

spot Reference Manual

0.3

Generated by Doxygen 1.4.0

Wed Jan 25 14:52:55 2006

Contents

1	spot Main Page	1
2	spot Module Index	2
3	spot Directory Hierarchy	3
4	spot Namespace Index	4
5	spot Hierarchical Index	4
6	spot Class Index	8
7	spot File Index	13
8	spot Page Index	17
9	spot Module Documentation	17
10	spot Directory Documentation	54
11	spot Namespace Documentation	61
12	spot Class Documentation	78
13	spot File Documentation	421
14	spot Page Documentation	500

1 spot Main Page

1.1 The Spot Library

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

See spot.lip6.fr for more information about this project.

1.2 This Document

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

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

1.3 Handy starting points

- [spot::ltl::formula](#) Base class for an LTL formulae.
- [spot::ltl::parse](#) Parsing a text string into a [spot::ltl::formula](#).
- [spot::tgba](#) Base class for Transition-based Generalized Büchi Automaton.
- [spot::ltl_to_tgba_fm](#) Convert a [spot::ltl::formula](#) into a [spot::tgba](#).
- [spot::tgba_product](#) On-the-fly product of two [spot::tgba](#).
- [spot::emptiness_check](#) Base class for all emptiness-check algorithms (see also module [Emptiness-checks](#))

2 spot Module Index

2.1 spot Modules

Here is a list of all modules:

LTL formulae	17
Essential LTL types	18
LTL Abstract Syntax Tree	18
LTL environments	19
Algorithms for LTL formulae	19
Input/Output of LTL formulae	19
Derivable visitors	22
Rewriting LTL formulae	23
Miscellaneous algorithms for LTL formulae	24
TGBA (Transition-based Generalized Büchi Automata)	17
Essential TGBA types	31
TGBA representations	32
TGBA algorithms	32
TGBA on-the-fly algorithms	33
Input/Output of TGBA	33
Decorating the dot output	42
Translating LTL formulae into TGBA	35
Algorithm patterns	37

TGBA simplifications	38
Miscellaneous algorithms on TGBA	40
Emptiness-checks	42
Emptiness-check algorithms for SSP	17
Emptiness-check algorithms	43
TGBA runs and supporting functions	51
Emptiness-check statistics	53
Miscellaneous helper algorithms	27
Hashing functions	29
Random functions	30

3 spot Directory Hierarchy

3.1 spot Directories

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

evtgba	54
evtgbalogs	54
evtgbaparse	55
gspn	55
ltlast	56
ltlenv	57
ltlparse	57
ltlvisit	57
misc	58
tgba	59
tgbaalogs	60
gtec	55
tgbaparse	61

4 spot Namespace Index

4.1 spot Namespace List

Here is a list of all namespaces with brief descriptions:

spot	61
spot::lfl	75

5 spot Hierarchical Index

5.1 spot Class Hierarchy

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

spot::barand< gen >	89
spot::bdd_less_than	107
spot::bfs_steps	108
spot::char_ptr_less_than	117
spot::lfl::const_visitor	124
spot::couvreur99_check_shy::successor	146
spot::couvreur99_check_shy::todo_item	147
spot::couvreur99_check_status	148
spot::delayed_simulation_relation	154
spot::direct_simulation_relation	154
spot::dotty_decorator	154
spot::tgba_run_dotty_decorator	371
spot::emptiness_check	169
spot::couvreur99_check	130
spot::couvreur99_check_shy	140
spot::emptiness_check_instantiator	172
spot::emptiness_check_result	174
spot::couvreur99_check_result	135
spot::lfl::environment	177
spot::lfl::declarative_environment	150

spot::ltl::default_environment	152
spot::evtgba	179
spot::evtgba_explicit	181
spot::evtgba_product	188
spot::evtgba_explicit::state	185
spot::evtgba_explicit::transition	186
spot::evtgba_iterator	187
spot::evtgba_reachable_iterator	191
spot::evtgba_reachable_iterator_breadth_first	194
spot::evtgba_reachable_iterator_depth_first	198
spot::explicit_connected_component_factory	203
spot::connected_component_hash_set_factory	123
spot::ltl::formula	204
spot::ltl::constant	125
spot::ltl::ref_formula	265
spot::ltl::atomic_prop	83
spot::ltl::binop	110
spot::ltl::multop	222
spot::ltl::unop	409
spot::ltl::formula_ptr_hash	207
spot::ltl::formula_ptr_less_than	208
spot::free_list	208
spot::bdd_allocator	90
spot::bdd_dict	95
spot::bdd_dict::annon_free_list	104
spot::gspn_exeption	211
spot::gspn_interface	212
spot::gspn_ssp_interface	214
spot::loopless_modular_mixed_radix_gray_code	216

spot::minato_isop	219
spot::minato_isop::local_vars	221
spot::lfl::multop::paircmp	229
spot::numbered_state_heap	230
spot::numbered_state_heap_hash_map	235
spot::numbered_state_heap_const_iterator	233
spot::numbered_state_heap_factory	234
spot::numbered_state_heap_hash_map_factory	238
spot::option_map	239
spot::ptr_hash< T >	261
spot::lfl::random_lfl	262
spot::lfl::random_lfl::op_proba	265
spot::rsymbol	269
spot::scc_stack	271
spot::scc_stack::connected_component	273
spot::explicit_connected_component	201
spot::connected_component_hash_set	120
spot::spoiler_node	277
spot::duplicator_node	157
spot::duplicator_node_delayed	161
spot::spoiler_node_delayed	279
spot::state	283
spot::state_bdd	285
spot::state_evtgba_explicit	287
spot::state_explicit	290
spot::state_product	292
spot::state_ptr_equal	295
spot::state_ptr_hash	296
spot::state_ptr_less_than	296

spot::string_hash	297
spot::symbol	297
spot::tgba	299
spot::tgba_bdd_concrete	306
spot::tgba_explicit	322
spot::tgba_reduc	354
spot::tgba_product	335
spot::tgba_tba_proxy	391
spot::tgba_sba_proxy	374
spot::tgba_bdd_core_data	316
spot::tgba_bdd_factory	321
spot::tgba_bdd_concrete_factory	312
spot::tgba_explicit::transition	331
spot::tgba_reachable_iterator	342
spot::tgba_reachable_iterator_breadth_first	346
spot::parity_game_graph	242
spot::parity_game_graph_delayed	247
spot::parity_game_graph_direct	253
spot::tgba_reduc	354
spot::tgba_reachable_iterator_depth_first	350
spot::tgba_run	369
spot::tgba_run::step	370
spot::tgba_statistics	380
spot::tgba_succ_iterator	380
spot::tgba_explicit_succ_iterator	332
spot::tgba_succ_iterator_concrete	383
spot::tgba_succ_iterator_product	387
spot::time_info	398
spot::timer	399

spot::timer_map	400
spot::unsigned_statistics	415
spot::ars_statistics	81
spot::acss_statistics	78
spot::couvreur99_check_result	135
spot::ec_statistics	165
spot::couvreur99_check	130
spot::unsigned_statistics_copy	416
spot::ltl::visitor	417
spot::ltl::clone_visitor	118
spot::ltl::simplify_f_g_visitor	274
spot::ltl::unabbreviate_logic_visitor	402
spot::ltl::unabbreviate_ltl_visitor	406
spot::ltl::postfix_visitor	259
spot::weight	419

6 spot Class Index

6.1 spot Class List

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

spot::acss_statistics (Accepting Cycle Search Space statistics)	78
spot::ars_statistics (Accepting Run Search statistics)	81
spot::ltl::atomic_prop (Atomic propositions)	83
spot::barand< gen > (Compute pseudo-random integer value between 0 and n included, following a binomial distribution for probability p)	89
spot::bdd_allocator (Manage ranges of variables)	90
spot::bdd_dict	95
spot::bdd_dict::annon_free_list	104
spot::bdd_less_than (Comparison functor for BDDs)	107
spot::bfs_steps (Make a BFS in a spot::tgba to compute a tgba_run::steps)	108
spot::ltl::binop (Binary operator)	110

spot::char_ptr_less_than (Strict Weak Ordering for <code>char*</code>)	117
spot::ltl::clone_visitor (Clone a formula)	118
spot::connected_component_hash_set	120
spot::connected_component_hash_set_factory (Factory for connected_component_hash_set)	123
spot::ltl::const_visitor (Formula visitor that cannot modify the formula)	124
spot::ltl::constant (A constant (True or False))	125
spot::couvereur99_check (An implementation of the Couvereur99 emptiness-check algorithm)	130
spot::couvereur99_check_result (Compute a counter example from a spot::couvereur99_check_status)	135
spot::couvereur99_check_shy (A version of spot::couvereur99_check that tries to visit known states first)	140
spot::couvereur99_check_shy::successor	146
spot::couvereur99_check_shy::todo_item	147
spot::couvereur99_check_status (The status of the emptiness-check on success)	148
spot::ltl::declarative_environment (A declarative environment)	150
spot::ltl::default_environment (A laxist environment)	152
spot::delayed_simulation_relation	154
spot::direct_simulation_relation	154
spot::dotty_decorator (Choose state and link styles for spot::dotty_reachable)	154
spot::duplicator_node (Duplicator node of parity game graph)	157
spot::duplicator_node_delayed (Duplicator node of parity game graph for delayed simulation)	161
spot::ec_statistics (Emptiness-check statistics)	165
spot::emptiness_check (Common interface to emptiness check algorithms)	169
spot::emptiness_check_instantiator	172
spot::emptiness_check_result (The result of an emptiness check)	174
spot::ltl::environment (An environment that describes atomic propositions)	177
spot::evtgba	179
spot::evtgba_explicit	181
spot::evtgba_explicit::state	185
spot::evtgba_explicit::transition (Explicit transitions (used by spot::evtgba_explicit))	186

spot::evtgba_iterator	187
spot::evtgba_product	188
spot::evtgba_reachable_iterator (Iterate over all reachable states of a spot::evtgba)	191
spot::evtgba_reachable_iterator_breadth_first (An implementation of spot::evtgba_reachable_iterator that browses states breadth first)	194
spot::evtgba_reachable_iterator_depth_first (An implementation of spot::evtgba_reachable_iterator that browses states depth first)	198
spot::explicit_connected_component (An SCC storing all its states explicitly)	201
spot::explicit_connected_component_factory (Abstract factory for explicit_connected_component)	203
spot::ltl::formula (An LTL formula)	204
spot::ltl::formula_ptr_hash (Hash Function for <code>const formula*</code>)	207
spot::ltl::formula_ptr_less_than (Strict Weak Ordering for <code>const formula*</code>)	208
spot::free_list (Manage list of free integers)	208
spot::gspn_exeption (An exeption used to forward GSPN errors)	211
spot::gspn_interface	212
spot::gspn_ssp_interface	214
spot::loopless_modular_mixed_radix_gray_code (Loopless modular mixed radix Gray code iteration)	216
spot::minato_isop (Generate an irredundant sum-of-products (ISOP) form of a BDD function)	219
spot::minato_isop::local_vars (Internal variables for minato_isop)	221
spot::ltl::multop (Multi-operand operators)	222
spot::ltl::multop::paircmp (Comparison functor used internally by ltl::multop)	229
spot::numbered_state_heap (Keep track of a large quantity of indexed states)	230
spot::numbered_state_heap_const_iterator (Iterator on numbered_state_heap objects)	233
spot::numbered_state_heap_factory (Abstract factory for numbered_state_heap)	234
spot::numbered_state_heap_hash_map (A straightforward implementation of numbered_state_heap with a hash map)	235
spot::numbered_state_heap_hash_map_factory (Factory for numbered_state_heap_hash_map)	238
spot::option_map (Manage a map of options)	239

spot::parity_game_graph (Parity game graph which compute a simulation relation)	242
spot::parity_game_graph_delayed	247
spot::parity_game_graph_direct (Parity game graph which compute the direct simulation relation)	253
spot::ltl::postfix_visitor (Apply an algorithm on each node of an AST, during a postfix traversal)	259
spot::ptr_hash< T > (A hash function for pointers)	261
spot::ltl::random_ltl (Generate random LTL formulae)	262
spot::ltl::random_ltl::op_proba	265
spot::ltl::ref_formula (A reference-counted LTL formula)	265
spot::rsymbol	269
spot::scc_stack	271
spot::scc_stack::connected_component	273
spot::ltl::simplify_f_g_visitor (Replace <code>true U f</code> and <code>false R g</code> by <code>F f</code> and <code>G g</code>)	274
spot::spoiler_node (Spoiler node of parity game graph)	277
spot::spoiler_node_delayed (Spoiler node of parity game graph for delayed simulation)	279
spot::state (Abstract class for states)	283
spot::state_bdd	285
spot::state_evtgba_explicit (States used by spot::tgba_evtgba_explicit)	287
spot::state_explicit	290
spot::state_product (A state for spot::tgba_product)	292
spot::state_ptr_equal (An Equivalence Relation for <code>state*</code>)	295
spot::state_ptr_hash (Hash Function for <code>state*</code>)	296
spot::state_ptr_less_than (Strict Weak Ordering for <code>state*</code>)	296
spot::string_hash (A hash function for strings)	297
spot::symbol	297
spot::tgba (A Transition-based Generalized Büchi Automaton)	299
spot::tgba_bdd_concrete (A concrete spot::tgba implemented using BDDs)	306
spot::tgba_bdd_concrete_factory (Helper class to build a spot::tgba_bdd_concrete object)	312
spot::tgba_bdd_core_data (Core data for a TGBA encoded using BDDs)	316

spot::tgba_bdd_factory (Abstract class for spot::tgba_bdd_concrete factories)	321
spot::tgba_explicit	322
spot::tgba_explicit::transition (Explicit transitions (used by spot::tgba_explicit))	331
spot::tgba_explicit_succ_iterator	332
spot::tgba_product (A lazy product. (States are computed on the fly.))	335
spot::tgba_reachable_iterator (Iterate over all reachable states of a spot::tgba)	342
spot::tgba_reachable_iterator_breadth_first (An implementation of spot::tgba_reachable_iterator that browses states breadth first)	346
spot::tgba_reachable_iterator_depth_first (An implementation of spot::tgba_reachable_iterator that browses states depth first)	350
spot::tgba_reduc	354
spot::tgba_run (An accepted run, for a tgba)	369
spot::tgba_run::step	370
spot::tgba_run_dotty_decorator (Highlight a spot::tgba_run on a spot::tgba)	371
spot::tgba_sba_proxy (Degeneralize a spot::tgba on the fly, producing an SBA)	374
spot::tgba_statistics	380
spot::tgba_succ_iterator (Iterate over the successors of a state)	380
spot::tgba_succ_iterator_concrete	383
spot::tgba_succ_iterator_product (Iterate over the successors of a product computed on the fly)	387
spot::tgba_tba_proxy (Degeneralize a spot::tgba on the fly, producing a TBA)	391
spot::time_info (A structure to record elapsed time in clock ticks)	398
spot::timer (A timekeeper that accumulate interval of time)	399
spot::timer_map (A map of timer, where each timer has a name)	400
spot::ltl::unabbreviate_logic_visitor (Clone and rewrite a formula to remove most of the abbreviated logical operators)	402
spot::ltl::unabbreviate_ltl_visitor (Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators)	406
spot::ltl::unop (Unary operators)	409
spot::unsigned_statistics	415
spot::unsigned_statistics_copy (Comparable statistics)	416
spot::ltl::visitor (Formula visitor that can modify the formula)	417

[spot::weight](#) (Manage for a given automaton a vector of counter indexed by its acceptance condition) [419](#)

7 spot File Index

7.1 spot File List

Here is a list of all files with brief descriptions:

gspn/common.hh	421
gspn/gspn.hh	422
gspn/spp.hh	423
evtgba/evtgba.hh	424
evtgba/evtgbaiter.hh	424
evtgba/explicit.hh	425
evtgba/product.hh	426
evtgba/symbol.hh	427
evtgbaalgos/dotty.hh	427
evtgbaalgos/reachiter.hh	429
evtgbaalgos/save.hh	431
evtgbaalgos/tgba2evtgba.hh	432
evtgbaparse/public.hh	433
ltlast/allnodes.hh (Define all LTL node types)	438
ltlast/atomic_prop.hh (LTL atomic propositions)	438
ltlast/binop.hh (LTL binary operators)	439
ltlast/constant.hh (LTL constants)	440
ltlast/formula.hh (LTL formula interface)	440
ltlast/multop.hh (LTL multi-operand operators)	442
ltlast/predecl.hh (Predeclare all LTL node types)	442
ltlast/refformula.hh (Reference-counted LTL formulae)	443
ltlast/unop.hh (LTL unary operators)	443
ltlast/visitor.hh (LTL visitor interface)	444
ltlenv/declenv.hh	445

ltlenv/defaultenv.hh	445
ltlenv/environment.hh	446
ltlparse/public.hh	435
ltlvisit/apcollect.hh	446
ltlvisit/basicreduce.hh	447
ltlvisit/clone.hh	448
ltlvisit/destroy.hh	449
ltlvisit/dotty.hh	428
ltlvisit/dump.hh	449
ltlvisit/length.hh	450
ltlvisit/lunabbrev.hh	450
ltlvisit/nenoform.hh	451
ltlvisit/postfix.hh	451
ltlvisit/randomltl.hh	452
ltlvisit/reduce.hh	452
ltlvisit/simpfg.hh	453
ltlvisit/syntimpl.hh	453
ltlvisit/tostring.hh	454
ltlvisit/tunabbrev.hh	455
misc/bareword.hh	455
misc/bddalloc.hh	456
misc/bddlt.hh	456
misc/escape.hh	457
misc/freelist.hh	457
misc/hash.hh	458
misc/hashfunc.hh	458
misc/ltstr.hh	459
misc/minato.hh	460
misc/modgray.hh	460

misc/optionmap.hh	460
misc/random.hh	461
misc/timer.hh	462
misc/version.hh	462
tgba/bdddict.hh	462
tgba/bddprint.hh	464
tgba/formula2bdd.hh	468
tgba/public.hh	436
tgba/state.hh	468
tgba/statebdd.hh	469
tgba/succiter.hh	470
tgba/succiterconcrete.hh	471
tgba/tgba.hh	471
tgba/tgababddconcrete.hh	473
tgba/tgababddconcretefactory.hh	473
tgba/tgababddconcreteproduct.hh	474
tgba/tgababddcoredata.hh	475
tgba/tgababddfactory.hh	475
tgba/tgbaexplicit.hh	476
tgba/tgbaproduct.hh	477
tgba/tgbareduc.hh	478
tgba/tgbatba.hh	480
tgbaalgos/bfssteps.hh	480
tgbaalgos/dotty.hh	429
tgbaalgos/dottydec.hh	481
tgbaalgos/dupeexp.hh	481
tgbaalgos/emptiness.hh	482
tgbaalgos/emptiness_stats.hh	483
tgbaalgos/gv04.hh	489

tgbaalgos/lbtt.hh	489
tgbaalgos/ltl2tgba_fm.hh	490
tgbaalgos/ltl2tgba_lacim.hh	490
tgbaalgos/magic.hh	491
tgbaalgos/neverclaim.hh	492
tgbaalgos/powerset.hh	492
tgbaalgos/projrun.hh	493
tgbaalgos/randomgraph.hh	494
tgbaalgos/reachiter.hh	430
tgbaalgos/reducerun.hh	494
tgbaalgos/reductgba_sim.hh	494
tgbaalgos/replayrun.hh	496
tgbaalgos/rundotdec.hh	496
tgbaalgos/save.hh	432
tgbaalgos/se05.hh	497
tgbaalgos/stats.hh	498
tgbaalgos/tau03.hh	498
tgbaalgos/tau03opt.hh	499
tgbaalgos/weight.hh	499
tgbaalgos/gtec/ce.hh	484
tgbaalgos/gtec/explscc.hh	485
tgbaalgos/gtec/gtec.hh	485
tgbaalgos/gtec/nsheap.hh	486
tgbaalgos/gtec/scctestack.hh	487
tgbaalgos/gtec/status.hh	488
tgbaalgos/public.hh	436

8 spot Page Index

8.1 spot Related Pages

Here is a list of all related documentation pages:

Bug List

[500](#)

9 spot Module Documentation

9.1 LTL formulae

Modules

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

9.1.1 Detailed Description

This module gathers types and definitions related to LTL formulae.

9.2 TGBA (Transition-based Generalized Büchi Automata)

Modules

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

9.2.1 Detailed Description

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

9.3 Emptiness-check algorithms for SSP

Functions

- `couvreur99_check * spot::couvreur99_check_ssp_semi (const tgba *ssp_automata)`
- `couvreur99_check * spot::couvreur99_check_ssp_shy_semi (const tgba *ssp_automata)`
- `couvreur99_check * spot::couvreur99_check_ssp_shy (const tgba *ssp_automata)`

9.3.1 Function Documentation

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

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

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

9.4 Essential LTL types

Classes

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

Functions

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

9.4.1 Function Documentation

9.4.1.1 `formula* clone (const formula *f)`

Clone a formula.

9.4.1.2 `void destroy (const formula *f)`

Destroys a formula.

9.5 LTL Abstract Syntax Tree

Classes

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

- class `spot::ltl::formula`
An LTL formula.
- class `spot::ltl::multop`
Multi-operand operators.
- class `spot::ltl::ref_formula`
A reference-counted LTL formula.
- class `spot::ltl::unop`
Unary operators.

9.6 LTL environments

Classes

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

9.6.1 Detailed Description

LTL environment implementations.

9.7 Algorithms for LTL formulae

Modules

- [Input/Output of LTL formulae](#)
- [Derivable visitors](#)
- [Rewriting LTL formulae](#)
- [Miscellaneous algorithms for LTL formulae](#)

9.8 Input/Output of LTL formulae

Classes

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

Typedefs

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

Functions

- formula * [spot::ltl::parse](#) (const std::string <l_string, [parse_error_list](#) &error_list, environment &env=default_environment::instance(), bool debug=false)
Build a formula from an LTL string.
- bool [spot::ltl::format_parse_errors](#) (std::ostream &os, const std::string <l_string, [parse_error_list](#) &error_list)
Format diagnostics produced by [spot::ltl::parse](#).
- std::ostream & [spot::ltl::dotty](#) (std::ostream &os, const formula *f)
Write a formula tree using dot's syntax.
- std::ostream & [spot::ltl::dump](#) (std::ostream &os, const formula *f)
Dump a formula tree.
- std::ostream & [spot::ltl::to_string](#) (const formula *f, std::ostream &os)
Output a formula as a (parsable) string.
- std::string [spot::ltl::to_string](#) (const formula *f)
Convert a formula into a (parsable) string.
- std::ostream & [spot::ltl::to_spin_string](#) (const formula *f, std::ostream &os)
Output a formula as a (parsable by Spin) string.
- std::string [spot::ltl::to_spin_string](#) (const formula *f)
Convert a formula into a (parsable by Spin) string.

9.8.1 Typedef Documentation

9.8.1.1 typedef std::pair<yy::location, std::string> [spot::ltl::parse_error](#)

A parse diagnostic with its location.

9.8.1.2 typedef std::list<[parse_error](#)> [spot::ltl::parse_error_list](#)

A list of parser diagnostics, as filled by parse.

9.8.2 Function Documentation

9.8.2.1 `std::ostream& dotty (std::ostream & os, const formula *f)`

Write a formula tree using dot's syntax.

Parameters:

os The stream where it should be output.

f The formula to translate.

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

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

Dump a formula tree.

Parameters:

os The stream where it should be output.

f The formula to dump.

This is useful to display a formula when debugging.

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

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

Parameters:

os Where diagnostics should be output.

ltl_string The string that were parsed.

error_list The error list filled by `spot::ltl::parse` while parsing *ltl_string*.

Returns:

`true` iff any diagnostic was output.

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

Build a formula from an LTL string.

Parameters:

ltl_string The string to parse.

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

env The environment into which parsing should take place.

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

Returns:

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

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered 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.

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

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

Parameters:

f The formula to translate.

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

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

Parameters:

f The formula to translate.

os The stream where it should be output.

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

Convert a formula into a (parsable) string.

Parameters:

f The formula to translate.

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

Output a formula as a (parsable) string.

Parameters:

f The formula to translate.

os The stream where it should be output.

9.9 Derivable visitors

Classes

- class [spot::ltl::clone_visitor](#)
Clone a formula.
- class [spot::ltl::unabbreviate_logic_visitor](#)
Clone and rewrite a formula to remove most of the abbreviated logical operators.
- class [spot::ltl::postfix_visitor](#)

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

- class `spot::ltl::simplify_f_g_visitor`
Replace `true U f` and `false R g` by `F f` and `G g`.
- class `spot::ltl::unabbreviate_ltl_visitor`
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

9.10 Rewriting LTL formulae

Enumerations

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

Functions

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

9.10.1 Enumeration Type Documentation

9.10.1.1 enum `reduce_options`

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

Enumeration values:

Reduce_None No reduction.

Reduce_Basics Basic reductions.

Reduce_Syntactic_Implications Somenzi & Bloem syntactic implication.

Reduce_Eventuality_And_Universality Etessami & Holzmann eventuality and universality reductions.

Reduce_All All reductions.

9.10.2 Function Documentation

9.10.2.1 formula* basic_reduce (const formula *f)

Basic rewritings.

9.10.2.2 formula* negative_normal_form (const formula *f, bool negated = false)

Build the negative normal form of f .

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

Parameters:

f The formula to normalize.

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

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

9.10.2.3 formula* reduce (const formula *f, int opt = Reduce_All)

Reduce a formula f .

Parameters:

f the formula to reduce

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

Returns:

the reduced formula

9.10.2.4 formula* simplify_f_g (const formula *f)

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

9.10.2.5 formula* unabbreviate_logic (const formula *f)

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

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

9.11 Miscellaneous algorithms for LTL formulae

Typedefs

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

Functions

- `atomic_prop_set * spot::ltl::atomic_prop_collect` (const formula *f, atomic_prop_set *s=0)
Return the set of atomic propositions occurring in a formula.
- bool `spot::ltl::is_GF` (const formula *f)
Whether a formula starts with GF.
- bool `spot::ltl::is_FG` (const formula *f)
Whether a formula starts with FG.
- int `spot::ltl::length` (const formula *f)
Compute the length of a formula.
- bool `spot::ltl::is_eventual` (const formula *f)
Check whether a formula is a pure eventuality.
- bool `spot::ltl::is_universal` (const formula *f)
Check whether a formula is purely universal.
- bool `spot::ltl::syntactic_implication` (const formula *f1, const formula *f2)
Syntactic implication.
- bool `spot::ltl::syntactic_implication_neg` (const formula *f1, const formula *f2, bool right)
Syntactic implication.

9.11.1 Typedef Documentation

9.11.1.1 `typedef std::set<atomic_prop*, formula_ptr_less_than> spot::ltl::atomic_prop_set`

Set of atomic propositions.

9.11.2 Function Documentation

9.11.2.1 `atomic_prop_set* atomic_prop_collect` (const formula *f, atomic_prop_set *s = 0)

Return the set of atomic propositions occurring in a formula.

Parameters:

f the formula to inspect

s an existing set to fill with atomic_propositions discovered, or 0 if the set should be allocated by the function.

Returns:

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

9.11.2.2 bool is_eventual (const formula *f)

Check whether a formula is a pure eventuality.

Pure eventuality formulae are defined in

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

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

9.11.2.3 bool is_FG (const formula *f)

Whether a formula starts with FG.

9.11.2.4 bool is_GF (const formula *f)

Whether a formula starts with GF.

9.11.2.5 bool is_universal (const formula *f)

Check whether a formula is purely universal.

Purely universal formulae are defined in

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

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

9.11.2.6 int length (const formula *f)

Compute the length of a formula.

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

9.11.2.7 bool syntactic_implication (const formula *f1, const formula *f2)

Syntactic implication.

This comes from

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

9.11.2.8 bool syntactic_implication_neg (const formula *f1, const formula *f2, bool right)

Syntactic implication.

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

See also:

[syntactic_implication](#)

9.12 Miscellaneous helper algorithms

Whether a word is bare.

Modules

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

Classes

- class [spot::bdd_allocator](#)
Manage ranges of variables.
- struct [spot::bdd_less_than](#)
Comparison functor for BDDs.
- class [spot::free_list](#)
Manage list of free integers.
- struct [spot::char_ptr_less_than](#)
Strict Weak Ordering for char.*
- class [spot::minato_isop](#)
Generate an irredundant sum-of-products (ISOP) form of a BDD function.

- class `spot::loopless_modular_mixed_radix_gray_code`
Loopless modular mixed radix Gray code iteration.
- class `spot::option_map`
Manage a map of options.
- struct `spot::time_info`
A structure to record elapsed time in clock ticks.
- class `spot::timer`
A timekeeper that accumulate interval of time.
- class `spot::timer_map`
A map of timer, where each timer has a name.

Functions

- bool `spot::is_bare_word` (const char *str)
- std::string `spot::quote_unless_bare_word` (const std::string &str)
Double-quote words that are not bare.
- std::ostream & `spot::escape_str` (std::ostream &os, const std::string &str)
Escape " and \ characters in str.
- std::string `spot::escape_str` (const std::string &str)
Escape " and \ characters in str.
- const char * `spot::version` ()
Return Spot's version.

9.12.1 Detailed Description

Whether a word is bare.

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

9.12.2 Function Documentation

9.12.2.1 std::string escape_str (const std::string & str)

Escape " and \ characters in *str*.

9.12.2.2 std::ostream& escape_str (std::ostream & os, const std::string & str)

Escape " and \ characters in *str*.

9.12.2.3 `bool is_bare_word (const char * str)`

9.12.2.4 `std::string quote_unless_bare_word (const std::string & str)`

Double-quote words that are not bare.

See also:

[is_bare_word](#)

9.12.2.5 `const char* version ()`

Return Spot's version.

9.13 Hashing functions

Classes

- struct [spot::ltl::formula_ptr_hash](#)
Hash Function for `const formula`.*
- struct [spot::ptr_hash< T >](#)
A hash function for pointers.
- struct [spot::string_hash](#)
A hash function for strings.
- struct [spot::state_ptr_hash](#)
Hash Function for `state`.*

Functions

- `size_t spot::wang32_hash (size_t key)`
Thomas Wang's 32 bit hash function.
- `size_t spot::knuth32_hash (size_t key)`
Knuth's Multiplicative hash function.

9.13.1 Function Documentation

9.13.1.1 `size_t knuth32_hash (size_t key)` `[inline]`

Knuth's Multiplicative hash function.

This function is suitable for hashing values whose high order bits do not vary much (ex. addresses of memory objects). Prefer [spot::wang32_hash\(\)](#) otherwise.
<http://www.concentric.net/~Ttwang/tech/addrhash.htm>

9.13.1.2 `size_t wang32_hash (size_t key)` [inline]

Thomas Wang's 32 bit hash function.

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

9.14 Random functions

Classes

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

Functions

- void `spot::srand` (unsigned int seed)
Reset the seed of the pseudo-random number generator.
- int `spot::rrand` (int min, int max)
Compute a pseudo-random integer value between min and max included.
- int `spot::mrnd` (int max)
Compute a pseudo-random integer value between 0 and max-1 included.
- double `spot::drand` ()
Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).
- double `spot::nrnd` ()
Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).
- double `spot::bmrand` ()
Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).
- int `spot::prand` (double p)
Return a pseudo-random positive integer value following a Poisson distribution with parameter p.

9.14.1 Function Documentation

9.14.1.1 `double bmrand` ()

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

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

9.14.1.2 double drand ()

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

See also:

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

9.14.1.3 int mrand (int *max*)

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

See also:

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

9.14.1.4 double nrand ()

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

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

9.14.1.5 int prand (double *p*)

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

Precondition:

$p > 0$

9.14.1.6 int rrand (int *min*, int *max*)

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

See also:

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

9.14.1.7 void srand (unsigned int *seed*)

Reset the seed of the pseudo-random number generator.

See also:

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

9.15 Essential TGBA types**Classes**

- class [spot::bdd_dict](#)
- class [spot::state](#)

Abstract class for states.

- struct `spot::state_ptr_less_than`
Strict Weak Ordering for `state`.*
- struct `spot::state_ptr_equal`
An Equivalence Relation for `state`.*
- struct `spot::state_ptr_hash`
Hash Function for `state`.*
- class `spot::tgba_succ_iterator`
Iterate over the successors of a state.
- class `spot::tgba`
A Transition-based Generalized Büchi Automaton.

9.16 TGBA representations

Classes

- class `spot::state_bdd`
- class `spot::tgba_succ_iterator_concrete`
- class `spot::tgba_bdd_concrete`
A concrete `spot::tgba` implemented using BDDs.
- class `spot::tgba_explicit`
- class `spot::state_explicit`
- class `spot::tgba_explicit_succ_iterator`
- class `spot::tgba_reduc`

9.17 TGBA algorithms

Modules

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

Functions

- `tgba_bdd_concrete * spot::product (const tgba_bdd_concrete *left, const tgba_bdd_concrete *right)`
Multiplies two `spot::tgba_bdd_concrete` automata.

9.17.1 Function Documentation

9.17.1.1 `tgba_bdd_concrete* product (const tgba_bdd_concrete * left, const tgba_bdd_concrete * right)`

Multiplies two `spot::tgba_bdd_concrete` automata.

This function builds the resulting product as another `spot::tgba_bdd_concrete` automaton.

9.18 TGBA on-the-fly algorithms

Classes

- class `spot::state_product`
A state for `spot::tgba_product`.
- class `spot::tgba_tba_proxy`
Degeneralize a `spot::tgba` on the fly, producing a TBA.
- class `spot::tgba_sba_proxy`
Degeneralize a `spot::tgba` on the fly, producing an SBA.

9.19 Input/Output of TGBA

Modules

- `Decorating the dot output`

Typedefs

- typedef `std::pair< yy::location, std::string >` `spot::tgba_parse_error`
A parse diagnostic with its location.
- typedef `std::list< tgba_parse_error >` `spot::tgba_parse_error_list`
A list of parser diagnostics, as filled by parse.

Functions

- `std::ostream & spot::dotty_reachable (std::ostream &os, const tgba *g, dotty_decorator *dd=dotty_decorator::instance())`
Print reachable states in dot format.
- `std::ostream & spot::lbt_reachable (std::ostream &os, const tgba *g)`
Print reachable states in LBTT format.
- `std::ostream & spot::never_claim_reachable (std::ostream &os, const tgba_sba_proxy *g, const ltl::formula *f=0)`
Print reachable states in Spin never claim format.

- `std::ostream & spot::tgba_save_reachable` (`std::ostream &os`, `const tgba *g`)
Save reachable states in text format.
- `tgba_explicit * spot::tgba_parse` (`const std::string &filename`, `tgba_parse_error_list &error_list`, `bdd_dict *dict`, `ltl::environment &env=ltl::default_environment::instance()`, `bool debug=false`)
Build a `spot::tgba_explicit` from a text file.
- `bool spot::format_tgba_parse_errors` (`std::ostream &os`, `const std::string &filename`, `tgba_parse_error_list &error_list`)
Format diagnostics produced by `spot::tgba_parse`.

9.19.1 Typedef Documentation

9.19.1.1 `typedef std::pair<yy::location, std::string> spot::tgba_parse_error`

A parse diagnostic with its location.

9.19.1.2 `typedef std::list<tgba_parse_error> spot::tgba_parse_error_list`

A list of parser diagnostics, as filled by parse.

9.19.2 Function Documentation

9.19.2.1 `std::ostream& dotty_reachable` (`std::ostream & os`, `const tgba * g`, `dotty_decorator * dd = dotty_decorator::instance()`)

Print reachable states in dot format.

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

9.19.2.2 `bool format_tgba_parse_errors` (`std::ostream & os`, `const std::string & filename`, `tgba_parse_error_list & error_list`)

Format diagnostics produced by `spot::tgba_parse`.

Parameters:

- `os` Where diagnostics should be output.
- `filename` The filename that should appear in the diagnostics.
- `error_list` The error list filled by `spot::ltl::parse` while parsing `ltl_string`.

Returns:

`true` iff any diagnostic was output.

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

Print reachable states in LBTT format.

Parameters:

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

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

Print reachable states in Spin never claim format.

Parameters:

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

9.19.2.5 `tgba_explicit* tgba_parse (const std::string & filename, tgba_parse_error_list & error_list, bdd_dict * dict, ltl::environment & env = ltl::default_environment::instance(), bool debug = false)`

Build a `spot::tgba_explicit` from a text file.

Parameters:

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

Returns:

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

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered 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.

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

Save reachable states in text format.

9.20 Translating LTL formulae into TGBA

Functions

- `tgba_explicit * spot::ltl_to_tgba_fm (const ltl::formula * f, bdd_dict * dict, bool exprop=false, bool symb_merge=true, bool branching_postponement=false, bool fair_loop_approx=false, const ltl::atomic_prop_set * unobs=0, int reduce_ltl=ltl::Reduce_None)`
Build a `spot::tgba_explicit` from an LTL formula.
- `tgba_bdd_concrete * spot::ltl_to_tgba_lacim (const ltl::formula * f, bdd_dict * dict)`
Build a `spot::tgba_bdd_concrete` from an LTL formula.

9.20.1 Function Documentation

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

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

This is based on the following paper.

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

Parameters:

f The formula to translate into an automaton.

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

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

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

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

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

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

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

reduce_ltl If this parameter is set, the LTL formulae representing each state of the automaton will be simplified using `spot::ltl::reduce()` before computing the successor. *reduce_ltl* should specify the type of reduction to apply as documented for `spot::ltl::reduce()`. This idea is taken from the following paper.

```
@InProceedings{ thirioux.02.fmics,
  author   = {Xavier Thirioux},
  title    = {Simple and Efficient Translation from {LTL} Formulas to
              {B\"u}chi Automata},
  booktitle = {Proceedings of the 7th International ERCIM Workshop in
              Formal Methods for Industrial Critical Systems (FMICS'02)},
  series   = {Electronic Notes in Theoretical Computer Science},
  volume   = {66(2)},
  publisher = {Elsevier},
  editor    = {Rance Cleaveland and Hubert Garavel},
  year     = {2002},
  month    = jul,
  address  = {M{\`a}laga, Spain}
}
```

Returns:

A `spot::tgba_explicit` that recognizes the language of *f*.

9.20.1.2 tgba_bdd_concrete* ltl_to_tgba_lacim (const ltl::formula *f, bdd_dict *dict)

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

This is based on the following paper.

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

Parameters:

f The formula to translate into an automaton.

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

Returns:

A `spot::tgba_bdd_concrete` that recognizes the language of *f*.

9.21 Algorithm patterns

Classes

- class `spot::tgba_reachable_iterator`

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

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

9.22 TGBA simplifications

Classes

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

Typedefs

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

Enumerations

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

Functions

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

9.22.1 Typedef Documentation

9.22.1.1 `typedef std::vector<duplicator_node*> spot::dn_v`

9.22.1.2 `typedef std::vector<const state*> spot::s_v`

9.22.1.3 `typedef std::vector<spoiler_node*> spot::sn_v`

9.22.2 Enumeration Type Documentation

9.22.2.1 `enum reduce_tgba_options`

Options for reduce.

Enumeration values:

Reduce_None No reduction.

Reduce_quotient_Dir_Sim Reduction of state using direct simulation relation.

Reduce_transition_Dir_Sim Reduction of transitions using direct simulation relation.

Reduce_quotient_Del_Sim Reduction of state using delayed simulation relation.

Reduce_transition_Del_Sim Reduction of transition using delayed simulation relation.

Reduce_Scc Reduction using SCC.

Reduce_All All reductions.

9.22.3 Function Documentation

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

To free a simulation relation.

9.22.3.2 void free_relation_simulation (direct_simulation_relation * rel)

To free a simulation relation.

9.22.3.3 delayed_simulation_relation* get_delayed_relation_simulation (const tgba * a, std::ostream & os, int opt = -1)

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

Bug

Does not work for generalized automata.

9.22.3.4 direct_simulation_relation* get_direct_relation_simulation (const tgba * a, std::ostream & os, int opt = -1)

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

9.22.3.5 tgba* reduc_tgba_sim (const tgba * a, int opt = Reduce_All)

Remove some node of the automata using a simulation relation.

Parameters:

a the automata to reduce.

opt a conjunction of [spot::reduce_tgba_options](#) specifying which optimizations to apply.

Returns:

the reduced automata.

9.23 Miscellaneous algorithms on TGBA**Classes**

- class [spot::bfs_steps](#)
Make a BFS in a [spot::tgba](#) to compute a [tgba_run::steps](#).
- struct [spot::tgba_statistics](#)

Functions

- tgba_explicit * [spot::tgba_dupexp_bfs](#) (const tgba * aut)
Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.
- tgba_explicit * [spot::tgba_dupexp_dfs](#) (const tgba * aut)
Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.
- tgba_explicit * [spot::tgba_powerset](#) (const tgba * aut)
Build a deterministic automaton, ignoring acceptance conditions.
- tgba * [spot::random_graph](#) (int n, float d, const [ltl::atomic_prop_set](#) * ap, bdd_dict * dict, int n_acc=0, float a=0.1, float t=0.5, [ltl::environment](#) * env=&[ltl::default_environment::instance\(\)](#))

Construct a tgba randomly.

- tgba_statistics [spot::stats_reachable](#) (const tgba *g)

Compute statistics for an automaton.

9.23.1 Function Documentation

9.23.1.1 tgba* random_graph (int *n*, float *d*, const [ltl::atomic_prop_set](#) * *ap*, bdd_dict * *dict*, int *n_acc* = 0, float *a* = 0.1, float *t* = 0.5, [ltl::environment](#) * *env* = <ltl::default_environment::instance())

Construct a tgba randomly.

Parameters:

- n* The number of states wanted in the automata (>0). All states will be connected, and there will be no dead state.
- d* The density of the automata. This is the probability (between 0.0 and 1.0), to add a transition between two states. All states have at least one outgoing transition, so *d* is considered only when adding the remaining transition. A density of 1 means all states will be connected to each other.
- ap* The list of atomic property that should label the transition.
- dict* The [bdd_dict](#) to used for this automata.
- n_acc* The number of acceptance sets to use.
- a* The probability (between 0.0 and 1.0) that a transition belongs to an acceptance set.
- t* The probability (between 0.0 and 1.0) that an atomic proposition is true.
- env* The environment in which to declare the acceptance conditions.

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

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

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

9.23.1.2 tgba_statistics stats_reachable (const tgba *g)

Compute statistics for an automaton.

9.23.1.3 tgba_explicit* tgba_dupexp_bfs (const tgba * aut)

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

9.23.1.4 tgba_explicit* tgba_dupexp_dfs (const tgba * aut)

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

9.23.1.5 tgba_explicit* tgba_powerset (const tgba * aut)

Build a deterministic automaton, ignoring acceptance conditions.

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

9.24 Decorating the dot output**Classes**

- class [spot::dotty_decorator](#)
Choose state and link styles for [spot::dotty_reachable](#).
- class [spot::tgba_run_dotty_decorator](#)
Highlight a [spot::tgba_run](#) on a [spot::tgba](#).

9.25 Emptiness-checks**Modules**

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

Classes

- class [spot::emptiness_check_result](#)
The result of an emptiness check.
- class [spot::emptiness_check](#)
Common interface to emptiness check algorithms.
- class [spot::emptiness_check_instantiator](#)

9.25.1 Detailed Description

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

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

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

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

9.26 Emptiness-check algorithms

Classes

- class `spot::couvreur99_check`
An implementation of the Couvreur99 emptiness-check algorithm.
- class `spot::couvreur99_check_shy`
A version of `spot::couvreur99_check` that tries to visit known states first.

Functions

- `emptiness_check * spot::couvreur99 (const tgba *a, option_map options=option_map(), const numbered_state_heap_factory *nshf=numbered_state_heap_hash_map_factory::instance())`
Check whether the language of an automate is empty.
- `emptiness_check * spot::explicit_gv04_check (const tgba *a, option_map o=option_map())`
Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.
- `emptiness_check * spot::explicit_magic_search (const tgba *a, option_map o=option_map())`
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * spot::bit_state_hashing_magic_search (const tgba *a, size_t size, option_map o=option_map())`
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * spot::magic_search (const tgba *a, option_map o=option_map())`
Wrapper for the two magic_search implementations.
- `emptiness_check * spot::explicit_se05_search (const tgba *a, option_map o=option_map())`
Returns an emptiness check on the `spot::tgba` automaton a.
- `emptiness_check * spot::bit_state_hashing_se05_search (const tgba *a, size_t size, option_map o=option_map())`
Returns an emptiness checker on the `spot::tgba` automaton a.

- `emptiness_check * spot::se05` (const tgba *a, option_map o)
Wrapper for the two se05 implementations.
- `emptiness_check * spot::explicit_tau03_search` (const tgba *a, option_map o=option_map())
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * spot::explicit_tau03_opt_search` (const tgba *a, option_map o=option_map())
Returns an emptiness checker on the `spot::tgba` automaton a.

9.26.1 Function Documentation

9.26.1.1 `emptiness_check* bit_state_hashing_magic_search` (const tgba *a, size_t size, option_map o = option_map())

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

Precondition:

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

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

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

Consequently, the detection of an acceptance cycle is not ensured.

The size of the heap is limited to
size bytes.

The implemented algorithm is the same as the one of `spot::explicit_magic_search`.

See also:

`spot::explicit_magic_search`

9.26.1.2 `emptiness_check* bit_state_hashing_se05_search` (const tgba *a, size_t size, option_map o = option_map())

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

Precondition:

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

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

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

Consequently, the detection of an acceptance cycle is not ensured.

The size of the heap is limited to

size bytes.

The implemented algorithm is the same as the one of [spot::explicit_se05_search](#).

See also:

[spot::explicit_se05_search](#)

9.26.13 emptiness_check* **couvreur99** (**const tgba * a**, **option_map options** = **option_map()**, **const numbered_state_heap_factory * nshf** = **numbered_state_heap_hash_map_factory::instance()**)

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}
}
```

A recursive definition of the algorithm would look as follows, but the implementation is of course not recursive. ($\langle \Sigma, Q, \delta, q, F \rangle$ is the automaton to check, H is an associative array mapping each state to its positive DFS order or 0 if it is dead, SCC is and ACC are two stacks.)

```
check(<Sigma, Q, delta, q, F>, H, SCC, ACC)
  if q is not in H // new state
    H[q] = H.size + 1
    SCC.push(<H[q], {}>)
    forall <a, s> : <q, _, a, s> in delta
      ACC.push(a)
      res = check(<Sigma, Q, delta, s, F>, H, SCC, ACC)
      if res
        return res
    <n, _> = SCC.top()
    if n = H[q]
      SCC.pop()
```

```

        mark_reachable_states_as_dead(<Sigma, Q, delta, q, F>, H$)
    return 0
else
    if H[q] = 0 // dead state
        ACC.pop()
        return true
    else // state in stack: merge SCC
        all = {}
        do
            <n, a> = SCC.pop()
            all = all union a union { ACC.pop() }
        until n <= H[q]
        SCC.push(<n, all>)
        if all != F
            return 0
        return new emptiness_check_result(necessary data)

```

check() returns 0 iff the automaton's language is empty. It returns an instance of `emptiness_check_result`. If the automaton accept a word. (Use `emptiness_check_result::accepting_run()` to extract an accepting run.)

There are two variants of this algorithm: `spot::couvreur99_check` and `spot::couvreur99_check_shy`. They differ in their memory usage, the number for successors computed before they are used and the way the depth first search is directed.

`spot::couvreur99_check` performs a straightforward depth first search. The DFS stacks store `tgba_succ_` iterators, so that only the iterators which really are explored are computed.

`spot::couvreur99_check_shy` tries to explore successors which are visited states first. this helps to merge SCCs and generally helps to produce shorter counter-examples. However this algorithm cannot stores unprocessed successors as `tgba_succ_iterators`: it must compute all successors of a state at once in order to decide which to explore first, and must keep a list of all unexplored successors in its DFS stack.

The `couvreur99()` function is a wrapper around these two flavors of the algorithm. *options* is an option map that specifies which algorithms should be used, and how.

The following options are available.

- "shy" : if non zero, then `spot::couvreur99_check_shy` is used, otherwise (and by default) `spot::couvreur99_check` is used.
- "poprem" : specifies how the algorithm should handle the destruction of non-accepting maximal strongly connected components. If `poprem` is non null, the algorithm will keep a list of all states of a SCC that are fully processed and should be removed once the MSCC is popped. If `poprem` is null (the default), the MSCC will be traversed again (i.e. generating the successors of the root recursively) for deletion. This is a choice between memory and speed.
- "group" : this options is used only by `spot::couvreur99_check_shy`. If non null (the default), the successors of all the states that belong to the same SCC will be considered when choosing a successor. Otherwise, only the successor of the topmost state on the DFS stack are considered.

9.26.1.4 emptiness_check* explicit_gv04_check (const tgba * a, option_map o = option_map())

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

Precondition:

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

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

```
@InProceedings{geldenhuys.04.tacas,
  author = {Jaco Geldenhuys and Antti Valmari},
  title = {Tarjan's Algorithm Makes On-the-Fly {LTL} Verification
    More Efficient},
  booktitle = {Proceedings of the 10th International Conference on Tools
    and Algorithms for the Construction and Analysis of Systems
    (TACAS'04)},
  editor = {Kurt Jensen and Andreas Podelski},
  pages = {205--219},
  year = {2004},
  publisher = {Springer-Verlag},
  series = {Lecture Notes in Computer Science},
  volume = {2988},
  isbn = {3-540-21299-X}
}
```

9.26.1.5 `emptiness_check* explicit_magic_search (const tgba * a, option_map o = option_map())`

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

Precondition:

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

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

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

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

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



```
end;
```

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

```
Article{      courcoubetis.92.fmsd,
  author      = {Costas Courcoubetis and Moshe Y. Vardi and Pierre
                 Wolper and Mihalis Yannakakis},
  title       = {Memory-Efficient Algorithm for the Verification of
                 Temporal Properties},
  journal     = {Formal Methods in System Design},
  pages       = {275--288},
  year       = {1992},
  volume     = {1}
}
```

Bug

The name is misleading. Magic-search is the algorithm from `godefroid.93.pstv`, not `courcoubetis.92.fmsd`.

9.26.1.6 `emptiness_check* explicit_se05_search (const tgba * a, option_map o = option_map())`

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

Precondition:

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

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

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

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

procedure dfs_red(s)
begin
  if s.color == cyan then
```

```

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

```

It is an adaptation to TBA of the one presented in

```

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

```

See also:

[spot::explicit_magic_search](#)

9.26.1.7 emptiness_check* explicit_tau03_opt_search (const tgba * *a*, option_map *o* = option_map())

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

Precondition:

The automaton *a* must have at least one acceptance condition.

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

```

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

procedure dfs_blue (s)
begin
    s.color = cyan;
    s.acc = emptyset;
    s.weight = weight;
    for all t in post(s) do
        let (s, l, a, t) be the edge from s to t;
        if t.color == white then
            for all b in a do
                weight[b] = weight[b] + 1;
            end for;
            call dfs_blue(t);
            for all b in a do
                weight[b] = weight[b] - 1;
            end for;
        end if;
    end for;
end;

```

```

        end for;
    end if;
    Acc = s.acc U a;
    if t.color == cyan &&
        (Acc U support(weight - t.weight) U t.acc) == all_acc then
        report a cycle;
    else if Acc not included in t.acc then
        t.acc := t.acc U Acc;
        call dfs_red(t, Acc);
    end if;
end for;
s.color = blue;
end;

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

```

This algorithm is a generalisation to TGBA of the one implemented in [spot::explicit_se05_search](#). It is based on the acceptance set labelling of states used in [spot::explicit_tau03_search](#). Moreover, it introduces a slight optimisation based on vectors of integers counting for each acceptance condition how many times the condition has been visited in the path stored in the blue stack. Such a vector is associated to each state of this stack.

9.26.1.8 emptiness_check* explicit_tau03_search (const tgba *a, option_map o = option_map())

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

Precondition:

The automaton *a* must have at least one acceptance condition.

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

```

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

procedure dfs_blue (s)
begin
    s.color = blue;
    s.acc = emptyset;
    for all t in post(s) do
        if t.color == white then
            call dfs_blue(t);
        end if;
    end for;
    for all t in post(s) do
        let (s, l, a, t) be the edge from s to t;

```

```

        if s.acc U a not included in t.acc then
            call dfs_red(t, a U s.acc);
        end if;
    end for;
    if s.acc == all_acc then
        report a cycle;
    end if;
end;

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

```

This algorithm is the one presented in

```

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

```

9.26.1.9 emptiness_check* magic_search (const tgba * a, option_map o = option_map())

Wrapper for the two magic_search implementations.

This wrapper calls `explicit_magic_search_search()` or `bit_state_hashing_magic_search()` according to the "bsh" option in the `option_map`. If "bsh" is set and non null, its value is used as the size of the hash map.

9.26.1.10 emptiness_check* se05 (const tgba * a, option_map o)

Wrapper for the two se05 implementations.

This wrapper calls `explicit_se05_search()` or `bit_state_hashing_se05_search()` according to the "bsh" option in the `option_map`. If "bsh" is set and non null, its value is used as the size of the hash map.

9.27 TGBA runs and supporting functions

Classes

- struct `spot::tgba_run`

An accepted run, for a tgba.

Functions

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

9.27.1 Function Documentation

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

Display a [tgba_run](#).

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

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

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

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

Project a [tgba_run](#) on a tgba.

If a [tgba_run](#) has been generated on a product, or any other on-the-fly algorithm with tgba operands,

Parameters:

- run* the run to replay
- a_run* the automata on which the run was generated
- a_proj* the automata on which to project the run

Returns:

- true iff the run could be completed

9.27.1.3 tgba_run* reduce_run (const tgba * *a*, const tgba_run * *org*)

Reduce an accepting run.

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

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

Replay a [tgba_run](#) on a tgba.

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

Parameters:

run the run to replay

a the automata on which to replay that run

os the stream on which the replay should be traced

debug if set the output will be more verbose and extra debugging informations will be output on failure

Returns:

true iff the run could be completed

9.27.1.5 tgba* tgba_run_to_tgba (const tgba * *a*, const tgba_run * *run*)

Return an explicit_tgba corresponding to *run* (i.e. comparable states are merged).

Precondition:

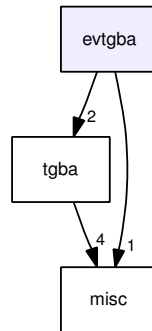
run must correspond to an actual run of the automaton *a*.

9.28 Emptiness-check statistics**Classes**

- struct [spot::unsigned_statistics](#)
- class [spot::unsigned_statistics_copy](#)
comparable statistics
- class [spot::ec_statistics](#)
Emptiness-check statistics.
- class [spot::ars_statistics](#)
Accepting Run Search statistics.
- class [spot::acss_statistics](#)
Accepting Cycle Search Space statistics.

10 spot Directory Documentation

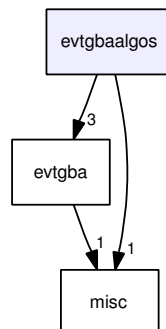
10.1 evtgba/ Directory Reference



Files

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

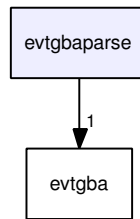
10.2 evtgbaalgos/ Directory Reference



Files

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

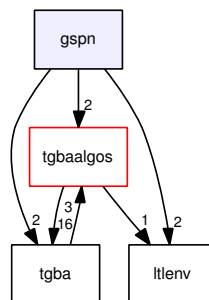
10.3 evtgbaparse/ Directory Reference



Files

- file [public.hh](#)

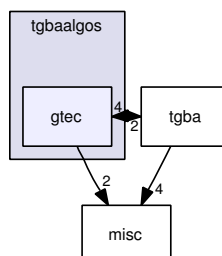
10.4 gspn/ Directory Reference



Files

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

10.5 tgbaalgos/gtec/ Directory Reference

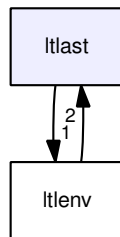


Files

- file [ce.hh](#)

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

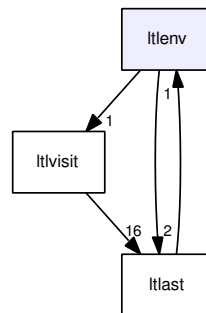
10.6 Itlast/ Directory Reference



Files

- file [allnodes.hh](#)
Define all LTL node types.
- file [atomic_prop.hh](#)
LTL atomic propositions.
- file [binop.hh](#)
LTL binary operators.
- file [constant.hh](#)
LTL constants.
- file [formula.hh](#)
LTL formula interface.
- file [multop.hh](#)
LTL multi-operand operators.
- file [predecl.hh](#)
Predeclare all LTL node types.
- file [refformula.hh](#)
Reference-counted LTL formulae.
- file [unop.hh](#)
LTL unary operators.
- file [visitor.hh](#)
LTL visitor interface.

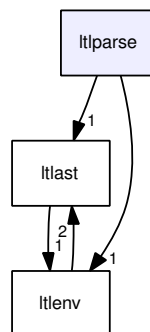
10.7 Itlenv/ Directory Reference



Files

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

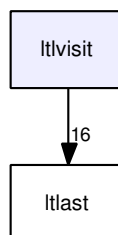
10.8 Itlparse/ Directory Reference



Files

- file [public.hh](#)

10.9 Itlvisit/ Directory Reference



Files

- file [apcollect.hh](#)
- file [basicreduce.hh](#)
- file [clone.hh](#)
- file [destroy.hh](#)
- file [dotty.hh](#)
- file [dump.hh](#)
- file [length.hh](#)
- file [lunabbrev.hh](#)
- file [nenofrm.hh](#)
- file [postfix.hh](#)
- file [randomltl.hh](#)
- file [reduce.hh](#)
- file [simpfg.hh](#)
- file [syntimpl.hh](#)
- file [tostring.hh](#)
- file [tunabbrev.hh](#)

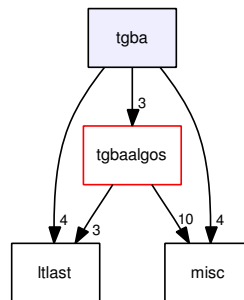
10.10 misc/ Directory Reference

misc

Files

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

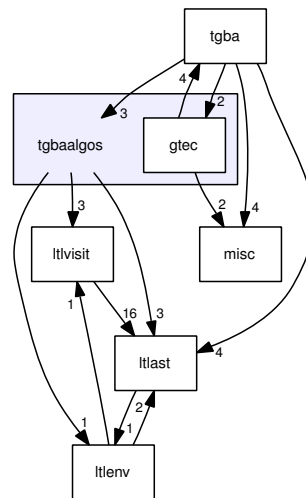
10.11 tgba/ Directory Reference



Files

- file [bdddict.hh](#)
- file [bddprint.hh](#)
- file [formula2bdd.hh](#)
- file [public.hh](#)
- file [state.hh](#)
- file [statebdd.hh](#)
- file [succiter.hh](#)
- file [succiterconcrete.hh](#)
- file [tgba.hh](#)
- file [tgbabddconcrete.hh](#)
- file [tgbabddconcretefactory.hh](#)
- file [tgbabddconcreteproduct.hh](#)
- file [tgbabddcoredata.hh](#)
- file [tgbabddfacyory.hh](#)
- file [tgbaexplicit.hh](#)
- file [tgbaproduct.hh](#)
- file [tgbareduc.hh](#)
- file [tgbatba.hh](#)

10.12 tgbaalgos/ Directory Reference



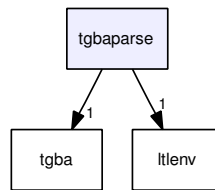
Directories

- directory [gtec](#)

Files

- file [bfssteps.hh](#)
- file [dotty.hh](#)
- file [dottydec.hh](#)
- file [dupexp.hh](#)
- file [emptiness.hh](#)
- file [emptiness_stats.hh](#)
- file [gv04.hh](#)
- file [lbt.hh](#)
- file [ltl2tgba_fm.hh](#)
- file [ltl2tgba_lacim.hh](#)
- file [magic.hh](#)
- file [neverclaim.hh](#)
- file [powerset.hh](#)
- file [projrun.hh](#)
- file [randomgraph.hh](#)
- file [reachiter.hh](#)
- file [reducerun.hh](#)
- file [reductgba_sim.hh](#)
- file [replayrun.hh](#)
- file [rundotdec.hh](#)
- file [save.hh](#)
- file [se05.hh](#)
- file [stats.hh](#)
- file [tau03.hh](#)
- file [tau03opt.hh](#)
- file [weight.hh](#)

10.13 tgbaparse/ Directory Reference



Files

- file [public.hh](#)

11 spot Namespace Documentation

11.1 spot Namespace Reference

Classes

- class [evtgba](#)
- class [evtgba_iterator](#)
- class [evtgba_explicit](#)
- class [state_evtgba_explicit](#)
States used by `spot::tgba_evtgba_explicit`.
- class [evtgba_product](#)
- class [symbol](#)
- class [rsymbol](#)
- class [evtgba_reachable_iterator](#)
Iterate over all reachable states of a `spot::evtgba`.
- class [evtgba_reachable_iterator_depth_first](#)
An implementation of `spot::evtgba_reachable_iterator` that browses states depth first.
- class [evtgba_reachable_iterator_breadth_first](#)
An implementation of `spot::evtgba_reachable_iterator` that browses states breadth first.
- class [bdd_allocator](#)
Manage ranges of variables.
- struct [bdd_less_than](#)
Comparison functor for BDDs.
- class [free_list](#)
Manage list of free integers.
- struct [ptr_hash](#)
A hash function for pointers.

- struct [string_hash](#)
A hash function for strings.
- struct [char_ptr_less_than](#)
Strict Weak Ordering for `char`.*
- class [minato_isop](#)
Generate an irredundant sum-of-products (ISOP) form of a BDD function.
- class [loopless_modular_mixed_radix_gray_code](#)
Loopless modular mixed radix Gray code iteration.
- class [option_map](#)
Manage a map of options.
- class [barand](#)
Compute pseudo-random integer value between 0 and `n` included, following a binomial distribution for probability `p`.
- struct [time_info](#)
A structure to record elapsed time in clock ticks.
- class [timer](#)
A timekeeper that accumulate interval of time.
- class [timer_map](#)
A map of timer, where each timer has a name.
- class [bdd_dict](#)
- class [state](#)
Abstract class for states.
- struct [state_ptr_less_than](#)
Strict Weak Ordering for `state`.*
- struct [state_ptr_equal](#)
An Equivalence Relation for `state`.*
- struct [state_ptr_hash](#)
Hash Function for `state`.*
- class [state_bdd](#)
- class [tgba_succ_iterator](#)
Iterate over the successors of a state.
- class [tgba_succ_iterator_concrete](#)
- class [tgba](#)
A Transition-based Generalized Büchi Automaton.

- class [tgba_bdd_concrete](#)
A concrete [spot::tgba](#) implemented using BDDs.
- class [tgba_bdd_concrete_factory](#)
Helper class to build a [spot::tgba_bdd_concrete](#) object.
- struct [tgba_bdd_core_data](#)
Core data for a TGBA encoded using BDDs.
- class [tgba_bdd_factory](#)
Abstract class for [spot::tgba_bdd_concrete](#) factories.
- class [tgba_explicit](#)
- class [state_explicit](#)
- class [tgba_explicit_succ_iterator](#)
- class [state_product](#)
A state for [spot::tgba_product](#).
- class [tgba_succ_iterator_product](#)
Iterate over the successors of a product computed on the fly.
- class [tgba_product](#)
A lazy product. (States are computed on the fly.).
- class [direct_simulation_relation](#)
- class [delayed_simulation_relation](#)
- class [tgba_reduc](#)
- class [tgba_tba_proxy](#)
Degeneralize a [spot::tgba](#) on the fly, producing a TBA.
- class [tgba_sba_proxy](#)
Degeneralize a [spot::tgba](#) on the fly, producing an SBA.
- class [bfs_steps](#)
Make a BFS in a [spot::tgba](#) to compute a [tgba_run::steps](#).
- class [dotty_decorator](#)
Choose state and link styles for [spot::dotty_reachable](#).
- class [emptiness_check_result](#)
The result of an emptiness check.
- class [emptiness_check](#)
Common interface to emptiness check algorithms.
- class [emptiness_check_instantiator](#)
- struct [tgba_run](#)
An accepted run, for a tgba.
- struct [unsigned_statistics](#)

- class [unsigned_statistics_copy](#)
comparable statistics
- class [ec_statistics](#)
Emptiness-check statistics.
- class [ars_statistics](#)
Accepting Run Search statistics.
- class [acss_statistics](#)
Accepting Cycle Search Space statistics.
- class [couvreur99_check_result](#)
Compute a counter example from a [spot::couvreur99_check_status](#).
- class [explicit_connected_component](#)
An SCC storing all its states explicitly.
- class [connected_component_hash_set](#)
- class [explicit_connected_component_factory](#)
Abstract factory for [explicit_connected_component](#).
- class [connected_component_hash_set_factory](#)
Factory for [connected_component_hash_set](#).
- class [couvreur99_check](#)
An implementation of the Couvreur99 emptiness-check algorithm.
- class [couvreur99_check_shy](#)
A version of [spot::couvreur99_check](#) that tries to visit known states first.
- class [numbered_state_heap_const_iterator](#)
Iterator on [numbered_state_heap](#) objects.
- class [numbered_state_heap](#)
Keep track of a large quantity of indexed states.
- class [numbered_state_heap_factory](#)
Abstract factory for [numbered_state_heap](#).
- class [numbered_state_heap_hash_map](#)
A straightforward implementation of [numbered_state_heap](#) with a hash map.
- class [numbered_state_heap_hash_map_factory](#)
Factory for [numbered_state_heap_hash_map](#).
- class [scc_stack](#)
- class [couvreur99_check_status](#)
The status of the emptiness-check on success.

- class [tgba_reachable_iterator](#)
Iterate over all reachable states of a [spot::tgba](#).
- class [tgba_reachable_iterator_depth_first](#)
An implementation of [spot::tgba_reachable_iterator](#) that browses states depth first.
- class [tgba_reachable_iterator_breadth_first](#)
An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.
- class [parity_game_graph](#)
Parity game graph which compute a simulation relation.
- class [spoiler_node](#)
Spoiler node of parity game graph.
- class [duplicator_node](#)
Duplicator node of parity game graph.
- class [parity_game_graph_direct](#)
Parity game graph which compute the direct simulation relation.
- class [spoiler_node_delayed](#)
Spoiler node of parity game graph for delayed simulation.
- class [duplicator_node_delayed](#)
Duplicator node of parity game graph for delayed simulation.
- class [parity_game_graph_delayed](#)
- class [tgba_run_dotty_decorator](#)
Highlight a [spot::tgba_run](#) on a [spot::tgba](#).
- struct [tgba_statistics](#)
- class [weight](#)
Manage for a given automaton a vector of counter indexed by its acceptance condition.
- class [gspn_exception](#)
An exception used to forward GSPN errors.
- class [gspn_interface](#)
- class [gspn_ssp_interface](#)

Namespaces

- namespace [ltl](#)

Typedefs

- typedef std::set< const [symbol](#) * > [symbol_set](#)
- typedef std::set< [rsymbol](#) > [rsymbol_set](#)
- typedef std::pair< yy::location, std::string > [evtgba_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [evtgba_parse_error](#) > [evtgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.
- typedef std::pair< const [spot::state](#) *, const [spot::state](#) * > [state_couple](#)
- typedef std::vector< [state_couple](#) * > [simulation_relation](#)
- typedef std::vector< [spoiler_node](#) * > [sn_v](#)
- typedef std::vector< [duplicator_node](#) * > [dn_v](#)
- typedef std::vector< const [state](#) * > [s_v](#)
- typedef std::pair< yy::location, std::string > [tgba_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [tgba_parse_error](#) > [tgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.

Enumerations

- enum [reduce_tgba_options](#) {
[Reduce_None](#) = 0, [Reduce_quotient_Dir_Sim](#) = 1, [Reduce_transition_Dir_Sim](#) = 2, [Reduce_](#)
[quotient_Del_Sim](#) = 4,
[Reduce_transition_Del_Sim](#) = 8, [Reduce_Scc](#) = 16, [Reduce_All](#) = -1U }
Options for reduce.

Functions

- std::ostream & [dotty_reachable](#) (std::ostream &os, const [evtgba](#) *g)
Print reachable states in dot format.
- std::ostream & [evtgba_save_reachable](#) (std::ostream &os, const [evtgba](#) *g)
Save reachable states in text format.
- [evtgba_explicit](#) * [tgba_to_evtgba](#) (const [tgba](#) *a)
Convert a tgba into an evtgba.
- [evtgba_explicit](#) * [evtgba_parse](#) (const std::string &filename, [evtgba_parse_error_list](#) &error_list, bool debug=false)
Build a [spot::evtgba_explicit](#) from a text file.
- bool [format_evtgba_parse_errors](#) (std::ostream &os, const std::string &filename, [evtgba_parse_](#)
[error_list](#) &error_list)
Format diagnostics produced by [spot::evtgba_parse](#).

- bool [is_bare_word](#) (const char *str)
- std::string [quote_unless_bare_word](#) (const std::string &str)
Double-quote words that are not bare.
- std::ostream & [escape_str](#) (std::ostream &os, const std::string &str)
Escape " and \ characters in str.
- std::string [escape_str](#) (const std::string &str)
Escape " and \ characters in str.
- size_t [wang32_hash](#) (size_t key)
Thomas Wang's 32 bit hash function.
- size_t [knuth32_hash](#) (size_t key)
Knuth's Multiplicative hash function.
- void [srand](#) (unsigned int seed)
Reset the seed of the pseudo-random number generator.
- int [rrand](#) (int min, int max)
Compute a pseudo-random integer value between min and max included.
- int [mrand](#) (int max)
Compute a pseudo-random integer value between 0 and max-1 included.
- double [drand](#) ()
Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).
- double [nrand](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).
- double [bmrand](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).
- int [prand](#) (double p)
Return a pseudo-random positive integer value following a Poisson distribution with parameter p.
- const char * [version](#) ()
Return Spot's version.
- std::ostream & [bdd_print_sat](#) (std::ostream &os, const [bdd_dict](#) *dict, bdd b)
Print a BDD as a list of literals.
- std::string [bdd_format_sat](#) (const [bdd_dict](#) *dict, bdd b)
Format a BDD as a list of literals.
- std::ostream & [bdd_print_acc](#) (std::ostream &os, const [bdd_dict](#) *dict, bdd b)
Print a BDD as a list of acceptance conditions.

- `std::ostream & bdd_print_accset` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a set of acceptance conditions.
- `std::string bdd_format_accset` (`const bdd_dict *dict`, `bdd b`)
Format a BDD as a set of acceptance conditions.
- `std::ostream & bdd_print_set` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a set.
- `std::string bdd_format_set` (`const bdd_dict *dict`, `bdd b`)
Format a BDD as a set.
- `std::ostream & bdd_print_formula` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a formula.
- `std::string bdd_format_formula` (`const bdd_dict *dict`, `bdd b`)
Format a BDD as a formula.
- `std::ostream & bdd_print_dot` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a diagram in dotty format.
- `std::ostream & bdd_print_table` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a table.
- `bdd formula_to_bdd` (`const ltl::formula *f`, `bdd_dict *d`, `void *for_me`)
- `const ltl::formula * bdd_to_formula` (`bdd f`, `const bdd_dict *d`)
- `tgba_bdd_concrete * product` (`const tgba_bdd_concrete *left`, `const tgba_bdd_concrete *right`)
Multiplies two spot::tgba_bdd_concrete automata.
- `std::ostream & dotty_reachable` (`std::ostream &os`, `const tgba *g`, `dotty_decorator *dd=dotty_decorator::instance()`)
Print reachable states in dot format.
- `tgba_explicit * tgba_dupexp_bfs` (`const tgba *aut`)
Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.
- `tgba_explicit * tgba_dupexp_dfs` (`const tgba *aut`)
Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.
- `std::ostream & print_tgba_run` (`std::ostream &os`, `const tgba *a`, `const tgba_run *run`)
Display a tgba_run.
- `tgba * tgba_run_to_tgba` (`const tgba *a`, `const tgba_run *run`)
Return an explicit_tgba corresponding to run (i.e. comparable states are merged).
- `emptiness_check * couvreur99` (`const tgba *a`, `option_map options=option_map()`, `const numbered_state_heap_factory *nshf=numbered_state_heap_hash_map_factory::instance()`)
Check whether the language of an automata is empty.
- `emptiness_check * explicit_gv04_check` (`const tgba *a`, `option_map o=option_map()`)

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

- `std::ostream & lbtt_reachable` (`std::ostream &os`, `const tgba *g`)
Print reachable states in LBTT format.
- `tgba_explicit * ltl_to_tgba_fm` (`const ltl::formula *f`, `bdd_dict *dict`, `bool exprop=false`, `bool symb_merge=true`, `bool branching_postponement=false`, `bool fair_loop_approx=false`, `const ltl::atomic_prop_set *unobs=0`, `int reduce_ltl=ltl::Reduce_None`)
Build a `spot::tgba_explicit` from an LTL formula.
- `tgba_bdd_concrete * ltl_to_tgba_lacim` (`const ltl::formula *f`, `bdd_dict *dict`)
Build a `spot::tgba_bdd_concrete` from an LTL formula.
- `emptiness_check * explicit_magic_search` (`const tgba *a`, `option_map o=option_map()`)
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * bit_state_hashing_magic_search` (`const tgba *a`, `size_t size`, `option_map o=option_map()`)
Returns an emptiness checker on the `spot::tgba` automaton a.
- `emptiness_check * magic_search` (`const tgba *a`, `option_map o=option_map()`)
Wrapper for the two `magic_search` implementations.
- `std::ostream & never_claim_reachable` (`std::ostream &os`, `const tgba_sba_proxy *g`, `const ltl::formula *f=0`)
Print reachable states in Spin never claim format.
- `tgba_explicit * tgba_powerset` (`const tgba *aut`)
Build a deterministic automaton, ignoring acceptance conditions.
- `tgba_run * project_tgba_run` (`const tgba *a_run`, `const tgba *a_proj`, `const tgba_run *run`)
Project a `tgba_run` on a `tgba`.
- `tgba * random_graph` (`int n`, `float d`, `const ltl::atomic_prop_set *ap`, `bdd_dict *dict`, `int n_acc=0`, `float a=0.1`, `float t=0.5`, `ltl::environment *env=<l::default_environment::instance()`)
Construct a `tgba` randomly.
- `tgba_run * reduce_run` (`const tgba *a`, `const tgba_run *org`)
Reduce an accepting run.
- `tgba * reduc_tgba_sim` (`const tgba *a`, `int opt=Reduce_All`)
Remove some node of the automata using a simulation relation.
- `direct_simulation_relation * get_direct_relation_simulation` (`const tgba *a`, `std::ostream &os`, `int opt=-1`)
Compute a direct simulation relation on state of `tgba` f.
- `delayed_simulation_relation * get_delayed_relation_simulation` (`const tgba *a`, `std::ostream &os`, `int opt=-1`)
- `void free_relation_simulation` (`direct_simulation_relation *rel`)

To free a simulation relation.

- void [free_relation_simulation](#) ([delayed_simulation_relation](#) *rel)
To free a simulation relation.
- bool [replay_tgba_run](#) (std::ostream &os, const [tgba](#) *a, const [tgba_run](#) *run, bool debug=false)
Replay a [tgba_run](#) on a [tgba](#).
- std::ostream & [tgba_save_reachable](#) (std::ostream &os, const [tgba](#) *g)
Save reachable states in text format.
- [emptiness_check](#) * [explicit_se05_search](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)())
Returns an emptiness check on the [spot::tgba](#) automaton a.
- [emptiness_check](#) * [bit_state_hashing_se05_search](#) (const [tgba](#) *a, [size_t](#) size, [option_map](#) o=[option_map](#)())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [emptiness_check](#) * [se05](#) (const [tgba](#) *a, [option_map](#) o)
Wrapper for the two [se05](#) implementations.
- [tgba_statistics](#) [stats_reachable](#) (const [tgba](#) *g)
Compute statistics for an automaton.
- [emptiness_check](#) * [explicit_tau03_search](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [emptiness_check](#) * [explicit_tau03_opt_search](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- [tgba_explicit](#) * [tgba_parse](#) (const std::string &filename, [tgba_parse_error_list](#) &error_list, [bdd_dict](#) *dict, [ltl::environment](#) &env=[ltl::default_environment::instance](#)(), bool debug=false)
Build a [spot::tgba_explicit](#) from a text file.
- bool [format_tgba_parse_errors](#) (std::ostream &os, const std::string &filename, [tgba_parse_error_list](#) &error_list)
Format diagnostics produced by [spot::tgba_parse](#).
- std::ostream & [operator<<](#) (std::ostream &os, const [gspn_exception](#) &e)
- [couvereur99_check](#) * [couvereur99_check_ssp_semi](#) (const [tgba](#) *ssp_automata)
- [couvereur99_check](#) * [couvereur99_check_ssp_shy_semi](#) (const [tgba](#) *ssp_automata)
- [couvereur99_check](#) * [couvereur99_check_ssp_shy](#) (const [tgba](#) *ssp_automata)

11.1.1 Typedef Documentation

11.1.1.1 typedef std::pair<yy::location, std::string> [spot::evtgba_parse_error](#)

A parse diagnostic with its location.

11.1.1.2 typedef std::list<[evtgba_parse_error](#)> [spot::evtgba_parse_error_list](#)

A list of parser diagnostics, as filled by parse.

11.1.1.3 `typedef std::set<rsymbol> spot::rsymbol_set`

11.1.1.4 `typedef std::vector<state_couple*> spot::simulation_relation`

11.1.1.5 `typedef std::pair<const spot::state*, const spot::state*> spot::state_couple`

11.1.1.6 `typedef std::set<const symbol*> spot::symbol_set`

11.1.2 Function Documentation

11.1.2.1 `std::string bdd_format_accset (const bdd_dict * dict, bdd b)`

Format a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

11.1.2.2 `std::string bdd_format_formula (const bdd_dict * dict, bdd b)`

Format a BDD as a formula.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

11.1.2.3 `std::string bdd_format_sat (const bdd_dict * dict, bdd b)`

Format a BDD as a list of literals.

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

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

11.1.2.4 std::string bdd_format_set (const bdd_dict * *dict*, bdd *b*)

Format a BDD as a set.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

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

Print a BDD as a list of acceptance conditions.

This is used when saving a TGBA.

Parameters:

os The output stream.

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

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

Print a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

Parameters:

os The output stream.

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

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

Print a BDD as a diagram in dotty format.

Parameters:

os The output stream.

dict The dictionary to use, to lookup variables.

b The BDD to print.

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

Print a BDD as a formula.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

11.1.2.9 std::ostream& bdd_print_sat (std::ostream & *os*, const bdd_dict * *dict*, bdd *b*)

Print a BDD as a list of literals.

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

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

11.1.2.10 std::ostream& bdd_print_set (std::ostream & *os*, const bdd_dict * *dict*, bdd *b*)

Print a BDD as a set.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

11.1.2.11 std::ostream& bdd_print_table (std::ostream & *os*, const bdd_dict * *dict*, bdd *b*)

Print a BDD as a table.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

11.1.2.12 const [ltl::formula](#)* bdd_to_formula (bdd *f*, const bdd_dict * *d*)**11.1.2.13 std::ostream& dotty_reachable (std::ostream & *os*, const evtgba * *g*)**

Print reachable states in dot format.

11.1.2.14 [evtgba_explicit](#)* [evtgba_parse](#) (const std::string & *filename*, [evtgba_parse_error_list](#) & *error_list*, bool *debug* = false)

Build a [spot::evtgba_explicit](#) from a text file.

Parameters:

filename The name of the file to parse.

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

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

Returns:

A pointer to the [evtgba](#) built from *filename*, or 0 if the file could not be opened.

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

Warning:

This function is not reentrant.

11.1.2.15 std::ostream& [evtgba_save_reachable](#) (std::ostream & *os*, const [evtgba](#) * *g*)

Save reachable states in text format.

11.1.2.16 bool [format_evtgba_parse_errors](#) (std::ostream & *os*, const std::string & *filename*, [evtgba_parse_error_list](#) & *error_list*)

Format diagnostics produced by [spot::evtgba_parse](#).

Parameters:

os Where diagnostics should be output.

filename The filename that should appear in the diagnostics.

error_list The error list filled by [spot::ltl::parse](#) while parsing *ltl_string*.

Returns:

true iff any diagnostic was output.

11.1.2.17 bdd [formula_to_bdd](#) (const [ltl::formula](#) * *f*, [bdd_dict](#) * *d*, void * *for_me*)

11.1.2.18 std::ostream& operator<< (std::ostream & *os*, const [gspn_exception](#) & *e*)

11.1.2.19 [evtgba_explicit](#)* [tgba_to_evtgba](#) (const [tgba](#) * *a*)

Convert a [tgba](#) into an [evtgba](#).

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

11.2 spot::ltl Namespace Reference

Classes

- class [atomic_prop](#)
Atomic propositions.
- class [binop](#)
Binary operator.
- class [constant](#)
A constant (True or False).
- class [formula](#)
An LTL formula.
- struct [formula_ptr_less_than](#)
Strict Weak Ordering for `const formula`.*
- struct [formula_ptr_hash](#)
Hash Function for `const formula`.*
- class [multop](#)
Multi-operand operators.
- class [ref_formula](#)
A reference-counted LTL formula.
- class [unop](#)
Unary operators.
- struct [visitor](#)
Formula visitor that can modify the formula.
- struct [const_visitor](#)
Formula visitor that cannot modify the formula.
- class [declarative_environment](#)
A declarative environment.
- class [default_environment](#)
A laxist environment.
- class [environment](#)
An environment that describes atomic propositions.
- class [clone_visitor](#)
Clone a formula.
- class [unabbreviate_logic_visitor](#)

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

- class [postfix_visitor](#)
Apply an algorithm on each node of an AST, during a postfix traversal.
- class [random_ltl](#)
Generate random LTL formulae.
- class [simplify_f_g_visitor](#)
Replace `true U f` and `false R g` by `F f` and `G g`.
- class [unabbreviate_ltl_visitor](#)
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

Typedefs

- typedef std::pair< yy::location, std::string > [parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [parse_error](#) > [parse_error_list](#)
A list of parser diagnostics, as filled by parse.
- typedef std::set< [atomic_prop](#) *, [formula_ptr_less_than](#) > [atomic_prop_set](#)
Set of atomic propositions.

Enumerations

- enum [reduce_options](#) {
 [Reduce_None](#) = 0, [Reduce_Basics](#) = 1, [Reduce_Syntactic_Implications](#) = 2, [Reduce_Eventuality_And_Universality](#) = 4,
 [Reduce_All](#) = -1U }
Options for `spot::ltl::reduce`.

Functions

- [formula](#) * [parse](#) (const std::string <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`.
- [atomic_prop_set](#) * [atomic_prop_collect](#) (const [formula](#) *f, [atomic_prop_set](#) *s=0)
Return the set of atomic propositions occurring in a formula.

- `formula * basic_reduce (const formula *f)`
Basic rewritings.
- `bool is_GF (const formula *f)`
Whether a formula starts with GF.
- `bool is_FG (const formula *f)`
Whether a formula starts with FG.
- `formula * clone (const formula *f)`
Clone a formula.
- `void destroy (const formula *f)`
Destroys a formula.
- `std::ostream & dotty (std::ostream &os, const formula *f)`
Write a formula tree using dot's syntax.
- `std::ostream & dump (std::ostream &os, const formula *f)`
Dump a formula tree.
- `int length (const formula *f)`
Compute the length of a formula.
- `formula * unabbreviate_logic (const formula *f)`
Clone and rewrite a formula to remove most of the abbreviated logical operators.
- `formula * negative_normal_form (const formula *f, bool negated=false)`
Build the negative normal form of f.
- `formula * reduce (const formula *f, int opt=Reduce_All)`
Reduce a formula f.
- `bool is_eventual (const formula *f)`
Check whether a formula is a pure eventuality.
- `bool is_universal (const formula *f)`
Check whether a formula is purely universal.
- `formula * simplify_f_g (const formula *f)`
Replace true U f and false R g by F f and G g.
- `bool syntactic_implication (const formula *f1, const formula *f2)`
Syntactic implication.
- `bool syntactic_implication_neg (const formula *f1, const formula *f2, bool right)`
Syntactic implication.
- `std::ostream & to_string (const formula *f, std::ostream &os)`
Output a formula as a (parsable) string.

- `std::string to_string (const formula *f)`
Convert a formula into a (parsable) string.
- `std::ostream & to_spin_string (const formula *f, std::ostream &os)`
Output a formula as a (parsable by Spin) string.
- `std::string to_spin_string (const formula *f)`
Convert a formula into a (parsable by Spin) string.
- `formula * unabbreviate_ltl (const formula *f)`
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

11.2.1 Function Documentation

11.2.1.1 `formula* unabbreviate_ltl (const formula *f)`

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

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

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

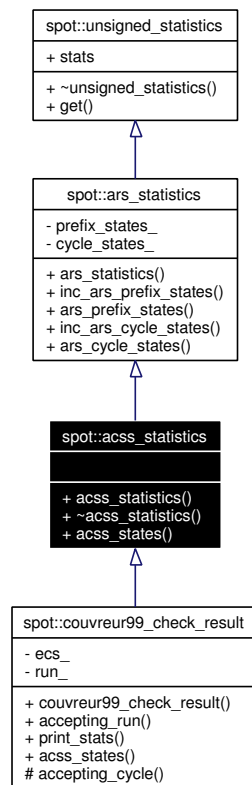
12 spot Class Documentation

12.1 `spot::acss_statistics` Class Reference

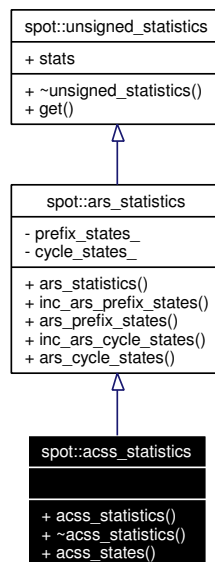
Accepting Cycle Search Space statistics.

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

Inheritance diagram for `spot::acss_statistics`:



Collaboration diagram for `spot::acss_statistics`:



Public Types

- `typedef unsigned(unsigned_statistics::* unsigned_fun)() const`
- `typedef std::map< const char *, unsigned_fun, char_ptr_less_than > stats_map`

Public Member Functions

- [acss_statistics](#) ()
- virtual [~acss_statistics](#) ()
- virtual unsigned [acss_states](#) () const =0
Number of states in the search space for the accepting cycle.
- void [inc_ars_prefix_states](#) ()
- unsigned [ars_prefix_states](#) () const
- void [inc_ars_cycle_states](#) ()
- unsigned [ars_cycle_states](#) () const
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

12.1.1 Detailed Description

Accepting Cycle Search Space statistics.

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

12.1.2 Member Typedef Documentation

12.1.2.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map` [inherited]

12.1.2.2 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const` [inherited]

12.1.3 Constructor & Destructor Documentation

12.1.3.1 `spot::acss_statistics::acss_statistics ()` [inline]

12.1.3.2 `virtual spot::acss_statistics::~~acss_statistics ()` [inline, virtual]

12.1.4 Member Function Documentation

12.1.4.1 `virtual unsigned spot::acss_statistics::acss_states () const` [pure virtual]

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

Implemented in [spot::couvreur99_check_result](#).

12.1.4.2 `unsigned spot::ars_statistics::ars_cycle_states () const` [inline, inherited]

12.1.4.3 `unsigned spot::ars_statistics::ars_prefix_states () const` [inline, inherited]

12.1.4.4 `unsigned spot::unsigned_statistics::get (const char * str) const` [inline, inherited]

12.1.4.5 `void spot::ars_statistics::inc_ars_cycle_states ()` [inline, inherited]

12.1.4.6 `void spot::ars_statistics::inc_ars_prefix_states ()` [inline, inherited]

12.1.5 Member Data Documentation

12.1.5.1 `stats_map spot::unsigned_statistics::stats` [inherited]

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

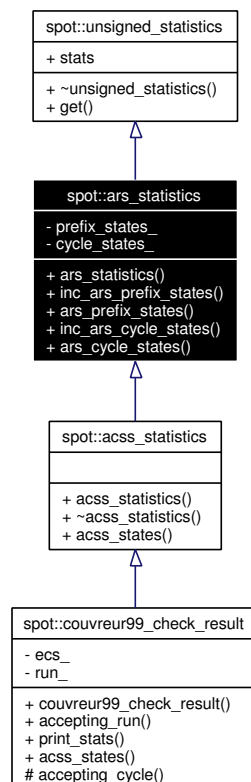
- [tgbaalgos/emptiness_stats.hh](#)

12.2 spot::ars_statistics Class Reference

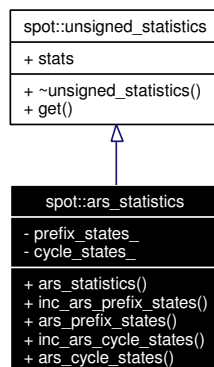
Accepting Run Search statistics.

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

Inheritance diagram for spot::ars_statistics:



Collaboration diagram for spot::ars_statistics:



Public Types

- `typedef unsigned(unsigned_statistics::* unsigned_fun)() const`
- `typedef std::map< const char *, unsigned_fun, char_ptr_less_than > stats_map`

Public Member Functions

- [ars_statistics](#) ()
- void [inc_ars_prefix_states](#) ()
- unsigned [ars_prefix_states](#) () const
- void [inc_ars_cycle_states](#) ()
- unsigned [ars_cycle_states](#) () const
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

Private Attributes

- unsigned [prefix_states_](#)
- unsigned [cycle_states_](#)
states visited to construct the prefix

12.2.1 Detailed Description

Accepting Run Search statistics.

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

12.2.2 Member Typedef Documentation

12.2.2.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map [inherited]`

12.2.2.2 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun())() const` [inherited]

12.2.3 Constructor & Destructor Documentation

12.2.3.1 `spot::ars_statistics::ars_statistics()` [inline]

12.2.4 Member Function Documentation

12.2.4.1 `unsigned spot::ars_statistics::ars_cycle_states() const` [inline]

12.2.4.2 `unsigned spot::ars_statistics::ars_prefix_states() const` [inline]

12.2.4.3 `unsigned spot::unsigned_statistics::get(const char * str) const` [inline, inherited]

12.2.4.4 `void spot::ars_statistics::inc_ars_cycle_states()` [inline]

12.2.4.5 `void spot::ars_statistics::inc_ars_prefix_states()` [inline]

12.2.5 Member Data Documentation

12.2.5.1 `unsigned spot::ars_statistics::cycle_states_` [private]
states visited to construct the prefix

12.2.5.2 `unsigned spot::ars_statistics::prefix_states_` [private]

12.2.5.3 `stats_map spot::unsigned_statistics::stats` [inherited]

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

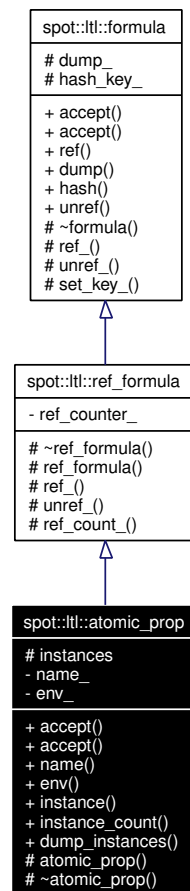
- [tgbaalgos/emptiness_stats.hh](#)

12.3 spot::ltl::atomic_prop Class Reference

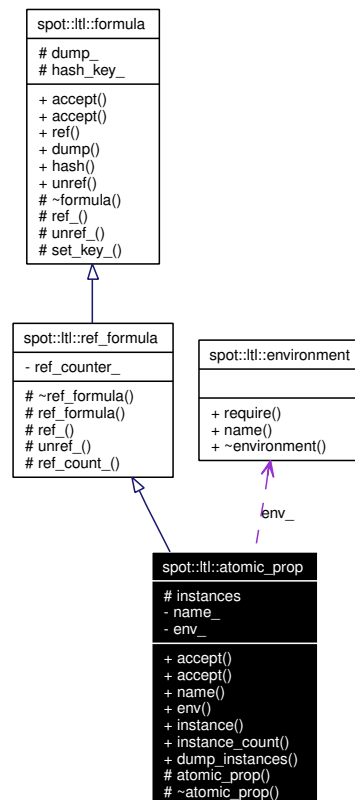
Atomic propositions.

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

Inheritance diagram for spot::ltl::atomic_prop:



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



Public Member Functions

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

Static Public Member Functions

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

Protected Types

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

Protected Member Functions

- [atomic_prop](#) (const std::string &name, [environment](#) &env)
- virtual [~atomic_prop](#) ()
- void [ref_](#) ()
increment reference counter if any
- bool [unref_](#) ()
decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).
- unsigned [ref_count_](#) ()
Number of references to this formula.
- void [set_key_](#) ()
Compute key_ from dump_.

Protected Attributes

- std::string [dump_](#)
The canonic representation of the formula.
- size_t [hash_key_](#)
The hash key of this formula.

Static Protected Attributes

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

Private Attributes

- std::string [name_](#)
- [environment](#) * [env_](#)

12.3.1 Detailed Description

Atomic propositions.

12.3.2 Member Typedef Documentation

12.3.2.1 typedef std::map<[pair](#), [atomic_prop](#)*> [spot::ltl::atomic_prop::map](#) [protected]

12.3.2.2 typedef std::pair<std::string, [environment](#)*> [spot::ltl::atomic_prop::pair](#) [protected]

12.3.3 Constructor & Destructor Documentation

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

12.3.3.2 virtual [spot::ltl::atomic_prop::~~atomic_prop](#) () [protected, virtual]

12.3.4 Member Function Documentation

12.3.4.1 virtual void [spot::ltl::atomic_prop::accept](#) ([const_visitor](#) & *visitor*) const [virtual]

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

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

12.3.4.2 virtual void [spot::ltl::atomic_prop::accept](#) ([visitor](#) & *visitor*) [virtual]

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

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

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

Return a canonic representation of the formula.

12.3.4.4 static std::ostream& [spot::ltl::atomic_prop::dump_instances](#) (std::ostream & *os*) [static]

List all instances of atomic propositions. For debugging.

12.3.4.5 [environment](#)& [spot::ltl::atomic_prop::env](#) () const

Get the environment of the atomic proposition.

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

Return a hash_key for the formula.

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

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

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

Number of instantiated atomic propositions. For debugging.

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

Get the name of the atomic proposition.

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

clone this node

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

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

increment reference counter if any

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

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

Number of references to this formula.

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

Compute key_ from dump_.

Should be called once in each object, after dump_ has been set.

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

release this node

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

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

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

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

12.3.5 Member Data Documentation

12.3.5.1 std::string spot::ltl::formula::dump_ [protected, inherited]

The canonic representation of the formula.

12.3.5.2 environment* spot::ltl::atomic_prop::env_ [private]

12.3.5.3 size_t spot::ltl::formula::hash_key_ [protected, inherited]

The hash key of this formula.

Initialized by [set_key_\(\)](#).

12.3.5.4 map spot::ltl::atomic_prop::instances [static, protected]

12.3.5.5 std::string spot::ltl::atomic_prop::name_ [private]

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

- [ltlast/atomic_prop.hh](#)

12.4 spot::barand< gen > Class Template Reference

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

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

Public Member Functions

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

Protected Attributes

- const int [n_](#)
- const double [m_](#)
- const double [s_](#)

12.4.1 Detailed Description

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

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

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

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

12.4.2 Constructor & Destructor Documentation

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

12.4.3 Member Function Documentation

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

12.4.4 Member Data Documentation

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

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

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

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

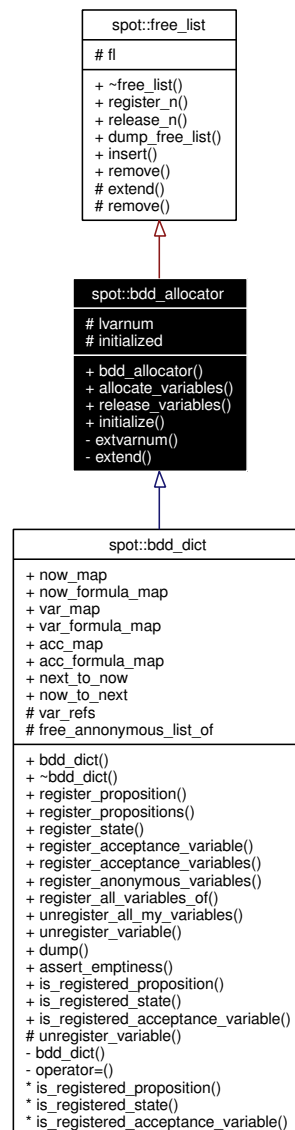
- [misc/random.hh](#)

12.5 spot::bdd_allocator Class Reference

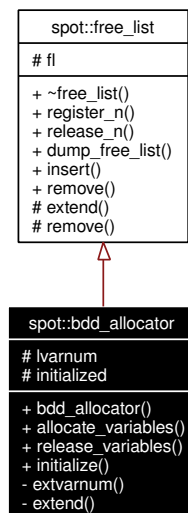
Manage ranges of variables.

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

Inheritance diagram for spot::bdd_allocator:



Collaboration diagram for `spot::bdd_allocator`:



Public Member Functions

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

Static Public Member Functions

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

Protected Attributes

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

Static Protected Attributes

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

Private Types

- typedef std::pair< int, int > [pos_lenght_pair](#)
Such pairs describe second free integer starting at first.
- typedef std::list< [pos_lenght_pair](#) > [free_list_type](#)

Private Member Functions

- void [extvarnum](#) (int more)
Require more variables.
- virtual int [extend](#) (int n)
- int [register_n](#) (int n)
Find n consecutive integers.
- void [release_n](#) (int base, int n)
Release n consecutive integers starting at base.
- std::ostream & [dump_free_list](#) (std::ostream &os) const
Dump the list to os for debugging.
- void [insert](#) (int base, int n)
Extend the list by inserting a new pos-lenght pair.
- void [remove](#) (int base, int n=0)
Remove n consecutive entries from the list, starting at base.
- void [remove](#) (free_list_type::iterator i, int base, int n)
Remove n consecutive entries from the list, starting at base.

Private Attributes

- [free_list_type](#) fl
Tracks unused BDD variables.

12.5.1 Detailed Description

Manage ranges of variables.

12.5.2 Member Typedef Documentation

12.5.2.1 typedef std::list<[pos_lenght_pair](#)> [spot::free_list::free_list_type](#) [protected, inherited]

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

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

12.5.3 Constructor & Destructor Documentation

12.5.3.1 `spot::bdd_allocator::bdd_allocator ()`

Default constructor.

12.5.4 Member Function Documentation

12.5.4.1 `int spot::bdd_allocator::allocate_variables (int n)`

Allocate *n* BDD variables.

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

Dump the list to *os* for debugging.

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

Allocate *n* integer.

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

Implements `spot::free_list`.

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

Require more variables.

12.5.4.5 `static void spot::bdd_allocator::initialize ()` [static]

Initialize the BDD library.

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

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

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

Find *n* consecutive integers.

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

Returns:

the first integer of the range

12.5.4.8 void spot::free_list::release_n (int *base*, int *n*) [inherited]

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

12.5.4.9 void spot::bdd_allocator::release_variables (int *base*, int *n*)

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

12.5.4.10 void spot::free_list::remove (free_list_type::iterator *i*, int *base*, int *n*) [protected, inherited]

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

12.5.4.11 void spot::free_list::remove (int *base*, int *n* = 0) [inherited]

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

12.5.5 Member Data Documentation**12.5.5.1 free_list_type spot::free_list::fl** [protected, inherited]

Tracks unused BDD variables.

12.5.5.2 bool spot::bdd_allocator::initialized [static, protected]

Whether the BDD library has been initialized.

12.5.5.3 int spot::bdd_allocator::lvarnum [protected]

number of variables in use in this allocator.

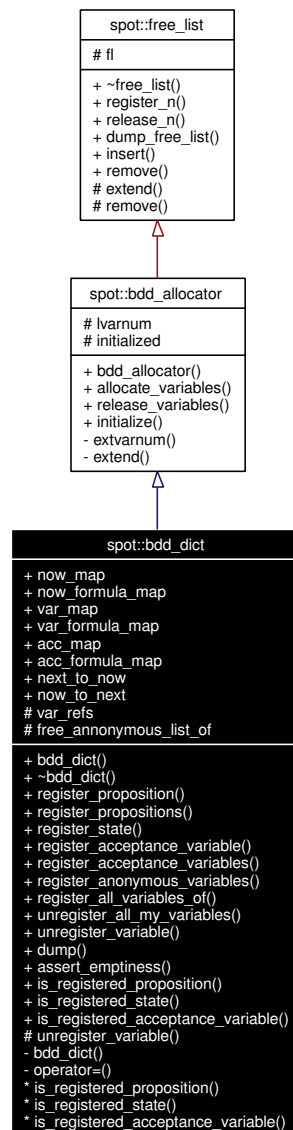
The documentation for this class was generated from the following file:

- [misc/bddalloc.hh](#)

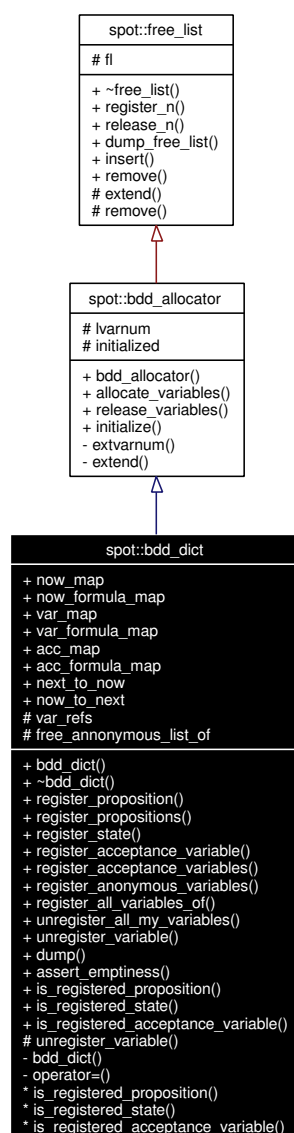
12.6 spot::bdd_dict Class Reference

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

Inheritance diagram for spot::bdd_dict:



Collaboration diagram for `spot::bdd_dict`:



Public Types

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

Public Member Functions

- [bdd_dict](#) ()
- [~bdd_dict](#) ()
- int [register_proposition](#) (const [ltl::formula](#) *f, const void *for_me)

Register an atomic proposition.

- void [register_propositions](#) (bdd f, const void *for_me)
Register BDD variables as atomic propositions.
- int [register_state](#) (const [ltl::formula](#) *f, const void *for_me)
Register a couple of Now/Next variables.
- int [register_acceptance_variable](#) (const [ltl::formula](#) *f, const void *for_me)
Register an atomic proposition.
- void [register_acceptance_variables](#) (bdd f, const void *for_me)
Register BDD variables as acceptance variables.
- int [register_anonymous_variables](#) (int n, const void *for_me)
Register anonymous BDD variables.
- void [register_all_variables_of](#) (const void *from_other, const void *for_me)
Duplicate the variable usage of another object.
- void [unregister_all_my_variables](#) (const void *me)
Release all variables used by an object.
- void [unregister_variable](#) (int var, const void *me)
Release a variable used by me.
- std::ostream & [dump](#) (std::ostream &os) const
Dump all variables for debugging.
- void [assert_emptiness](#) () const
Make sure the dictionary is empty.
- int [allocate_variables](#) (int n)
Allocate n BDD variables.
- void [release_variables](#) (int base, int n)
Release n BDD variables starting at base.
- bool [is_registered_proposition](#) (const [ltl::formula](#) *f, const void *by_me)
- bool [is_registered_state](#) (const [ltl::formula](#) *f, const void *by_me)
- bool [is_registered_acceptance_variable](#) (const [ltl::formula](#) *f, const void *by_me)

Static Public Member Functions

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

Public Attributes

- [fv_map now_map](#)
Maps formulae to "Now" BDD variables.
- [vf_map now_formula_map](#)
Maps "Now" BDD variables to formulae.
- [fv_map var_map](#)
Maps atomic propositions to BDD variables.
- [vf_map var_formula_map](#)
Maps BDD variables to atomic propositions.
- [fv_map acc_map](#)
Maps acceptance conditions to BDD variables.
- [vf_map acc_formula_map](#)
Maps BDD variables to acceptance conditions.
- `bddPair * next_to_now`
Map Next variables to Now variables.
- `bddPair * now_to_next`
Map Now variables to Next variables.

Protected Types

- `typedef std::set< const void * > ref_set`
BDD-variable reference counts.
- `typedef std::map< int, ref_set > vr_map`
- `typedef std::map< const void *, annon_free_list > free_anonymous_list_of_type`
List of unused anonymous variable number for each automaton.

Protected Member Functions

- `void unregister_variable (vr_map::iterator &cur, const void *me)`

Protected Attributes

- [vr_map var_refs](#)
- [free_anonymous_list_of_type free_anonymous_list_of](#)
- `int lvarnum`
number of variables in use in this allocator.

Static Protected Attributes

- static bool [initialized](#)

Whether the BDD library has been initialized.

Private Member Functions

- [bdd_dict](#) (const [bdd_dict](#) &other)
- [bdd_dict](#) & [operator=](#) (const [bdd_dict](#) &other)

Classes

- class [annon_free_list](#)

12.6.1 Detailed Description

Map BDD variables to formulae.

12.6.2 Member Typedef Documentation

12.6.2.1 `typedef std::map<const void*, annon_free_list> spot::bdd_dict::free_anonymous_list_of_type [protected]`

List of unused anonymous variable number for each automaton.

12.6.2.2 `typedef std::map<const ltl::formula*, int> spot::bdd_dict::fv_map`

Formula-to-BDD-variable maps.

12.6.2.3 `typedef std::set<const void*> spot::bdd_dict::ref_set [protected]`

BDD-variable reference counts.

12.6.2.4 `typedef std::map<int, const ltl::formula*> spot::bdd_dict::vf_map`

BDD-variable-to-formula maps.

12.6.2.5 `typedef std::map<int, ref_set> spot::bdd_dict::vr_map [protected]`

12.6.3 Constructor & Destructor Documentation

12.6.3.1 `spot::bdd_dict::bdd_dict ()`

12.6.3.2 `spot::bdd_dict::~~bdd_dict ()`

12.6.3.3 `spot::bdd_dict::bdd_dict (const bdd_dict &other) [private]`

12.6.4 Member Function Documentation

12.6.4.1 int spot::bdd_allocator::allocate_variables (int *n*) [inherited]

Allocate *n* BDD variables.

12.6.4.2 void spot::bdd_dict::assert_emptiness () const

Make sure the dictionary is empty.

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

12.6.4.3 std::ostream& spot::bdd_dict::dump (std::ostream & *os*) const

Dump all variables for debugging.

Parameters:

os The output stream.

12.6.4.4 static void spot::bdd_allocator::initialize () [static, inherited]

Initialize the BDD library.

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

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

Check whether formula *f* has already been registered by *by_me*.

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

12.6.4.8 bdd_dict& spot::bdd_dict::operator= (const bdd_dict & *other*) [private]

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

Register an atomic proposition.

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

Returns:

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

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

Register BDD variables as acceptance variables.

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

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

Duplicate the variable usage of another object.

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

12.6.4.12 int spot::bdd_dict::register_anonymous_variables (int *n*, const void **for_me*)

Register anonymous BDD variables.

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

Returns:

The variable number. Use [bdd_ithvar\(\)](#) or [bdd_nithvar\(\)](#) to convert this to a BDD.

12.6.4.13 int spot::bdd_dict::register_proposition (const [ltl::formula](#) **f*, const void **for_me*)

Register an atomic proposition.

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

Returns:

The variable number. Use [bdd_ithvar\(\)](#) or [bdd_nithvar\(\)](#) to convert this to a BDD.

12.6.4.14 void spot::bdd_dict::register_propositions (bdd *f*, const void **for_me*)

Register BDD variables as atomic propositions.

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

12.6.4.15 int spot::bdd_dict::register_state (const [ltl::formula](#) **f*, const void **for_me*)

Register a couple of Now/Next variables.

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

Returns:

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

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

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

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

Release all variables used by an object.

Usually called in the destructor if *me*.

12.6.4.18 void `spot::bdd_dict::unregister_variable` (vr_map::iterator & *cur*, const void * *me*) [*protected*]

12.6.4.19 void `spot::bdd_dict::unregister_variable` (int *var*, const void * *me*)

Release a variable used by *me*.

12.6.5 Member Data Documentation

12.6.5.1 vf_map `spot::bdd_dict::acc_formula_map`

Maps BDD variables to acceptance conditions.

12.6.5.2 fv_map `spot::bdd_dict::acc_map`

Maps acceptance conditions to BDD variables.

12.6.5.3 free_anonymous_list_of_type `spot::bdd_dict::free_anonymous_list_of` [*protected*]

12.6.5.4 bool `spot::bdd_allocator::initialized` [*static*, *protected*, *inherited*]

Whether the BDD library has been initialized.

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

number of variables in use in this allocator.

12.6.5.6 bddPair* `spot::bdd_dict::next_to_now`

Map Next variables to Now variables.

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

12.6.5.7 vf_map `spot::bdd_dict::now_formula_map`

Maps "Now" BDD variables to formulae.

12.6.5.8 fv_map `spot::bdd_dict::now_map`

Maps formulae to "Now" BDD variables.

12.6.5.9 bddPair* spot::bdd_dict::now_to_next

Map Now variables to Next variables.

Use with BuDDy's bdd_replace() function.

12.6.5.10 vf_map spot::bdd_dict::var_formula_map

Maps BDD variables to atomic propositions.

12.6.5.11 fv_map spot::bdd_dict::var_map

Maps atomic propositions to BDD variables.

12.6.5.12 vr_map spot::bdd_dict::var_refs [protected]

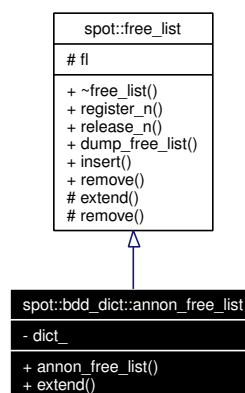
The documentation for this class was generated from the following file:

- [tgba/bdddict.hh](#)

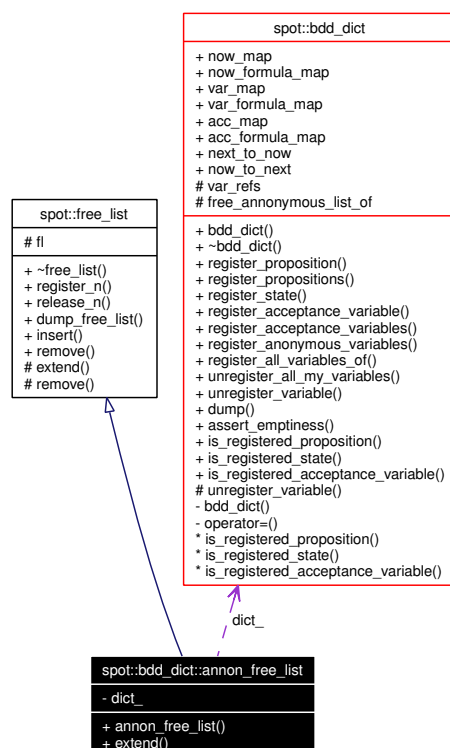
12.7 spot::bdd_dict::annon_free_list Class Reference

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

Inheritance diagram for spot::bdd_dict::annon_free_list:



Collaboration diagram for spot::bdd_dict::annon_free_list:



Public Member Functions

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

Protected Types

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

Protected Member Functions

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

Protected Attributes

- [free_list_type](#) fl
Tracks unused BDD variables.

Private Attributes

- [bdd_dict](#) * dict_

12.7.1 Member Typedef Documentation

12.7.1.1 typedef `std::list<pos_lenght_pair>` [spot::free_list::free_list_type](#) [protected, inherited]

12.7.1.2 typedef `std::pair<int, int>` [spot::free_list::pos_lenght_pair](#) [protected, inherited]

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

12.7.2 Constructor & Destructor Documentation

12.7.2.1 `spot::bdd_dict::annon_free_list::annon_free_list (bdd_dict * d = 0)`

12.7.3 Member Function Documentation

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

Dump the list to `os` for debugging.

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

Allocate `n` integer.

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

Implements [spot::free_list](#).

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

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

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

Find *n* consecutive integers.

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

Returns:

the first integer of the range

12.7.3.5 void spot::free_list::release_n (int *base*, int *n*) [inherited]

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

12.7.3.6 void spot::free_list::remove (free_list_type::iterator *i*, int *base*, int *n*) [protected, inherited]

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

12.7.3.7 void spot::free_list::remove (int *base*, int *n* = 0) [inherited]

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

12.7.4 Member Data Documentation**12.7.4.1 bdd_dict* spot::bdd_dict::annon_free_list::dict_** [private]**12.7.4.2 free_list_type spot::free_list::fl** [protected, inherited]

Tracks unused BDD variables.

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

- [tgba/bdddict.hh](#)

12.8 spot::bdd_less_than Struct Reference

Comparison functor for BDDs.

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

Public Member Functions

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

12.8.1 Detailed Description

Comparison functor for BDDs.

12.8.2 Member Function Documentation

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

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

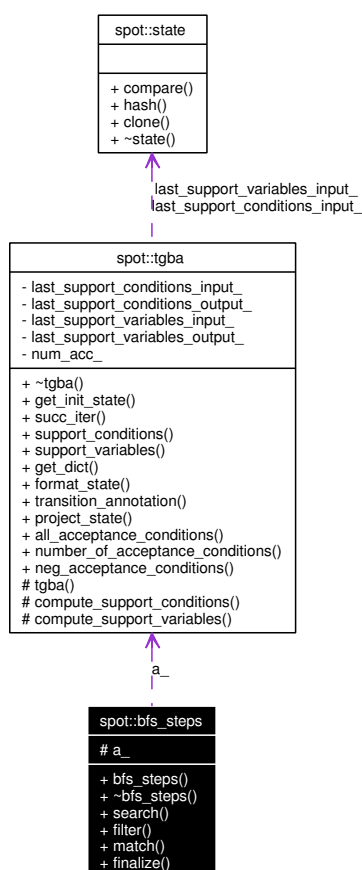
- [misc/bddlt.hh](#)

12.9 spot::bfs_steps Class Reference

Make a BFS in a [spot::tgba](#) to compute a [tgba_run::steps](#).

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

Collaboration diagram for spot::bfs_steps:



Public Member Functions

- [bfs_steps](#) (const [tgba](#) *a)
- virtual [~bfs_steps](#) ()
- const [state](#) * [search](#) (const [state](#) *start, [tgba_run::steps](#) &l)

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

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

Protected Attributes

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

12.9.1 Detailed Description

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

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

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

12.9.2 Constructor & Destructor Documentation

12.9.2.1 spot::bfs_steps::bfs_steps (const tgba * a)

12.9.2.2 virtual spot::bfs_steps::~bfs_steps () [virtual]

12.9.3 Member Function Documentation

12.9.3.1 virtual const state* spot::bfs_steps::filter (const state * s) [pure virtual]

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

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

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

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

12.9.3.2 virtual void spot::bfs_steps::finalize (const std::map< const state *, tgba_run::step, state_ptr_less_than > & father, const tgba_run::step & s, const state * start, tgba_run::steps & l) [virtual]

Append the resulting path to the resulting run.

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

12.9.3.3 virtual bool spot::bfs_steps::match (tgba_run::step & step, const state * dest) [pure virtual]

Whether a new transition completes a path.

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

Parameters:

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

Returns:

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

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

12.9.3.4 const state* spot::bfs_steps::search (const state * start, tgba_run::steps & l)

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

Returns:

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

12.9.4 Member Data Documentation

12.9.4.1 const tgba* spot::bfs_steps::a_ [protected]

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

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

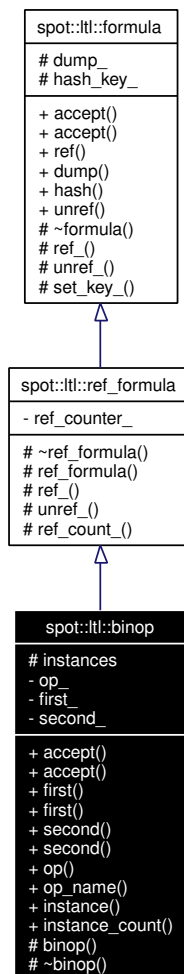
- [tgbaalgos/bfssteps.hh](#)

12.10 spot::ltl::binop Class Reference

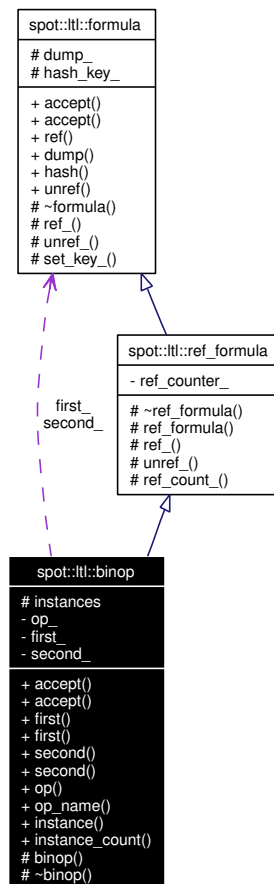
Binary operator.

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

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



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



Public Types

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

Public Member Functions

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

Get the second operand.

- `formula * second ()`

Get the second operand.

- `type op () const`

Get the type of this operator.

- `const char * op_name () const`

Get the type of this operator, as a string.

- `formula * ref ()`

clone this node

- `const std::string & dump () const`

Return a canonic representation of the formula.

- `const size_t hash () const`

Return a hash_key for the formula.

Static Public Member Functions

- static `binop * instance (type op, formula *first, formula *second)`
- static unsigned `instance_count ()`

Number of instantiated binary operators. For debugging.

- static void `unref (formula *f)`

release this node

Protected Types

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

Protected Member Functions

- `binop (type op, formula *first, formula *second)`
- virtual `~binop ()`
- void `ref_ ()`

increment reference counter if any

- bool `unref_ ()`

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

- unsigned `ref_count_ ()`

Number of references to this formula.

- void [set_key_\(\)](#)
Compute key_ from dump_.

Protected Attributes

- std::string [dump_](#)
The canonic representation of the formula.
- size_t [hash_key_](#)
The hash key of this formula.

Static Protected Attributes

- static [map instances](#)

Private Attributes

- type [op_](#)
- formula * [first_](#)
- formula * [second_](#)

12.10.1 Detailed Description

Binary operator.

12.10.2 Member Typedef Documentation

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

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

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

12.10.3 Member Enumeration Documentation

12.10.3.1 `enum spot::ltl::binop::type`

Different kinds of binary opertaors

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

Enumeration values:

Xor

Implies

*Equiv**U**R*

12.10.4 Constructor & Destructor Documentation

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

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

12.10.5 Member Function Documentation

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

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

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

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

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

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

12.10.5.3 const std::string& spot::ltl::formula::dump () const [inherited]

Return a canonic representation of the formula.

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

Get the first operand.

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

Get the first operand.

12.10.5.6 const size_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash_key for the formula.

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

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

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

Number of instantiated binary operators. For debugging.

12.10.5.9 *type* spot::ltl::binop::op () const

Get the type of this operator.

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

Get the type of this operator, as a string.

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

clone this node

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

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

increment reference counter if any

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

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

Number of references to this formula.

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

Get the second operand.

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

Get the second operand.

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

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

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

release this node

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

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

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

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

12.10.6 Member Data Documentation**12.10.6.1** `std::string spot::ltl::formula::dump_` [protected, inherited]

The canonic representation of the formula.

12.10.6.2 `formula* spot::ltl::binop::first_` [private]

12.10.6.3 `size_t spot::ltl::formula::hash_key_` [protected, inherited]

The hash key of this formula.

Initialized by `set_key_()`.

12.10.6.4 `map spot::ltl::binop::instances` [static, protected]

12.10.6.5 `type spot::ltl::binop::op_` [private]

12.10.6.6 `formula* spot::ltl::binop::second_` [private]

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

- `ltlast/binop.hh`

12.11 `spot::char_ptr_less_than` Struct Reference

Strict Weak Ordering for `char*`.

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

Public Member Functions

- `bool operator()` (`const char *left`, `const char *right`) `const`

12.11.1 Detailed Description

Strict Weak Ordering for `char*`.

This is meant to be used as a comparison functor for STL `map` whose key are of type `const char*`.

For instance here is how one could declare a map of `const state*`.

```
std::map<const char*, int, spot::state_ptr_less_than> seen;
```

12.11.2 Member Function Documentation

12.11.2.1 `bool spot::char_ptr_less_than::operator()` (`const char * left`, `const char * right`) `const` [inline]

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

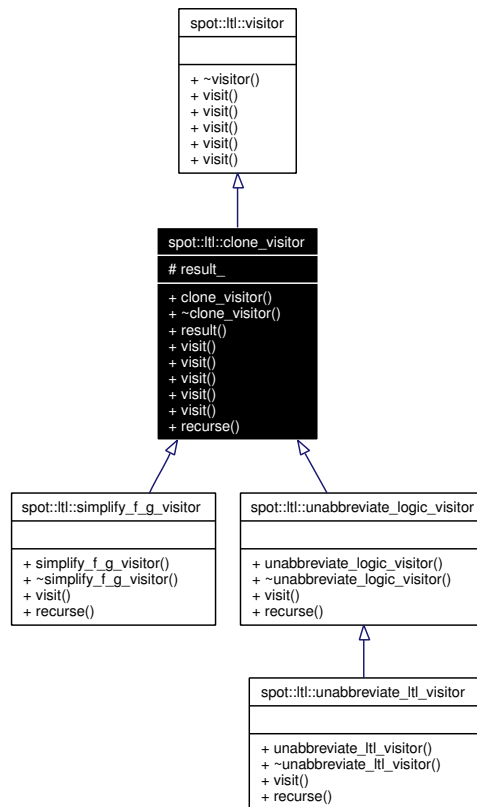
- `misc/ltstr.hh`

12.12 spot::ltl::clone_visitor Class Reference

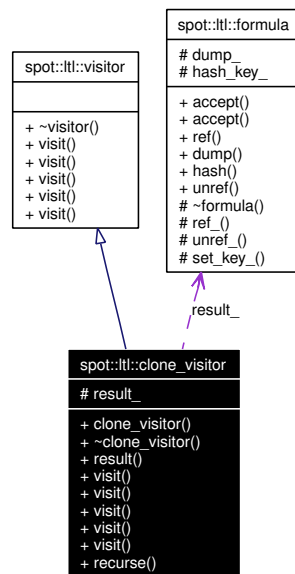
Clone a formula.

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

Inheritance diagram for spot::ltl::clone_visitor:



Collaboration diagram for spot::ltl::clone_visitor:



Public Member Functions

- [clone_visitor](#) ()
- virtual [~clone_visitor](#) ()
- [formula](#) * [result](#) () const
- void [visit](#) ([atomic_prop](#) *ap)
- void [visit](#) ([unop](#) *uo)
- void [visit](#) ([binop](#) *bo)
- void [visit](#) ([multop](#) *mo)
- void [visit](#) ([constant](#) *c)
- virtual [formula](#) * [recurse](#) ([formula](#) *f)

Protected Attributes

- [formula](#) * [result_](#)

12.12.1 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.

12.12.2 Constructor & Destructor Documentation

12.12.2.1 spot::ltl::clone_visitor::clone_visitor ()

12.12.2.2 virtual spot::ltl::clone_visitor::~~clone_visitor () [virtual]

12.12.3 Member Function Documentation

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

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

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

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

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

12.12.3.4 void [spot::ltl::clone_visitor::visit](#) ([multop](#) **mo*) [virtual]

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

12.12.3.5 void [spot::ltl::clone_visitor::visit](#) ([binop](#) **bo*) [virtual]

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

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

12.12.3.6 void [spot::ltl::clone_visitor::visit](#) ([unop](#) **uo*) [virtual]

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

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

12.12.3.7 void [spot::ltl::clone_visitor::visit](#) ([atomic_prop](#) **ap*) [virtual]

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

12.12.4 Member Data Documentation

12.12.4.1 [formula](#)* [spot::ltl::clone_visitor::result_](#) [protected]

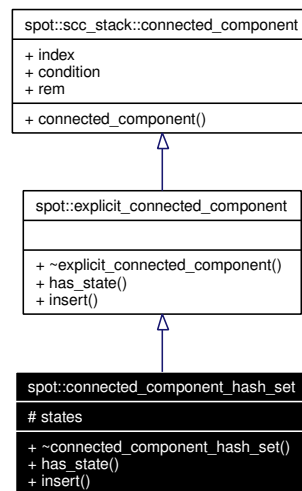
The documentation for this class was generated from the following file:

- [ltlvisit/clone.hh](#)

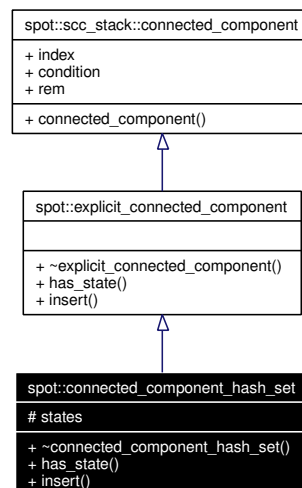
12.13 spot::connected_component_hash_set Class Reference

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

Inheritance diagram for [spot::connected_component_hash_set](#):



Collaboration diagram for `spot::connected_component_hash_set`:



Public Member Functions

- virtual `~connected_component_hash_set()`
- virtual const `state * has_state (const state *s)` const
Check if the SCC contains states s.
- virtual void `insert (const state *s)`
Insert a new state in the SCC.

Public Attributes

- int `index`
Index of the SCC.

- bdd [condition](#)
- `std::list< const state * > rem`

Protected Types

- `typedef Sgi::hash_set< const state *, state_ptr_hash, state_ptr_equal > set_type`

Protected Attributes

- [set_type](#) [states](#)

12.13.1 Detailed Description

A straightforward implementation of [explicit_connected_component](#) using a hash.

12.13.2 Member Typedef Documentation

12.13.2.1 `typedef Sgi::hash_set<const state*, state_ptr_hash, state_ptr_equal> spot::connected_component_hash_set::set_type [protected]`

12.13.3 Constructor & Destructor Documentation

12.13.3.1 `virtual spot::connected_component_hash_set::~~connected_component_hash_set () [inline, virtual]`

12.13.4 Member Function Documentation

12.13.4.1 `virtual const state* spot::connected_component_hash_set::has_state (const state * s) const [virtual]`

Check if the SCC contains states *s*.

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

Implements [spot::explicit_connected_component](#).

12.13.4.2 `virtual void spot::connected_component_hash_set::insert (const state * s) [virtual]`

Insert a new state in the SCC.

Implements [spot::explicit_connected_component](#).

12.13.5 Member Data Documentation

12.13.5.1 `bdd spot::scc_stack::connected_component::condition [inherited]`

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

12.13.5.2 int [spot::scc_stack::connected_component::index](#) [inherited]

Index of the SCC.

12.13.5.3 std::list<const [state*](#)> [spot::scc_stack::connected_component::rem](#) [inherited]**12.13.5.4** set_type [spot::connected_component_hash_set::states](#) [protected]

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

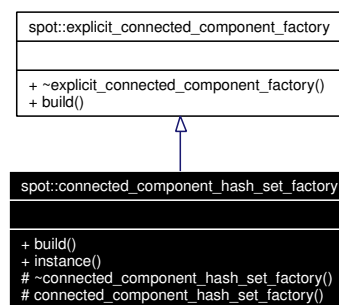
- [tgbaalgos/gtec/explscc.hh](#)

12.14 spot::connected_component_hash_set_factory Class Reference

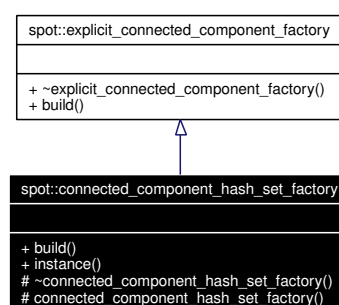
Factory for [connected_component_hash_set](#).

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

Inheritance diagram for spot::connected_component_hash_set_factory:



Collaboration diagram for spot::connected_component_hash_set_factory:

**Public Member Functions**

- virtual [connected_component_hash_set](#) * `build ()` const
Create an [explicit_connected_component](#).

Static Public Member Functions

- static const [connected_component_hash_set_factory](#) * [instance](#) ()

Get the unique instance of this class.

Protected Member Functions

- virtual [~connected_component_hash_set_factory](#) ()
- [connected_component_hash_set_factory](#) ()

Construction is forbidden.

12.14.1 Detailed Description

Factory for [connected_component_hash_set](#).

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

12.14.2 Constructor & Destructor Documentation

12.14.2.1 virtual [spot::connected_component_hash_set_factory::~connected_component_hash_set_factory](#) () [inline, protected, virtual]

12.14.2.2 [spot::connected_component_hash_set_factory::connected_component_hash_set_factory](#) () [protected]

Construction is forbidden.

12.14.3 Member Function Documentation

12.14.3.1 virtual [connected_component_hash_set*](#) [spot::connected_component_hash_set_factory::build](#) () const [virtual]

Create an [explicit_connected_component](#).

Implements [spot::explicit_connected_component_factory](#).

12.14.3.2 static const [connected_component_hash_set_factory*](#) [spot::connected_component_hash_set_factory::instance](#) () [static]

Get the unique instance of this class.

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

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

12.15 spot::ltl::const_visitor Struct Reference

Formula visitor that cannot modify the formula.

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

Public Member Functions

- virtual [~const_visitor](#) ()
- 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

12.15.1 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.

12.15.2 Constructor & Destructor Documentation

12.15.2.1 virtual [spot::ltl::const_visitor::~~const_visitor](#) () [inline, virtual]

12.15.3 Member Function Documentation

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

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

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

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

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

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

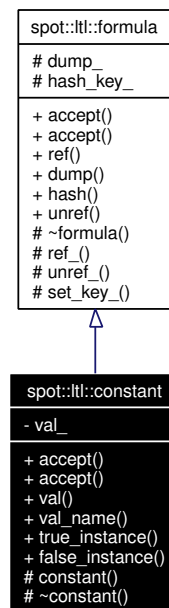
- [ltlast/visitor.hh](#)

12.16 spot::ltl::constant Class Reference

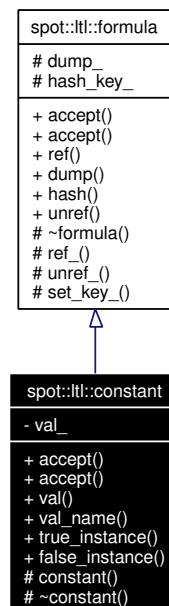
A constant (True or False).

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

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



Collaboration diagram for `spot::ltl::constant`:



Public Types

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

Public Member Functions

- virtual void `accept` (`visitor &v`)

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

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

Static Public Member Functions

- static `constant * true_instance ()`
Get the sole instance of spot::ltl::constant::constant(True).
- static `constant * false_instance ()`
Get the sole instance of spot::ltl::constant::constant(False).
- static void `unref (formula *f)`
release this node

Protected Member Functions

- `constant (type val)`
- virtual `~constant ()`
- virtual void `ref_ ()`
increment reference counter if any
- virtual bool `unref_ ()`
decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).
- void `set_key_ ()`
Compute key_ from dump_.

Protected Attributes

- std::string [dump_](#)
The canonic representation of the formula.
- size_t [hash_key_](#)
The hash key of this formula.

Private Attributes

- [type val_](#)

12.16.1 Detailed Description

A constant (True or False).

12.16.2 Member Enumeration Documentation**12.16.2.1 enum [spot::ltl::constant::type](#)**

Enumeration values:

False

True

12.16.3 Constructor & Destructor Documentation

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

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

12.16.4 Member Function Documentation

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

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

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

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

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

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

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

Return a canonic representation of the formula.

12.16.4.4 static [constant*](#) spot::ltl::constant::false_instance () [static]

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

12.16.4.5 const size_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash_key for the formula.

12.16.4.6 [formula*](#) spot::ltl::formula::ref () [inherited]

clone this node

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

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

increment reference counter if any

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

12.16.4.8 void spot::ltl::formula::set_key_ () [protected, inherited]

Compute key_ from dump_.

Should be called once in each object, after dump_ has been set.

12.16.4.9 static [constant*](#) spot::ltl::constant::true_instance () [static]

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

12.16.4.10 static void spot::ltl::formula::unref ([formula *f](#)) [static, inherited]

release this node

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

12.16.4.11 virtual bool spot::ltl::formula::unref_ () [protected, virtual, inherited]

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

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

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

Return the value of the constant.

12.16.4.13 const char* spot::ltl::constant::val_name () const

Return the value of the constant as a string.

12.16.5 Member Data Documentation

12.16.5.1 `std::string spot::lfl::formula::dump_` [protected, inherited]

The canonic representation of the formula.

12.16.5.2 `size_t spot::lfl::formula::hash_key_` [protected, inherited]

The hash key of this formula.

Initialized by `set_key_()`.

12.16.5.3 `type spot::lfl::constant::val_` [private]

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

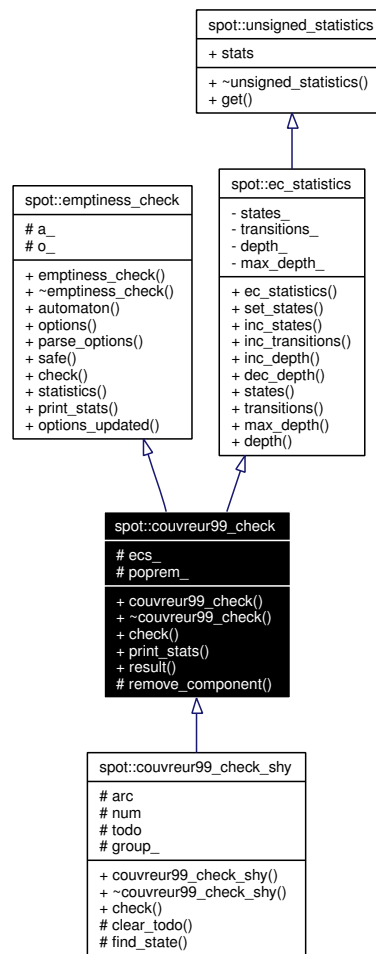
- `lflast/constant.hh`

12.17 `spot::couvreur99_check` Class Reference

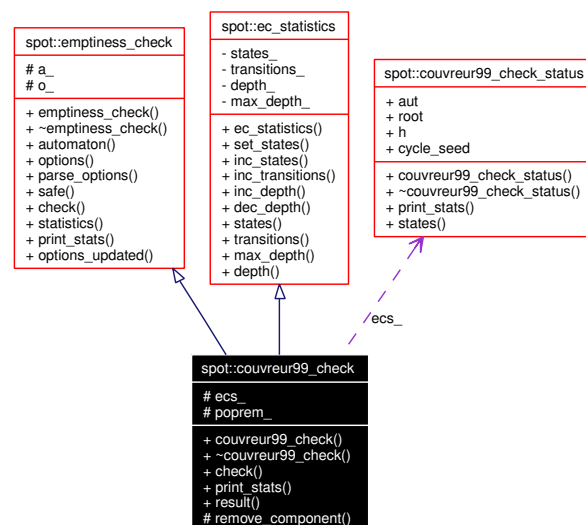
An implementation of the Couvreur99 emptiness-check algorithm.

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

Inheritance diagram for `spot::couvreur99_check`:



Collaboration diagram for spot::couvreur99_check:



Public Types

- typedef unsigned(unsigned_statistics::* [unsigned_fun](#))() const
- typedef std::map< const char *, [unsigned_fun](#), [char_ptr_less_than](#) > [stats_map](#)

Public Member Functions

- [couvreur99_check](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)(), const [numbered_state_heap_factory](#) *nshf=[numbered_state_heap_hash_map_factory::instance](#)())
- virtual [~couvreur99_check](#) ()
- virtual [emptiness_check_result](#) * [check](#) ()
Check whether the automaton's language is empty.
- virtual std::ostream & [print_stats](#) (std::ostream &os) const
Print statistics, if any.
- const [couvreur99_check_status](#) * [result](#) () const
Return the status of the emptiness-check.
- const [tgba](#) * [automaton](#) () const
The automaton that this emptiness-check inspects.
- const [option_map](#) & [options](#) () const
Return the options parametrizing how the emptiness check is realized.
- const char * [parse_options](#) (char *options)
Modify the algorithm options.
- virtual bool [safe](#) () const
Return false iff accepting_run() can return 0 for non-empty automata.
- virtual const [unsigned_statistics](#) * [statistics](#) () const
Return statistics, if available.
- virtual void [options_updated](#) (const [option_map](#) &old)
Notify option updates.
- void [set_states](#) (unsigned n)
- void [inc_states](#) ()
- void [inc_transitions](#) ()
- void [inc_depth](#) (unsigned n=1)
- void [dec_depth](#) (unsigned n=1)
- unsigned [states](#) () const
- unsigned [transitions](#) () const
- unsigned [max_depth](#) () const
- unsigned [depth](#) () const
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

Protected Member Functions

- void `remove_component` (const `state` *start_delete)

Remove a strongly component from the hash.

Protected Attributes

- `couvreur99_check_status` * `ecs_`
- bool `poprem_`

Whether to store the state to be removed.

- const `tgba` * `a_`

The automaton.

- `option_map` `o_`

The options.

12.17.1 Detailed Description

An implementation of the Couvreur99 emptiness-check algorithm.

See the documentation for `spot::couvreur99`.

12.17.2 Member Typedef Documentation

12.17.2.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map` [inherited]

12.17.2.2 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const` [inherited]

12.17.3 Constructor & Destructor Documentation

12.17.3.1 `spot::couvreur99_check::couvreur99_check (const tgba * a, option_map o = option_map(), const numbered_state_heap_factory * nshf = numbered_state_heap_hash_map_factory::instance())`

12.17.3.2 `virtual spot::couvreur99_check::~~couvreur99_check ()` [virtual]

12.17.4 Member Function Documentation

12.17.4.1 `const tgba* spot::emptiness_check::automaton () const` [inline, inherited]

The automaton that this emptiness-check inspects.

12.17.4.2 `virtual emptiness_check_result* spot::couvreur99_check::check ()` [virtual]

Check whether the automaton's language is empty.

Implements `spot::emptiness_check`.

Reimplemented in `spot::couvreur99_check_shy`.

12.17.4.3 `void spot::ec_statistics::dec_depth (unsigned n = 1)` [inline, inherited]**12.17.4.4** `unsigned spot::ec_statistics::depth () const` [inline, inherited]**12.17.4.5** `unsigned spot::unsigned_statistics::get (const char * str) const` [inline, inherited]**12.17.4.6** `void spot::ec_statistics::inc_depth (unsigned n = 1)` [inline, inherited]**12.17.4.7** `void spot::ec_statistics::inc_states ()` [inline, inherited]**12.17.4.8** `void spot::ec_statistics::inc_transitions ()` [inline, inherited]**12.17.4.9** `unsigned spot::ec_statistics::max_depth () const` [inline, inherited]**12.17.4.10** `const option_map& spot::emptiness_check::options () const` [inline, inherited]

Return the options parametrizing how the emptiness check is realized.

12.17.4.11 `virtual void spot::emptiness_check::options_updated (const option_map & old)` [virtual, inherited]

Notify option updates.

12.17.4.12 `const char* spot::emptiness_check::parse_options (char * options)` [inherited]

Modify the algorithm options.

12.17.4.13 `virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const` [virtual]

Print statistics, if any.

Reimplemented from `spot::emptiness_check`.

12.17.4.14 `void spot::couvreur99_check::remove_component (const state * start_delete)` [protected]

Remove a strongly component from the hash.

This function remove all accessible state from a given state. In other words, it removes the strongly connected component that contains this state.

12.17.4.15 `const couvreur99_check_status* spot::couvreur99_check::result () const`

Return the status of the emptiness-check.

When `check()` succeed, the status should be passed along to `spot::counter_example`.

This status should not be deleted, it is a pointer to a member of this class that will be deleted when the `couvreur99` object is deleted.

12.17.4.16 `virtual bool spot::emptiness_check::safe () const` [virtual, inherited]

Return false iff accepting_run() can return 0 for non-empty automata.

12.17.4.17 `void spot::ec_statistics::set_states (unsigned n)` [inline, inherited]**12.17.4.18** `unsigned spot::ec_statistics::states () const` [inline, inherited]**12.17.4.19** `virtual const unsigned_statistics* spot::emptiness_check::statistics () const` [virtual, inherited]

Return statistics, if available.

12.17.4.20 `unsigned spot::ec_statistics::transitions () const` [inline, inherited]**12.17.5** Member Data Documentation**12.17.5.1** `const tgba* spot::emptiness_check::a_` [protected, inherited]

The automaton.

12.17.5.2 `couvreur99_check_status* spot::couvreur99_check::ecs_` [protected]**12.17.5.3** `option_map spot::emptiness_check::o_` [protected, inherited]

The options.

12.17.5.4 `bool spot::couvreur99_check::poprem_` [protected]

Whether to store the state to be removed.

12.17.5.5 `stats_map spot::unsigned_statistics::stats` [inherited]

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

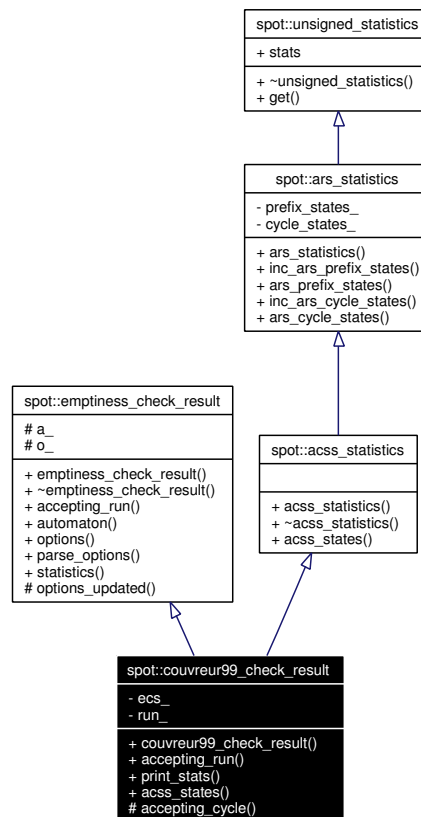
- `tgbaalgos/gtec/gtec.hh`

12.18 spot::couvreur99_check_result Class Reference

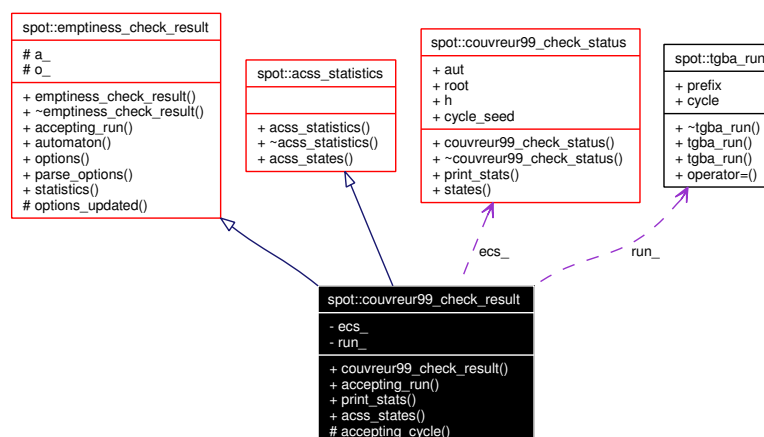
Compute a counter example from a `spot::couvreur99_check_status`.

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


Inheritance diagram for spot::couvreur99_check_result:



Collaboration diagram for spot::couvreur99_check_result:



Public Types

- typedef unsigned(unsigned_statistics::* [unsigned_fun](#))() const
- typedef std::map< const char *, [unsigned_fun](#), [char_ptr_less_than](#) > stats_map

Public Member Functions

- `couvreur99_check_result` (const `couvreur99_check_status` *ecs, `option_map` o=`option_map`())
- virtual `tgba_run` * `accepting_run` ()
Return a run accepted by the automata passed to the emptiness check.
- void `print_stats` (std::ostream &os) const
- virtual unsigned `acss_states` () const
Number of states in the search space for the accepting cycle.
- const `tgba` * `automaton` () const
The automaton on which an `accepting_run()` was found.
- const `option_map` & `options` () const
Return the options parametrizing how the accepting run is computed.
- const char * `parse_options` (char *options)
Modify the algorithm options.
- virtual const unsigned `statistics` * `statistics` () const
Return statistics, if available.
- void `inc_ars_prefix_states` ()
- unsigned `ars_prefix_states` () const
- void `inc_ars_cycle_states` ()
- unsigned `ars_cycle_states` () const
- unsigned `get` (const char *str) const

Public Attributes

- `stats_map` stats

Protected Member Functions

- void `accepting_cycle` ()
- virtual void `options_updated` (const `option_map` &old)
Notify option updates.

Protected Attributes

- const `tgba` * `a_`
The automaton.
- `option_map` `o_`
The options.

Private Attributes

- const [couvereur99_check_status](#) * [ecs_](#)
- [tgba_run](#) * [run_](#)

12.18.1 Detailed Description

Compute a counter example from a [spot::couvereur99_check_status](#).

12.18.2 Member Typedef Documentation

12.18.2.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map` [inherited]

12.18.2.2 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const` [inherited]

12.18.3 Constructor & Destructor Documentation

12.18.3.1 `spot::couvereur99_check_result::couvereur99_check_result (const couvreur99_check_status * ecs, option_map o = option_map())`

12.18.4 Member Function Documentation

12.18.4.1 `void spot::couvereur99_check_result::accepting_cycle ()` [protected]

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

12.18.4.2 `virtual tgba_run* spot::couvereur99_check_result::accepting_run ()` [virtual]

Return a run accepted by the automata passed to the emptiness check.

This method might actually compute the acceptance run. (Not all emptiness check algorithms actually produce a counter-example as a side-effect of checking emptiness, some need some post-processing.)

This can also return 0 if the emptiness check algorithm cannot produce a counter example (that does not mean there is no counter-example; the mere existence of an instance of this class asserts the existence of a counter-example).

Reimplemented from [spot::emptiness_check_result](#).

12.18.4.3 `virtual unsigned spot::couvereur99_check_result::acss_states () const` [virtual]

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

Implements [spot::acss_statistics](#).

12.18.4.4 `unsigned spot::ars_statistics::ars_cycle_states () const` [inline, inherited]

12.18.4.5 `unsigned spot::ars_statistics::ars_prefix_states () const` [inline, inherited]

12.18.4.6 `const tgba* spot::emptiness_check_result::automaton () const` [inline, inherited]

The automaton on which an [accepting_run\(\)](#) was found.

12.18.4.7 `unsigned spot::unsigned_statistics::get (const char * str) const` [inline, inherited]

12.18.4.8 `void spot::ars_statistics::inc_ars_cycle_states ()` [inline, inherited]

12.18.4.9 `void spot::ars_statistics::inc_ars_prefix_states ()` [inline, inherited]

12.18.4.10 `const option_map& spot::emptiness_check_result::options () const` [inline, inherited]

Return the options parametrizing how the accepting run is computed.

12.18.4.11 `virtual void spot::emptiness_check_result::options_updated (const option_map & old)` [protected, virtual, inherited]

Notify option updates.

12.18.4.12 `const char* spot::emptiness_check_result::parse_options (char * options)` [inherited]

Modify the algorithm options.

12.18.4.13 `void spot::couvreur99_check_result::print_stats (std::ostream & os) const`

12.18.4.14 `virtual const unsigned_statistics* spot::emptiness_check_result::statistics () const` [virtual, inherited]

Return statistics, if available.

12.18.5 Member Data Documentation

12.18.5.1 `const tgba* spot::emptiness_check_result::a_` [protected, inherited]

The automaton.

12.18.5.2 `const couvreur99_check_status* spot::couvreur99_check_result::ecs_` [private]

12.18.5.3 `option_map spot::emptiness_check_result::o_` [protected, inherited]

The options.

12.18.5.4 `tgba_run* spot::couvreur99_check_result::run_` [private]

12.18.5.5 stats_map spot::unsigned_statistics::stats [inherited]

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

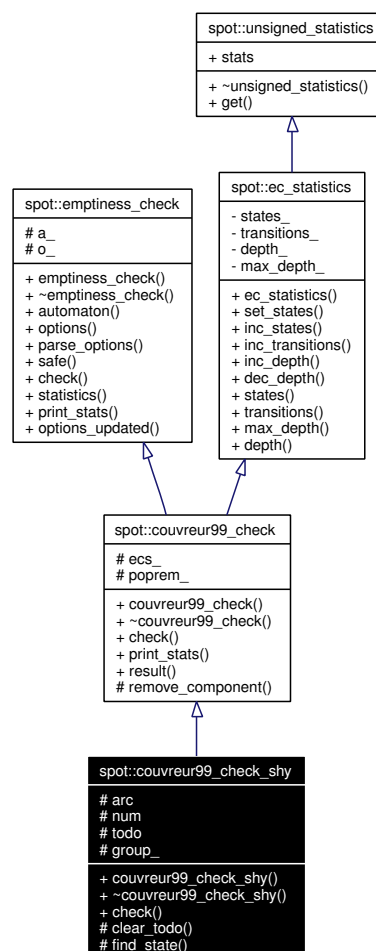
- [tgbaalgos/gtec/ce.hh](#)

12.19 spot::couvreur99_check_shy Class Reference

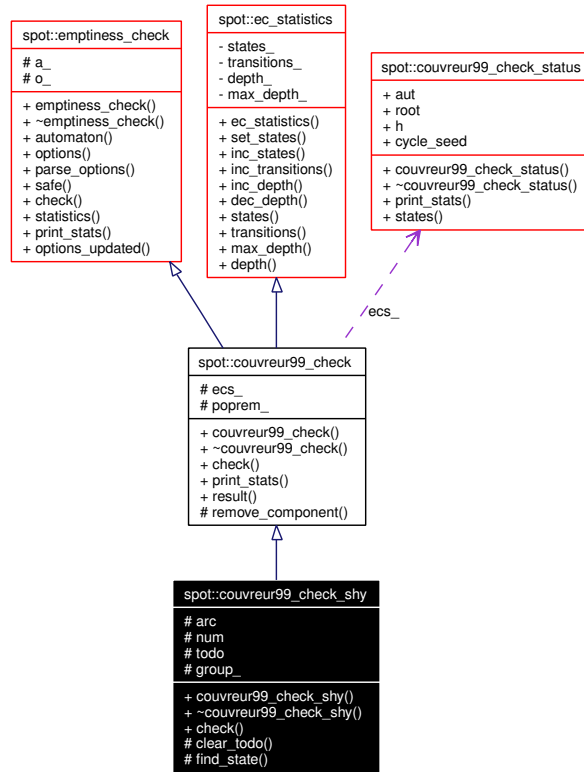
A version of [spot::couvreur99_check](#) that tries to visit known states first.

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

Inheritance diagram for spot::couvreur99_check_shy:



Collaboration diagram for spot::couvreur99_check_shy:



Public Types

- typedef unsigned(unsigned_statistics::* [unsigned_fun](#))() const
- typedef std::map< const char *, [unsigned_fun](#), [char_ptr_less_than](#) > [stats_map](#)

Public Member Functions

- [couvreur99_check_shy](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)(), const [numbered_state_heap_factory](#) *nshf=[numbered_state_heap_hash_map_factory::instance\(\)](#))
- virtual [~couvreur99_check_shy](#) ()
- virtual [emptiness_check_result](#) * [check](#) ()
Check whether the automaton's language is empty.
- virtual std::ostream & [print_stats](#) (std::ostream &os) const
Print statistics, if any.
- const [couvreur99_check_status](#) * [result](#) () const
Return the status of the emptiness-check.
- const [tgba](#) * [automaton](#) () const
The automaton that this emptiness-check inspects.
- const [option_map](#) & [options](#) () const
Return the options parametrizing how the emptiness check is realized.

- const char * [parse_options](#) (char *options)
Modify the algorithm options.
- virtual bool [safe](#) () const
Return false iff accepting_run() can return 0 for non-empty automata.
- virtual const [unsigned_statistics](#) * [statistics](#) () const
Return statistics, if available.
- virtual void [options_updated](#) (const [option_map](#) &old)
Notify option updates.
- void [set_states](#) (unsigned n)
- void [inc_states](#) ()
- void [inc_transitions](#) ()
- void [inc_depth](#) (unsigned n=1)
- void [dec_depth](#) (unsigned n=1)
- unsigned [states](#) () const
- unsigned [transitions](#) () const
- unsigned [max_depth](#) () const
- unsigned [depth](#) () const
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

Protected Types

- typedef std::list< [successor](#) > [succ_queue](#)
- typedef std::list< [todo_item](#) > [todo_list](#)

Protected Member Functions

- void [clear_todo](#) ()
- virtual int * [find_state](#) (const [state](#) *s)
find the SCC number of a unprocessed state.
- void [remove_component](#) (const [state](#) *start_delete)
Remove a strongly component from the hash.

Protected Attributes

- std::stack< bdd > [arc](#)
- int [num](#)
- [todo_list](#) todo
- bool [group_](#)
- [couvreur99_check_status](#) * [ecs_](#)

- bool [poprem_](#)
Whether to store the state to be removed.
- const [tgba](#) * [a_](#)
The automaton.
- [option_map](#) [o_](#)
The options.

Classes

- struct [successor](#)
- struct [todo_item](#)

12.19.1 Detailed Description

A version of [spot::couvreur99_check](#) that tries to visit known states first.

See the documentation for [spot::couvreur99](#).

12.19.2 Member Typedef Documentation

12.19.2.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map` [inherited]

12.19.2.2 `typedef std::list<successor> spot::couvreur99_check_shy::succ_queue` [protected]

12.19.2.3 `typedef std::list<todo_item> spot::couvreur99_check_shy::todo_list` [protected]

12.19.2.4 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const` [inherited]

12.19.3 Constructor & Destructor Documentation

12.19.3.1 `spot::couvreur99_check_shy::couvreur99_check_shy (const tgba * a, option_map o = option_map (), const numbered_state_heap_factory * nshf = numbered_state_heap_hash_map_factory::instance ())`

12.19.3.2 `virtual spot::couvreur99_check_shy::~~couvreur99_check_shy ()` [virtual]

12.19.4 Member Function Documentation

12.19.4.1 `const tgba* spot::emptiness_check::automaton () const` [inline, inherited]

The automaton that this emptiness-check inspects.

12.19.4.2 virtual [emptiness_check_result*](#) spot::couvreur99_check_shy::check () [virtual]

Check whether the automaton's language is empty.

Reimplemented from [spot::couvreur99_check](#).

12.19.4.3 void spot::couvreur99_check_shy::clear_todo () [protected]**12.19.4.4** void spot::ec_statistics::dec_depth (unsigned *n* = 1) [inline, inherited]**12.19.4.5** unsigned spot::ec_statistics::depth () const [inline, inherited]**12.19.4.6** virtual int* spot::couvreur99_check_shy::find_state (const [state](#) * *s*) [protected, virtual]

find the SCC number of a unprocessed state.

Sometimes we want to modify some of the above structures when looking up a new state. This happens for instance when find() must perform inclusion checking and add new states to process to TODO during this step. (Because TODO must be known, sub-classing [spot::numbered_state_heap](#) is not enough.) Then overriding this method is the way to go.

12.19.4.7 unsigned spot::unsigned_statistics::get (const char * *str*) const [inline, inherited]**12.19.4.8** void spot::ec_statistics::inc_depth (unsigned *n* = 1) [inline, inherited]**12.19.4.9** void spot::ec_statistics::inc_states () [inline, inherited]**12.19.4.10** void spot::ec_statistics::inc_transitions () [inline, inherited]**12.19.4.11** unsigned spot::ec_statistics::max_depth () const [inline, inherited]**12.19.4.12** const [option_map&](#) spot::emptiness_check::options () const [inline, inherited]

Return the options parametrizing how the emptiness check is realized.

12.19.4.13 virtual void spot::emptiness_check::options_updated (const [option_map](#) & *old*) [virtual, inherited]

Notify option updates.

12.19.4.14 const char* spot::emptiness_check::parse_options (char * *options*) [inherited]

Modify the algorithm options.

12.19.4.15 `virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const` [virtual, inherited]

Print statistics, if any.

Reimplemented from [spot::emptiness_check](#).

12.19.4.16 `void spot::couvreur99_check::remove_component (const state * start_delete)` [protected, inherited]

Remove a strongly component from the hash.

This function remove all accessible state from a given state. In other words, it removes the strongly connected component that contains this state.

12.19.4.17 `const couvreur99_check_status* spot::couvreur99_check::result () const` [inherited]

Return the status of the emptiness-check.

When [check\(\)](#) succeed, the status should be passed along to `spot::counter_example`.

This status should not be deleted, it is a pointer to a member of this class that will be deleted when the `couvreur99` object is deleted.

12.19.4.18 `virtual bool spot::emptiness_check::safe () const` [virtual, inherited]

Return false iff `accepting_run()` can return 0 for non-empty automata.

12.19.4.19 `void spot::ec_statistics::set_states (unsigned n)` [inline, inherited]

12.19.4.20 `unsigned spot::ec_statistics::states () const` [inline, inherited]

12.19.4.21 `virtual const unsigned_statistics* spot::emptiness_check::statistics () const` [virtual, inherited]

Return statistics, if available.

12.19.4.22 `unsigned spot::ec_statistics::transitions () const` [inline, inherited]

12.19.5 Member Data Documentation

12.19.5.1 `const tgba* spot::emptiness_check::a_` [protected, inherited]

The automaton.

12.19.5.2 `std::stack<bdd> spot::couvreur99_check_shy::arc` [protected]

12.19.5.3 `couvreur99_check_status* spot::couvreur99_check::ecs_` [protected, inherited]

12.19.5.4 `bool spot::couvreur99_check_shy::group_` [protected]

12.19.5.5 int [spot::couvreur99_check_shy::num](#) [protected]

12.19.5.6 [option_map](#) [spot::emptiness_check::o](#) [protected, inherited]

The options.

12.19.5.7 bool [spot::couvreur99_check::poprem](#) [protected, inherited]

Whether to store the state to be removed.

12.19.5.8 [stats_map](#) [spot::unsigned_statistics::stats](#) [inherited]

12.19.5.9 [todo_list](#) [spot::couvreur99_check_shy::todo](#) [protected]

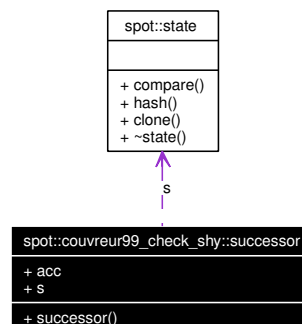
The documentation for this class was generated from the following file:

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

12.20 spot::couvreur99_check_shy::successor Struct Reference

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

Collaboration diagram for spot::couvreur99_check_shy::successor:



Public Member Functions

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

Public Attributes

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

12.20.1 Constructor & Destructor Documentation

12.20.1.1 [spot::couvreur99_check_shy::successor::successor](#) (bdd [acc](#), const [spot::state](#) * [s](#))
[inline]

12.20.2 Member Data Documentation

12.20.2.1 bdd [spot::couvreur99_check_shy::successor::acc](#)

12.20.2.2 const [spot::state](#)* [spot::couvreur99_check_shy::successor::s](#)

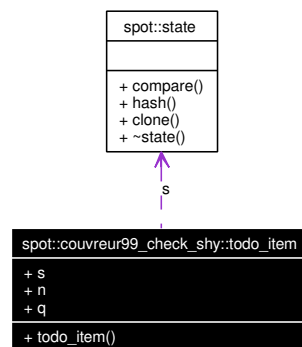
The documentation for this struct was generated from the following file:

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

12.21 spot::couvreur99_check_shy::todo_item Struct Reference

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

Collaboration diagram for spot::couvreur99_check_shy::todo_item:



Public Member Functions

- [todo_item](#) (const [state](#) *s, int n)

Public Attributes

- const [state](#) * s
- int n
- [succ_queue](#) q

12.21.1 Constructor & Destructor Documentation

12.21.1.1 spot::couvreur99_check_shy::todo_item::todo_item (const [state](#) * s, int n) `[inline]`

12.21.2 Member Data Documentation

12.21.2.1 int [spot::couvreur99_check_shy::todo_item::n](#)

12.21.2.2 [succ_queue](#) [spot::couvreur99_check_shy::todo_item::q](#)

12.21.2.3 const state* spot::couvereur99_check_shy::todo_item::s

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

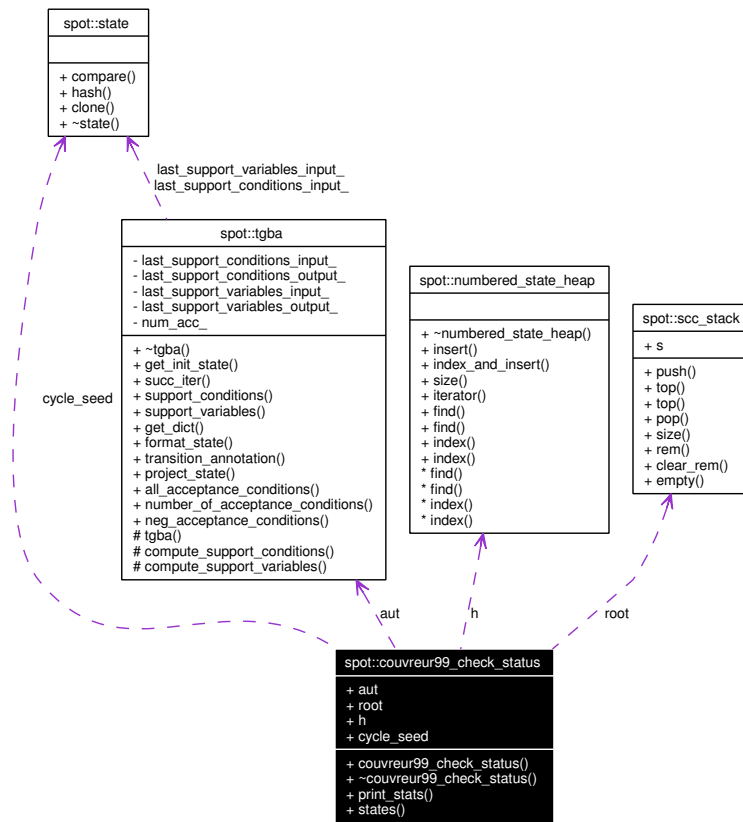
- [tgbaalgos/gtec/gtec.hh](#)

12.22 spot::couvereur99_check_status Class Reference

The status of the emptiness-check on success.

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

Collaboration diagram for spot::couvereur99_check_status:



Public Member Functions

- [couvereur99_check_status](#) (const [tgba](#) *aut, const [numbered_state_heap_factory](#) *nshf)
- [~couvereur99_check_status](#) ()
- void [print_stats](#) (std::ostream &os) const
Output statistics about this object.
- int [states](#) () const
Return the number of states visited by the search.

Public Attributes

- const [tgba](#) * [aut](#)
- [scc_stack](#) root
- [numbered_state_heap](#) * [h](#)
Heap of visited states.
- const [state](#) * [cycle_seed](#)

12.22.1 Detailed Description

The status of the emptiness-check on success.

This contains everything needed to construct a counter-example: the automata, the stack of SCCs traversed by the counter-example, and the heap of visited states with their indexes.

12.22.2 Constructor & Destructor Documentation

12.22.2.1 [spot::couvreur99_check_status::couvreur99_check_status](#) (const [tgba](#) * [aut](#), const [numbered_state_heap_factory](#) * [nshf](#))

12.22.2.2 [spot::couvreur99_check_status::~~couvreur99_check_status](#) ()

12.22.3 Member Function Documentation

12.22.3.1 [void spot::couvreur99_check_status::print_stats](#) (std::ostream & [os](#)) const

Output statistics about this object.

12.22.3.2 [int spot::couvreur99_check_status::states](#) () const

Return the number of states visited by the search.

12.22.4 Member Data Documentation

12.22.4.1 const [tgba](#)* [spot::couvreur99_check_status::aut](#)

12.22.4.2 const [state](#)* [spot::couvreur99_check_status::cycle_seed](#)

12.22.4.3 [numbered_state_heap](#)* [spot::couvreur99_check_status::h](#)

Heap of visited states.

12.22.4.4 [scc_stack](#) [spot::couvreur99_check_status::root](#)

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

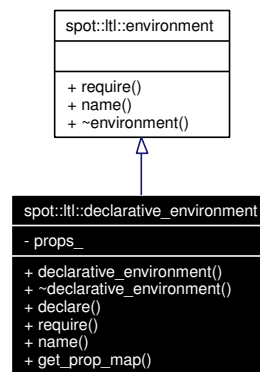
- [tgbaalgos/gtec/status.hh](#)

12.23 spot::ltl::declarative_environment Class Reference

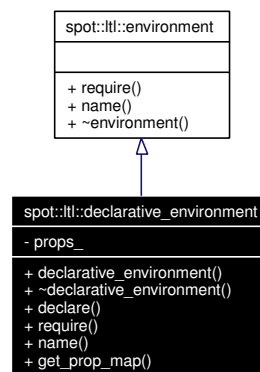
A declarative environment.

```
#include <ltlenv/declenv.hh>
```

Inheritance diagram for spot::ltl::declarative_environment:



Collaboration diagram for spot::ltl::declarative_environment:



Public Types

- typedef std::map< const std::string, [ltl::atomic_prop](#) * > [prop_map](#)

Public Member Functions

- [declarative_environment](#) ()
- [~declarative_environment](#) ()
- bool [declare](#) (const std::string &prop_str)
- virtual [ltl::formula](#) * [require](#) (const std::string &prop_str)

Obtain the formula associated to prop_str.

- virtual const std::string & [name](#) ()

Get the name of the environment.

- `const prop_map & get_prop_map () const`
Get the map of atomic proposition known to this environment.

Private Attributes

- `prop_map props_`

12.23.1 Detailed Description

A declarative environment.

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

12.23.2 Member Typedef Documentation

12.23.2.1 `typedef std::map<const std::string, ltl::atomic_prop*> spot::ltl::declarative_environment::prop_map`

12.23.3 Constructor & Destructor Documentation

12.23.3.1 `spot::ltl::declarative_environment::declarative_environment ()`

12.23.3.2 `spot::ltl::declarative_environment::~~declarative_environment ()`

12.23.4 Member Function Documentation

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

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

12.23.4.2 `const prop_map& spot::ltl::declarative_environment::get_prop_map () const`

Get the map of atomic proposition known to this environment.

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

Get the name of the environment.

Implements `spot::ltl::environment`.

12.23.4.4 `virtual ltl::formula* spot::ltl::declarative_environment::require (const std::string & prop_str) [virtual]`

Obtain the formula associated to `prop_str`.

Usually `prop_str`, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated `spot::ltl::atomic_prop`.

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a [spot::ltl::formula](#) instead of an [spot::ltl::atomic_prop](#), because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

Returns:

0 iff *prop_str* is not part of the environment, or the associated [spot::ltl::formula](#) otherwise.

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

12.23.5 Member Data Documentation

12.23.5.1 [prop_map](#) [spot::ltl::declarative_environment::props_](#) [private]

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

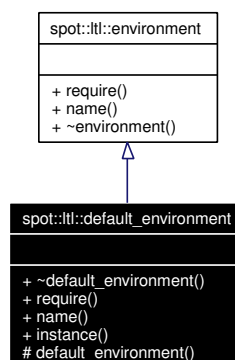
- [ltlenv/declenv.hh](#)

12.24 spot::ltl::default_environment Class Reference

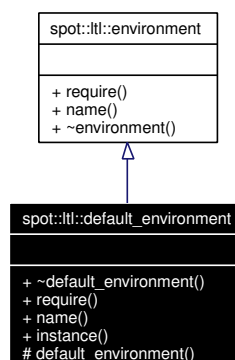
A laxist environment.

```
#include <ltlenv/defaultenv.hh>
```

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



Collaboration diagram for [spot::ltl::default_environment](#):



Public Member Functions

- virtual [~default_environment](#) ()
- virtual [formula](#) * [require](#) (const std::string &prop_str)
Obtain the formula associated to prop_str.
- virtual const std::string & [name](#) ()
Get the name of the environment.

Static Public Member Functions

- static [default_environment](#) & [instance](#) ()
Get the sole instance of spot::ltl::default_environment.

Protected Member Functions

- [default_environment](#) ()

12.24.1 Detailed Description

A laxist environment.

This environment recognizes all atomic propositions.

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

12.24.2 Constructor & Destructor Documentation

12.24.2.1 virtual spot::ltl::default_environment::~[~default_environment](#) () [virtual]

12.24.2.2 spot::ltl::default_environment::default_environment () [protected]

12.24.3 Member Function Documentation

12.24.3.1 static [default_environment](#)& spot::ltl::default_environment::instance () [static]

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

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

Get the name of the environment.

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

12.24.3.3 `virtual formula* spot::ltl::default_environment::require (const std::string & prop_str)` [virtual]

Obtain the formula associated to *prop_str*.

Usually *prop_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated `spot::ltl::atomic_prop`.

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a `spot::ltl::formula` instead of an `spot::ltl::atomic_prop`, because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

Returns:

0 iff *prop_str* is not part of the environment, or the associated `spot::ltl::formula` otherwise.

Implements `spot::ltl::environment`.

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

- [ltlenv/defaultenv.hh](#)

12.25 `spot::delayed_simulation_relation` Class Reference

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

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

- [tgba/tgbareduc.hh](#)

12.26 `spot::direct_simulation_relation` Class Reference

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

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

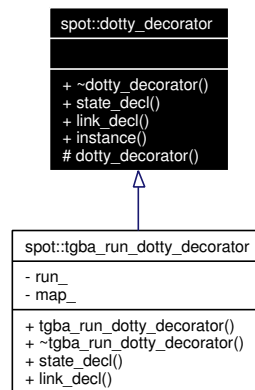
- [tgba/tgbareduc.hh](#)

12.27 `spot::dotty_decorator` Class Reference

Choose state and link styles for `spot::dotty_reachable`.

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

Inheritance diagram for `spot::dotty_decorator`:



Public Member Functions

- virtual [~dotty_decorator](#) ()
- virtual std::string [state_decl](#) (const [tgba](#) *a, const [state](#) *s, int n, [tgba_succ_iterator](#) *si, const std::string &label)
Compute the style of a state.
- virtual std::string [link_decl](#) (const [tgba](#) *a, const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si, const std::string &label)
Compute the style of a link.

Static Public Member Functions

- static [dotty_decorator](#) * [instance](#) ()
Get the unique instance of the default [dotty_decorator](#).

Protected Member Functions

- [dotty_decorator](#) ()

12.27.1 Detailed Description

Choose state and link styles for [spot::dotty_reachable](#).

12.27.2 Constructor & Destructor Documentation

12.27.2.1 virtual [spot::dotty_decorator::~~dotty_decorator](#) () [virtual]

12.27.2.2 [spot::dotty_decorator::dotty_decorator](#) () [protected]

12.27.3 Member Function Documentation

12.27.3.1 static `dotty_decorator*` `spot::dotty_decorator::instance()` [static]

Get the unique instance of the default `dotty_decorator`.

12.27.3.2 virtual `std::string` `spot::dotty_decorator::link_decl` (`const tgba * a`, `const state * in_s`, `int in`, `const state * out_s`, `int out`, `const tgba_succ_iterator * si`, `const std::string & label`) [virtual]

Compute the style of a link.

This function should output a string of the form `[label="foo", style=bar, ...]`. The default implementation will simply output `[label="LABEL"]` with `LABEL` replaced by the value of `label`.

Parameters:

- a* the automaton being drawn
- in_s* the source state of the transition being drawn (owned by the caller)
- in* the unique number associated to *in_s*
- out_s* the destination state of the transition being drawn (owned by the caller)
- out* the unique number associated to *out_s*
- si* an iterator over the successors of *in_s*, pointing to the current transition (owned by the caller and cannot be iterated)
- label* the computed name of this state

Reimplemented in `spot::tgba_run_dotty_decorator`.

12.27.3.3 virtual `std::string` `spot::dotty_decorator::state_decl` (`const tgba * a`, `const state * s`, `int n`, `tgba_succ_iterator * si`, `const std::string & label`) [virtual]

Compute the style of a state.

This function should output a string of the form `[label="foo", style=bar, ...]`. The default implementation will simply output `[label="LABEL"]` with `LABEL` replaced by the value of `label`.

Parameters:

- a* the automaton being drawn
- s* the state being drawn (owned by the caller)
- n* a unique number for this state
- si* an iterator over the successors of this state (owned by the caller, but can be freely iterated)
- label* the computed name of this state

Reimplemented in `spot::tgba_run_dotty_decorator`.

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

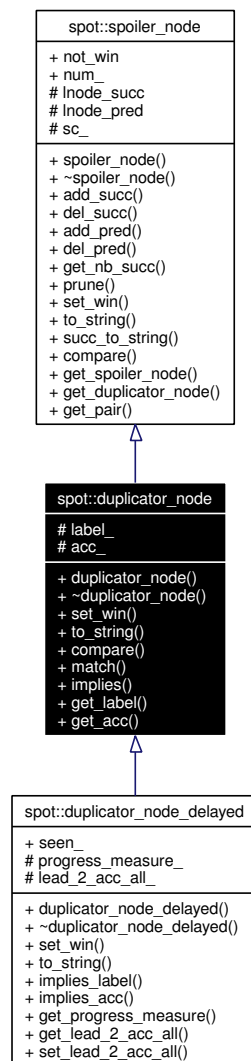
- `tgbaalgorithms/dottydec.hh`

12.28 spot::duplicator_node Class Reference

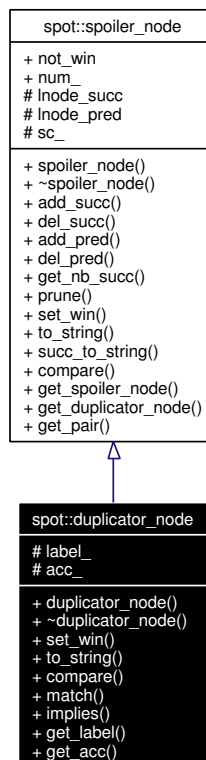
Duplicator node of parity game graph.

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

Inheritance diagram for spot::duplicator_node:



Collaboration diagram for spot::duplicator_node:



Public Member Functions

- `duplicator_node` (const `state` *d_node, const `state` *s_node, bdd l, bdd a, int num)
- virtual `~duplicator_node` ()
- virtual bool `set_win` ()
- virtual std::string `to_string` (const `tgba` *a)
- virtual bool `compare` (`spoiler_node` *n)
- bool `match` (bdd l, bdd a)
- bool `implies` (bdd l, bdd a)
- bdd `get_label` () const
- bdd `get_acc` () const
- bool `add_succ` (`spoiler_node` *n)

Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.

- void `del_succ` (`spoiler_node` *n)
- virtual void `add_pred` (`spoiler_node` *n)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const `state` * `get_spoiler_node` ()
- const `state` * `get_duplicator_node` ()
- `state_couple` * `get_pair` ()

Public Attributes

- bool [not_win](#)
- int [num_](#)

Protected Attributes

- bdd [label_](#)
- bdd [acc_](#)
- [sn_v](#) * [lnode_succ](#)
- [sn_v](#) * [lnode_pred](#)
- [state_couple](#) * [sc_](#)

12.28.1 Detailed Description

Duplicator node of parity game graph.

12.28.2 Constructor & Destructor Documentation

12.28.2.1 spot::duplicator_node::duplicator_node (const [state](#) * [d_node](#), const [state](#) * [s_node](#), bdd [l](#), bdd [a](#), int [num](#))

12.28.2.2 virtual spot::duplicator_node::~~duplicator_node () [virtual]

12.28.3 Member Function Documentation

12.28.3.1 virtual void spot::spoiler_node::add_pred ([spoiler_node](#) * [n](#)) [virtual, inherited]

12.28.3.2 bool spot::spoiler_node::add_succ ([spoiler_node](#) * [n](#)) [inherited]

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

12.28.3.3 virtual bool spot::duplicator_node::compare ([spoiler_node](#) * [n](#)) [virtual]

Reimplemented from [spot::spoiler_node](#).

12.28.3.4 virtual void spot::spoiler_node::del_pred () [virtual, inherited]

12.28.3.5 void spot::spoiler_node::del_succ ([spoiler_node](#) * [n](#)) [inherited]

12.28.3.6 bdd spot::duplicator_node::get_acc () const

12.28.3.7 const [state](#)* spot::spoiler_node::get_duplicator_node () [inherited]

12.28.3.8 bdd spot::duplicator_node::get_label () const

12.28.3.9 `int spot::spoiler_node::get_nb_succ ()` [inherited]

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

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

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

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

12.28.3.14 `bool spot::spoiler_node::prune ()` [inherited]

12.28.3.15 `virtual bool spot::duplicator_node::set_win ()` [virtual]

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

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

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

Reimplemented from `spot::spoiler_node`.

Reimplemented in `spot::duplicator_node_delayed`.

12.28.4 Member Data Documentation

12.28.4.1 `bdd spot::duplicator_node::acc_` [protected]

12.28.4.2 `bdd spot::duplicator_node::label_` [protected]

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

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

12.28.4.5 `bool spot::spoiler_node::not_win` [inherited]

12.28.4.6 `int spot::spoiler_node::num_` [inherited]

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

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

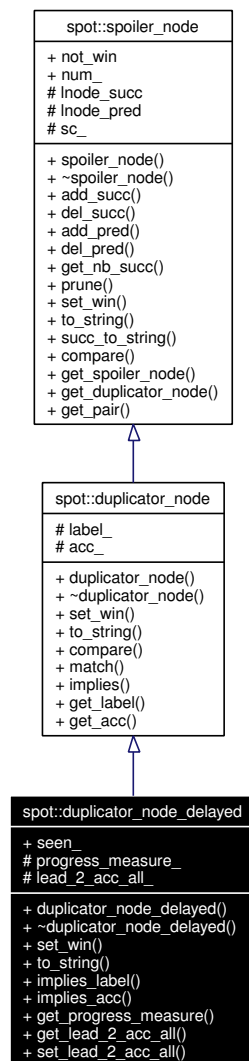
- `tgbaalgos/reductgba_sim.hh`

12.29 spot::duplicator_node_delayed Class Reference

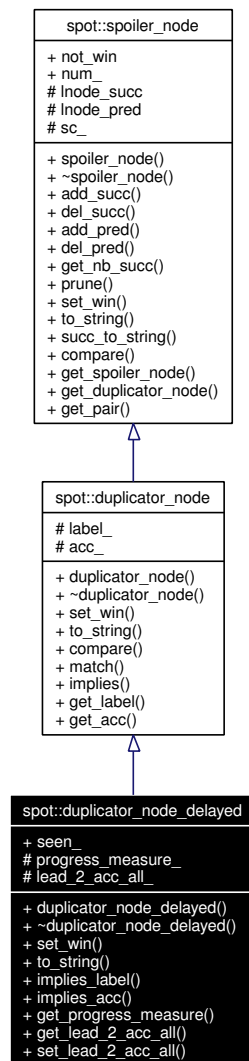
Duplicator node of parity game graph for delayed simulation.

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

Inheritance diagram for spot::duplicator_node_delayed:



Collaboration diagram for spot::duplicator_node_delayed:



Public Member Functions

- `duplicator_node_delayed` (const `state` *d_node, const `state` *s_node, bdd l, bdd a, int num)
- `~duplicator_node_delayed` ()
- bool `set_win` ()

Return true if the progress_measure has changed.

- virtual std::string `to_string` (const `tgba` *a)
- bool `implies_label` (bdd l)
- bool `implies_acc` (bdd a)
- int `get_progress_measure` ()
- bool `get_lead_2_acc_all` ()
- bool `set_lead_2_acc_all` (bdd acc=bddfalse)
- virtual bool `compare` (`spoiler_node` *n)
- bool `match` (bdd l, bdd a)
- bool `implies` (bdd l, bdd a)
- bdd `get_label` () const

- bdd [get_acc](#) () const
- bool [add_succ](#) (spoiler_node *n)
Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.
- void [del_succ](#) (spoiler_node *n)
- virtual void [add_pred](#) (spoiler_node *n)
- virtual void [del_pred](#) ()
- int [get_nb_succ](#) ()
- bool [prune](#) ()
- virtual std::string [succ_to_string](#) ()
- const state * [get_spoiler_node](#) ()
- const state * [get_duplicator_node](#) ()
- state_couple * [get_pair](#) ()

Public Attributes

- bool [seen_](#)
- bool [not_win](#)
- int [num_](#)

Protected Attributes

- int [progress_measure_](#)
- bool [lead_2_acc_all_](#)
- bdd [label_](#)
- bdd [acc_](#)
- sn_v * [lnode_succ](#)
- sn_v * [lnode_pred](#)
- state_couple * [sc_](#)

12.29.1 Detailed Description

Duplicator node of parity game graph for delayed simulation.

12.29.2 Constructor & Destructor Documentation

12.29.2.1 spot::duplicator_node_delayed::duplicator_node_delayed (const state * [d_node](#), const state * [s_node](#), bdd [l](#), bdd [a](#), int [num](#))

12.29.2.2 spot::duplicator_node_delayed::~~duplicator_node_delayed ()

12.29.3 Member Function Documentation

12.29.3.1 virtual void spot::spoiler_node::add_pred (spoiler_node * [n](#)) [virtual, inherited]

12.29.3.2 bool spot::spoiler_node::add_succ (spoiler_node * [n](#)) [inherited]

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

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

Reimplemented from `spot::spoiler_node`.

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

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

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

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

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

12.29.3.9 `bool spot::duplicator_node_delayed::get_lead_2_acc_all ()`

12.29.3.10 `int spot::spoiler_node::get_nb_succ () [inherited]`

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

12.29.3.12 `int spot::duplicator_node_delayed::get_progress_measure ()`

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

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

12.29.3.15 `bool spot::duplicator_node_delayed::implies_acc (bdd a)`

12.29.3.16 `bool spot::duplicator_node_delayed::implies_label (bdd l)`

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

12.29.3.18 `bool spot::spoiler_node::prune () [inherited]`

12.29.3.19 `bool spot::duplicator_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

12.29.3.20 `bool spot::duplicator_node_delayed::set_win () [virtual]`

Return true if the progress_measure has changed.

Reimplemented from `spot::duplicator_node`.

12.29.3.21 virtual std::string spot::spoiler_node::succ_to_string () [virtual, inherited]

12.29.3.22 virtual std::string spot::duplicator_node_delayed::to_string (const tgba * a) [virtual]

Reimplemented from spot::duplicator_node.

12.29.4 Member Data Documentation

12.29.4.1 bdd spot::duplicator_node::acc_ [protected, inherited]

12.29.4.2 bdd spot::duplicator_node::label_ [protected, inherited]

12.29.4.3 bool spot::duplicator_node_delayed::lead_2_acc_all_ [protected]

12.29.4.4 sn_v* spot::spoiler_node::lnode_pred [protected, inherited]

12.29.4.5 sn_v* spot::spoiler_node::lnode_succ [protected, inherited]

12.29.4.6 bool spot::spoiler_node::not_win [inherited]

12.29.4.7 int spot::spoiler_node::num_ [inherited]

12.29.4.8 int spot::duplicator_node_delayed::progress_measure_ [protected]

12.29.4.9 state_couple* spot::spoiler_node::sc_ [protected, inherited]

12.29.4.10 bool spot::duplicator_node_delayed::seen_

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

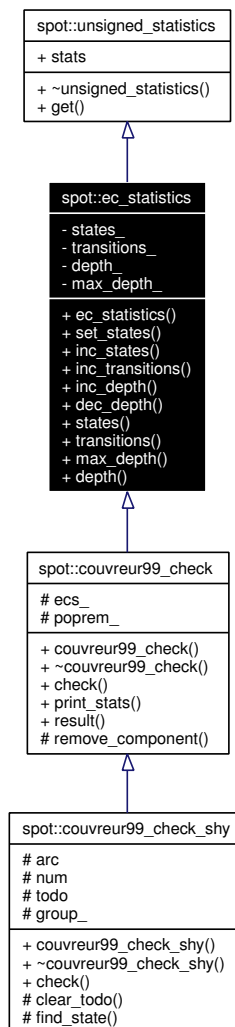
- tgbaalgos/reductgba_sim.hh

12.30 spot::ec_statistics Class Reference

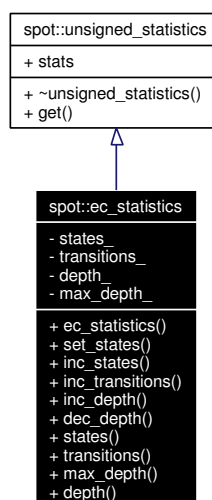
Emptiness-check statistics.

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

Inheritance diagram for spot::ec_statistics:



Collaboration diagram for `spot::ec_statistics`:



Public Types

- typedef unsigned(unsigned_statistics::* [unsigned_fun](#))() const
- typedef std::map< const char *, [unsigned_fun](#), [char_ptr_less_than](#) > [stats_map](#)

Public Member Functions

- [ec_statistics](#) ()
- void [set_states](#) (unsigned n)
- void [inc_states](#) ()
- void [inc_transitions](#) ()
- void [inc_depth](#) (unsigned n=1)
- void [dec_depth](#) (unsigned n=1)
- unsigned [states](#) () const
- unsigned [transitions](#) () const
- unsigned [max_depth](#) () const
- unsigned [depth](#) () const
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

Private Attributes

- unsigned [states_](#)
- unsigned [transitions_](#)
number of disctint visited states
- unsigned [depth_](#)
number of visited transitions
- unsigned [max_depth_](#)
maximal depth of the stack(s)

12.30.1 Detailed Description

Emptiness-check statistics.

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

12.30.2 Member Typedef Documentation

12.30.2.1 typedef std::map<const char*, [unsigned_fun](#), [char_ptr_less_than](#)> [spot::unsigned_statistics::stats_map](#) [inherited]

12.30.2.2 typedef unsigned(unsigned_statistics::* [spot::unsigned_statistics::unsigned_fun](#))() const [inherited]

12.30.3 Constructor & Destructor Documentation

12.30.3.1 `spot::ec_statistics::ec_statistics ()` [\[inline\]](#)

12.30.4 Member Function Documentation

12.30.4.1 `void spot::ec_statistics::dec_depth (unsigned n = 1)` [\[inline\]](#)

12.30.4.2 `unsigned spot::ec_statistics::depth () const` [\[inline\]](#)

12.30.4.3 `unsigned spot::unsigned_statistics::get (const char * str) const` [\[inline, inherited\]](#)

12.30.4.4 `void spot::ec_statistics::inc_depth (unsigned n = 1)` [\[inline\]](#)

12.30.4.5 `void spot::ec_statistics::inc_states ()` [\[inline\]](#)

12.30.4.6 `void spot::ec_statistics::inc_transitions ()` [\[inline\]](#)

12.30.4.7 `unsigned spot::ec_statistics::max_depth () const` [\[inline\]](#)

12.30.4.8 `void spot::ec_statistics::set_states (unsigned n)` [\[inline\]](#)

12.30.4.9 `unsigned spot::ec_statistics::states () const` [\[inline\]](#)

12.30.4.10 `unsigned spot::ec_statistics::transitions () const` [\[inline\]](#)

12.30.5 Member Data Documentation

12.30.5.1 `unsigned spot::ec_statistics::depth_` [\[private\]](#)
number of visited transitions

12.30.5.2 `unsigned spot::ec_statistics::max_depth_` [\[private\]](#)
maximal depth of the stack(s)

12.30.5.3 `unsigned spot::ec_statistics::states_` [\[private\]](#)

12.30.5.4 `stats_map spot::unsigned_statistics::stats` [\[inherited\]](#)

12.30.5.5 unsigned [spot::ec_statistics::transitions_](#) [private]

number of disctint visited states

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

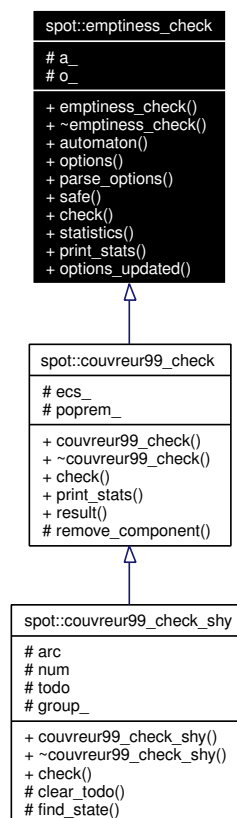
- [tgbaalgos/emptiness_stats.hh](#)

12.31 spot::emptiness_check Class Reference

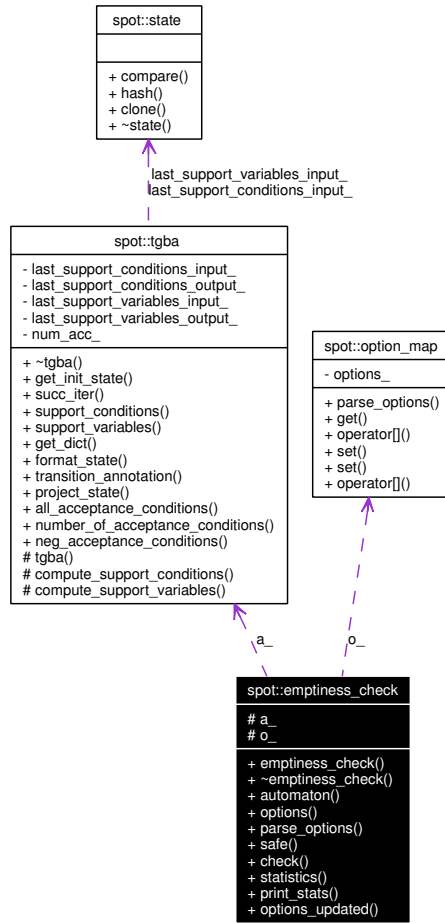
Common interface to emptiness check algorithms.

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

Inheritance diagram for spot::emptiness_check:



Collaboration diagram for spot::emptiness_check:



Public Member Functions

- `emptiness_check` (const `tgba` *a, `option_map` o=`option_map`())
- virtual `~emptiness_check` ()
- const `tgba` * `automaton` () const
The automaton that this emptiness-check inspects.
- const `option_map` & `options` () const
Return the options parametrizing how the emptiness check is realized.
- const char * `parse_options` (char *options)
Modify the algorithm options.
- virtual bool `safe` () const
Return false iff accepting_run() can return 0 for non-empty automata.
- virtual `emptiness_check_result` * `check` ()=0
Check whether the automaton contain an accepting run.
- virtual const `unsigned_statistics` * `statistics` () const
Return statistics, if available.

- virtual `std::ostream & print_stats (std::ostream &os) const`
Print statistics, if any.
- virtual void `options_updated (const option_map &old)`
Notify option updates.

Protected Attributes

- const `tgba * a_`
The automaton.
- `option_map o_`
The options.

12.31.1 Detailed Description

Common interface to emptiness check algorithms.

12.31.2 Constructor & Destructor Documentation

12.31.2.1 `spot::emptiness_check::emptiness_check (const tgba * a, option_map o = option_map\(\))` [`inline`]

12.31.2.2 virtual `spot::emptiness_check::~emptiness_check ()` [`virtual`]

12.31.3 Member Function Documentation

12.31.3.1 const `tgba* spot::emptiness_check::automaton () const` [`inline`]

The automaton that this emptiness-check inspects.

12.31.3.2 virtual `emptiness_check_result* spot::emptiness_check::check ()` [`pure virtual`]

Check whether the automaton contain an accepting run.

Return 0 if the automaton accepts no run. Return an instance of `emptiness_check_result` otherwise. This instance might allow to obtain one sample acceptance run. The result has to be destroyed before the `emptiness_check` instance that generated it.

Some `emptiness_check` algorithms may allow `check\(\)` to be called several time, but generally you should not assume that.

Some `emptiness_check` algorithms, especially those using bit state hashing may return 0 even if the automaton is not empty.

See also:

`safe\(\)`

Implemented in `spot::couvreur99_check`, and `spot::couvreur99_check_shy`.

12.31.3.3 `const option_map& spot::emptiness_check::options () const` `[inline]`

Return the options parametrizing how the emptiness check is realized.

12.31.3.4 `virtual void spot::emptiness_check::options_updated (const option_map & old)`
`[virtual]`

Notify option updates.

12.31.3.5 `const char* spot::emptiness_check::parse_options (char * options)`

Modify the algorithm options.

12.31.3.6 `virtual std::ostream& spot::emptiness_check::print_stats (std::ostream & os) const`
`[virtual]`

Print statistics, if any.

Reimplemented in [spot::couvreur99_check](#).

12.31.3.7 `virtual bool spot::emptiness_check::safe () const` `[virtual]`

Return false iff accepting_run() can return 0 for non-empty automata.

12.31.3.8 `virtual const unsigned_statistics* spot::emptiness_check::statistics () const` `[virtual]`

Return statistics, if available.

12.31.4 Member Data Documentation

12.31.4.1 `const tgba* spot::emptiness_check::a_` `[protected]`

The automaton.

12.31.4.2 `option_map spot::emptiness_check::o_` `[protected]`

The options.

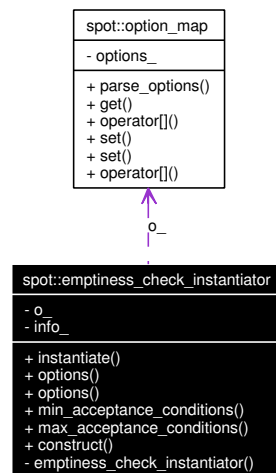
The documentation for this class was generated from the following file:

- [tgbaalgos/emptiness.hh](#)

12.32 spot::emptiness_check_instantiator Class Reference

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

Collaboration diagram for spot::emptiness_check_instantiator:



Public Member Functions

- `emptiness_check * instantiate (const tgba *a) const`
Actually instantiate the emptiness check, for a.
- `const option_map & options () const`
- `option_map & options ()`
- `unsigned int min_acceptance_conditions () const`
Minimum number of acceptance conditions supported by the emptiness check.
- `unsigned int max_acceptance_conditions () const`
Maximum number of acceptance conditions supported by the emptiness check.

Static Public Member Functions

- `static emptiness_check_instantiator * construct (const char *name, const char **err)`
Create an emptiness-check instantiator, given the name of an emptiness check.

Private Member Functions

- `emptiness_check_instantiator (option_map o, void *i)`

Private Attributes

- `option_map o_`
- `void * info_`

12.32.1 Constructor & Destructor Documentation

12.32.1.1 `spot::emptiness_check_instantiator::emptiness_check_instantiator (option_map o, void * i) [private]`

12.32.2 Member Function Documentation

12.32.2.1 static [emptiness_check_instantiator](#)* spot::emptiness_check_instantiator::construct (const char * *name*, const char ** *err*) [static]

Create an emptiness-check instantiator, given the name of an emptiness check.

name should have the form "name" or "name(options)".

On error, the function returns 0. If the name of the algorithm was unknown, **err* will be set to *name*. If some fragment of the options could not be parsed, **err* will point to that fragment.

12.32.2.2 [emptiness_check](#)* spot::emptiness_check_instantiator::instantiate (const [tgba](#) * *a*) const

Actually instantiate the emptiness check, for *a*.

12.32.2.3 unsigned int spot::emptiness_check_instantiator::max_acceptance_conditions () const

Maximum number of acceptance conditions supported by the emptiness check.

Returns:

−1U if no upper bound exists.

12.32.2.4 unsigned int spot::emptiness_check_instantiator::min_acceptance_conditions () const

Minimum number of acceptance conditions supported by the emptiness check.

12.32.2.5 [option_map](#)& spot::emptiness_check_instantiator::options () [inline]

12.32.2.6 const [option_map](#)& spot::emptiness_check_instantiator::options () const [inline]

Accessor to the options.

12.32.3 Member Data Documentation

12.32.3.1 void* [spot::emptiness_check_instantiator::info_](#) [private]

12.32.3.2 [option_map](#) [spot::emptiness_check_instantiator::o_](#) [private]

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

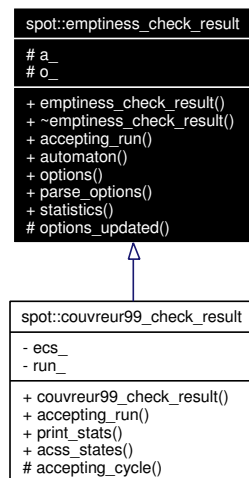
- [tgbaalgos/emptiness.hh](#)

12.33 spot::emptiness_check_result Class Reference

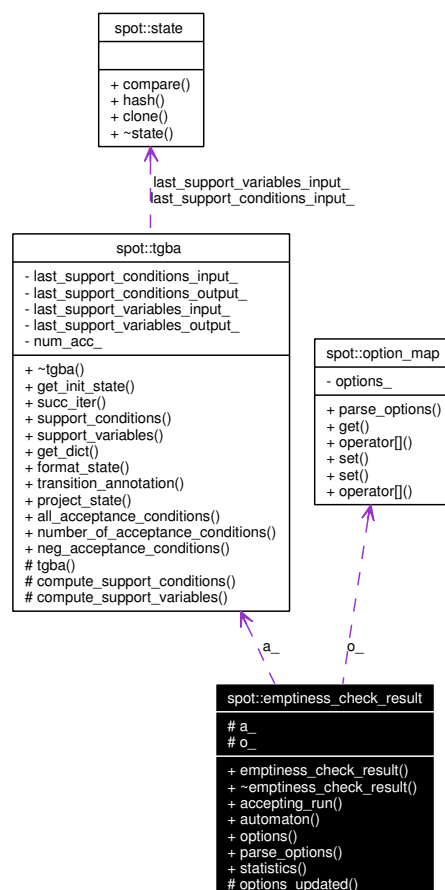
The result of an emptiness check.

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

Inheritance diagram for spot::emptiness_check_result:



Collaboration diagram for `spot::emptiness_check_result`:



Public Member Functions

- `emptiness_check_result` (const `tgba` *a, `option_map` o=`option_map`())
- virtual `~emptiness_check_result` ()

- virtual [tgba_run](#) * [accepting_run](#) ()
Return a run accepted by the automata passed to the emptiness check.
- const [tgba](#) * [automaton](#) () const
The automaton on which an [accepting_run\(\)](#) was found.
- const [option_map](#) & [options](#) () const
Return the options parametrizing how the accepting run is computed.
- const char * [parse_options](#) (char *options)
Modify the algorithm options.
- virtual const [unsigned_statistics](#) * [statistics](#) () const
Return statistics, if available.

Protected Member Functions

- virtual void [options_updated](#) (const [option_map](#) &old)
Notify option updates.

Protected Attributes

- const [tgba](#) * [a_](#)
The automaton.
- [option_map](#) [o_](#)
The options.

12.33.1 Detailed Description

The result of an emptiness check.

Instances of these class should not last longer than the instances of [emptiness_check](#) that produced them as they may reference data internal to the check.

12.33.2 Constructor & Destructor Documentation

12.33.2.1 `spot::emptiness_check_result::emptiness_check_result (const tgba * a, option_map o = option_map()) [inline]`

12.33.2.2 `virtual spot::emptiness_check_result::~emptiness_check_result () [inline, virtual]`

12.33.3 Member Function Documentation

12.33.3.1 virtual [tgba_run*](#) spot::emptiness_check_result::accepting_run () [virtual]

Return a run accepted by the automata passed to the emptiness check.

This method might actually compute the acceptance run. (Not all emptiness check algorithms actually produce a counter-example as a side-effect of checking emptiness, some need some post-processing.)

This can also return 0 if the emptiness check algorithm cannot produce a counter example (that does not mean there is no counter-example; the mere existence of an instance of this class asserts the existence of a counter-example).

Reimplemented in [spot::couvreur99_check_result](#).

12.33.3.2 const [tgba*](#) spot::emptiness_check_result::automaton () const [inline]

The automaton on which an [accepting_run\(\)](#) was found.

12.33.3.3 const [option_map&](#) spot::emptiness_check_result::options () const [inline]

Return the options parametrizing how the accepting run is computed.

12.33.3.4 virtual void spot::emptiness_check_result::options_updated (const [option_map](#) & old) [protected, virtual]

Notify option updates.

12.33.3.5 const char* spot::emptiness_check_result::parse_options (char * options)

Modify the algorithm options.

12.33.3.6 virtual const [unsigned_statistics*](#) spot::emptiness_check_result::statistics () const [virtual]

Return statistics, if available.

12.33.4 Member Data Documentation

12.33.4.1 const [tgba*](#) spot::emptiness_check_result::a_ [protected]

The automaton.

12.33.4.2 [option_map](#) spot::emptiness_check_result::o_ [protected]

The options.

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

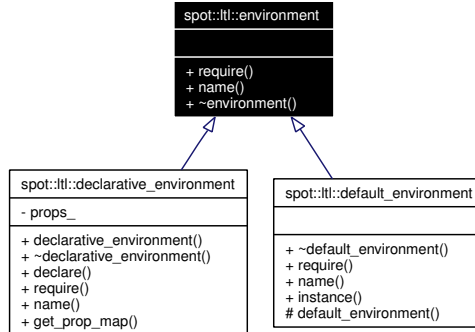
- [tgbaalgorithms/emptiness.hh](#)

12.34 spot::ltl::environment Class Reference

An environment that describes atomic propositions.

```
#include <ltlenv/environment.hh>
```

Inheritance diagram for spot::ltl::environment:



Public Member Functions

- virtual [formula](#) * [require](#) (const std::string &prop_str)=0
Obtain the formula associated to prop_str.
- virtual const std::string & [name](#) ()=0
Get the name of the environment.
- virtual [~environment](#) ()

12.34.1 Detailed Description

An environment that describes atomic propositions.

12.34.2 Constructor & Destructor Documentation

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

12.34.3 Member Function Documentation

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

Get the name of the environment.

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

12.34.3.2 virtual [formula](#)* spot::ltl::environment::require (const std::string &prop_str) [pure virtual]

Obtain the formula associated to prop_str.

Usually prop_str, is the name of an atomic proposition, and spot::ltl::require simply returns the associated [spot::ltl::atomic_prop](#).

Note this is not a const method. Some environments will "create" the atomic proposition when requested.

We return a [spot::ltl::formula](#) instead of an [spot::ltl::atomic_prop](#), because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

Returns:

0 iff *prop_str* is not part of the environment, or the associated [spot::ltl::formula](#) otherwise.

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

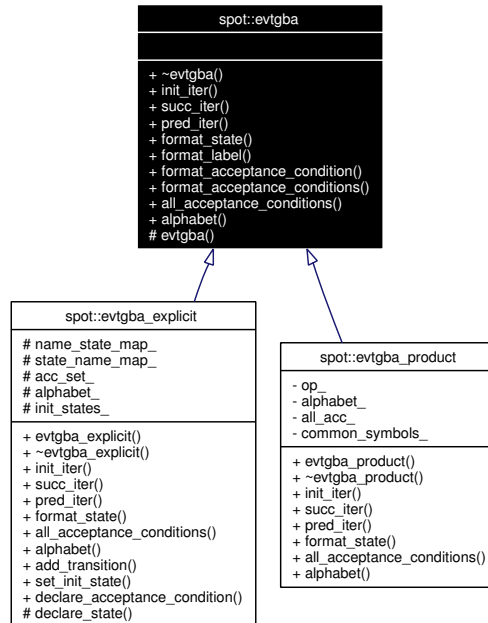
The documentation for this class was generated from the following file:

- [ltlenv/environment.hh](#)

12.35 spot::evtgba Class Reference

```
#include <evtgba/evtgba.hh>
```

Inheritance diagram for `spot::evtgba`:



Public Member Functions

- virtual `~evtgba ()`
- virtual `evtgba_iterator * init_iter () const =0`
- virtual `evtgba_iterator * succ_iter (const state *s) const =0`
- virtual `evtgba_iterator * pred_iter (const state *s) const =0`
- virtual `std::string format_state (const state *state) const =0`

Format the state as a string for printing.

- virtual `std::string format_label (const symbol *symbol) const`
- virtual `std::string format_acceptance_condition (const symbol *symbol) const`
- virtual `std::string format_acceptance_conditions (const symbol_set &symset) const`

- virtual const [symbol_set](#) & [all_acceptance_conditions](#) () const =0
Return the set of all acceptance conditions used by this automaton.
- virtual const [symbol_set](#) & [alphabet](#) () const =0

Protected Member Functions

- [evtgba](#) ()

12.35.1 Constructor & Destructor Documentation

12.35.1.1 [spot::evtgba::evtgba](#) () [protected]

12.35.1.2 virtual [spot::evtgba::~~evtgba](#) () [virtual]

12.35.2 Member Function Documentation

12.35.2.1 virtual const [symbol_set](#)& [spot::evtgba::all_acceptance_conditions](#) () const [pure virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implemented in [spot::evtgba_explicit](#), and [spot::evtgba_product](#).

12.35.2.2 virtual const [symbol_set](#)& [spot::evtgba::alphabet](#) () const [pure virtual]

Implemented in [spot::evtgba_explicit](#), and [spot::evtgba_product](#).

12.35.2.3 virtual std::string [spot::evtgba::format_acceptance_condition](#) (const [symbol](#) * *symbol*) const [virtual]

12.35.2.4 virtual std::string [spot::evtgba::format_acceptance_conditions](#) (const [symbol_set](#) & *symbol_set*) const [virtual]

12.35.2.5 virtual std::string [spot::evtgba::format_label](#) (const [symbol](#) * *symbol*) const [virtual]

12.35.2.6 virtual std::string [spot::evtgba::format_state](#) (const [state](#) * *state*) const [pure virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::evtgba_product](#).

12.35.2.7 virtual [evtgba_iterator](#)* [spot::evtgba::init_iter\(\)](#) const [pure virtual]

Implemented in [spot::evtgba_explicit](#), and [spot::evtgba_product](#).

12.35.2.8 virtual [evtgba_iterator](#)* [spot::evtgba::pred_iter](#) (const [state](#) * s) const [pure virtual]

Implemented in [spot::evtgba_product](#).

12.35.2.9 virtual [evtgba_iterator](#)* [spot::evtgba::succ_iter](#) (const [state](#) * s) const [pure virtual]

Implemented in [spot::evtgba_product](#).

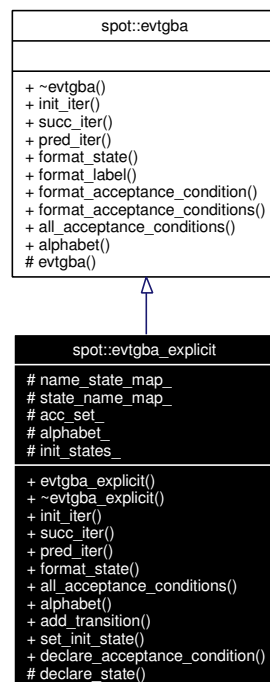
The documentation for this class was generated from the following file:

- [evtgba/evtgba.hh](#)

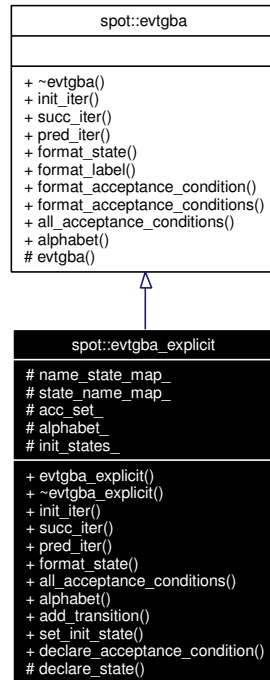
12.36 spot::evtgba_explicit Class Reference

```
#include <evtgba/explicit.hh>
```

Inheritance diagram for [spot::evtgba_explicit](#):



Collaboration diagram for [spot::evtgba_explicit](#):



Public Types

- typedef std::list< [transition](#) * > [transition_list](#)

Public Member Functions

- [evtgba_explicit](#) ()
- virtual [~evtgba_explicit](#) ()
- virtual [evtgba_iterator](#) * [init_iter](#) () const
- virtual [evtgba_iterator](#) * [succ_iter](#) (const [spot::state](#) *s) const
- virtual [evtgba_iterator](#) * [pred_iter](#) (const [spot::state](#) *s) const
- virtual std::string [format_state](#) (const [spot::state](#) *state) const
- virtual const [symbol_set](#) & [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual const [symbol_set](#) & [alphabet](#) () const
- [transition](#) * [add_transition](#) (const std::string &source, const [rsymbol](#) &label, [rsymbol_set](#) acc, const std::string &dest)
- void [set_init_state](#) (const std::string &name)
Designate name as initial state.
- void [declare_acceptance_condition](#) (const [rsymbol](#) &acc)
- virtual [evtgba_iterator](#) * [succ_iter](#) (const [state](#) *s) const =0
- virtual [evtgba_iterator](#) * [pred_iter](#) (const [state](#) *s) const =0
- virtual std::string [format_state](#) (const [state](#) *state) const =0
Format the state as a string for printing.
- virtual std::string [format_label](#) (const [symbol](#) *symbol) const

- virtual std::string [format_acceptance_condition](#) (const [symbol](#) *[symbol](#)) const
- virtual std::string [format_acceptance_conditions](#) (const [symbol_set](#) &symset) const

Protected Types

- typedef Sgi::hash_map< const std::string, [evtgba_explicit::state](#) *, [string_hash](#) > [ns_map](#)
- typedef Sgi::hash_map< const [evtgba_explicit::state](#) *, std::string, [ptr_hash](#)< [evtgba_explicit::state](#) > > [sn_map](#)

Protected Member Functions

- [state](#) * [declare_state](#) (const std::string &name)

Protected Attributes

- [ns_map](#) [name_state_map_](#)
- [sn_map](#) [state_name_map_](#)
- [symbol_set](#) [acc_set_](#)
- [symbol_set](#) [alphabet_](#)
- [transition_list](#) [init_states_](#)

Classes

- struct [state](#)
- struct [transition](#)

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

12.36.1 Member Typedef Documentation

12.36.1.1 typedef Sgi::hash_map<const std::string, [evtgba_explicit::state](#)*, [string_hash](#)> [spot::evtgba_explicit::ns_map](#) [protected]

12.36.1.2 typedef Sgi::hash_map<const [evtgba_explicit::state](#)*, std::string, [ptr_hash](#)<[evtgba_explicit::state](#)> > [spot::evtgba_explicit::sn_map](#) [protected]

12.36.1.3 typedef std::list<[transition](#)*> [spot::evtgba_explicit::transition_list](#)

12.36.2 Constructor & Destructor Documentation

12.36.2.1 [spot::evtgba_explicit::evtgba_explicit](#) ()

12.36.2.2 virtual [spot::evtgba_explicit::~~evtgba_explicit](#) () [virtual]

12.36.3 Member Function Documentation

12.36.3.1 [transition*](#) `spot::evtgba_explicit::add_transition (const std::string & source, const rsymbol & label, rsymbol_set acc, const std::string & dest)`

12.36.3.2 `virtual const symbol_set& spot::evtgba_explicit::all_acceptance_conditions () const` `[virtual]`

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::evtgba](#).

12.36.3.3 `virtual const symbol_set& spot::evtgba_explicit::alphabet () const` `[virtual]`

Implements [spot::evtgba](#).

12.36.3.4 `void spot::evtgba_explicit::declare_acceptance_condition (const rsymbol & acc)`

12.36.3.5 `state* spot::evtgba_explicit::declare_state (const std::string & name)` `[protected]`

12.36.3.6 `virtual std::string spot::evtgba::format_acceptance_condition (const symbol * symbol) const` `[virtual, inherited]`

12.36.3.7 `virtual std::string spot::evtgba::format_acceptance_conditions (const symbol_set & sym-set) const` `[virtual, inherited]`

12.36.3.8 `virtual std::string spot::evtgba::format_label (const symbol * symbol) const` `[virtual, inherited]`

12.36.3.9 `virtual std::string spot::evtgba::format_state (const state * state) const` `[pure virtual, inherited]`

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in [spot::evtgba_product](#).

12.36.3.10 `virtual std::string spot::evtgba_explicit::format_state (const spot::state * state) const` `[virtual]`

12.36.3.11 `virtual evtgba_iterator* spot::evtgba_explicit::init_iter () const` `[virtual]`

Implements [spot::evtgba](#).

12.36.3.12 virtual [evtgba_iterator](#)* spot::evtgba::pred_iter (const [state](#) * s) const [pure virtual, inherited]

Implemented in [spot::evtgba_product](#).

12.36.3.13 virtual [evtgba_iterator](#)* spot::evtgba_explicit::pred_iter (const [spot::state](#) * s) const [virtual]

12.36.3.14 void spot::evtgba_explicit::set_init_state (const std::string & name)

Designate *name* as initial state.

Can be called multiple times in case there is several initial states.

12.36.3.15 virtual [evtgba_iterator](#)* spot::evtgba::succ_iter (const [state](#) * s) const [pure virtual, inherited]

Implemented in [spot::evtgba_product](#).

12.36.3.16 virtual [evtgba_iterator](#)* spot::evtgba_explicit::succ_iter (const [spot::state](#) * s) const [virtual]

12.36.4 Member Data Documentation

12.36.4.1 [symbol_set](#) spot::evtgba_explicit::acc_set_ [protected]

12.36.4.2 [symbol_set](#) spot::evtgba_explicit::alphabet_ [protected]

12.36.4.3 [transition_list](#) spot::evtgba_explicit::init_states_ [protected]

12.36.4.4 [ns_map](#) spot::evtgba_explicit::name_state_map_ [protected]

12.36.4.5 [sn_map](#) spot::evtgba_explicit::state_name_map_ [protected]

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

- [evtgba/explicit.hh](#)

12.37 spot::evtgba_explicit::state Struct Reference

```
#include <evtgba/explicit.hh>
```

Public Attributes

- [transition_list](#) in
- [transition_list](#) out

12.37.1 Member Data Documentation

12.37.1.1 [transition_list](#) spot::evtgba_explicit::state::in

12.37.1.2 [transition_list](#) spot::evtgba_explicit::state::out

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

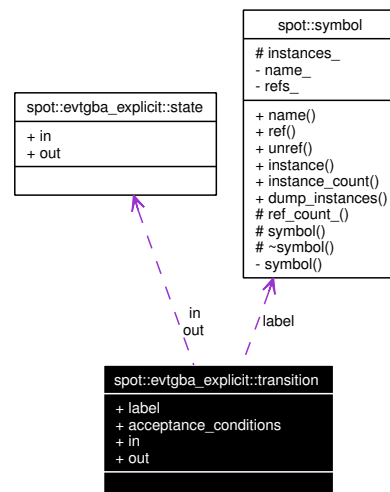
- [evtgba/explicit.hh](#)

12.38 spot::evtgba_explicit::transition Struct Reference

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

```
#include <evtgba/explicit.hh>
```

Collaboration diagram for spot::evtgba_explicit::transition:



Public Attributes

- const [symbol](#) * [label](#)
- [symbol_set](#) [acceptance_conditions](#)
- [state](#) * [in](#)
- [state](#) * [out](#)

12.38.1 Detailed Description

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

12.38.2 Member Data Documentation

12.38.2.1 [symbol_set](#) spot::evtgba_explicit::transition::acceptance_conditions

12.38.2.2 [state](#)* spot::evtgba_explicit::transition::in

12.38.2.3 `const symbol* spot::evtgba_explicit::transition::label`

12.38.2.4 `state* spot::evtgba_explicit::transition::out`

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

- [evtgba/explicit.hh](#)

12.39 spot::evtgba_iterator Class Reference

```
#include <evtgba/evtgbaiter.hh>
```

Public Member Functions

- virtual `~evtgba_iterator ()`
- virtual void `first ()=0`
- virtual void `next ()=0`
- virtual bool `done () const =0`
- virtual const `state * current_state () const =0`
- virtual const `symbol * current_label () const =0`
- virtual `symbol_set current_acceptance_conditions () const =0`

12.39.1 Constructor & Destructor Documentation

12.39.1.1 `virtual spot::evtgba_iterator::~~evtgba_iterator () [inline, virtual]`

12.39.2 Member Function Documentation

12.39.2.1 `virtual symbol_set spot::evtgba_iterator::current_acceptance_conditions () const [pure virtual]`

12.39.2.2 `virtual const symbol* spot::evtgba_iterator::current_label () const [pure virtual]`

12.39.2.3 `virtual const state* spot::evtgba_iterator::current_state () const [pure virtual]`

12.39.2.4 `virtual bool spot::evtgba_iterator::done () const [pure virtual]`

12.39.2.5 `virtual void spot::evtgba_iterator::first () [pure virtual]`

12.39.2.6 `virtual void spot::evtgba_iterator::next () [pure virtual]`

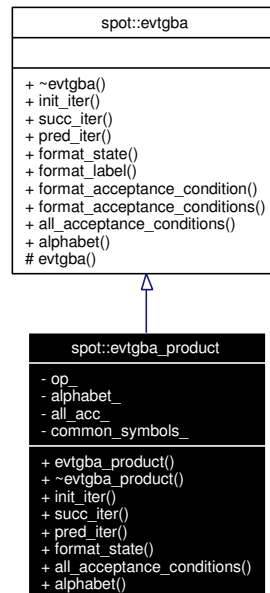
The documentation for this class was generated from the following file:

- [evtgba/evtgbaiter.hh](#)

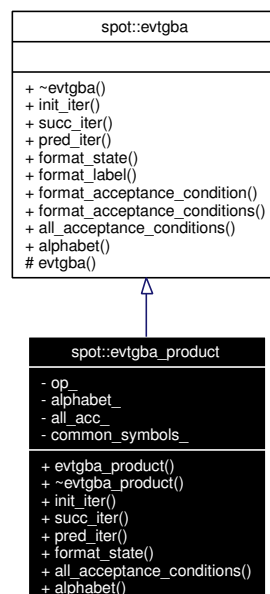
12.40 spot::evtgba_product Class Reference

```
#include <evtgba/product.hh>
```

Inheritance diagram for spot::evtgba_product:



Collaboration diagram for spot::evtgba_product:



Public Types

- typedef std::vector< const [evtgba](#) * > [evtgba_product_operands](#)
- typedef std::map< const [symbol](#) *, std::set< int > > [common_symbol_table](#)

Public Member Functions

- [evtgba_product](#) (const [evtgba_product_operands](#) &op)
- virtual [~evtgba_product](#) ()
- virtual [evtgba_iterator](#) * [init_iter](#) () const
- virtual [evtgba_iterator](#) * [succ_iter](#) (const [state](#) *s) const
- virtual [evtgba_iterator](#) * [pred_iter](#) (const [state](#) *s) const
- virtual std::string [format_state](#) (const [state](#) *state) const
Format the state as a string for printing.
- virtual const [symbol_set](#) & [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual const [symbol_set](#) & [alphabet](#) () const
- virtual std::string [format_label](#) (const [symbol](#) *symbol) const
- virtual std::string [format_acceptance_condition](#) (const [symbol](#) *symbol) const
- virtual std::string [format_acceptance_conditions](#) (const [symbol_set](#) &symset) const

Private Attributes

- const [evtgba_product_operands](#) op_
- [symbol_set](#) alphabet_
- [symbol_set](#) all_acc_
- [common_symbol_table](#) common_symbols_

12.40.1 Member Typedef Documentation

12.40.1.1 typedef std::map<const [symbol](#)*, std::set<int> > [spot::evtgba_product::common_symbol_table](#)

12.40.1.2 typedef std::vector<const [evtgba](#)*> [spot::evtgba_product::evtgba_product_operands](#)

12.40.2 Constructor & Destructor Documentation

12.40.2.1 [spot::evtgba_product::evtgba_product](#) (const [evtgba_product_operands](#) &op)

12.40.2.2 virtual [spot::evtgba_product::~~evtgba_product](#) () [virtual]

12.40.3 Member Function Documentation

12.40.3.1 virtual const [symbol_set](#)& [spot::evtgba_product::all_acceptance_conditions](#) () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::evtgba](#).

12.40.3.2 virtual const [symbol_set](#)& spot::evtgba_product::alphabet () const [virtual]

Implements [spot::evtgba](#).

12.40.3.3 virtual std::string spot::evtgba::format_acceptance_condition (const [symbol](#) * *symbol*) const [virtual, inherited]

12.40.3.4 virtual std::string spot::evtgba::format_acceptance_conditions (const [symbol_set](#) & *sym-set*) const [virtual, inherited]

12.40.3.5 virtual std::string spot::evtgba::format_label (const [symbol](#) * *symbol*) const [virtual, inherited]

12.40.3.6 virtual std::string spot::evtgba_product::format_state (const [state](#) * *state*) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::evtgba](#).

12.40.3.7 virtual [evtgba_iterator](#)* spot::evtgba_product::init_iter () const [virtual]

Implements [spot::evtgba](#).

12.40.3.8 virtual [evtgba_iterator](#)* spot::evtgba_product::pred_iter (const [state](#) * *s*) const [virtual]

Implements [spot::evtgba](#).

12.40.3.9 virtual [evtgba_iterator](#)* spot::evtgba_product::succ_iter (const [state](#) * *s*) const [virtual]

Implements [spot::evtgba](#).

12.40.4 Member Data Documentation

12.40.4.1 [symbol_set](#) spot::evtgba_product::all_acc_ [private]

12.40.4.2 [symbol_set](#) spot::evtgba_product::alphabet_ [private]

12.40.4.3 [common_symbol_table](#) spot::evtgba_product::common_symbols_ [private]

12.40.4.4 const [evtgba_product_operands](#) spot::evtgba_product::op_ [private]

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

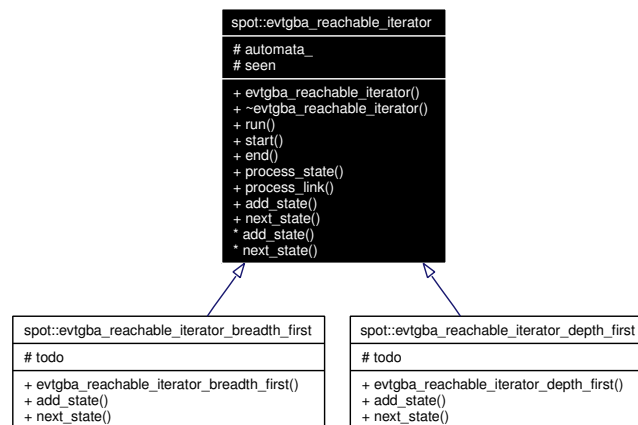
- [evtgba/product.hh](#)

12.41 spot::evtgba_reachable_iterator Class Reference

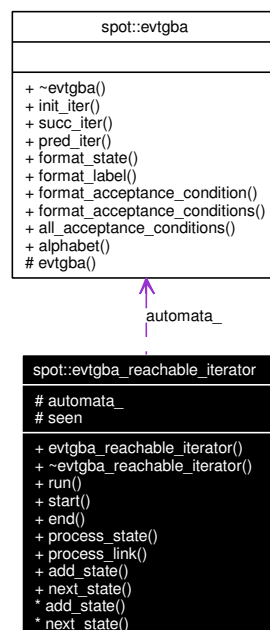
Iterate over all reachable states of a [spot::evtgba](#).

```
#include <evtgbaalgos/reachiter.hh>
```

Inheritance diagram for spot::evtgba_reachable_iterator:



Collaboration diagram for spot::evtgba_reachable_iterator:



Public Member Functions

- [evtgba_reachable_iterator](#) (const [evtgba](#) *a)
- virtual [~evtgba_reachable_iterator](#) ()
- void [run](#) ()

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

- virtual void [start](#) (int n)
Called by [run\(\)](#) before starting its iteration.
- virtual void [end](#) ()
Called by [run\(\)](#) once all states have been explored.
- virtual void [process_state](#) (const [state](#) *s, int n, [evtgba_iterator](#) *si)
- virtual void [process_link](#) (int in, int out, const [evtgba_iterator](#) *si)

Todo list management.

Called by [run\(\)](#) to register newly discovered states.

[spot::evtgba_reachable_iterator_depth_first](#) and [spot::evtgba_reachable_iterator_breadth_first](#) offer two precanned implementations for these functions.

- virtual void [add_state](#) (const [state](#) *s)=0
- virtual const [state](#) * [next_state](#) ()=0
Called by [run\(\)](#) to obtain the.

Protected Types

- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [seen_map](#)

Protected Attributes

- const [evtgba](#) * [automata_](#)
The [spot::evtgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

12.41.1 Detailed Description

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

12.41.2 Member Typedef Documentation

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

12.41.3 Constructor & Destructor Documentation

12.41.3.1 [spot::evtgba_reachable_iterator::evtgba_reachable_iterator](#) (const [evtgba](#) * a)

12.41.3.2 virtual [spot::evtgba_reachable_iterator::~~evtgba_reachable_iterator](#) () [virtual]

12.41.4 Member Function Documentation

12.41.4.1 `virtual void spot::evtgba_reachable_iterator::add_state (const state * s)` [pure virtual]

Implemented in [spot::evtgba_reachable_iterator_depth_first](#), and [spot::evtgba_reachable_iterator_breadth_first](#).

12.41.4.2 `virtual void spot::evtgba_reachable_iterator::end ()` [virtual]

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

12.41.4.3 `virtual const state* spot::evtgba_reachable_iterator::next_state ()` [pure virtual]

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

Implemented in [spot::evtgba_reachable_iterator_depth_first](#), and [spot::evtgba_reachable_iterator_breadth_first](#).

12.41.4.4 `virtual void spot::evtgba_reachable_iterator::process_link (int in, int out, const evtgba_iterator * si)` [virtual]

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

Parameters:

in The source state number.

out The destination state number.

si The [spot::evtgba_iterator](#) positionned on the current transition.

12.41.4.5 `virtual void spot::evtgba_reachable_iterator::process_state (const state * s, int n, evtgba_iterator * si)` [virtual]

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

Parameters:

s The current state.

n An unique number assigned to *s*.

si The [spot::evtgba_iterator](#) for *s*.

12.41.4.6 `void spot::evtgba_reachable_iterator::run ()`

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

12.41.4.7 `virtual void spot::evtgba_reachable_iterator::start (int n)` [virtual]

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

Parameters:

n The number of initial states.

12.41.5 Member Data Documentation

12.41.5.1 `const evtgba*` `spot::evtgba_reachable_iterator::automata_` [protected]

The `spot::evtgba` to explore.

12.41.5.2 `seen_map` `spot::evtgba_reachable_iterator::seen` [protected]

States already seen.

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

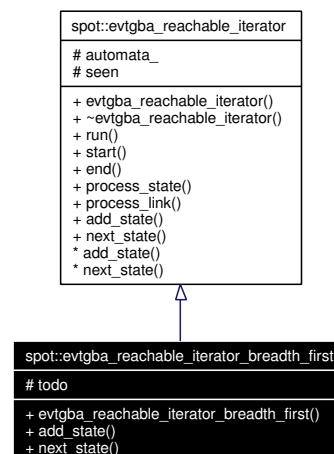
- `evtgbalogs/reachiter.hh`

12.42 spot::evtgba_reachable_iterator_breadth_first Class Reference

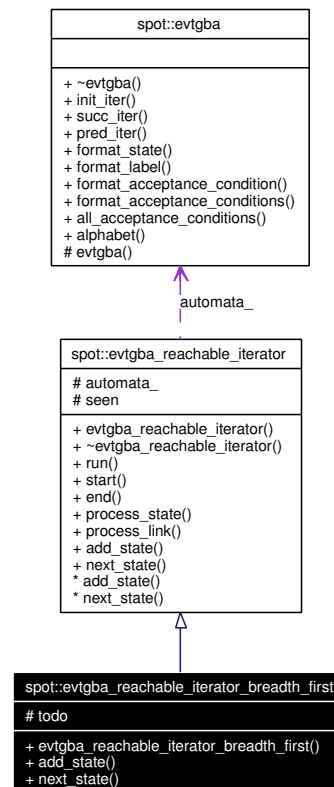
An implementation of `spot::evtgba_reachable_iterator` that browses states breadth first.

```
#include <evtgbalogs/reachiter.hh>
```

Inheritance diagram for `spot::evtgba_reachable_iterator_breadth_first`:



Collaboration diagram for `spot::evtgba_reachable_iterator_breadth_first`:



Public Member Functions

- `evtgba_reachable_iterator_breadth_first` (const `evtgba` *a)
- virtual void `add_state` (const `state` *s)
- virtual const `state` * `next_state` ()
Called by `run()` to obtain the.
- void `run` ()
Iterate over all reachable states of a `spot::evtgba`.
- virtual void `start` (int n)
Called by `run()` before starting its iteration.
- virtual void `end` ()
Called by `run()` once all states have been explored.
- virtual void `process_state` (const `state` *s, int n, `evtgba_iterator` *si)
- virtual void `process_link` (int in, int out, const `evtgba_iterator` *si)

Protected Types

- typedef `Sgi::hash_map`< const `state` *, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

Protected Attributes

- `std::deque< const state * >` `todo`
A queue of states yet to explore.
- `const evtgba * automata_`
The [spot::evtgba](#) to explore.
- `seen_map` `seen`
States already seen.

12.42.1 Detailed Description

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

12.42.2 Member Typedef Documentation

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

12.42.3 Constructor & Destructor Documentation

12.42.3.1 `spot::evtgba_reachable_iterator_breadth_first::evtgba_reachable_iterator_breadth_first (const evtgba * a)`

12.42.4 Member Function Documentation

12.42.4.1 `virtual void spot::evtgba_reachable_iterator_breadth_first::add_state (const state * s)`
[virtual]

Implements [spot::evtgba_reachable_iterator](#).

12.42.4.2 `virtual void spot::evtgba_reachable_iterator::end ()` [virtual, inherited]

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

12.42.4.3 `virtual const state* spot::evtgba_reachable_iterator_breadth_first::next_state ()`
[virtual]

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

Implements [spot::evtgba_reachable_iterator](#).

12.42.4.4 `virtual void spot::evtgba_reachable_iterator::process_link (int in, int out, const evtgba_iterator * si)` [virtual, inherited]

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

Parameters:

in The source state number.

out The destination state number.

si The [spot::evtgba_iterator](#) positionned on the current transition.

12.42.4.5 virtual void spot::evtgba_reachable_iterator::process_state (const [state](#) * *s*, int *n*, [evtgba_iterator](#) * *si*) [virtual, inherited]

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

Parameters:

s The current state.

n An unique number assigned to *s*.

si The [spot::evtgba_iterator](#) for *s*.

12.42.4.6 void spot::evtgba_reachable_iterator::run () [inherited]

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

12.42.4.7 virtual void spot::evtgba_reachable_iterator::start (int *n*) [virtual, inherited]

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

Parameters:

n The number of initial states.

12.42.5 Member Data Documentation

12.42.5.1 const [evtgba](#)* [spot::evtgba_reachable_iterator::automata_](#) [protected, inherited]

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

12.42.5.2 [seen_map](#) [spot::evtgba_reachable_iterator::seen](#) [protected, inherited]

States already seen.

12.42.5.3 [std::deque](#)<const [state](#)*> [spot::evtgba_reachable_iterator_breadth_first::todo](#) [protected]

A queue of states yet to explore.

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

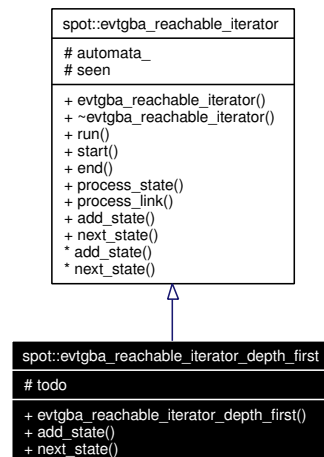
- [evtgbaaalgos/reachiter.hh](#)

12.43 spot::evtgba_reachable_iterator_depth_first Class Reference

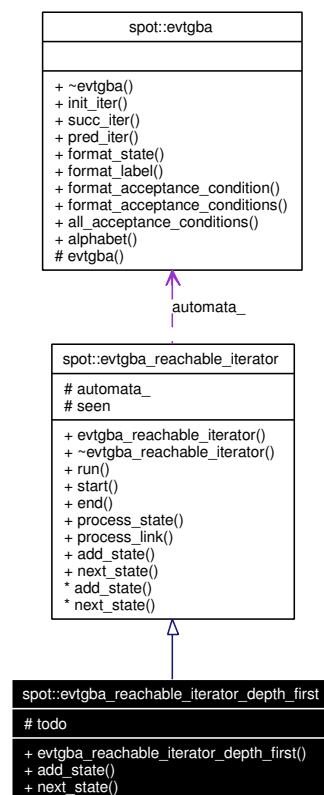
An implementation of [spot::evtgba_reachable_iterator](#) that browses states depth first.

```
#include <evtgbaalgos/reachiter.hh>
```

Inheritance diagram for spot::evtgba_reachable_iterator_depth_first:



Collaboration diagram for spot::evtgba_reachable_iterator_depth_first:



Public Member Functions

- [evtgba_reachable_iterator_depth_first](#) (const [evtgba](#) *a)
- virtual void [add_state](#) (const [state](#) *s)
- virtual const [state](#) * [next_state](#) ()
Called by [run\(\)](#) to obtain the.
- void [run](#) ()
Iterate over all reachable states of a [spot::evtgba](#).
- virtual void [start](#) (int n)
Called by [run\(\)](#) before starting its iteration.
- virtual void [end](#) ()
Called by [run\(\)](#) once all states have been explored.
- virtual void [process_state](#) (const [state](#) *s, int n, [evtgba_iterator](#) *si)
- virtual void [process_link](#) (int in, int out, const [evtgba_iterator](#) *si)

Protected Types

- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [seen_map](#)

Protected Attributes

- std::stack< const [state](#) * > [todo](#)
A stack of states yet to explore.
- const [evtgba](#) * [automata_](#)
The [spot::evtgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

12.43.1 Detailed Description

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

12.43.2 Member Typedef Documentation

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

12.43.3 Constructor & Destructor Documentation

12.43.3.1 [spot::evtgba_reachable_iterator_depth_first::evtgba_reachable_iterator_depth_first](#)
(const [evtgba](#) * a)

12.43.4 Member Function Documentation

12.43.4.1 `virtual void spot::evtgba_reachable_iterator_depth_first::add_state (const state * s)` `[virtual]`

Implements [spot::evtgba_reachable_iterator](#).

12.43.4.2 `virtual void spot::evtgba_reachable_iterator::end ()` `[virtual, inherited]`

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

12.43.4.3 `virtual const state* spot::evtgba_reachable_iterator_depth_first::next_state ()` `[virtual]`

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

Implements [spot::evtgba_reachable_iterator](#).

12.43.4.4 `virtual void spot::evtgba_reachable_iterator::process_link (int in, int out, const evtgba_iterator * si)` `[virtual, inherited]`

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

Parameters:

in The source state number.

out The destination state number.

si The [spot::evtgba_iterator](#) positionned on the current transition.

12.43.4.5 `virtual void spot::evtgba_reachable_iterator::process_state (const state * s, int n, evtgba_iterator * si)` `[virtual, inherited]`

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

Parameters:

s The current state.

n An unique number assigned to *s*.

si The [spot::evtgba_iterator](#) for *s*.

12.43.4.6 `void spot::evtgba_reachable_iterator::run ()` `[inherited]`

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

12.43.4.7 `virtual void spot::evtgba_reachable_iterator::start (int n)` `[virtual, inherited]`

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

Parameters:

n The number of initial states.

12.43.5 Member Data Documentation

12.43.5.1 `const` `evtgba*` `spot::evtgba_reachable_iterator::automata_` [protected, inherited]

The `spot::evtgba` to explore.

12.43.5.2 `seen_map` `spot::evtgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.43.5.3 `std::stack<const` `state*>` `spot::evtgba_reachable_iterator_depth_first::todo` [protected]

A stack of states yet to explore.

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

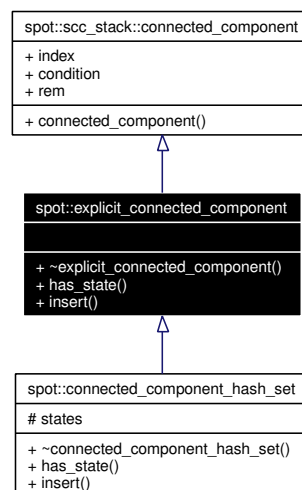
- `evtgbaalgos/reachiter.hh`

12.44 spot::explicit_connected_component Class Reference

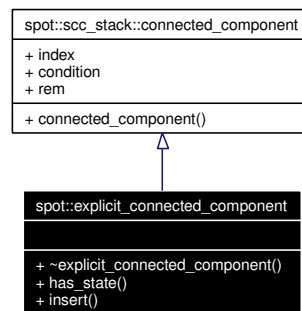
An SCC storing all its states explicitly.

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

Inheritance diagram for `spot::explicit_connected_component`:



Collaboration diagram for `spot::explicit_connected_component`:



Public Member Functions

- virtual `~explicit_connected_component()`
- virtual const `state * has_state (const state *s) const =0`
Check if the SCC contains states `s`.
- virtual void `insert (const state *s)=0`
Insert a new state in the SCC.

Public Attributes

- int `index`
Index of the SCC.
- bdd `condition`
- `std::list< const state * > rem`

12.44.1 Detailed Description

An SCC storing all its states explicitly.

12.44.2 Constructor & Destructor Documentation

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

12.44.3 Member Function Documentation

12.44.3.1 virtual const `state* spot::explicit_connected_component::has_state (const state *s) const`
`[pure virtual]`

Check if the SCC contains states `s`.

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

Implemented in `spot::connected_component_hash_set`.

12.44.3.2 `virtual void spot::explicit_connected_component::insert (const state * s) [pure virtual]`

Insert a new state in the SCC.

Implemented in [spot::connected_component_hash_set](#).

12.44.4 Member Data Documentation

12.44.4.1 `bdd spot::scc_stack::connected_component::condition [inherited]`

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

12.44.4.2 `int spot::scc_stack::connected_component::index [inherited]`

Index of the SCC.

12.44.4.3 `std::list<const state*> spot::scc_stack::connected_component::rem [inherited]`

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

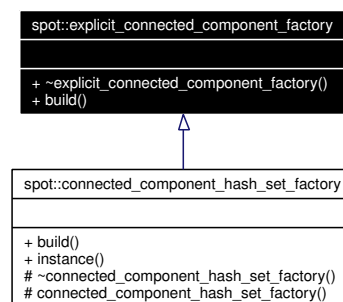
- [tgbaalgos/gtec/explscg.hh](#)

12.45 spot::explicit_connected_component_factory Class Reference

Abstract factory for [explicit_connected_component](#).

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

Inheritance diagram for `spot::explicit_connected_component_factory`:



Public Member Functions

- virtual `~explicit_connected_component_factory ()`
- virtual `explicit_connected_component * build () const =0`
Create an [explicit_connected_component](#).

12.45.1 Detailed Description

Abstract factory for [explicit_connected_component](#).

12.45.2 Constructor & Destructor Documentation

12.45.2.1 virtual `spot::explicit_connected_component_factory::~explicit_connected_component_factory()` [`inline`, `virtual`]

12.45.3 Member Function Documentation

12.45.3.1 virtual `explicit_connected_component*` `spot::explicit_connected_component_factory::build()` `const` [`pure virtual`]

Create an `explicit_connected_component`.

Implemented in `spot::connected_component_hash_set_factory`.

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

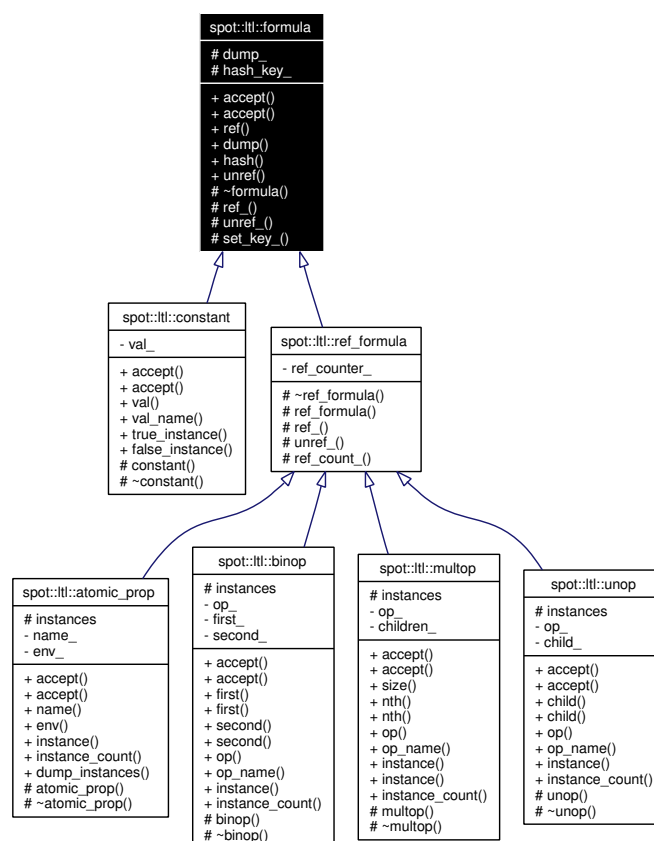
- `tgbaalgos/gtec/explsc.h`

12.46 spot::ltl::formula Class Reference

An LTL formula.

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

Inheritance diagram for `spot::ltl::formula`:



Public Member Functions

- virtual void `accept (visitor &v)=0`
Entry point for vspot::ltl::visitor instances.
- virtual void `accept (const_visitor &v) const =0`
Entry point for vspot::ltl::const_visitor instances.
- `formula * ref ()`
clone this node
- const std::string & `dump () const`
Return a canonic representation of the formula.
- const size_t `hash () const`
Return a hash_key for the formula.

Static Public Member Functions

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

Protected Member Functions

- virtual `~formula ()`
- virtual void `ref_ ()`
increment reference counter if any
- virtual bool `unref_ ()`
decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).
- void `set_key_ ()`
Compute key_ from dump_.

Protected Attributes

- std::string `dump_`
The canonic representation of the formula.
- size_t `hash_key_`
The hash key of this formula.

12.46.1 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](#).

12.46.2 Constructor & Destructor Documentation

12.46.2.1 `virtual spot::ltl::formula::~formula ()` [protected, virtual]

12.46.3 Member Function Documentation

12.46.3.1 `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](#).

12.46.3.2 `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](#).

12.46.3.3 `const std::string& spot::ltl::formula::dump () const`

Return a canonic representation of the formula.

12.46.3.4 `const size_t spot::ltl::formula::hash () const` [inline]

Return a hash_key for the formula.

12.46.3.5 `formula* spot::ltl::formula::ref ()`

clone this node

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

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

increment reference counter if any

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

12.46.3.7 `void spot::ltl::formula::set_key_ ()` [protected]

Compute key_ from dump_.

Should be called once in each object, after dump_ has been set.

12.46.3.8 static void spot::ltl::formula::unref (formula *f) [static]

release this node

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

12.46.3.9 virtual bool spot::ltl::formula::unref_ () [protected, virtual]

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

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

12.46.4 Member Data Documentation**12.46.4.1 std::string spot::ltl::formula::dump_ [protected]**

The canonic representation of the formula.

12.46.4.2 size_t spot::ltl::formula::hash_key_ [protected]

The hash key of this formula.

Initialized by [set_key_\(\)](#).

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

- [ltlast/formula.hh](#)

12.47 spot::ltl::formula_ptr_hash Struct Reference

Hash Function for `const formula*`.

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

Public Member Functions

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

12.47.1 Detailed Description

Hash Function for `const formula*`.

This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `const formula*`.

For instance here is how one could declare a map of `const :: formula*`.

```
// Remember how many times each formula has been seen.
Sgi::hash_map<const spot::ltl::formula*, int,
             const spot::ltl::formula_ptr_hash> seen;
```


12.47.2 Member Function Documentation

12.47.2.1 size_t spot::ltl::formula_ptr_hash::operator() (const formula * *that*) const [inline]

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

- ltlast/formula.hh

12.48 spot::ltl::formula_ptr_less_than Struct Reference

Strict Weak Ordering for const formula*.

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

Public Member Functions

- bool operator() (const formula *left, const formula *right) const

12.48.1 Detailed Description

Strict Weak Ordering for const formula*.

This is meant to be used as a comparison functor for STL map whose key are of type const formula*.

For instance here is how one could declare a map of const :: formula*.

```
// Remember how many times each formula has been seen.
std::map<const spot::ltl::formula*, int,
    spot::formula_ptr_less_than> seen;
```

12.48.2 Member Function Documentation

12.48.2.1 bool spot::ltl::formula_ptr_less_than::operator() (const formula * *left*, const formula * *right*) const [inline]

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

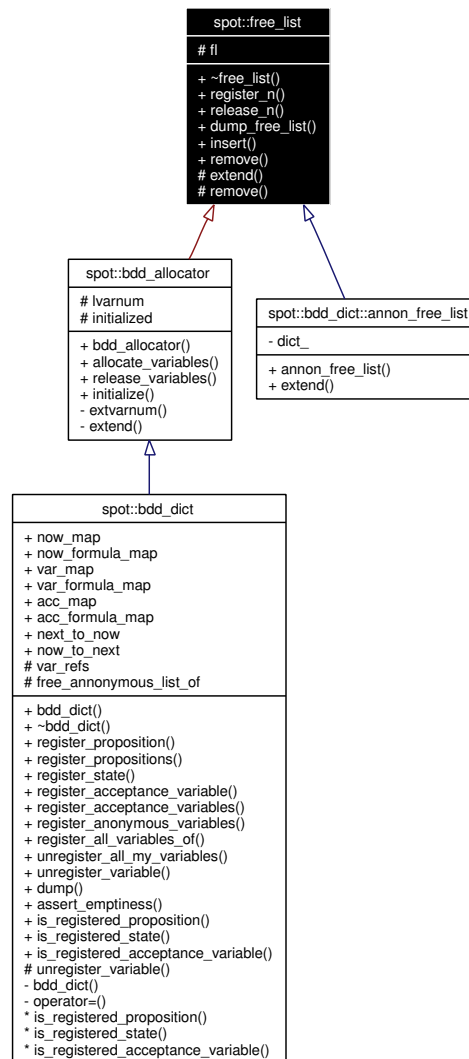
- ltlast/formula.hh

12.49 spot::free_list Class Reference

Manage list of free integers.

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

Inheritance diagram for spot::free_list:



Public Member Functions

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

Protected Types

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

Protected Member Functions

- virtual `int` `extend` (`int` n)=0
- void `remove` (`free_list_type::iterator` i, `int` base, `int` n)
Remove n consecutive entries from the list, starting at base.

Protected Attributes

- `free_list_type` fl
Tracks unused BDD variables.

12.49.1 Detailed Description

Manage list of free integers.

12.49.2 Member Typedef Documentation

12.49.2.1 typedef `std::list<pos_lenght_pair>` `spot::free_list::free_list_type` [protected]

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

Such pairs describe second free integer starting at first.

12.49.3 Constructor & Destructor Documentation

12.49.3.1 virtual `spot::free_list::~free_list` () [virtual]

12.49.4 Member Function Documentation

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

Dump the list to *os* for debugging.

12.49.4.2 `virtual int spot::free_list::extend (int n)` [protected, pure virtual]

Allocate *n* integer.

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

Implemented in `spot::bdd_allocator`, and `spot::bdd_dict::annon_free_list`.

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

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

12.49.4.4 `int spot::free_list::register_n (int n)`

Find *n* consecutive integers.

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

Returns:

the first integer of the range

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

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

12.49.4.6 `void spot::free_list::remove (free_list_type::iterator i, int base, int n)` [protected]

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

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

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

12.49.5 Member Data Documentation**12.49.5.1** `free_list_type spot::free_list::fl` [protected]

Tracks unused BDD variables.

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

- `misc/freelist.hh`

12.50 `spot::gspn_exception` Class Reference

An exception used to forward GSPN errors.

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

Public Member Functions

- [gspn_exeption](#) (const std::string &where, int err)
- int [get_err](#) () const
- std::string [get_where](#) () const

Private Attributes

- int [err_](#)
- std::string [where_](#)

12.50.1 Detailed Description

An exeption used to forward GSPN errors.

12.50.2 Constructor & Destructor Documentation

12.50.2.1 `spot::gspn_exeption::gspn_exeption (const std::string & where, int err)` `[inline]`

12.50.3 Member Function Documentation

12.50.3.1 `int spot::gspn_exeption::get_err () const` `[inline]`

12.50.3.2 `std::string spot::gspn_exeption::get_where () const` `[inline]`

12.50.4 Member Data Documentation

12.50.4.1 `int spot::gspn_exeption::err_` `[private]`

12.50.4.2 `std::string spot::gspn_exeption::where_` `[private]`

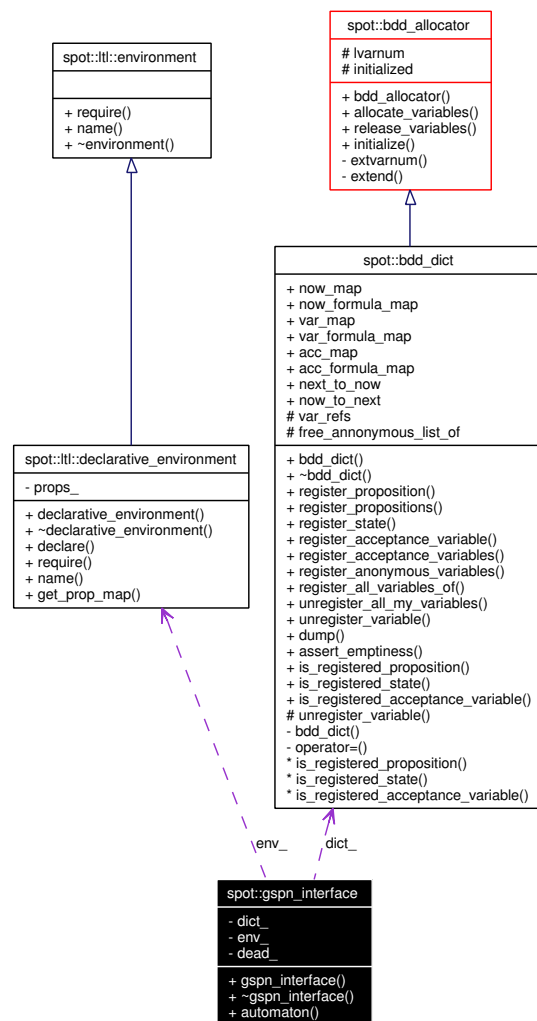
The documentation for this class was generated from the following file:

- [gspn/common.hh](#)

12.51 spot::gspn_interface Class Reference

```
#include <gspn/gspn.hh>
```

Collaboration diagram for spot::gspn_interface:



Public Member Functions

- `gspn_interface` (int argc, char **argv, `bdd_dict` *dict, `ltl::declarative_environment` &env, const std::string &dead="true")
- `~gspn_interface` ()
- `tgba * automaton` () const

Private Attributes

- `bdd_dict` * `dict_`
- `ltl::declarative_environment` & `env_`
- const std::string `dead_`

12.51.1 Constructor & Destructor Documentation

12.51.1.1 `spot::gspn_interface::gspn_interface` (int *argc*, char ** *argv*, `bdd_dict` * *dict*, `ltl::declarative_environment` & *env*, const std::string & *dead* = "true")

12.51.1.2 spot::gspn_interface::~~gspn_interface ()

12.51.2 Member Function Documentation

12.51.2.1 tgba* spot::gspn_interface::automaton () const

12.51.3 Member Data Documentation

12.51.3.1 const std::string spot::gspn_interface::dead_ [private]

12.51.3.2 bdd_dict* spot::gspn_interface::dict_ [private]

12.51.3.3 ltl::declarative_environment& spot::gspn_interface::env_ [private]

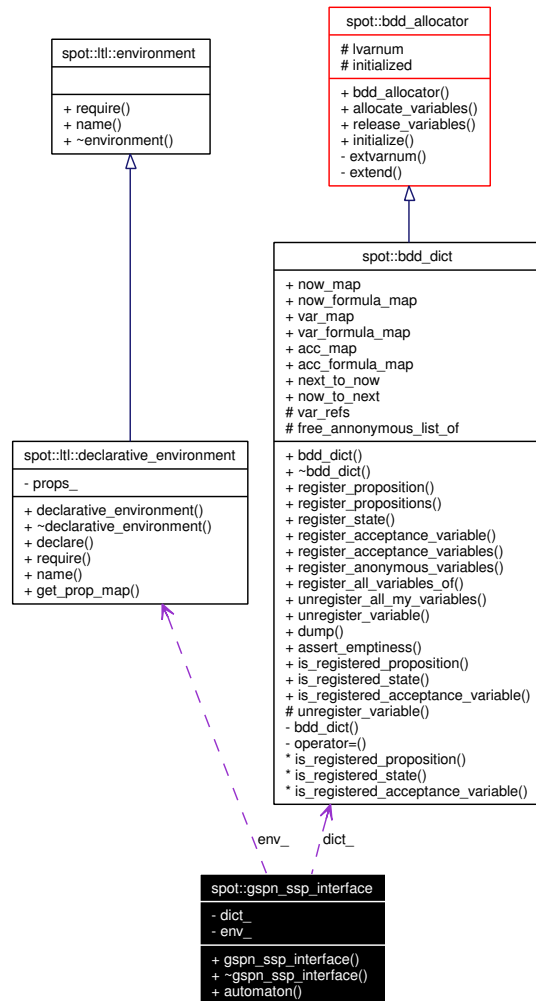
The documentation for this class was generated from the following file:

- [gspn/gspn.hh](#)

12.52 spot::gspn_ssp_interface Class Reference

```
#include <gspn/ssp.hh>
```

Collaboration diagram for spot::gspn_ssp_interface:



Public Member Functions

- `gspn_ssp_interface` (int argc, char **argv, `bdd_dict` *dict, const `ltl::declarative_environment` &env, bool inclusion=false)
- `~gspn_ssp_interface` ()
- `tgba * automaton` (const `tgba` *operand) const

Private Attributes

- `bdd_dict` * dict_
- const `ltl::declarative_environment` & env_

12.52.1 Constructor & Destructor Documentation

12.52.1.1 `spot::gspn_ssp_interface::gspn_ssp_interface` (int argc, char ** argv, `bdd_dict` * dict, const `ltl::declarative_environment` & env, bool inclusion = false)

12.52.1.2 `spot::gspn_ssp_interface::~~gspn_ssp_interface` ()

12.52.2 Member Function Documentation

12.52.2.1 [tgba*](#) [spot::gspn_ssp_interface::automaton](#) (const [tgba](#) * *operand*) const

12.52.3 Member Data Documentation

12.52.3.1 [bdd_dict*](#) [spot::gspn_ssp_interface::dict_](#) [private]

12.52.3.2 const [ltl::declarative_environment&](#) [spot::gspn_ssp_interface::env_](#) [private]

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

- [gspn/ssp.hh](#)

12.53 spot::loopless_modular_mixed_radix_gray_code Class Reference

Loopless modular mixed radix Gray code iteration.

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

Public Member Functions

- [loopless_modular_mixed_radix_gray_code](#) (int n)
- virtual [~loopless_modular_mixed_radix_gray_code](#) ()

iteration over an element in a tuple

The class does not know how to modify the elements of the tuple (Knuth's a_j 's). These changes are therefore abstracted using the `a_first()`, `a_next()`, and `a_last()` abstract functions. These need to be implemented in subclasses as appropriate.

- virtual void [a_first](#) (int j)=0
Reset a_j to its initial value.
- virtual void [a_next](#) (int j)=0
Advance a_j to its next value.
- virtual bool [a_last](#) (int j) const =0
Whether a_j is on its last value.

iteration over all the tuples

- void [first](#) ()
Reset the iteration to the first tuple.
- bool [last](#) () const
Whether this the last tuple.
- bool [done](#) () const
Whether all tuple have been explored.
- int [next](#) ()
Update one item of the tuple and return its position.

Protected Attributes

- int `n_`
- bool `done_`
- int * `a_`
- int * `f_`
- int * `m_`
- int * `s_`
- int * `non_one_radixes_`

12.53.1 Detailed Description

Loopless modular mixed radix Gray code iteration.

This class is based on the loopless modular mixed radix gray code algorithm described in exercise 77 of "The Art of Computer Programming", Pre-Fascicle 2A (Draft of section 7.2.1.1: generating all n-tuples) by Donald E. Knuth.

The idea is to enumerate the set of all n-tuples $(a_0, a_1, \dots, a_{n-1})$ where each a_j range over a distinct set (this is the *mixed radix* part), so that only one a_j changes between two successive tuples of the iteration (that is the *Gray code* part), and that this changes occurs always in the same direction, cycling over the set a_j must cover (i.e., *modular*). The algorithm is *loopless* in that computing the next tuple done without any loop, i.e., in constant time.

This class does not need to know the type of the a_j , it will handle them indirectly through three methods: `a_first()`, `a_next()`, and `a_last()`. These methods need to be implemented in a subclass for the particular type of a_j at hand.

The class itself offers four functions to control the iteration over the set of all the $(a_0, a_1, \dots, a_{n-1})$ tuples: `first()`, `next()`, `last()`, and `done()`. These functions are usually used as follows:

```
for (g.first(); !g.done(); g.next())
    use the tuple
```

How to use the tuple of course depends on the way it as been stored in the subclass.

Finally, let's mention two differences between this algorithm and the one in Knuth's book. This version of the algorithm does not need to know the radixes (i.e., the size of set of each a_j) beforehand: it will discover them on-the-fly when `a_last(j)` first return true. It will also work with a_j that cannot be changed. (This is achieved by reindexing the elements through `non_one_radixes_`, to consider only the elements with a non-singleton range.)

12.53.2 Constructor & Destructor Documentation

12.53.2.1 spot::loopless_modular_mixed_radix_gray_code::loopless_modular_mixed_radix_gray_code (int *n*)

Constructor.

Parameters:

- n* The size of the tuples to enumerate.

12.53.2.2 virtual spot::loopless_modular_mixed_radix_gray_code::~loopless_modular_mixed_radix_gray_code () [virtual]

12.53.3 Member Function Documentation

12.53.3.1 `virtual void spot::loopless_modular_mixed_radix_gray_code::a_first (int j) [pure virtual]`

Reset a_j to its initial value.

12.53.3.2 `virtual bool spot::loopless_modular_mixed_radix_gray_code::a_last (int j) const [pure virtual]`

Whether a_j is on its last value.

12.53.3.3 `virtual void spot::loopless_modular_mixed_radix_gray_code::a_next (int j) [pure virtual]`

Advance a_j to its next value.

This will never be called if `a_last(j)` is true.

12.53.3.4 `bool spot::loopless_modular_mixed_radix_gray_code::done () const [inline]`

Whether all tuple have been explored.

12.53.3.5 `void spot::loopless_modular_mixed_radix_gray_code::first ()`

Reset the iteration to the first tuple.

This must be called before calling any of `next()`, `last()`, or `done()`.

12.53.3.6 `bool spot::loopless_modular_mixed_radix_gray_code::last () const [inline]`

Whether this the last tuple.

At this point it is still OK to call `next()`, and then `done()` will become true.

12.53.3.7 `int spot::loopless_modular_mixed_radix_gray_code::next ()`

Update one item of the tuple and return its position.

`next()` should never be called if `done()` is true. If it is called on the last tuple (i.e., `last()` is true), it will return -1. Otherwise it will update one a_j of the tuple through one the a_j handling functions, and return j .

12.53.4 Member Data Documentation

12.53.4.1 `int* spot::loopless_modular_mixed_radix_gray_code::a_ [protected]`

12.53.4.2 `bool spot::loopless_modular_mixed_radix_gray_code::done_ [protected]`

12.53.4.3 `int* spot::loopless_modular_mixed_radix_gray_code::f_ [protected]`

12.53.4.4 `int* spot::loopless_modular_mixed_radix_gray_code::m_ [protected]`

12.53.4.5 `int` `spot::loopless_modular_mixed_radix_gray_code::n_` [protected]

12.53.4.6 `int*` `spot::loopless_modular_mixed_radix_gray_code::non_one_radixes_` [protected]

12.53.4.7 `int*` `spot::loopless_modular_mixed_radix_gray_code::s_` [protected]

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

- `misc/modgray.hh`

12.54 `spot::minato_isop` Class Reference

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

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

Public Member Functions

- `minato_isop` (bdd input)
Constructor.
 - *input* The BDD function to translate in ISOP.
- `minato_isop` (bdd input, bdd vars)
Constructor.
 - *input* The BDD function to translate in ISOP.
 - *vars* The set of BDD variables to factorize in input.
- `bdd next ()`
Compute the next sum term of the ISOP form. Return `bddfalse` when all terms have been output.

Private Attributes

- `std::stack< local_vars > todo_`
- `std::stack< bdd > cube_`
- `bdd ret_`

Classes

- `struct local_vars`
Internal variables for `minato_isop`.

12.54.1 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}
}
```

12.54.2 Constructor & Destructor Documentation

12.54.2.1 spot::minato_isop::minato_isop (bdd *input*)

Constructor.

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

12.54.2.2 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*.

12.54.3 Member Function Documentation

12.54.3.1 bdd spot::minato_isop::next ()

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

12.54.4 Member Data Documentation

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

12.54.4.2 bdd spot::minato_isop::ret_ [private]

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

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

- [misc/minato.hh](#)

12.55 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](#)

12.55.1 Detailed Description

Internal variables for [minato_isop](#).

12.55.2 Member Enumeration Documentation

12.55.2.1 anonymous enum

Enumeration values:

FirstStep

SecondStep

ThirdStep

FourthStep

12.55.3 Constructor & Destructor Documentation

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

12.55.4 Member Data Documentation

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

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

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

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

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

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

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

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

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

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

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

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

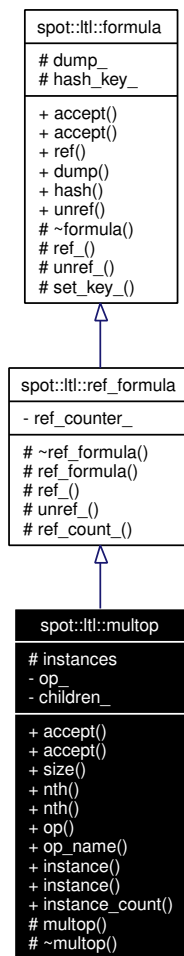
- misc/[minato.hh](#)

12.56 spot::ltl::multop Class Reference

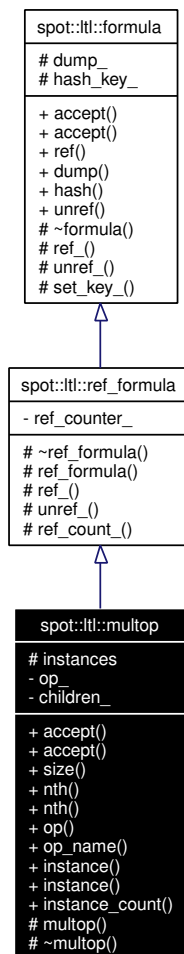
Multi-operand operators.

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

Inheritance diagram for spot::ltl::multop:



Collaboration diagram for `spot::ltl::multop`:



Public Types

- typedef `std::vector< formula * >` `vec`
List of formulae.
- enum `type` { `Or`, `And` }

Public Member Functions

- virtual void `accept (visitor &v)`
Entry point for `vspot::ltl::visitor` instances.
- virtual void `accept (const_visitor &v) const`
Entry point for `vspot::ltl::const_visitor` instances.
- unsigned `size ()` const
Get the number of children.
- const `formula * nth (unsigned n)` const
*Get the *n*th children.*

- `formula * nth` (unsigned n)
Get the nth children.
- `type op ()` const
Get the type of this operator.
- `const char * op_name ()` const
Get the type of this operator, as a string.
- `formula * ref ()`
clone this node
- `const std::string & dump ()` const
Return a canonic representation of the formula.
- `const size_t hash ()` const
Return a hash_key for the formula.

Static Public Member Functions

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

Protected Types

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

Protected Member Functions

- `multop` (type op, `vec *v`)
- virtual `~multop ()`
- void `ref_ ()`
increment reference counter if any
- bool `unref_ ()`

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

- unsigned `ref_count_()`
Number of references to this formula.
- void `set_key_()`
Compute key_ from dump_.

Protected Attributes

- std::string `dump_`
The canonic representation of the formula.
- size_t `hash_key_`
The hash key of this formula.

Static Protected Attributes

- static `map instances`

Private Attributes

- type `op_`
- `vec * children_`

Classes

- struct `paircmp`
Comparison functor used internally by `ltl::multop`.

12.56.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

12.56.2 Member Typedef Documentation

12.56.2.1 `typedef std::map<pair, formula*, paircmp> spot::ltl::multop::map` [protected]

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

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

List of formulae.

12.56.3 Member Enumeration Documentation

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

Enumeration values:

Or

And

12.56.4 Constructor & Destructor Documentation

12.56.4.1 [spot::ltl::multop::multop](#) ([type op](#), [vec * v](#)) [protected]

12.56.4.2 [virtual spot::ltl::multop::~~multop](#) () [protected, virtual]

12.56.5 Member Function Documentation

12.56.5.1 [virtual void spot::ltl::multop::accept](#) ([const_visitor & v](#)) [const](#) [virtual]

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

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

12.56.5.2 [virtual void spot::ltl::multop::accept](#) ([visitor & v](#)) [virtual]

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

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

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

Return a canonic representation of the formula.

12.56.5.4 [const size_t spot::ltl::formula::hash](#) () [const](#) [inline, inherited]

Return a hash_key for the formula.

12.56.5.5 [static formula* spot::ltl::multop::instance](#) ([type op](#), [vec * v](#)) [static]

Build a [spot::ltl::multop](#) with many children.

Same as the other [instance\(\)](#) function, but take a vector of formula in argument. This vector is acquired by the [spot::ltl::multop](#) class, the caller should allocate it with `new`, but not use it (especially not destroy it) after it has been passed to [spot::ltl::multop](#).

This functions can perform slight optimizations and may not return an [ltl::multop](#) objects. For instance if the vector contain only one unique element, this this formula will be returned as-is.

12.56.5.6 [static formula* spot::ltl::multop::instance](#) ([type op](#), [formula * first](#), [formula * second](#)) [static]

Build a [spot::ltl::multop](#) with two children.

If one of the children itself is a `spot::ltl::multop` with the same type, it will be merged. I.e., children if that child will be added, and that child itself will be destroyed. This allows incremental building of n-ary `ltl::multop`.

This functions can perform slight optimizations and may not return an `ltl::multop` objects. For instance if `first` and `second` are equal, that formula is returned as-is.

12.56.5.7 static unsigned spot::ltl::multop::instance_count () [static]

Number of instantiated multi-operand operators. For debugging.

12.56.5.8 formula* spot::ltl::multop::nth (unsigned n)

Get the nth children.

Starting with $n = 0$.

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

Get the nth children.

Starting with $n = 0$.

12.56.5.10 type spot::ltl::multop::op () const

Get the type of this operator.

12.56.5.11 const char* spot::ltl::multop::op_name () const

Get the type of this operator, as a string.

12.56.5.12 formula* spot::ltl::formula::ref () [inherited]

clone this node

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

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

increment reference counter if any

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

12.56.5.14 unsigned spot::ltl::ref_formula::ref_count_ () [protected, inherited]

Number of references to this formula.

12.56.5.15 void spot::ltl::formula::set_key_ () [protected, inherited]

Compute key_ from dump_.

Should be called once in each object, after `dump_` has been set.

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

Get the number of children.

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

release this node

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

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

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

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

12.56.6 Member Data Documentation**12.56.6.1** `vec* spot::ltl::multop::children_ [private]`**12.56.6.2** `std::string spot::ltl::formula::dump_ [protected, inherited]`

The canonic representation of the formula.

12.56.6.3 `size_t spot::ltl::formula::hash_key_ [protected, inherited]`

The hash key of this formula.

Initialized by `set_key_()`.

12.56.6.4 `map spot::ltl::multop::instances [static, protected]`**12.56.6.5** `type spot::ltl::multop::op_ [private]`

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

- `ltlast/multop.hh`

12.57 `spot::ltl::multop::pairemp` Struct Reference

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

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

Public Member Functions

- `bool operator() (const pair &p1, const pair &p2) const`

12.57.1 Detailed Description

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

12.57.2 Member Function Documentation

12.57.2.1 `bool spot::ltl::multop::paircmp::operator() (const pair & p1, const pair & p2) const` `[inline]`

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

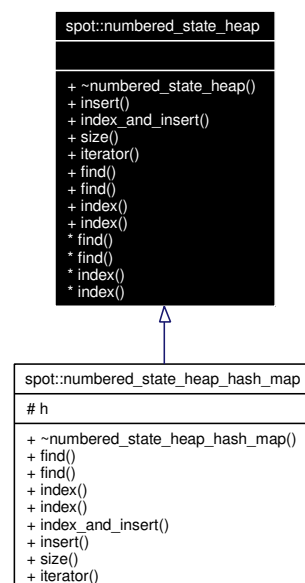
- [ltlast/multop.hh](#)

12.58 spot::numbered_state_heap Class Reference

Keep track of a large quantity of indexed states.

```
#include <tgbaaalgos/gtec/nsheap.hh>
```

Inheritance diagram for `spot::numbered_state_heap`:



Public Types

- `typedef std::pair< const state *, int * > state_index_p`
- `typedef std::pair< const state *, int > state_index`

Public Member Functions

- virtual `~numbered_state_heap ()`
- virtual void `insert (const state *s, int index)=0`

Add a new state s with index index.

- virtual int & [index_and_insert](#) (const [state](#) *&s)=0
Get the index of a state, and insert that state if it is missing.
- virtual int [size](#) () const =0
The number of stored states.
- virtual [numbered_state_heap_const_iterator](#) * [iterator](#) () const =0
Return an iterator on the states/indexes pairs.
- virtual [state_index](#) [find](#) (const [state](#) *s) const =0
Is state in the heap?
- virtual [state_index_p](#) [find](#) (const [state](#) *s)=0
- virtual [state_index](#) [index](#) (const [state](#) *s) const =0
Return the index of an existing state.
- virtual [state_index_p](#) [index](#) (const [state](#) *s)=0

12.58.1 Detailed Description

Keep track of a large quantity of indexed states.

12.58.2 Member Typedef Documentation

12.58.2.1 `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index`

12.58.2.2 `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p`

12.58.3 Constructor & Destructor Documentation

12.58.3.1 `virtual spot::numbered_state_heap::~~numbered_state_heap () [inline, virtual]`

12.58.4 Member Function Documentation

12.58.4.1 `virtual state_index_p spot::numbered_state_heap::find (const state * s) [pure virtual]`

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.2 `virtual state_index spot::numbered_state_heap::find (const state * s) const [pure virtual]`

Is state in the heap?

Returns a pair (0,0) if *s* is not in the heap. or a pair (*p*, *i*) if there is a clone *p* of *s* *i* in the heap with index. If *s* is in the heap and is different from *p* it will be freed.

These functions are called by the algorithm to check whether a successor is a new state to explore or an already visited state.

These functions can be redefined to search for more than an equal match. For example we could redefine it to check state inclusion.

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.3 `virtual state_index_p spot::numbered_state_heap::index (const state * s) [pure virtual]`

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.4 `virtual state_index spot::numbered_state_heap::index (const state * s) const [pure virtual]`

Return the index of an existing state.

This is mostly similar to `find()`, except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.5 `virtual int& spot::numbered_state_heap::index_and_insert (const state *& s) [pure virtual]`

Get the index of a state, and insert that state if it is missing.

If a clone of *s* is already in the hash table, *s* will be deleted and replaced by the address of the clone used.

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.6 `virtual void spot::numbered_state_heap::insert (const state * s, int index) [pure virtual]`

Add a new state *s* with index *index*.

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.7 `virtual numbered_state_heap_const_iterator* spot::numbered_state_heap::iterator () const [pure virtual]`

Return an iterator on the states/indexes pairs.

Implemented in [spot::numbered_state_heap_hash_map](#).

12.58.4.8 `virtual int spot::numbered_state_heap::size () const [pure virtual]`

The number of stored states.

Implemented in [spot::numbered_state_heap_hash_map](#).

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

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

12.59 spot::numbered_state_heap_const_iterator Class Reference

Iterator on [numbered_state_heap](#) objects.

```
#include <tgbaaalgos/gtec/nsheap.hh>
```

Public Member Functions

- virtual [~numbered_state_heap_const_iterator](#) ()
- virtual void [first](#) ()=0
Iteration.
- virtual void [next](#) ()=0
- virtual bool [done](#) () const =0
- virtual const [state](#) * [get_state](#) () const =0
Inspection.
- virtual int [get_index](#) () const =0

12.59.1 Detailed Description

Iterator on [numbered_state_heap](#) objects.

12.59.2 Constructor & Destructor Documentation

12.59.2.1 virtual [spot::numbered_state_heap_const_iterator::~~numbered_state_heap_const_iterator](#) () [inline, virtual]

12.59.3 Member Function Documentation

12.59.3.1 virtual bool [spot::numbered_state_heap_const_iterator::done](#) () const [pure virtual]

12.59.3.2 virtual void [spot::numbered_state_heap_const_iterator::first](#) () [pure virtual]

Iteration.

12.59.3.3 virtual int [spot::numbered_state_heap_const_iterator::get_index](#) () const [pure virtual]

12.59.3.4 virtual const [state](#)* [spot::numbered_state_heap_const_iterator::get_state](#) () const [pure virtual]

Inspection.

12.59.3.5 virtual void spot::numbered_state_heap_const_iterator::next () [pure virtual]

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

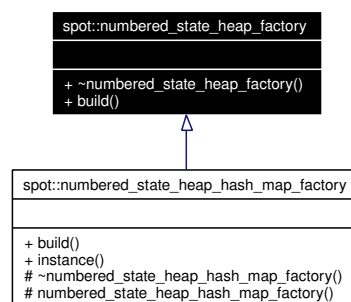
- [tgbaalgos/gtec/nsheap.hh](#)

12.60 spot::numbered_state_heap_factory Class Reference

Abstract factory for [numbered_state_heap](#).

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

Inheritance diagram for spot::numbered_state_heap_factory:



Public Member Functions

- virtual [~numbered_state_heap_factory](#) ()
- virtual [numbered_state_heap](#) * [build](#) () const =0

12.60.1 Detailed Description

Abstract factory for [numbered_state_heap](#).

12.60.2 Constructor & Destructor Documentation

12.60.2.1 virtual spot::numbered_state_heap_factory::~~numbered_state_heap_factory ()
 [inline, virtual]

12.60.3 Member Function Documentation

12.60.3.1 virtual [numbered_state_heap](#)* spot::numbered_state_heap_factory::build () const
 [pure virtual]

Implemented in [spot::numbered_state_heap_hash_map_factory](#).

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

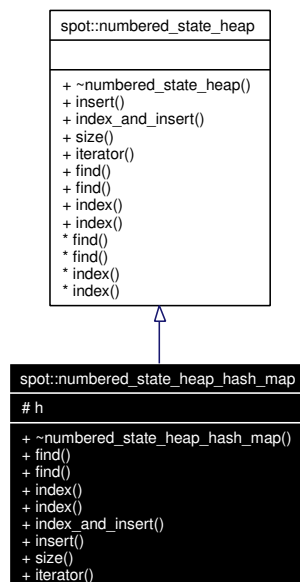
- [tgbaalgos/gtec/nsheap.hh](#)

12.61 spot::numbered_state_heap_hash_map Class Reference

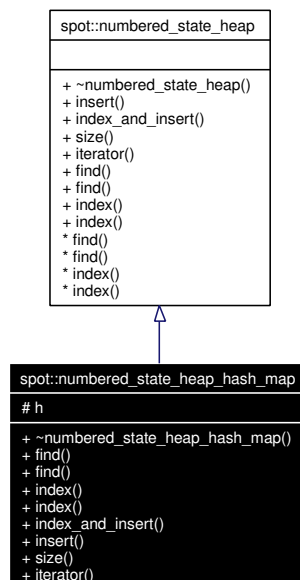
A straightforward implementation of [numbered_state_heap](#) with a hash map.

```
#include <tgbalgorithms/gtec/nsheap.hh>
```

Inheritance diagram for spot::numbered_state_heap_hash_map:



Collaboration diagram for spot::numbered_state_heap_hash_map:



Public Types

- typedef `Sgi::hash_map< const state *, int, state_ptr_hash, state_ptr_equal >` `hash_type`

- typedef std::pair< const [state](#) *, int * > [state_index_p](#)
- typedef std::pair< const [state](#) *, int > [state_index](#)

Public Member Functions

- virtual [~numbered_state_heap_hash_map](#) ()
- virtual [state_index](#) find (const [state](#) *s) const
Is state in the heap?
- virtual [state_index_p](#) find (const [state](#) *s)
- virtual [state_index](#) index (const [state](#) *s) const
Return the index of an existing state.
- virtual [state_index_p](#) index (const [state](#) *s)
- virtual int & [index_and_insert](#) (const [state](#) *&s)
Get the index of a state, and insert that state if it is missing.
- virtual void [insert](#) (const [state](#) *s, int index)
Add a new state s with index index.
- virtual int [size](#) () const
The number of stored states.
- virtual [numbered_state_heap_const_iterator](#) * [iterator](#) () const
Return an iterator on the states/indexes pairs.

Protected Attributes

- [hash_type](#) h
Map of visited states.

12.61.1 Detailed Description

A straightforward implementation of [numbered_state_heap](#) with a hash map.

12.61.2 Member Typedef Documentation

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

12.61.2.2 typedef std::pair<const [state](#)*, int> [spot::numbered_state_heap::state_index](#)
[inherited]

12.61.2.3 typedef std::pair<const [state](#)*, int*> [spot::numbered_state_heap::state_index_p](#)
[inherited]

12.61.3 Constructor & Destructor Documentation

12.61.3.1 virtual spot::numbered_state_heap_hash_map::~~numbered_state_heap_hash_map ()
[virtual]

12.61.4 Member Function Documentation

12.61.4.1 virtual state_index_p spot::numbered_state_heap_hash_map::find (const state * s)
[virtual]

Implements [spot::numbered_state_heap](#).

12.61.4.2 virtual state_index spot::numbered_state_heap_hash_map::find (const state * s) const
[virtual]

Is state in the heap?

Returns a pair (0,0) if *s* is not in the heap. or a pair (p, i) if there is a clone *p* of *s* *i* in the heap with index. If *s* is in the heap and is different from *p* it will be freed.

These functions are called by the algorithm to check whether a successor is a new state to explore or an already visited state.

These functions can be redefined to search for more than an equal match. For example we could redefine it to check state inclusion.

Implements [spot::numbered_state_heap](#).

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

Implements [spot::numbered_state_heap](#).

12.61.4.4 virtual state_index spot::numbered_state_heap_hash_map::index (const state * s) const
[virtual]

Return the index of an existing state.

This is mostly similar to [find\(\)](#), except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

Implements [spot::numbered_state_heap](#).

12.61.4.5 virtual int& spot::numbered_state_heap_hash_map::index_and_insert (const state *& s)
[virtual]

Get the index of a state, and insert that state if it is missing.

If a clone of *s* is already in the hash table, *s* will be deleted and replaced by the address of the clone used.

Implements [spot::numbered_state_heap](#).

12.61.4.6 virtual void spot::numbered_state_heap_hash_map::insert (const state * s, int index)
[virtual]

Add a new state *s* with index *index*.

Implements [spot::numbered_state_heap](#).

12.61.4.7 virtual [numbered_state_heap_const_iterator*](#) [spot::numbered_state_heap_hash_map::iterator \(\) const](#) [virtual]

Return an iterator on the states/indexes pairs.

Implements [spot::numbered_state_heap](#).

12.61.4.8 virtual int [spot::numbered_state_heap_hash_map::size \(\) const](#) [virtual]

The number of stored states.

Implements [spot::numbered_state_heap](#).

12.61.5 Member Data Documentation

12.61.5.1 [hash_type](#) [spot::numbered_state_heap_hash_map::h](#) [protected]

Map of visited states.

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

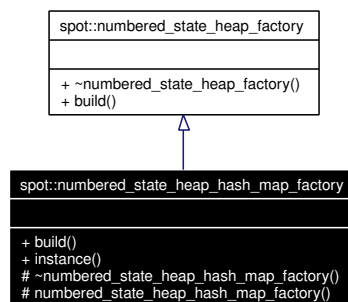
- [tgbaalgos/gtec/nsheap.hh](#)

12.62 spot::numbered_state_heap_hash_map_factory Class Reference

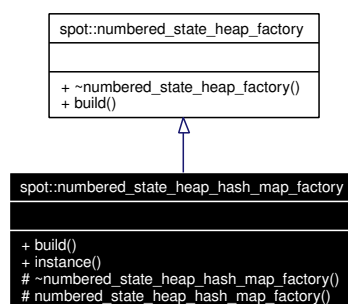
Factory for [numbered_state_heap_hash_map](#).

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

Inheritance diagram for [spot::numbered_state_heap_hash_map_factory](#):



Collaboration diagram for [spot::numbered_state_heap_hash_map_factory](#):



Public Member Functions

- virtual [numbered_state_heap_hash_map](#) * [build](#) () const

Static Public Member Functions

- static const [numbered_state_heap_hash_map_factory](#) * [instance](#) ()
Get the unique instance of this class.

Protected Member Functions

- virtual [~numbered_state_heap_hash_map_factory](#) ()
- [numbered_state_heap_hash_map_factory](#) ()

12.62.1 Detailed Description

Factory for [numbered_state_heap_hash_map](#).

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

12.62.2 Constructor & Destructor Documentation

12.62.2.1 virtual [spot::numbered_state_heap_hash_map_factory::~numbered_state_heap_hash_map_factory](#) () [inline, protected, virtual]

12.62.2.2 [spot::numbered_state_heap_hash_map_factory::numbered_state_heap_hash_map_factory](#) () [protected]

12.62.3 Member Function Documentation

12.62.3.1 virtual [numbered_state_heap_hash_map](#)* [spot::numbered_state_heap_hash_map_factory::build](#) () const [virtual]

Implements [spot::numbered_state_heap_factory](#).

12.62.3.2 static const [numbered_state_heap_hash_map_factory](#)* [spot::numbered_state_heap_hash_map_factory::instance](#) () [static]

Get the unique instance of this class.

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

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

12.63 spot::option_map Class Reference

Manage a map of options.

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


Public Member Functions

- const char * [parse_options](#) (const char *options)
Add the parsed options to the map.
- int [get](#) (const char *option, int def=0) const
Get the value of option.
- int [operator\[\]](#) (const char *option) const
Get the value of option.
- int [set](#) (const char *option, int val, int def=0)
Set the value of option to val.
- void [set](#) (const [option_map](#) &o)
Acquire all the settings of o.
- int & [operator\[\]](#) (const char *option)
Get a reference to the current value of option.

Private Attributes

- std::map< std::string, int > [options_](#)

Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [option_map](#) &m)
Print the [option_map](#) m.

12.63.1 Detailed Description

Manage a map of options.

Each option is defined by a string and is associated to an integer value.

12.63.2 Member Function Documentation

12.63.2.1 int spot::option_map::get (const char * *option*, int *def* = 0) const

Get the value of *option*.

Returns:

The value associated to *option* if it exists, *def* otherwise.

See also:

[operator\[\]\(\)](#)

12.63.2.2 `int& spot::option_map::operator[] (const char * option)`

Get a reference to the current value of *option*.

12.63.2.3 `int spot::option_map::operator[] (const char * option) const`

Get the value of *option*.

Returns:

The value associated to *option* if it exists, 0 otherwise.

See also:

[get\(\)](#)

12.63.2.4 `const char* spot::option_map::parse_options (const char * options)`

Add the parsed options to the map.

options are separated by a space, comma, semicolon or tabulation and can be optionnaly followed by an integer value (preceded by an equal sign). If not specified, the default value is 1.

The following three lines are equivalent.

```
optA !optB optC=4194304
optA=1, optB=0, optC=4096K
optC = 4M; optA !optB
```

Returns:

A non-null pointer to the option for which an expected integer value cannot be parsed.

12.63.2.5 `void spot::option_map::set (const option_map & o)`

Acquire all the settings of *o*.

12.63.2.6 `int spot::option_map::set (const char * option, int val, int def = 0)`

Set the value of *option* to *val*.

Returns:

The previous value associated to *option* if declared, or *def* otherwise.

12.63.3 Friends And Related Function Documentation

12.63.3.1 `std::ostream& operator<< (std::ostream & os, const option_map & m)` [[friend](#)]

Print the [option_map](#) *m*.

12.63.4 Member Data Documentation

12.63.4.1 `std::map<std::string, int> spot::option_map::options_` [[private](#)]

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

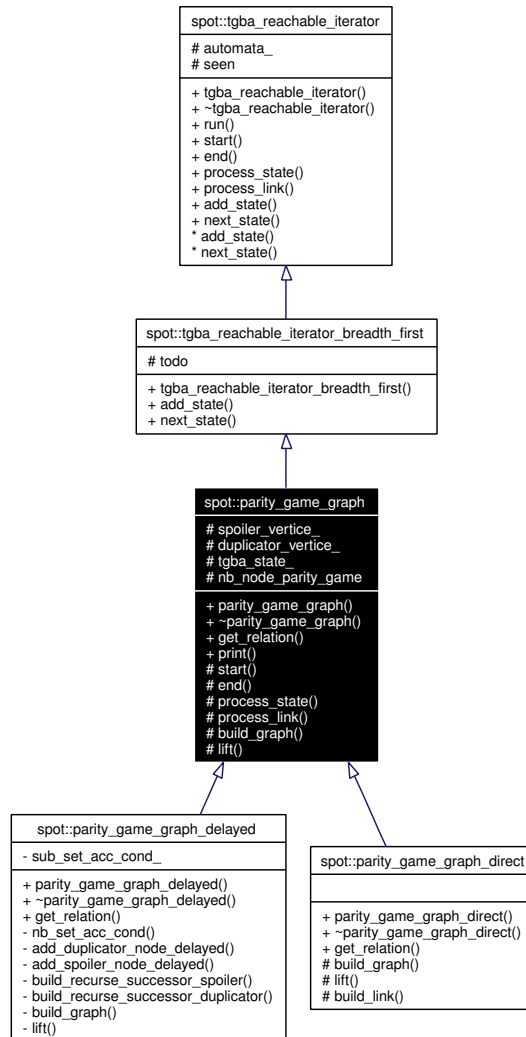
- [misc/optionmap.hh](#)

12.64 spot::parity_game_graph Class Reference

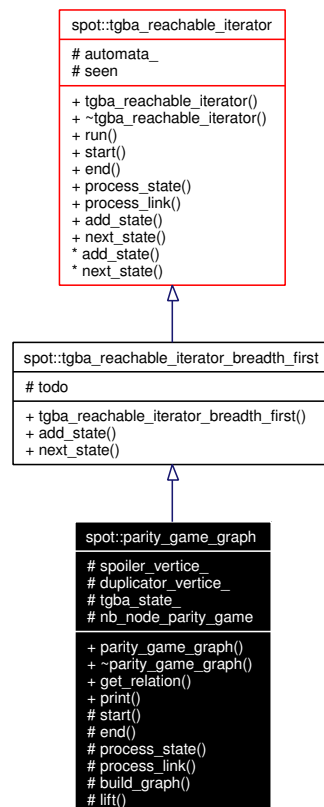
Parity game graph which compute a simulation relation.

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

Inheritance diagram for spot::parity_game_graph:



Collaboration diagram for spot::parity_game_graph:



Public Member Functions

- `parity_game_graph` (const `tgba` *a)
- virtual `~parity_game_graph` ()
- virtual `simulation_relation` * `get_relation` ()=0
- void `print` (std::ostream &os)
- virtual void `add_state` (const `state` *s)
- virtual const `state` * `next_state` ()
Called by `run()` to obtain the.
- void `run` ()
Iterate over all reachable states of a `spot::tgba`.
- virtual void `process_link` (const `state` *in_s, int in, const `state` *out_s, int out, const `tgba_succ_iterator` *si)

Protected Types

- typedef `Sgi::hash_map`< const `state` *, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

Protected Member Functions

- void `start` ()
Called by `run()` before starting its iteration.

- void `end` ()
Called by `run()` once all states have been explored.
- void `process_state` (const `state` *s, int n, `tgba_succ_iterator` *si)
- void `process_link` (int in, int out, const `tgba_succ_iterator` *si)
- virtual void `build_graph` ()=0
Compute each node of the graph.
- virtual void `lift` ()=0
Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

Protected Attributes

- `sn_v spoiler_vertice_`
- `dn_v duplicator_vertice_`
- `s_v tgba_state_`
- int `nb_node_parity_game`
- `std::deque< const state * > todo`
A queue of states yet to explore.
- const `tgba` * `automata_`
The `spot::tgba` to explore.
- `seen_map` `seen`
States already seen.

12.64.1 Detailed Description

Parity game graph which compute a simulation relation.

12.64.2 Member Typedef Documentation

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

Reimplemented in `spot::tgba_reduc`.

12.64.3 Constructor & Destructor Documentation

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

12.64.3.2 `virtual spot::parity_game_graph::~~parity_game_graph ()` [virtual]

12.64.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

12.64.4.2 `virtual void spot::parity_game_graph::build_graph ()` [protected, pure virtual]

Compute each node of the graph.

Implemented in [spot::parity_game_graph_direct](#), and [spot::parity_game_graph_delayed](#).

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

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

Reimplemented from [spot::tgba_reachable_iterator](#).

12.64.4.4 `virtual simulation_relation* spot::parity_game_graph::get_relation ()` [pure virtual]

Implemented in [spot::parity_game_graph_direct](#), and [spot::parity_game_graph_delayed](#).

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

Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

Implemented in [spot::parity_game_graph_direct](#), and [spot::parity_game_graph_delayed](#).

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

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

Implements [spot::tgba_reachable_iterator](#).

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

12.64.4.8 `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba_succ_iterator * si)` [virtual, inherited]

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

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The [spot::tgba_succ_iterator](#) positionned on the current transition.

The *in_s* and *out_s* states are owned by the [spot::tgba_reachable_iterator](#) instance and destroyed when the instance is destroyed.

12.64.4.9 void `spot::parity_game_graph::process_link` (int *in*, int *out*, const [tgba_succ_iterator](#) * *si*)
[protected]

12.64.4.10 void `spot::parity_game_graph::process_state` (const [state](#) * *s*, int *n*, [tgba_succ_iterator](#) * *si*) [protected, virtual]

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

Parameters:

- s* The current state.
- n* A unique number assigned to *s*.
- si* The [spot::tgba_succ_iterator](#) for *s*.

Reimplemented from [spot::tgba_reachable_iterator](#).

12.64.4.11 void `spot::tgba_reachable_iterator::run` () [inherited]

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

This is a template method that will call `add_state()`, `next_state()`, `start()`, `end()`, `process_state()`, and `process_link()`, while it iterate over state.

12.64.4.12 void `spot::parity_game_graph::start` () [protected, virtual]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

12.64.5 Member Data Documentation

12.64.5.1 const [tgba](#)* `spot::tgba_reachable_iterator::automata_` [protected, inherited]

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

12.64.5.2 [dn_v](#) `spot::parity_game_graph::duplicator_vertice_` [protected]

12.64.5.3 int `spot::parity_game_graph::nb_node_parity_game` [protected]

12.64.5.4 [seen_map](#) `spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.64.5.5 [sn_v](#) `spot::parity_game_graph::spoiler_vertice_` [protected]

12.64.5.6 [s_v](#) `spot::parity_game_graph::tgba_state_` [protected]

12.64.5.7 std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo

[protected, inherited]

A queue of states yet to explore.

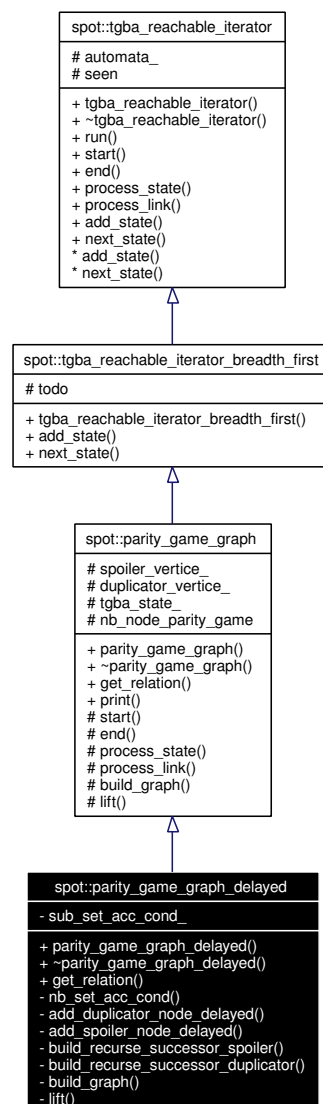
The documentation for this class was generated from the following file:

- tgbaalgos/reductgba_sim.hh

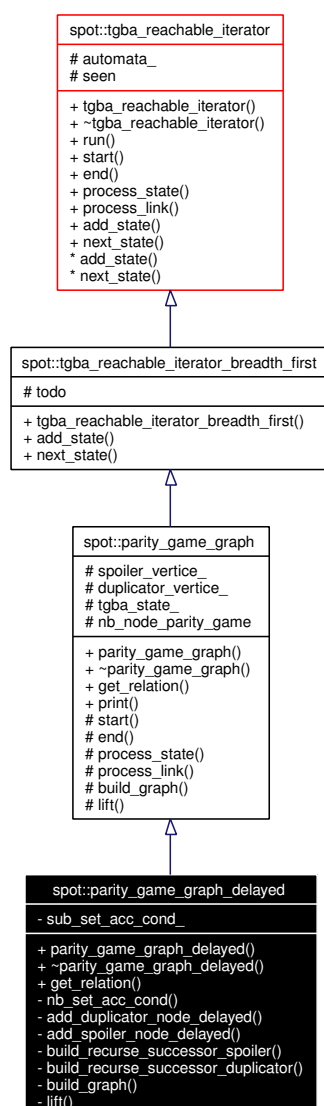
12.65 spot::parity_game_graph_delayed Class Reference

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

Inheritance diagram for spot::parity_game_graph_delayed:



Collaboration diagram for spot::parity_game_graph_delayed:



Public Member Functions

- [parity_game_graph_delayed](#) (const [tgba](#) *a)
- [~parity_game_graph_delayed](#) ()
- virtual [delayed_simulation_relation](#) * [get_relation](#) ()
- void [print](#) (std::ostream &os)
- virtual void [process_link](#) (const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si)
- virtual void [add_state](#) (const [state](#) *s)
- virtual const [state](#) * [next_state](#) ()

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

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

Protected Types

- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [seen_map](#)

Protected Member Functions

- void [start](#) ()
Called by [run\(\)](#) before starting its iteration.
- void [end](#) ()
Called by [run\(\)](#) once all states have been explored.
- void [process_state](#) (const [state](#) *s, int n, [tgba_succ_iterator](#) *si)
- void [process_link](#) (int in, int out, const [tgba_succ_iterator](#) *si)

Protected Attributes

- [sn_v](#) [spoiler_vertice_](#)
- [dn_v](#) [duplicator_vertice_](#)
- [s_v](#) [tgba_state_](#)
- int [nb_node_parity_game](#)
- std::deque< const [state](#) * > [todo](#)
A queue of states yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

Private Types

- typedef std::vector< bdd > [bdd_v](#)

Private Member Functions

- int [nb_set_acc_cond](#) ()
Return the number of acceptance condition.
- [duplicator_node_delayed](#) * [add_duplicator_node_delayed](#) (const [spot::state](#) *sn, const [spot::state](#) *dn, bdd acc, bdd label, int nb)
- [spoiler_node_delayed](#) * [add_spoiler_node_delayed](#) (const [spot::state](#) *sn, const [spot::state](#) *dn, bdd acc, int nb)
- void [build_recurse_successor_spoiler](#) ([spoiler_node](#) *sn, std::ostream &os)
- void [build_recurse_successor_duplicator](#) ([duplicator_node](#) *dn, [spoiler_node](#) *sn, std::ostream &os)
- virtual void [build_graph](#) ()
Compute the couple as for direct simulation.,.

- virtual void [lift](#) ()

The Jurdzinski's lifting algorithm.

Private Attributes

- [bdd_v sub_set_acc_cond_](#)

12.65.1 Detailed Description

Parity game graph which computes the delayed simulation relation as explained in

```
@InProceedings{etessami.01.alp,
  author = {Kousha Etessami and Thomas Wilke and Rebecca A. Schuller},
  title = {Fair Simulation Relations, Parity Games, and State Space
    Reduction for Buchi Automata},
  booktitle = {Proceedings of the 28th international colloquium on
    Automata, Languages and Programming},
  pages = {694--707},
  year = {2001},
  editor = {Fernando Orejas and Paul G. Spirakis and Jan van Leeuwen},
  volume = {2076},
  series = {Lecture Notes in Computer Science},
  address = {Crete, Greece},
  month = {July},
  publisher = {Springer-Verlag}
}
```

12.65.2 Member Typedef Documentation

12.65.2.1 typedef std::vector<bdd> [spot::parity_game_graph_delayed::bdd_v](#) [private]

Vector which contain all the sub-set of the set of acceptance condition.

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

Reimplemented in [spot::tgba_reduc](#).

12.65.3 Constructor & Destructor Documentation

12.65.3.1 [spot::parity_game_graph_delayed::parity_game_graph_delayed](#) (const [tgba](#) * *a*)

12.65.3.2 [spot::parity_game_graph_delayed::~~parity_game_graph_delayed](#) ()

12.65.4 Member Function Documentation

12.65.4.1 [duplicator_node_delayed*](#) [spot::parity_game_graph_delayed::add_duplicator_node_delayed](#) (const [spot::state](#) * *sn*, const [spot::state](#) * *dn*, bdd *acc*, bdd *label*, int *nb*) [private]

12.65.4.2 `spoiler_node_delayed*` `spot::parity_game_graph_delayed::add_spoiler_node_delayed` (const `spot::state` * *sn*, const `spot::state` * *dn*, bdd *acc*, int *nb*) [private]

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

Implements `spot::tgba_reachable_iterator`.

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

Compute the couple as for direct simulation,.

Implements `spot::parity_game_graph`.

12.65.4.5 `void` `spot::parity_game_graph_delayed::build_recurse_successor_duplicator` (`duplicator_node` * *dn*, `spoiler_node` * *sn*, `std::ostream` & *os*) [private]

12.65.4.6 `void` `spot::parity_game_graph_delayed::build_recurse_successor_spoiler` (`spoiler_node` * *sn*, `std::ostream` & *os*) [private]

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

Called by `run()` once all states have been explored.

Reimplemented from `spot::tgba_reachable_iterator`.

12.65.4.8 `virtual delayed_simulation_relation*` `spot::parity_game_graph_delayed::get_relation` () [virtual]

Implements `spot::parity_game_graph`.

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

The Jurdzinski's lifting algorithm.

Implements `spot::parity_game_graph`.

12.65.4.10 `int` `spot::parity_game_graph_delayed::nb_set_acc_cond` () [private]

Return the number of acceptance condition.

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

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

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

12.65.4.13 `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba_succ_iterator * si)` [virtual, inherited]

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

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The [spot::tgba_succ_iterator](#) positionned on the current transition.

The *in_s* and *out_s* states are owned by the [spot::tgba_reachable_iterator](#) instance and destroyed when the instance is destroyed.

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

12.65.4.15 `void spot::parity_game_graph::process_state (const state * s, int n, tgba_succ_iterator * si)` [protected, virtual, inherited]

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

Parameters:

s The current state.

n A unique number assigned to *s*.

si The [spot::tgba_succ_iterator](#) for *s*.

Reimplemented from [spot::tgba_reachable_iterator](#).

12.65.4.16 `void spot::tgba_reachable_iterator::run ()` [inherited]

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

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

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

Reimplemented from [spot::tgba_reachable_iterator](#).

12.65.5 Member Data Documentation

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

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

12.65.5.2 `dn_v spot::parity_game_graph::duplicator_vertice_` [protected, inherited]

12.65.5.3 `int spot::parity_game_graph::nb_node_parity_game` [protected, inherited]

12.65.5.4 `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.65.5.5 `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

12.65.5.6 `bdd_v spot::parity_game_graph_delayed::sub_set_acc_cond_` [private]

12.65.5.7 `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

12.65.5.8 `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo`
[protected, inherited]

A queue of states yet to explore.

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

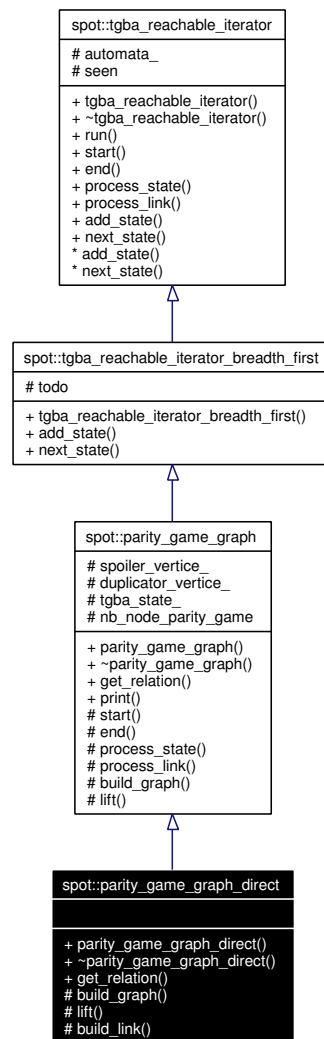
- `tgbaalgos/reductgba_sim.hh`

12.66 `spot::parity_game_graph_direct` Class Reference

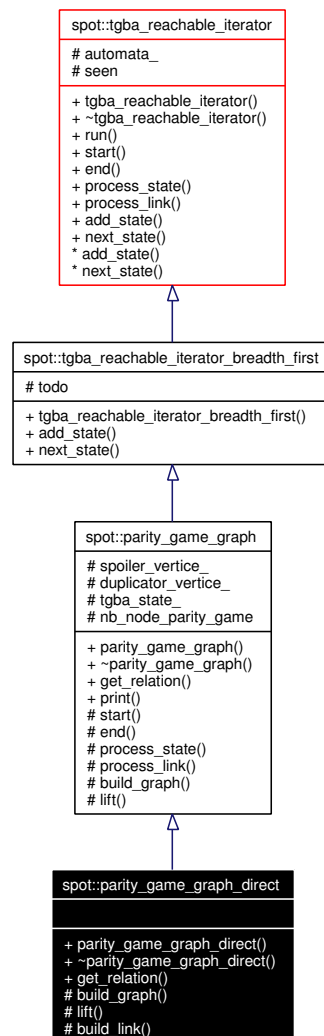
Parity game graph which compute the direct simulation relation.

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

Inheritance diagram for `spot::parity_game_graph_direct`:



Collaboration diagram for spot::parity_game_graph_direct:



Public Member Functions

- [parity_game_graph_direct](#) (const [tgba](#) *a)
- [~parity_game_graph_direct](#) ()
- virtual [direct_simulation_relation](#) * [get_relation](#) ()
- void [print](#) (std::ostream &os)
- virtual void [process_link](#) (const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si)
- virtual void [add_state](#) (const [state](#) *s)
- virtual const [state](#) * [next_state](#) ()

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

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

Protected Types

- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [seen_map](#)

Protected Member Functions

- virtual void [build_graph](#) ()
Compute each node of the graph.
- virtual void [lift](#) ()
Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.
- void [build_link](#) ()
- void [start](#) ()
Called by [run\(\)](#) before starting its iteration.
- void [end](#) ()
Called by [run\(\)](#) once all states have been explored.
- void [process_state](#) (const [state](#) *s, int n, [tgba_succ_iterator](#) *si)
- void [process_link](#) (int in, int out, const [tgba_succ_iterator](#) *si)

Protected Attributes

- [sn_v](#) [spoiler_vertice_](#)
- [dn_v](#) [duplicator_vertice_](#)
- [s_v](#) [tgba_state_](#)
- int [nb_node_parity_game](#)
- std::deque< const [state](#) * > [todo](#)
A queue of states yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

12.66.1 Detailed Description

Parity game graph which compute the direct simulation relation.

12.66.2 Member Typedef Documentation

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

Reimplemented in [spot::tgba_reduc](#).

12.66.3 Constructor & Destructor Documentation

12.66.3.1 spot::parity_game_graph_direct::parity_game_graph_direct (const [tgba](#) * *a*)

12.66.3.2 spot::parity_game_graph_direct::~~parity_game_graph_direct ()

12.66.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

12.66.4.2 virtual void spot::parity_game_graph_direct::build_graph () [protected, virtual]

Compute each node of the graph.

Implements [spot::parity_game_graph](#).

12.66.4.3 void spot::parity_game_graph_direct::build_link () [protected]

12.66.4.4 void spot::parity_game_graph::end () [protected, virtual, inherited]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

12.66.4.5 virtual [direct_simulation_relation](#)* spot::parity_game_graph_direct::get_relation ()
[virtual]

Implements [spot::parity_game_graph](#).

12.66.4.6 virtual void spot::parity_game_graph_direct::lift () [protected, virtual]

Remove edge from spoiler to duplicator that make duplicator loose. Spoiler node whose still have some link, reveal a direct simulation relation.

Implements [spot::parity_game_graph](#).

12.66.4.7 virtual const [state](#)* spot::tgba_reachable_iterator_breadth_first::next_state ()
[virtual, inherited]

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

Implements [spot::tgba_reachable_iterator](#).

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

12.66.4.9 virtual void spot::tgba_reachable_iterator::process_link (const [state](#) * *in_s*, int *in*, const [state](#) * *out_s*, int *out*, const [tgba_succ_iterator](#) * *si*) [virtual, inherited]

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

Parameters:

- in_s* The source state
- in* The source state number.
- out_s* The destination state
- out* The destination state number.
- si* The [spot::tgba_succ_iterator](#) positionned on the current transition.

The *in_s* and *out_s* states are owned by the [spot::tgba_reachable_iterator](#) instance and destroyed when the instance is destroyed.

12.66.4.10 void spot::parity_game_graph::process_link (int *in*, int *out*, const [tgba_succ_iterator](#) * *si*) [protected, inherited]

12.66.4.11 void spot::parity_game_graph::process_state (const [state](#) * *s*, int *n*, [tgba_succ_iterator](#) * *si*) [protected, virtual, inherited]

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

Parameters:

- s* The current state.
- n* A unique number assigned to *s*.
- si* The [spot::tgba_succ_iterator](#) for *s*.

Reimplemented from [spot::tgba_reachable_iterator](#).

12.66.4.12 void spot::tgba_reachable_iterator::run () [inherited]

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

12.66.4.13 void spot::parity_game_graph::start () [protected, virtual, inherited]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

12.66.5 Member Data Documentation

12.66.5.1 const [tgba](#)* spot::tgba_reachable_iterator::automata_ [protected, inherited]

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

12.66.5.2 [dn_v](#) spot::parity_game_graph::duplicator_vertice_ [protected, inherited]

12.66.5.3 int spot::parity_game_graph::nb_node_parity_game [protected, inherited]

12.66.5.4 `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.66.5.5 `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

12.66.5.6 `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

12.66.5.7 `std::deque<const state*>` `spot::tgba_reachable_iterator_breadth_first::todo`
[protected, inherited]

A queue of states yet to explore.

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

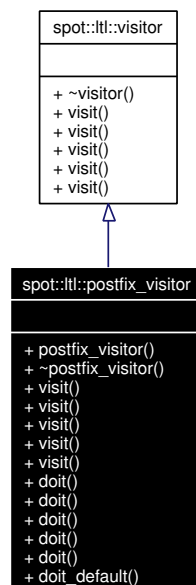
- [tgbaalgorithms/reductgba_sim.hh](#)

12.67 spot::ltl::postfix_visitor Class Reference

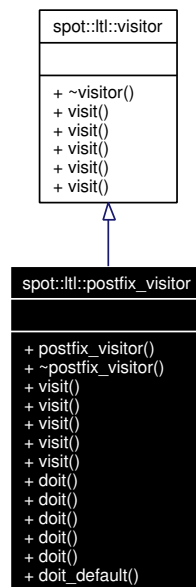
Apply an algorithm on each node of an AST, during a postfix traversal.

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

Inheritance diagram for `spot::ltl::postfix_visitor`:



Collaboration diagram for spot::ltl::postfix_visitor:



Public Member Functions

- [postfix_visitor \(\)](#)
- virtual [~postfix_visitor \(\)](#)
- void [visit \(atomic_prop *ap\)](#)
- void [visit \(unop *uo\)](#)
- void [visit \(binop *bo\)](#)
- void [visit \(multop *mo\)](#)
- void [visit \(constant *c\)](#)
- virtual void [doit \(atomic_prop *ap\)](#)
- virtual void [doit \(unop *uo\)](#)
- virtual void [doit \(binop *bo\)](#)
- virtual void [doit \(multop *mo\)](#)
- virtual void [doit \(constant *c\)](#)
- virtual void [doit_default \(formula *f\)](#)

12.67.1 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.

12.67.2 Constructor & Destructor Documentation

12.67.2.1 spot::ltl::postfix_visitor::postfix_visitor ()

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

12.67.3 Member Function Documentation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

12.67.3.11 void spot::ltl::postfix_visitor::visit (**atomic_prop** * *ap*) [virtual]

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

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

- [ltlvisit/postfix.hh](#)

12.68 spot::ptr_hash< T > Struct Template Reference

A hash function for pointers.

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

Public Member Functions

- [size_t operator\(\)](#) (const T *p) const

12.68.1 Detailed Description

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

A hash function for pointers.

12.68.2 Member Function Documentation

12.68.2.1 `template<class T> size_t spot::ptr_hash< T >::operator() (const T * p) const`
`[inline]`

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

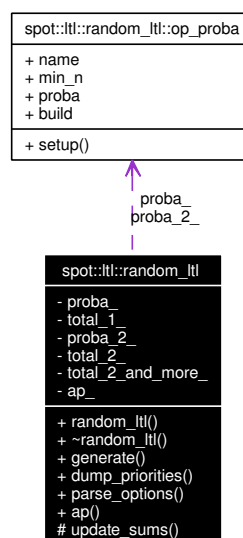
- misc/[hash.hh](#)

12.69 spot::ltl::random_ltl Class Reference

Generate random LTL formulae.

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

Collaboration diagram for spot::ltl::random_ltl:



Public Member Functions

- `random_ltl` (const `atomic_prop_set` *ap)
Create a random LTL generator using atomic propositions from ap.
- `~random_ltl` ()
- `formula * generate` (int n) const
Generate a formula of size n.
- `std::ostream & dump_priorities` (std::ostream &os) const
Print the priorities of each operator, constants, and atomic propositions.
- `const char * parse_options` (char *options)
Update the priorities used to generate LTL formulae.
- `const atomic_prop_set * ap` () const
Return the set of atomic proposition used to build formulae.

Protected Member Functions

- void [update_sums](#) ()

Private Attributes

- [op_proba](#) * [proba_](#)
- double [total_1_](#)
- [op_proba](#) * [proba_2_](#)
- double [total_2_](#)
- double [total_2_and_more_](#)
- const [atomic_prop_set](#) * [ap_](#)

Classes

- struct [op_proba](#)

12.69.1 Detailed Description

Generate random LTL formulae.

This class recursively construct LTL formulae of a given size. The formulae will use the use atomic propositions from the set of proposition passed to the constructor, in addition to the constant and all LTL operators supported by Spot.

By default each operator has equal chance to be selected. Also, each atomic proposition has as much chance as each constant (i.e., true and false) to be picked. This can be tuned using [parse_options\(\)](#).

12.69.2 Constructor & Destructor Documentation**12.69.2.1 spot::ltl::random_ltl::random_ltl (const [atomic_prop_set](#) * *ap*)**

Create a random LTL generator using atomic propositions from *ap*.

12.69.2.2 spot::ltl::random_ltl::~~[random_ltl](#) ()**12.69.3 Member Function Documentation****12.69.3.1 const [atomic_prop_set](#)* spot::ltl::random_ltl::ap () const** [inline]

Return the set of atomic proposition used to build formulae.

12.69.3.2 std::ostream& spot::ltl::random_ltl::dump_priorities (std::ostream & *os*) const

Print the priorities of each operator, constants, and atomic propositions.

12.69.3.3 [formula](#)* spot::ltl::random_ltl::generate (int *n*) const

Generate a formula of size *n*.

It is possible to obtain formulae that are smaller than *n*, because some simple simplifications are performed by the AST. (For instance the formula $a \mid a$ is automatically reduced to a by [spot::ltl::multop](#).)

Furthermore, for the purpose of this generator, `a | b | c` has length 5, while it has length 4 for `spot::ltl::length()`.

12.69.3.4 `const char* spot::ltl::random_ltl::parse_options (char * options)`

Update the priorities used to generate LTL formulae.

The initial priorities are as follows.

<code>ap</code>	<code>n</code>
<code>false</code>	<code>1</code>
<code>true</code>	<code>1</code>
<code>not</code>	<code>1</code>
<code>F</code>	<code>1</code>
<code>G</code>	<code>1</code>
<code>X</code>	<code>1</code>
<code>equiv</code>	<code>1</code>
<code>implies</code>	<code>1</code>
<code>xor</code>	<code>1</code>
<code>R</code>	<code>1</code>
<code>U</code>	<code>1</code>
<code>and</code>	<code>1</code>
<code>or</code>	<code>1</code>

Where `n` is the number of atomic propositions in the set passed to the constructor.

This means that each operator has equal chance to be selected. Also, each atomic proposition has as much chance as each constant (i.e., true and false) to be picked. This can be

These priorities can be altered using this function. *options* should be comma-separated list of KEY=VALUE assignments, using keys from the above list. For instance "`xor=0, F=3`" will prevent `xor` from being used, and will raise the relative probability of occurrences of the `F` operator.

12.69.3.5 `void spot::ltl::random_ltl::update_sums ()` [protected]

12.69.4 Member Data Documentation

12.69.4.1 `const atomic_prop_set* spot::ltl::random_ltl::ap_` [private]

12.69.4.2 `op_proba* spot::ltl::random_ltl::proba_` [private]

12.69.4.3 `op_proba* spot::ltl::random_ltl::proba_2_` [private]

12.69.4.4 `double spot::ltl::random_ltl::total_1_` [private]

12.69.4.5 `double spot::ltl::random_ltl::total_2_` [private]

12.69.4.6 `double spot::ltl::random_ltl::total_2_and_more_` [private]

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

- `ltlvisit/randomltl.hh`

12.70 `spot::ltl::random_ltl::op_proba` Struct Reference

Public Types

- typedef `formula` `*(* builder)(const random_ltl *rl, int n)`

Public Member Functions

- void `setup` (`const char *name`, `int min_n`, `builder build`)

Public Attributes

- `const char * name`
- `int min_n`
- `double proba`
- `builder build`

12.70.1 Member Typedef Documentation

12.70.1.1 typedef `formula``*(* spot::ltl::random_ltl::op_proba::builder)(const random_ltl *rl, int n)`

12.70.2 Member Function Documentation

12.70.2.1 void `spot::ltl::random_ltl::op_proba::setup` (`const char * name`, `int min_n`, `builder build`)

12.70.3 Member Data Documentation

12.70.3.1 `builder spot::ltl::random_ltl::op_proba::build`

12.70.3.2 int `spot::ltl::random_ltl::op_proba::min_n`

12.70.3.3 `const char* spot::ltl::random_ltl::op_proba::name`

12.70.3.4 double `spot::ltl::random_ltl::op_proba::proba`

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

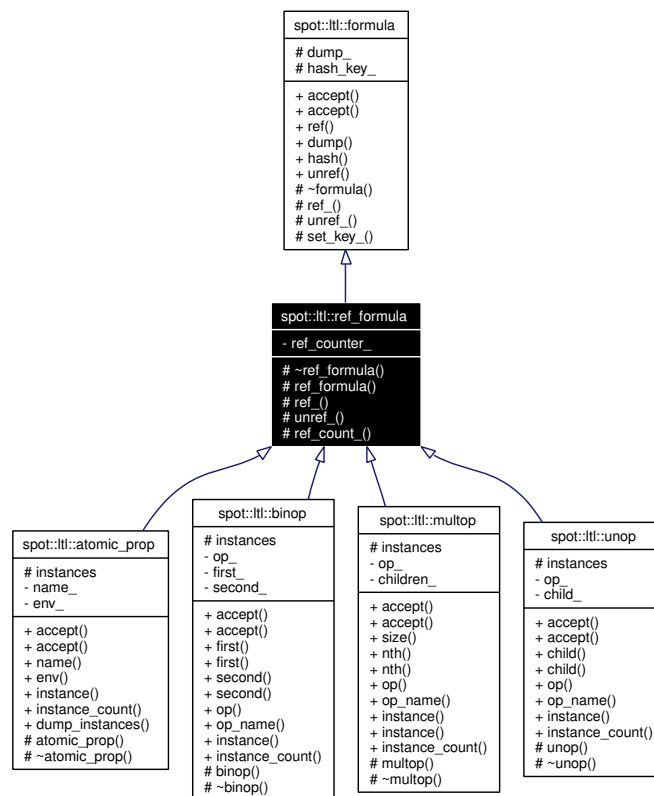
- `ltlvisit/randomltl.hh`

12.71 `spot::ltl::ref_formula` Class Reference

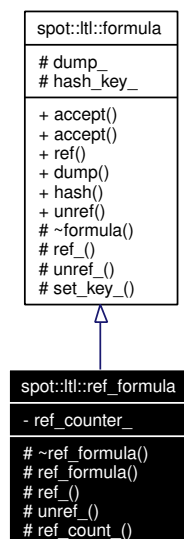
A reference-counted LTL formula.

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

Inheritance diagram for `spot::ltl::ref_formula`:



Collaboration diagram for `spot::ltl::ref_formula`:



Public Member Functions

- virtual void `accept` (`visitor` &`v`)=0

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

- virtual void `accept` (`const_visitor` &`v`) const =0
Entry point for vspot::ltl::const_visitor instances.
- `formula` * `ref` ()
clone this node
- const std::string & `dump` () const
Return a canonic representation of the formula.
- const size_t `hash` () const
Return a hash_key for the formula.

Static Public Member Functions

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

Protected Member Functions

- virtual `~ref_formula` ()
- `ref_formula` ()
- void `ref_` ()
increment reference counter if any
- bool `unref_` ()
decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).
- unsigned `ref_count_` ()
Number of references to this formula.
- void `set_key_` ()
Compute key_ from dump_.

Protected Attributes

- std::string `dump_`
The canonic representation of the formula.
- size_t `hash_key_`
The hash key of this formula.

Private Attributes

- unsigned `ref_counter_`

12.71.1 Detailed Description

A reference-counted LTL formula.

12.71.2 Constructor & Destructor Documentation

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

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

12.71.3 Member Function Documentation

12.71.3.1 `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`.

12.71.3.2 `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`.

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

Return a canonic representation of the formula.

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

Return a hash_key for the formula.

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

clone this node

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

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

increment reference counter if any

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

12.71.3.7 `unsigned spot::ltl::ref_formula::ref_count_ ()` [protected]

Number of references to this formula.

12.71.3.8 void spot::ltl::formula::set_key_() [protected, inherited]

Compute key_ from dump_.

Should be called once in each object, after dump_ has been set.

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

release this node

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

12.71.3.10 bool spot::ltl::ref_formula::unref_() [protected, virtual]

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

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

12.71.4 Member Data Documentation**12.71.4.1 std::string spot::ltl::formula::dump_** [protected, inherited]

The canonic representation of the formula.

12.71.4.2 size_t spot::ltl::formula::hash_key_ [protected, inherited]

The hash key of this formula.

Initialized by [set_key_\(\)](#).

12.71.4.3 unsigned spot::ltl::ref_formula::ref_counter_ [private]

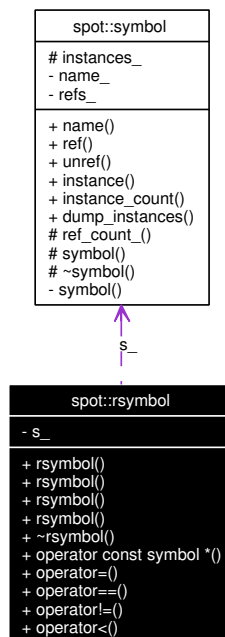
The documentation for this class was generated from the following file:

- [ltlast/refformula.hh](#)

12.72 spot::rsymbol Class Reference

```
#include <evtgba/symbol.hh>
```

Collaboration diagram for spot::rsymbol:



Public Member Functions

- `rsymbol` (const `symbol` *s)
- `rsymbol` (const std::string &s)
- `rsymbol` (const char *s)
- `rsymbol` (const `rsymbol` &rs)
- `~rsymbol` ()
- `operator const symbol * () const`
- `const rsymbol & operator=` (const `rsymbol` &rs)
- `bool operator==` (const `rsymbol` &rs) const
- `bool operator!=` (const `rsymbol` &rs) const
- `bool operator<` (const `rsymbol` &rs) const

Private Attributes

- const `symbol` * `s_`

12.72.1 Constructor & Destructor Documentation

12.72.1.1 `spot::rsymbol::rsymbol` (const `symbol` *s) [inline]

12.72.1.2 `spot::rsymbol::rsymbol` (const std::string &s) [inline]

12.72.1.3 `spot::rsymbol::rsymbol` (const char *s) [inline]

12.72.1.4 `spot::rsymbol::rsymbol` (const `rsymbol` &rs) [inline]

12.72.1.5 spot::rsymbol::~~rsymbol () [inline]

12.72.2 Member Function Documentation

12.72.2.1 spot::rsymbol::operator const rsymbol * () const [inline]

12.72.2.2 bool spot::rsymbol::operator!= (const rsymbol & rs) const [inline]

12.72.2.3 bool spot::rsymbol::operator< (const rsymbol & rs) const [inline]

12.72.2.4 const rsymbol& spot::rsymbol::operator= (const rsymbol & rs) [inline]

12.72.2.5 bool spot::rsymbol::operator== (const rsymbol & rs) const [inline]

12.72.3 Member Data Documentation

12.72.3.1 const rsymbol* spot::rsymbol::s_ [private]

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

- evtgba/symbol.hh

12.73 spot::scc_stack Class Reference

```
#include <tgbaalgorithms/gtec/sccstack.hh>
```

Public Types

- typedef std::list< connected_component > stack_type

Public Member Functions

- void push (int index)
Stack a new SCC with index index.
- connected_component & top ()
Access the top SCC.
- const connected_component & top () const
Access the top SCC.
- void pop ()
Pop the top SCC.
- size_t size () const
How many SCC are in stack.

- `std::list< const state * > & rem ()`

The `rem` member of the top SCC.

- `unsigned clear_rem \(\)`
- `bool empty \(\) const`

Is the stack empty?

Public Attributes

- `stack_type s`

Classes

- `struct connected_component`

12.73.1 Member Typedef Documentation

12.73.1.1 `typedef std::list<connected_component> spot::scc_stack::stack_type`

12.73.2 Member Function Documentation

12.73.2.1 `unsigned spot::scc_stack::clear_rem \(\)`

Purge all `rem` members.

Returns:

the number of elements cleared.

12.73.2.2 `bool spot::scc_stack::empty \(\) const`

Is the stack empty?

12.73.2.3 `void spot::scc_stack::pop \(\)`

Pop the top SCC.

12.73.2.4 `void spot::scc_stack::push \(int index\)`

Stack a new SCC with index *index*.

12.73.2.5 `std::list<const state*>& spot::scc_stack::rem \(\)`

The `rem` member of the top SCC.

12.73.2.6 `size_t spot::scc_stack::size \(\) const`

How many SCC are in stack.

12.73.2.7 `const connected_component& spot::scc_stack::top () const`

Access the top SCC.

12.73.2.8 `connected_component& spot::scc_stack::top ()`

Access the top SCC.

12.73.3 Member Data Documentation**12.73.3.1** `stack_type spot::scc_stack::s`

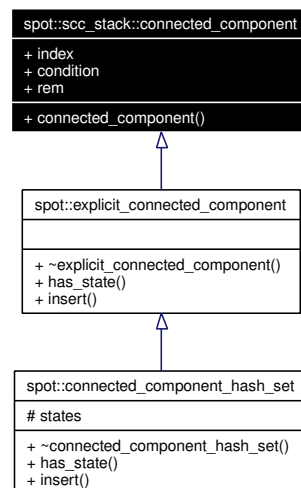
The documentation for this class was generated from the following file:

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

12.74 spot::scc_stack::connected_component Struct Reference

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

Inheritance diagram for spot::scc_stack::connected_component:

**Public Member Functions**

- `connected_component` (int `index`=-1)

Public Attributes

- int `index`
Index of the SCC.
- bdd `condition`
- `std::list< const state * >` `rem`

12.74.1 Constructor & Destructor Documentation

12.74.1.1 `spot::scc_stack::connected_component::connected_component (int index = -1)`

12.74.2 Member Data Documentation

12.74.2.1 `bdd spot::scc_stack::connected_component::condition`

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

12.74.2.2 `int spot::scc_stack::connected_component::index`

Index of the SCC.

12.74.2.3 `std::list<const state*> spot::scc_stack::connected_component::rem`

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

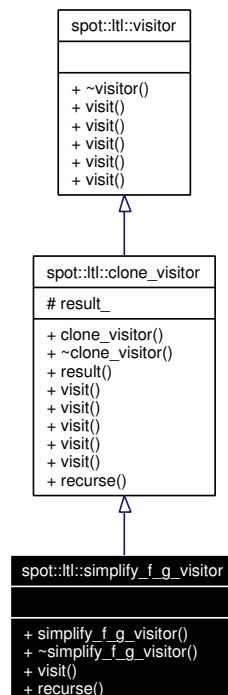
- [tgbaalgos/gtec/sccstack.hh](#)

12.75 spot::ltl::simplify_f_g_visitor Class Reference

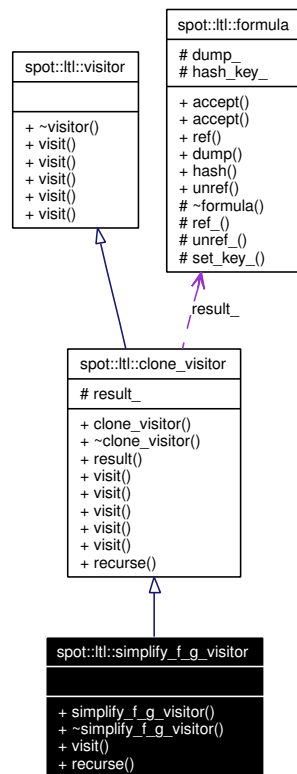
Replace `true U f` and `false R g` by `F f` and `G g`.

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

Inheritance diagram for `spot::ltl::simplify_f_g_visitor`:



Collaboration diagram for spot::ltl::simplify_f_g_visitor:



Public Member Functions

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

Protected Attributes

- [formula *result_](#)

Private Types

- [typedef clone_visitor super](#)

12.75.1 Detailed Description

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

12.75.2 Member Typedef Documentation

12.75.2.1 `typedef clone_visitor spot::ltl::simplify_f_g_visitor::super [private]`

12.75.3 Constructor & Destructor Documentation

12.75.3.1 `spot::ltl::simplify_f_g_visitor::simplify_f_g_visitor ()`

12.75.3.2 `virtual spot::ltl::simplify_f_g_visitor::~~simplify_f_g_visitor () [virtual]`

12.75.4 Member Function Documentation

12.75.4.1 `virtual formula* spot::ltl::simplify_f_g_visitor::recurse (formula * f) [virtual]`

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

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

12.75.4.3 `void spot::ltl::clone_visitor::visit (constant * c) [virtual, inherited]`

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

12.75.4.4 `void spot::ltl::clone_visitor::visit (multop * mo) [virtual, inherited]`

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

12.75.4.5 `void spot::ltl::clone_visitor::visit (unop * uo) [virtual, inherited]`

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

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

12.75.4.6 `void spot::ltl::clone_visitor::visit (atomic_prop * ap) [virtual, inherited]`

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

12.75.4.7 `void spot::ltl::simplify_f_g_visitor::visit (binop * bo) [virtual]`

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

12.75.5 Member Data Documentation

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

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

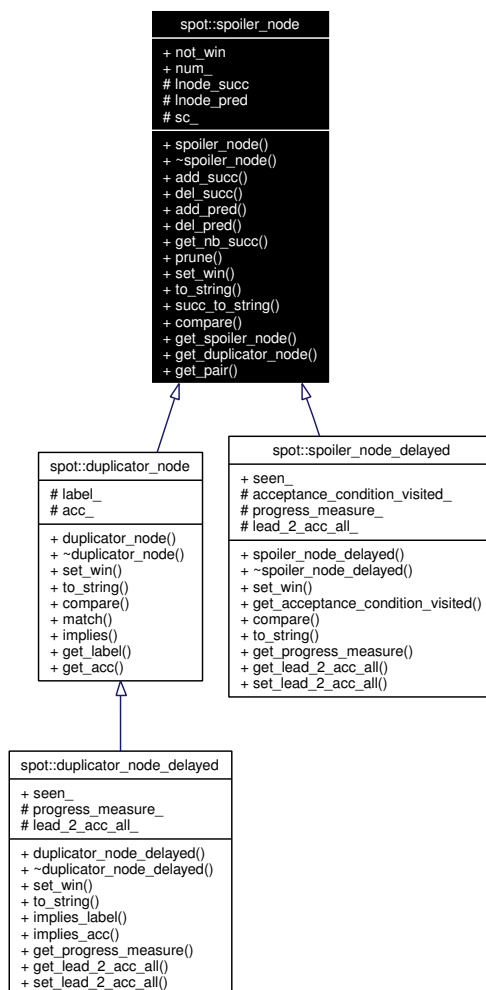
- [ltlvisit/simpfg.hh](#)

12.76 spot::spoiler_node Class Reference

Spoiler node of parity game graph.

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

Inheritance diagram for spot::spoiler_node:



Public Member Functions

- `spoiler_node` (const `state` *d_node, const `state` *s_node, int num)
- virtual `~spoiler_node` ()
- bool `add_succ` (spoiler_node *n)

Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.

- void `del_succ` (spoiler_node *n)
- virtual void `add_pred` (spoiler_node *n)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()

- virtual bool `set_win ()`
- virtual std::string `to_string (const tgba *a)`
- virtual std::string `succ_to_string ()`
- virtual bool `compare (spoiler_node *n)`
- const `state * get_spoiler_node ()`
- const `state * get_duplicator_node ()`
- `state_couple * get_pair ()`

Public Attributes

- bool `not_win`
- int `num_`

Protected Attributes

- `sn_v * lnode_succ`
- `sn_v * lnode_pred`
- `state_couple * sc_`

12.76.1 Detailed Description

Spoiler node of parity game graph.

12.76.2 Constructor & Destructor Documentation

12.76.2.1 `spot::spoiler_node::spoiler_node (const state * d_node, const state * s_node, int num)`

12.76.2.2 `virtual spot::spoiler_node::~spoiler_node ()` [virtual]

12.76.3 Member Function Documentation

12.76.3.1 `virtual void spot::spoiler_node::add_pred (spoiler_node * n)` [virtual]

12.76.3.2 `bool spot::spoiler_node::add_succ (spoiler_node * n)`

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

12.76.3.3 `virtual bool spot::spoiler_node::compare (spoiler_node * n)` [virtual]

Reimplemented in `spot::duplicator_node`, and `spot::spoiler_node_delayed`.

12.76.3.4 `virtual void spot::spoiler_node::del_pred ()` [virtual]

12.76.3.5 `void spot::spoiler_node::del_succ (spoiler_node * n)`

12.76.3.6 `const state* spot::spoiler_node::get_duplicator_node ()`

12.76.3.7 `int spot::spoiler_node::get_nb_succ ()`

12.76.3.8 `state_couple* spot::spoiler_node::get_pair ()`

12.76.3.9 `const state* spot::spoiler_node::get_spoiler_node ()`

12.76.3.10 `bool spot::spoiler_node::prune ()`

12.76.3.11 `virtual bool spot::spoiler_node::set_win ()` [virtual]

Reimplemented in `spot::duplicator_node`, `spot::spoiler_node_delayed`, and `spot::duplicator_node_delayed`.

12.76.3.12 `virtual std::string spot::spoiler_node::succ_to_string ()` [virtual]

12.76.3.13 `virtual std::string spot::spoiler_node::to_string (const tgba * a)` [virtual]

Reimplemented in `spot::duplicator_node`, `spot::spoiler_node_delayed`, and `spot::duplicator_node_delayed`.

12.76.4 Member Data Documentation

12.76.4.1 `sn_v* spot::spoiler_node::lnode_pred` [protected]

12.76.4.2 `sn_v* spot::spoiler_node::lnode_succ` [protected]

12.76.4.3 `bool spot::spoiler_node::not_win`

12.76.4.4 `int spot::spoiler_node::num_`

12.76.4.5 `state_couple* spot::spoiler_node::sc_` [protected]

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

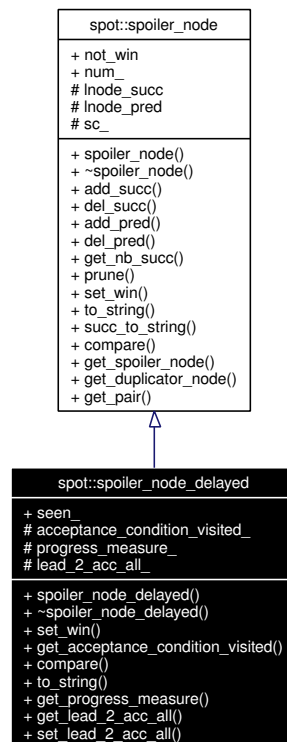
- `tgbaalgos/reductgba_sim.hh`

12.77 `spot::spoiler_node_delayed` Class Reference

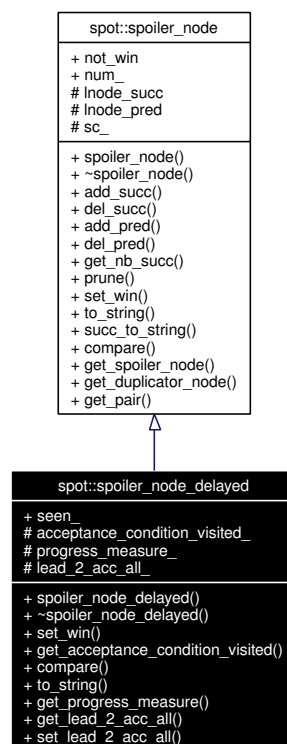
Spoiler node of parity game graph for delayed simulation.

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

Inheritance diagram for `spot::spoiler_node_delayed`:



Collaboration diagram for `spot::spoiler_node_delayed`:



Public Member Functions

- `spoiler_node_delayed` (const `state` *`d_node`, const `state` *`s_node`, bdd `a`, int `num`)
- `~spoiler_node_delayed` ()
- bool `set_win` ()

Return true if the progress_measure has changed.

- bdd `get_acceptance_condition_visited` () const
- virtual bool `compare` (`spoiler_node` *`n`)
- virtual std::string `to_string` (const `tgba` *`a`)
- int `get_progress_measure` () const
- bool `get_lead_2_acc_all` ()
- bool `set_lead_2_acc_all` (bdd `acc`=bddfalse)
- bool `add_succ` (`spoiler_node` *`n`)

Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.

- void `del_succ` (`spoiler_node` *`n`)
- virtual void `add_pred` (`spoiler_node` *`n`)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const `state` * `get_spoiler_node` ()
- const `state` * `get_duplicator_node` ()
- `state_couple` * `get_pair` ()

Public Attributes

- bool `seen_`
- bool `not_win`
- int `num_`

Protected Attributes

- bdd `acceptance_condition_visited_`
- int `progress_measure_`
- bool `lead_2_acc_all_`
- `sn_v` * `lnode_succ`
- `sn_v` * `lnode_pred`
- `state_couple` * `sc_`

12.77.1 Detailed Description

Spoiler node of parity game graph for delayed simulation.

12.77.2 Constructor & Destructor Documentation

12.77.2.1 `spot::spoiler_node_delayed::spoiler_node_delayed` (const `state` * `d_node`, const `state` * `s_node`, bdd `a`, int `num`)

12.77.2.2 `spot::spoiler_node_delayed::~spoiler_node_delayed ()`

12.77.3 Member Function Documentation

12.77.3.1 `virtual void spot::spoiler_node::add_pred (spoiler_node * n) [virtual, inherited]`

12.77.3.2 `bool spot::spoiler_node::add_succ (spoiler_node * n) [inherited]`

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

12.77.3.3 `virtual bool spot::spoiler_node_delayed::compare (spoiler_node * n) [virtual]`

Reimplemented from `spot::spoiler_node`.

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

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

12.77.3.6 `bdd spot::spoiler_node_delayed::get_acceptance_condition_visited () const`

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

12.77.3.8 `bool spot::spoiler_node_delayed::get_lead_2_acc_all ()`

12.77.3.9 `int spot::spoiler_node::get_nb_succ () [inherited]`

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

12.77.3.11 `int spot::spoiler_node_delayed::get_progress_measure () const`

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

12.77.3.13 `bool spot::spoiler_node::prune () [inherited]`

12.77.3.14 `bool spot::spoiler_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

12.77.3.15 `bool spot::spoiler_node_delayed::set_win () [virtual]`

Return true if the progress_measure has changed.

Reimplemented from `spot::spoiler_node`.

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

12.77.3.17 virtual std::string spot::spoiler_node_delayed::to_string (const [tgba](#) * a) [virtual]

Reimplemented from [spot::spoiler_node](#).

12.77.4 Member Data Documentation

12.77.4.1 bdd [spot::spoiler_node_delayed::acceptance_condition_visited_](#) [protected]

a Bdd for retain all the acceptance condition that a node has visited.

12.77.4.2 bool [spot::spoiler_node_delayed::lead_2_acc_all_](#) [protected]

12.77.4.3 [sn_v*](#) [spot::spoiler_node::lnode_pred](#) [protected, inherited]

12.77.4.4 [sn_v*](#) [spot::spoiler_node::lnode_succ](#) [protected, inherited]

12.77.4.5 bool [spot::spoiler_node::not_win](#) [inherited]

12.77.4.6 int [spot::spoiler_node::num_](#) [inherited]

12.77.4.7 int [spot::spoiler_node_delayed::progress_measure_](#) [protected]

12.77.4.8 [state_couple*](#) [spot::spoiler_node::sc_](#) [protected, inherited]

12.77.4.9 bool [spot::spoiler_node_delayed::seen_](#)

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

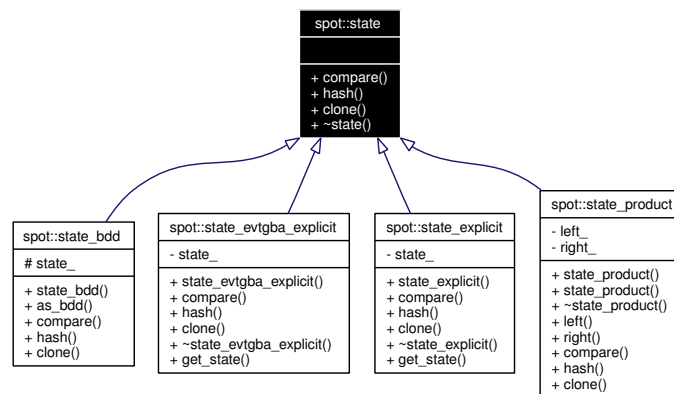
- [tgbaalgorithms/reductgba_sim.hh](#)

12.78 spot::state Class Reference

Abstract class for states.

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

Inheritance diagram for spot::state:



Public Member Functions

- virtual int [compare](#) (const [state](#) *other) const =0
Compares two states (that come from the same automaton).
- virtual size_t [hash](#) () const =0
Hash a state.
- virtual [state](#) * [clone](#) () const =0
Duplicate a state.
- virtual [~state](#) ()

12.78.1 Detailed Description

Abstract class for states.

12.78.2 Constructor & Destructor Documentation

12.78.2.1 virtual [spot::state::~state](#) () [inline, virtual]

12.78.3 Member Function Documentation

12.78.3.1 virtual [state](#)* [spot::state::clone](#) () const [pure virtual]

Duplicate a state.

Implemented in [spot::state_evtgba_explicit](#), [spot::state_bdd](#), [spot::state_explicit](#), and [spot::state_product](#).

12.78.3.2 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](#).

12.78.3.3 virtual size_t spot::state::hash () const [pure virtual]

Hash a state.

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

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

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

Implemented in [spot::state_evtgba_explicit](#), [spot::state_bdd](#), [spot::state_explicit](#), and [spot::state_product](#).

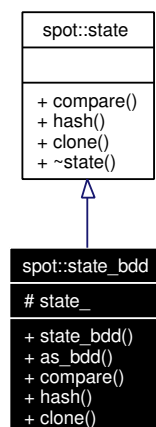
The documentation for this class was generated from the following file:

- [tgba/state.hh](#)

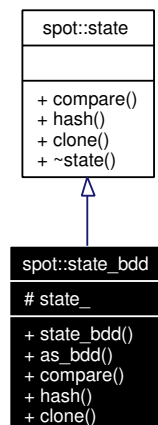
12.79 spot::state_bdd Class Reference

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

Inheritance diagram for `spot::state_bdd`:



Collaboration diagram for `spot::state_bdd`:



Public Member Functions

- `state_bdd` (bdd s)
- virtual bdd `as_bdd` () const
Return the BDD part of the state.
- virtual int `compare` (const `state` *other) const
Compares two states (that come from the same automaton).
- virtual size_t `hash` () const
Hash a state.
- virtual `state_bdd` * `clone` () const
Duplicate a state.

Protected Attributes

- bdd `state_`
BDD representation of the state.

12.79.1 Detailed Description

A state whose representation is a BDD.

12.79.2 Constructor & Destructor Documentation

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

12.79.3 Member Function Documentation

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

Return the BDD part of the state.

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

Duplicate a state.

Implements [spot::state](#).

12.79.3.3 `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](#).

12.79.3.4 `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](#).

12.79.4 Member Data Documentation**12.79.4.1** `bdd spot::state_bdd::state_` `[protected]`

BDD representation of the state.

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

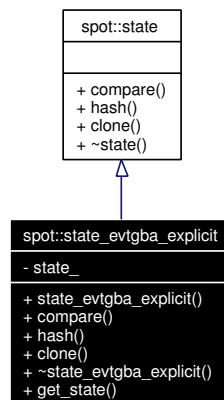
- [tgba/statebdd.hh](#)

12.80 `spot::state_evtgba_explicit` Class Reference

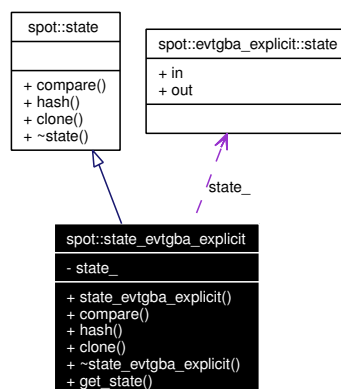
States used by `spot::tgba_evtgba_explicit`.

```
#include <evtgba/explicit.hh>
```

Inheritance diagram for `spot::state_evtgba_explicit`:



Collaboration diagram for `spot::state_evtgba_explicit`:



Public Member Functions

- `state_evtgba_explicit` (const `evtgba_explicit::state` *s)
- virtual int `compare` (const `spot::state` *other) const
- virtual size_t `hash` () const
Hash a state.
- virtual `state_evtgba_explicit` * `clone` () const
Duplicate a state.
- virtual `~state_evtgba_explicit` ()
- const `evtgba_explicit::state` * `get_state` () const
- virtual int `compare` (const `state` *other) const =0
Compares two states (that come from the same automaton).

Private Attributes

- const `evtgba_explicit::state` * `state_`

12.80.1 Detailed Description

States used by `spot::tgba_evtgba_explicit`.

12.80.2 Constructor & Destructor Documentation

12.80.2.1 `spot::state_evtgba_explicit::state_evtgba_explicit (const evtgba_explicit::state * s) [inline]`

12.80.2.2 `virtual spot::state_evtgba_explicit::~state_evtgba_explicit () [inline, virtual]`

12.80.3 Member Function Documentation

12.80.3.1 `virtual state_evtgba_explicit* spot::state_evtgba_explicit::clone () const [virtual]`

Duplicate a state.

Implements [spot::state](#).

12.80.3.2 `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](#).

12.80.3.3 `virtual int spot::state_evtgba_explicit::compare (const spot::state * other) const [virtual]`

12.80.3.4 `const evtgba_explicit::state* spot::state_evtgba_explicit::get_state () const`

12.80.3.5 `virtual size_t spot::state_evtgba_explicit::hash () const [virtual]`

Hash a state.

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

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

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build

the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

12.80.4 Member Data Documentation

12.80.4.1 `const evtgba_explicit::state* spot::state_evtgba_explicit::state_` [private]

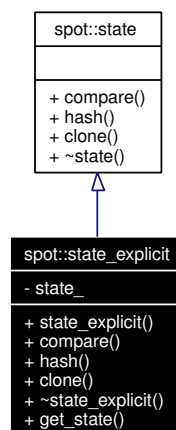
The documentation for this class was generated from the following file:

- [evtgba/explicit.hh](#)

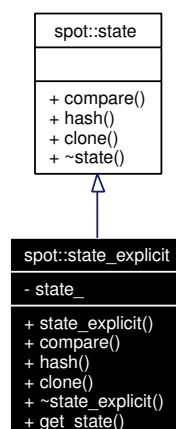
12.81 spot::state_explicit Class Reference

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

Inheritance diagram for `spot::state_explicit`:



Collaboration diagram for `spot::state_explicit`:



Public Member Functions

- [state_explicit](#) (const [tgba_explicit::state](#) *s)
- virtual int [compare](#) (const [spot::state](#) *other) const
- virtual size_t [hash](#) () const
Hash a state.
- virtual [state_explicit](#) * [clone](#) () const
Duplicate a state.
- virtual [~state_explicit](#) ()
- const [tgba_explicit::state](#) * [get_state](#) () const
- virtual int [compare](#) (const [state](#) *other) const =0
Compares two states (that come from the same automaton).

Private Attributes

- const [tgba_explicit::state](#) * [state_](#)

12.81.1 Detailed Description

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

12.81.2 Constructor & Destructor Documentation

12.81.2.1 [spot::state_explicit::state_explicit](#) (const [tgba_explicit::state](#) *s) [inline]

12.81.2.2 virtual [spot::state_explicit::~~state_explicit](#) () [inline, virtual]

12.81.3 Member Function Documentation

12.81.3.1 virtual [state_explicit](#)* [spot::state_explicit::clone](#) () const [virtual]

Duplicate a state.

Implements [spot::state](#).

12.81.3.2 virtual int [spot::state::compare](#) (const [state](#) * other) const [pure virtual, inherited]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also:

[spot::state_ptr_less_than](#)

Implemented in [spot::state_bdd](#), and [spot::state_product](#).

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

12.81.3.4 const tgba_explicit::state* spot::state_explicit::get_state () const

12.81.3.5 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`.

12.81.4 Member Data Documentation

12.81.4.1 const tgba_explicit::state* spot::state_explicit::state_ [private]

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

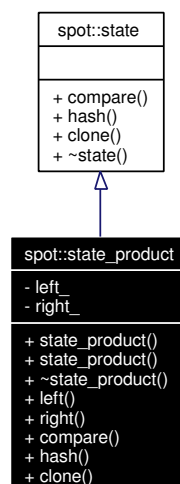
- `tgba/tgbaexplicit.hh`

12.82 spot::state_product Class Reference

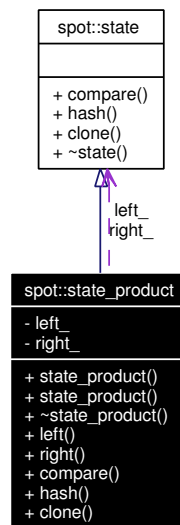
A state for `spot::tgba_product`.

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

Inheritance diagram for `spot::state_product`:



Collaboration diagram for spot::state_product:



Public Member Functions

- [state_product](#) ([state](#) *left, [state](#) *right)
Constructor.
- [state_product](#) (const [state_product](#) &o)
Copy constructor.
- virtual [~state_product](#) ()
- [state](#) * [left](#) () const
- [state](#) * [right](#) () const
- virtual int [compare](#) (const [state](#) *other) const
Compares two states (that come from the same automaton).
- virtual size_t [hash](#) () const
Hash a state.
- virtual [state_product](#) * [clone](#) () const
Duplicate a state.

Private Attributes

- [state](#) * [left_](#)
State from the left automaton.
- [state](#) * [right_](#)
State from the right automaton.

12.82.1 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.

12.82.2 Constructor & Destructor Documentation

12.82.2.1 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.

12.82.2.2 spot::state_product::state_product (const state_product & o)

Copy constructor.

12.82.2.3 virtual spot::state_product::~~state_product () [virtual]

12.82.3 Member Function Documentation

12.82.3.1 virtual state_product* spot::state_product::clone () const [virtual]

Duplicate a state.

Implements [spot::state](#).

12.82.3.2 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](#).

12.82.3.3 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](#).

12.82.3.4 [state*](#) `spot::state_product::left () const` `[inline]`

12.82.3.5 [state*](#) `spot::state_product::right () const` `[inline]`

12.82.4 Member Data Documentation

12.82.4.1 [state*](#) `spot::state_product::left_` `[private]`

State from the left automaton.

12.82.4.2 [state*](#) `spot::state_product::right_` `[private]`

State from the right automaton.

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

- [tgba/tgbaproduct.hh](#)

12.83 spot::state_ptr_equal Struct Reference

An Equivalence Relation for `state*`.

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

Public Member Functions

- `bool operator() (const state *left, const state *right) const`

12.83.1 Detailed Description

An Equivalence Relation for `state*`.

This is meant to be used as a comparison functor for `Sgi hash_map` whose key are of type `state*`.

For instance here is how one could declare a map of `state*`.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::state*, int, spot::state_ptr_hash,
              spot::state_ptr_equal> seen;
```

12.83.2 Member Function Documentation

12.83.2.1 `bool spot::state_ptr_equal::operator() (const state * left, const state * right) const` `[inline]`

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

- [tgba/state.hh](#)

12.84 spot::state_ptr_hash Struct Reference

Hash Function for state*.

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

Public Member Functions

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

12.84.1 Detailed Description

Hash Function for state*.

This is meant to be used as a hash functor for Sgi's hash_map whose key are of type state*.

For instance here is how one could declare a map of state*.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::state*, int, spot::state_ptr_hash,
              spot::state_ptr_equal> seen;
```

12.84.2 Member Function Documentation

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

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

- [tgba/state.hh](#)

12.85 spot::state_ptr_less_than Struct Reference

Strict Weak Ordering for state*.

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

Public Member Functions

- `bool operator() (const state *left, const state *right) const`

12.85.1 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;
```

12.85.2 Member Function Documentation

12.85.2.1 `bool spot::state_ptr_less_than::operator() (const state * left, const state * right) const` `[inline]`

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

- [tgba/state.hh](#)

12.86 spot::string_hash Struct Reference

A hash function for strings.

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

Public Member Functions

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

12.86.1 Detailed Description

A hash function for strings.

12.86.2 Member Function Documentation

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

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

- [misc/hash.hh](#)

12.87 spot::symbol Class Reference

```
#include <evtgba/symbol.hh>
```

Public Member Functions

- `const std::string & name () const`
- `void ref () const`
- `void unref () const`

Static Public Member Functions

- `static const symbol * instance (const std::string &name)`
- `static unsigned instance_count ()`
Number of instantiated atomic propositions. For debugging.
- `static std::ostream & dump_instances (std::ostream &os)`
List all instances of atomic propositions. For debugging.

Protected Types

- typedef std::map< const std::string, const [symbol](#) * > [map](#)

Protected Member Functions

- int [ref_count_](#) () const
- [symbol](#) (const std::string *name)
- [~symbol](#) ()

Static Protected Attributes

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

Private Member Functions

- [symbol](#) (const [symbol](#) &)

Private Attributes

- const std::string * [name_](#)
Undefined.
- int [refs_](#)

12.87.1 Member Typedef Documentation

12.87.1.1 typedef std::map<const std::string, const [symbol](#)*> [spot::symbol::map](#) [protected]

12.87.2 Constructor & Destructor Documentation

12.87.2.1 [spot::symbol::symbol](#) (const std::string * *name*) [protected]

12.87.2.2 [spot::symbol::~~symbol](#) () [protected]

12.87.2.3 [spot::symbol::symbol](#) (const [symbol](#) &) [private]

12.87.3 Member Function Documentation

12.87.3.1 static std::ostream& [spot::symbol::dump_instances](#) (std::ostream & *os*) [static]

List all instances of atomic propositions. For debugging.

12.87.3.2 static const [symbol](#)* [spot::symbol::instance](#) (const std::string & *name*) [static]

12.87.3.3 static unsigned spot::symbol::instance_count () [static]

Number of instantiated atomic propositions. For debugging.

12.87.3.4 const std::string& spot::symbol::name () const

12.87.3.5 void spot::symbol::ref () const

12.87.3.6 int spot::symbol::ref_count_ () const [protected]

12.87.3.7 void spot::symbol::unref () const

12.87.4 Member Data Documentation

12.87.4.1 map spot::symbol::instances_ [static, protected]

12.87.4.2 const std::string* spot::symbol::name_ [private]

Undefined.

12.87.4.3 int spot::symbol::refs_ [mutable, private]

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

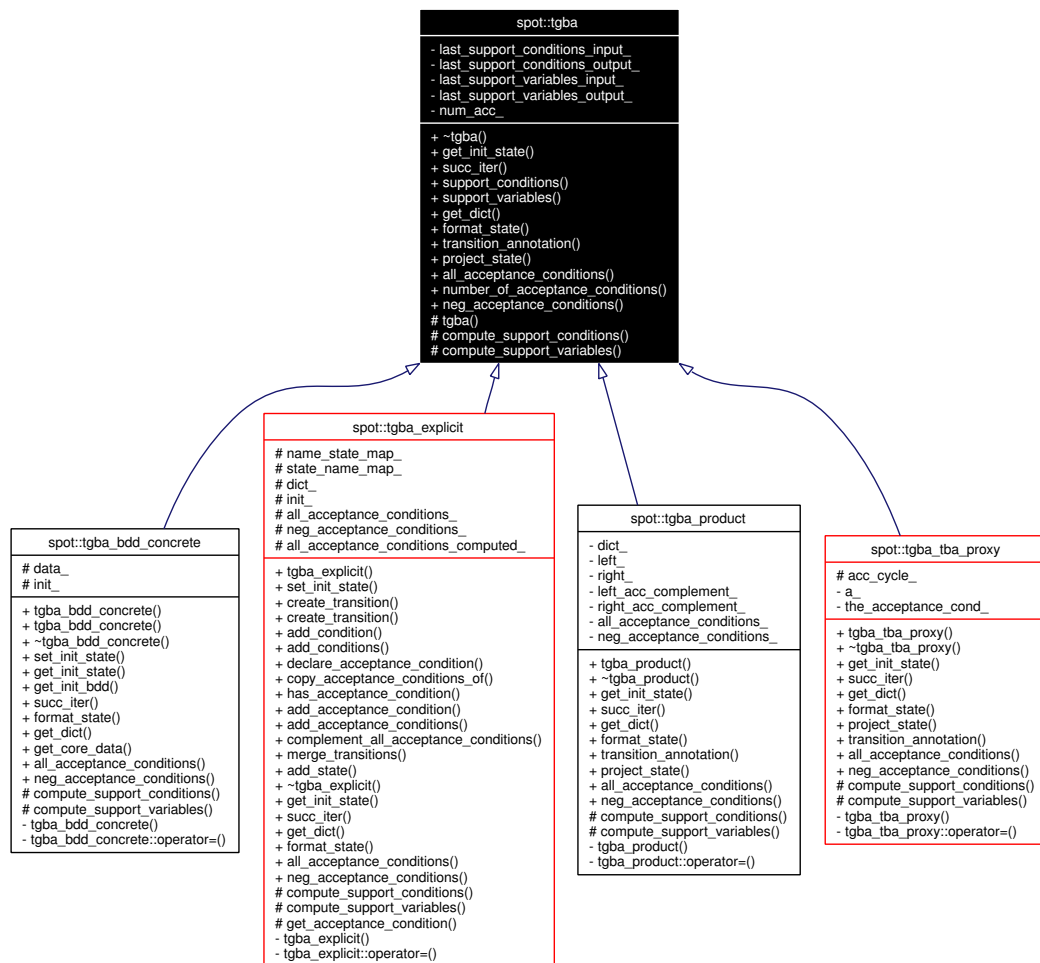
- evtgba/[symbol.hh](#)

12.88 spot::tgba Class Reference

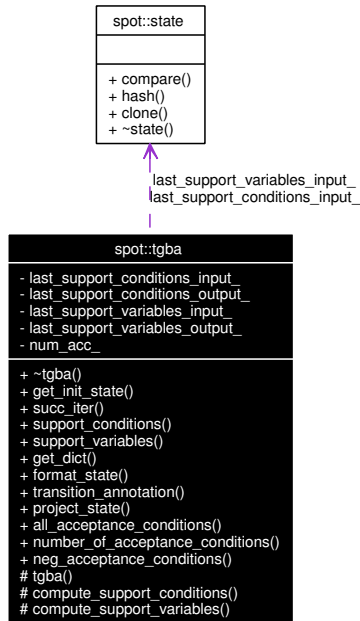
A Transition-based Generalized Büchi Automaton.

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

Inheritance diagram for spot::tgba:



Collaboration diagram for spot::tgba:



Public Member Functions

- virtual `~tgba()`
- virtual `state * get_init_state()` const =0
Get the initial state of the automaton.
- virtual `tgba_succ_iterator * succ_iter` (const `state *local_state`, const `state *global_state=0`, const `tgba *global_automaton=0`) const =0
Get an iterator over the successors of local_state.
- bdd `support_conditions` (const `state *state`) const
Get a formula that must hold whatever successor is taken.
- bdd `support_variables` (const `state *state`) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual bdd `dict * get_dict()` const =0
Get the dictionary associated to the automaton.
- virtual std::string `format_state` (const `state *state`) const =0
Format the state as a string for printing.
- virtual std::string `transition_annotation` (const `tgba_succ_iterator *t`) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual `state * project_state` (const `state *s`, const `tgba *t`) const
Project a state on an automaton.
- virtual bdd `all_acceptance_conditions` () const =0
Return the set of all acceptance conditions used by this automaton.

- virtual unsigned int `number_of_acceptance_conditions` () const
The number of acceptance conditions.
- virtual bdd `neg_acceptance_conditions` () const =0
Return the conjunction of all negated acceptance variables.

Protected Member Functions

- `tgba` ()
- virtual bdd `compute_support_conditions` (const `state` *state) const =0
Do the actual computation of `tgba::support_conditions()`.
- virtual bdd `compute_support_variables` (const `state` *state) const =0
Do the actual computation of `tgba::support_variables()`.

Private Attributes

- const `state` * `last_support_conditions_input_`
- bdd `last_support_conditions_output_`
- const `state` * `last_support_variables_input_`
- bdd `last_support_variables_output_`
- int `num_acc_`

12.88.1 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.

12.88.2 Constructor & Destructor Documentation

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

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

12.88.3 Member Function Documentation

12.88.3.1 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](#).

12.88.3.2 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](#).

12.88.3.3 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](#).

12.88.3.4 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](#).

12.88.3.5 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](#).

12.88.3.6 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](#).

12.88.3.7 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](#).

12.88.3.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual]`

The number of acceptance conditions.

12.88.3.9 `virtual state* spot::tgba::project_state (const state *s, const tgba *t) const [virtual]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

Returns:

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

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

12.88.3.10 `virtual tgba_succ_iterator* spot::tgba::succ_iter (const state *local_state, const state *global_state = 0, const tgba *global_automaton = 0) const [pure virtual]`

Get an iterator over the successors of *local_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global_automaton* designate the root `spot::tgba`, and *global_state* its state. This two objects can be used by [succ_iter\(\)](#) to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, *global_state* is not adopted by `succ_iter`.

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

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

12.88.3.11 bdd spot::tgba::support_conditions (const state * state) const

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

12.88.3.12 bdd spot::tgba::support_variables (const state * state) const

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

12.88.3.13 virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters:

t a non-done `tgba_succ_iterator` for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

12.88.4 Member Data Documentation

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

12.88.4.2 `bdd spot::tgba::last_support_conditions_output_` [mutable, private]

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

12.88.4.4 `bdd spot::tgba::last_support_variables_output_` [mutable, private]

12.88.4.5 `int spot::tgba::num_acc_` [mutable, private]

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

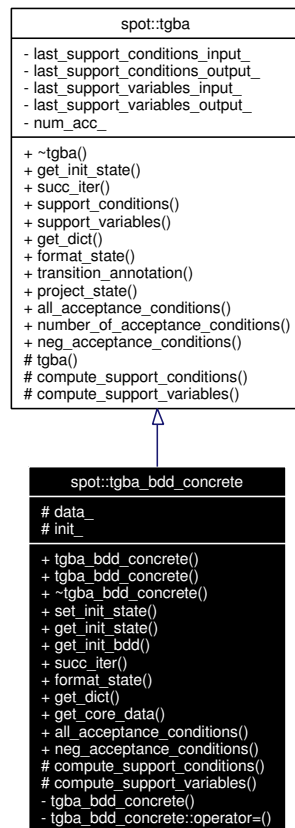
- [tgba/tgba.hh](#)

12.89 spot::tgba_bdd_concrete Class Reference

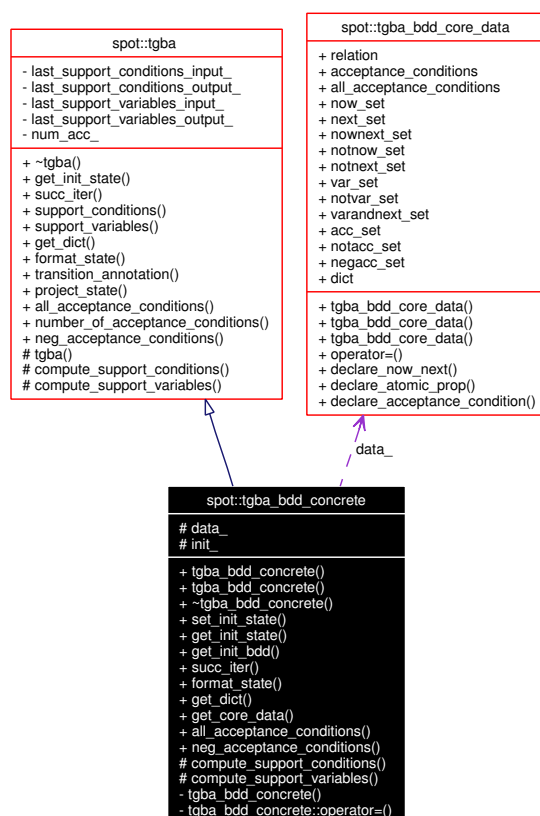
A concrete [spot::tgba](#) implemented using BDDs.

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

Inheritance diagram for spot::tgba_bdd_concrete:



Collaboration diagram for spot::tgba_bdd_concrete:



Public Member Functions

- `tgba_bdd_concrete` (const `tgba_bdd_factory` &fact)
Construct a `tgba_bdd_concrete` with unknown initial state.
- `tgba_bdd_concrete` (const `tgba_bdd_factory` &fact, bdd init)
Construct a `tgba_bdd_concrete` with known initial state.
- virtual `~tgba_bdd_concrete` ()
- virtual void `set_init_state` (bdd s)
Set the initial state.
- virtual `state_bdd` * `get_init_state` () const
Get the initial state of the automaton.
- bdd `get_init_bdd` () const
Get the initial state directly as a BDD.
- virtual `tgba_succ_iterator_concrete` * `succ_iter` (const `state` *local_state, const `state` *global_state=0, const `tgba` *global_automaton=0) const
Get an iterator over the successors of local_state.
- virtual std::string `format_state` (const `state` *state) const
Format the state as a string for printing.

- virtual `bdd_dict * get_dict ()` const
Get the dictionary associated to the automaton.
- const `tgba_bdd_core_data & get_core_data ()` const
Get the core data associated to this automaton.
- virtual `bdd all_acceptance_conditions ()` const
Return the set of all acceptance conditions used by this automaton.
- virtual `bdd neg_acceptance_conditions ()` const
Return the conjunction of all negated acceptance variables.
- `bdd support_conditions (const state *state)` const
Get a formula that must hold whatever successor is taken.
- `bdd support_variables (const state *state)` const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual `std::string transition_annotation (const tgba_succ_iterator *t)` const
Return a possible annotation for the transition pointed to by the iterator.
- virtual `state * project_state (const state *s, const tgba *t)` const
Project a state on an automaton.
- virtual `unsigned int number_of_acceptance_conditions ()` const
The number of acceptance conditions.

Protected Member Functions

- virtual `bdd compute_support_conditions (const state *state)` const
Do the actual computation of `tgba::support_conditions()`.
- virtual `bdd compute_support_variables (const state *state)` const
Do the actual computation of `tgba::support_variables()`.

Protected Attributes

- `tgba_bdd_core_data data_`
Core data associated to the automaton.
- `bdd init_`
Initial state.

Private Member Functions

- [tgba_bdd_concrete](#) (const [tgba_bdd_concrete](#) &)
- [tgba_bdd_concrete](#) & [tgba_bdd_concrete::operator=](#) (const [tgba_bdd_concrete](#) &)

12.89.1 Detailed Description

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

12.89.2 Constructor & Destructor Documentation

12.89.2.1 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.

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

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

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

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

12.89.3 Member Function Documentation

12.89.3.1 virtual bdd spot::tgba_bdd_concrete::all_acceptance_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

12.89.3.2 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](#).

12.89.3.3 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](#).

12.89.3.4 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](#).

12.89.3.5 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.

12.89.3.6 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](#).

12.89.3.7 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;
```

12.89.3.8 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](#).

12.89.3.9 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](#).

12.89.3.10 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const` [virtual, inherited]

The number of acceptance conditions.

12.89.3.11 `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const` [virtual, inherited]

Project a state on an automaton.

This converts *s*, into that corresponding `spot::state` for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata 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`.

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

Set the initial state.

12.89.3.13 `virtual tgba_succ_iterator_concrete* spot::tgba_bdd_concrete::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const` [virtual]

Get an iterator over the successors of *local_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global_automaton* designate the root `spot::tgba`, and *global_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, *global_state* is not adopted by `succ_iter`.

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

Implements `spot::tgba`.

12.89.3.14 `bdd spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

12.89.3.15 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

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

12.89.3.17 `virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const` [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters:

t a non-done [tgba_succ_iterator](#) for this automata

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

12.89.4 Member Data Documentation

12.89.4.1 `tgba_bdd_core_data spot::tgba_bdd_concrete::data_ [protected]`

Core data associated to the automaton.

12.89.4.2 `bdd spot::tgba_bdd_concrete::init_ [protected]`

Initial state.

The documentation for this class was generated from the following file:

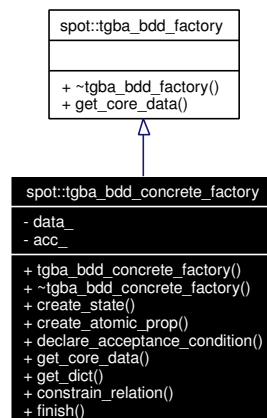
- [tgba/tgababddconcrete.hh](#)

12.90 `spot::tgba_bdd_concrete_factory` Class Reference

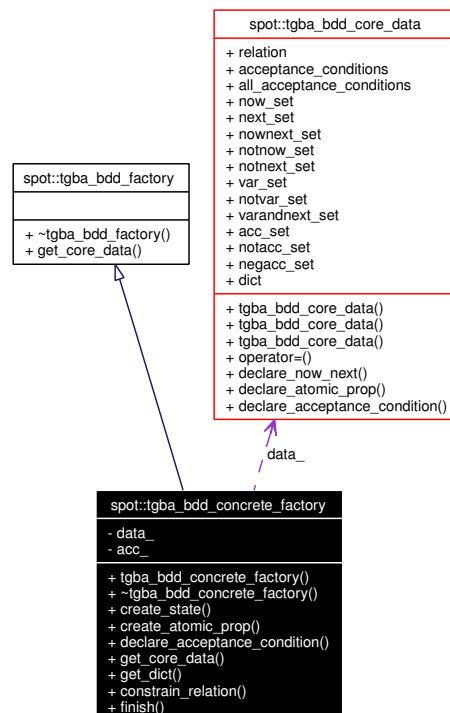
Helper class to build a [spot::tgba_bdd_concrete](#) object.

```
#include <tgba/tgababddconcretefactory.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, ltl::formula_ptr_hash > acc_map_`

Private Attributes

- `tgba_bdd_core_data data_`
Core data for the new automata.
- `acc_map_ acc_`
BDD associated to each acceptance condition.

12.90.1 Detailed Description

Helper class to build a `spot::tgba_bdd_concrete` object.

12.90.2 Member Typedef Documentation

12.90.2.1 `typedef Sgi::hash_map<const ltl::formula*, bdd, ltl::formula_ptr_hash> spot::tgba_bdd_concrete_factory::acc_map_ [private]`

12.90.3 Constructor & Destructor Documentation

12.90.3.1 `spot::tgba_bdd_concrete_factory::tgba_bdd_concrete_factory (bdd_dict * dict)`

12.90.3.2 `virtual spot::tgba_bdd_concrete_factory::~~tgba_bdd_concrete_factory () [virtual]`

12.90.4 Member Function Documentation

12.90.4.1 `void spot::tgba_bdd_concrete_factory::constrain_relation (bdd new_rel)`

Add a new constraint to the relation.

12.90.4.2 `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()`.

12.90.4.3 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()`.

12.90.4.4 void spot::tgba_bdd_concrete_factory::declare_acceptance_condition (bdd b, const ltl::formula *a)

Declare an acceptance condition.

Formula such as ' $f \text{ U } g$ ' or ' $F g$ ' make the promise that ' g ' will be fulfilled eventually. So once one of this formula has been translated into a BDD, we use `declare_acceptance_condition()` to associate all other states to the acceptance set of ' g '.

Parameters:

b a BDD indicating which variables are in the acceptance set

a the formula associated

12.90.4.5 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()`.

12.90.4.6 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`.

12.90.4.7 bdd_dict* spot::tgba_bdd_concrete_factory::get_dict () const**12.90.5 Member Data Documentation****12.90.5.1 acc_map spot::tgba_bdd_concrete_factory::acc_ [private]**

BDD associated to each acceptance condition.

12.90.5.2 tgba_bdd_core_data spot::tgba_bdd_concrete_factory::data_ [private]

Core data for the new automata.

The documentation for this class was generated from the following file:

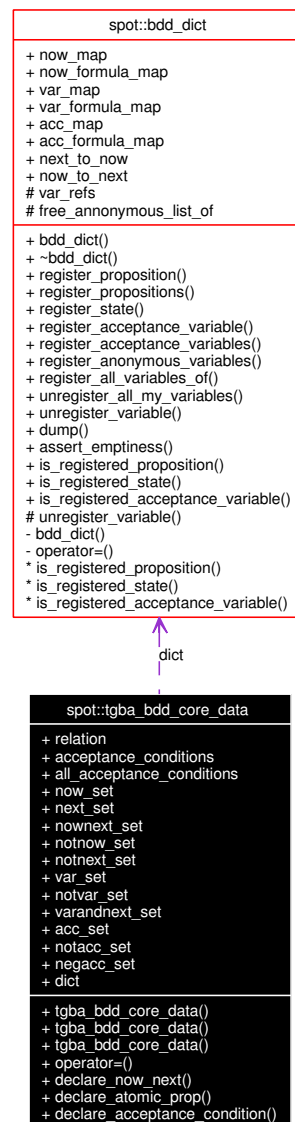
- [tgba/tgbabddconcretefactory.hh](#)

12.91 spot::tgba_bdd_core_data Struct Reference

Core data for a TGBA encoded using BDDs.

```
#include <tgba/tgbabddcoredata.hh>
```

Collaboration diagram for spot::tgba_bdd_core_data:



Public Member Functions

- [tgba_bdd_core_data](#) ([bdd_dict](#) *dict)
Default constructor.
- [tgba_bdd_core_data](#) (const [tgba_bdd_core_data](#) ©)
Copy 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.

12.91.1 Detailed Description

Core data for a TGBA encoded using BDDs.

12.91.2 Constructor & Destructor Documentation

12.91.2.1 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.

12.91.2.2 spot::tgba_bdd_core_data::tgba_bdd_core_data (const tgba_bdd_core_data & copy)

Copy constructor.

12.91.2.3 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.

12.91.3 Member Function Documentation

12.91.3.1 void spot::tgba_bdd_core_data::declare_acceptance_condition (bdd prom)

Update the variable sets to take a new acceptance condition into account.

12.91.3.2 void spot::tgba_bdd_core_data::declare_atomic_prop (bdd var)

Update the variable sets to take a new automic proposition into account.

12.91.3.3 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.

12.91.3.4 const tgba_bdd_core_data& spot::tgba_bdd_core_data::operator= (const tgba_bdd_core_data & *copy*)**12.91.4 Member Data Documentation****12.91.4.1 bdd spot::tgba_bdd_core_data::acc_set**

The (positive) conjunction of all variables which are acceptance conditions.

12.91.4.2 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 $(Acc[a] \& !Acc[b] \& !Acc[c]) \mid (!Acc[a] \& Acc[b] \& !Acc[c])$.

Acceptance 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.

12.91.4.3 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 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.

12.91.4.4 bdd_dict* spot::tgba_bdd_core_data::dict

The dictionary used by the automata.

12.91.4.5 bdd spot::tgba_bdd_core_data::negacc_set

The negative conjunction of all variables which are acceptance conditions.

12.91.4.6 `bdd spot::tgba_bdd_core_data::next_set`

The conjunction of all Next variables, in their positive form.

12.91.4.7 `bdd spot::tgba_bdd_core_data::notacc_set`

The (positive) conjunction of all variables which are not acceptance conditions.

12.91.4.8 `bdd spot::tgba_bdd_core_data::notnext_set`

The (positive) conjunction of all variables which are not Next variables.

12.91.4.9 `bdd spot::tgba_bdd_core_data::notnow_set`

The (positive) conjunction of all variables which are not Now variables.

12.91.4.10 `bdd spot::tgba_bdd_core_data::notvar_set`

The (positive) conjunction of all variables which are not atomic propositions.

12.91.4.11 `bdd spot::tgba_bdd_core_data::now_set`

The conjunction of all Now variables, in their positive form.

12.91.4.12 `bdd spot::tgba_bdd_core_data::nownext_set`

The conjunction of all Now and Next variables, in their positive form.

12.91.4.13 `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

12.91.4.14 `bdd spot::tgba_bdd_core_data::var_set`

The (positive) conjunction of all variables which are atomic propositions.

12.91.4.15 `bdd spot::tgba_bdd_core_data::varandnext_set`

The (positive) conjunction of all Next variables and atomic propositions.

The documentation for this struct was generated from the following file:

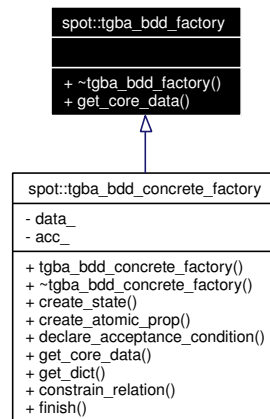
- [tgba/tgabddcoredata.hh](#)

12.92 spot::tgba_bdd_factory Class Reference

Abstract class for [spot::tgba_bdd_concrete](#) factories.

```
#include <tgba/tgabddfactory.hh>
```

Inheritance diagram for spot::tgba_bdd_factory:



Public Member Functions

- virtual `~tgba_bdd_factory ()`
- virtual const `tgba_bdd_core_data & get_core_data () const =0`

Get the core data for the new automata.

12.92.1 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.

12.92.2 Constructor & Destructor Documentation

12.92.2.1 virtual `spot::tgba_bdd_factory::~~tgba_bdd_factory ()` [`inline`, `virtual`]

12.92.3 Member Function Documentation

12.92.3.1 virtual const `tgba_bdd_core_data& spot::tgba_bdd_factory::get_core_data () const` [`pure virtual`]

Get the core data for the new automata.

Implemented in [spot::tgba_bdd_concrete_factory](#).

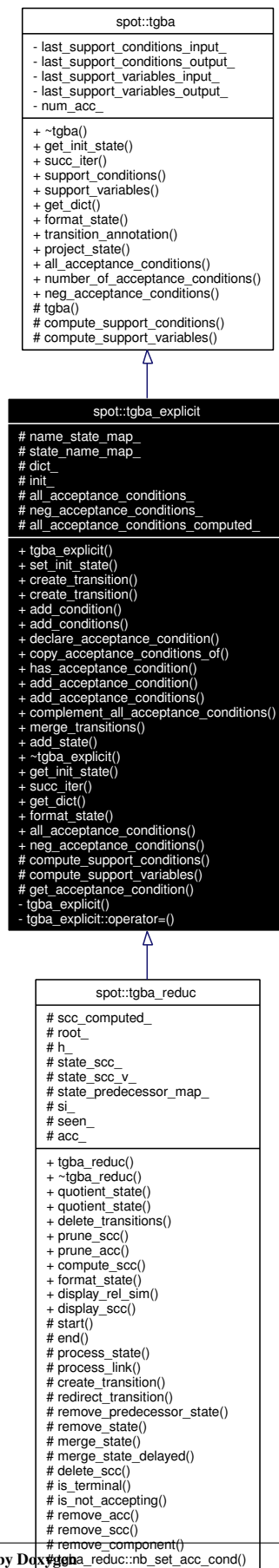
The documentation for this class was generated from the following file:

- [tgba/tgabddfactory.hh](#)

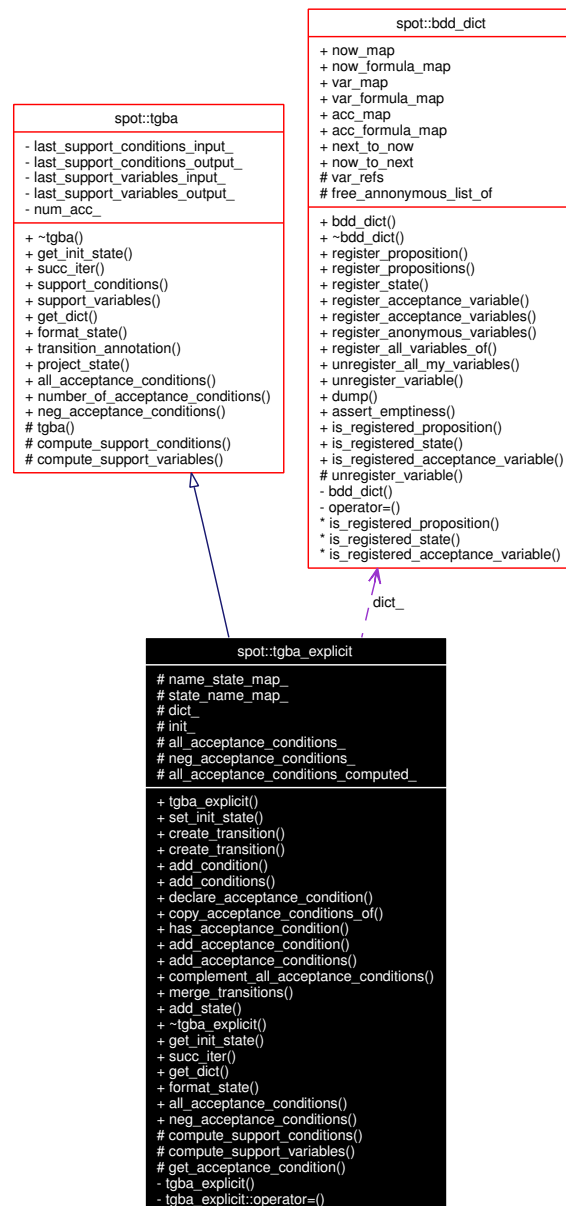
12.93 spot::tgba_explicit Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for spot::tgba_explicit:



Collaboration diagram for spot::tgba_explicit:



Public Types

- typedef std::list< [transition](#) * > [state](#)

Public Member Functions

- [tgba_explicit](#) ([bdd_dict](#) *dict)
- [state](#) * [set_init_state](#) (const std::string &state)
- [transition](#) * [create_transition](#) (const std::string &source, const std::string &dest)
- [transition](#) * [create_transition](#) ([state](#) *source, const [state](#) *dest)

- void [add_condition](#) ([transition](#) *t, const [ltl::formula](#) *f)
- void [add_conditions](#) ([transition](#) *t, bdd f)
This assumes that all variables in f are known from dict.
- void [declare_acceptance_condition](#) (const [ltl::formula](#) *f)
- void [copy_acceptance_conditions_of](#) (const [tgba](#) *a)
Copy the acceptance conditions of a tgba.
- bool [has_acceptance_condition](#) (const [ltl::formula](#) *f) const
- void [add_acceptance_condition](#) ([transition](#) *t, const [ltl::formula](#) *f)
- void [add_acceptance_conditions](#) ([transition](#) *t, bdd f)
This assumes that all acceptance conditions in f are known from dict.
- void [complement_all_acceptance_conditions](#) ()
- void [merge_transitions](#) ()
- [state](#) * [add_state](#) (const std::string &name)
- virtual [~tgba_explicit](#) ()
- virtual [spot::state](#) * [get_init_state](#) () const
Get the initial state of the automaton.
- virtual [tgba_succ_iterator](#) * [succ_iter](#) (const [spot::state](#) *local_state, const [spot::state](#) *global_state=0, const [tgba](#) *global_automaton=0) const
- virtual bdd [dict](#) * [get_dict](#) () const
Get the dictionary associated to the automaton.
- virtual std::string [format_state](#) (const [spot::state](#) *state) const
- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- virtual [tgba_succ_iterator](#) * [succ_iter](#) (const [state](#) *local_state, const [state](#) *global_state=0, const [tgba](#) *global_automaton=0) const =0
Get an iterator over the successors of local_state.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual std::string [format_state](#) (const [state](#) *state) const =0
Format the state as a string for printing.
- virtual std::string [transition_annotation](#) (const [tgba_succ_iterator](#) *t) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual [state](#) * [project_state](#) (const [state](#) *s, const [tgba](#) *t) const
Project a state on an automaton.

- virtual unsigned int [number_of_acceptance_conditions](#) () const
The number of acceptance conditions.

Protected Types

- typedef Sgi::hash_map< const std::string, [tgba_explicit::state](#) *, [string_hash](#) > [ns_map](#)
- typedef Sgi::hash_map< const [tgba_explicit::state](#) *, std::string, [ptr_hash](#)< [tgba_explicit::state](#) > > [sn_map](#)

Protected Member Functions

- virtual bdd [compute_support_conditions](#) (const [spot::state](#) *state) const
- virtual bdd [compute_support_variables](#) (const [spot::state](#) *state) const
- bdd [get_acceptance_condition](#) (const [ltl::formula](#) *f)
- virtual bdd [compute_support_conditions](#) (const [state](#) *state) const =0
Do the actual computation of [tgba::support_conditions\(\)](#).
- virtual bdd [compute_support_variables](#) (const [state](#) *state) const =0
Do the actual computation of [tgba::support_variables\(\)](#).

Protected Attributes

- [ns_map](#) [name_state_map_](#)
- [sn_map](#) [state_name_map_](#)
- bdd_dict * [dict_](#)
- [tgba_explicit::state](#) * [init_](#)
- bdd [all_acceptance_conditions_](#)
- bdd [neg_acceptance_conditions_](#)
- bool [all_acceptance_conditions_computed_](#)

Private Member Functions

- [tgba_explicit](#) (const [tgba_explicit](#) &other)
- [tgba_explicit](#) & [tgba_explicit::operator=](#) (const [tgba_explicit](#) &other)

Classes

- struct [transition](#)
Explicit transitions (used by [spot::tgba_explicit](#)).

12.93.1 Detailed Description

Explicit representation of a [spot::tgba](#).

12.93.2 Member Typedef Documentation

12.93.2.1 `typedef Sgi::hash_map<const std::string, tgba_explicit::state*, string_hash> spot::tgba_explicit::ns_map` [protected]

12.93.2.2 `typedef Sgi::hash_map<const tgba_explicit::state*, std::string, ptr_hash<tgba_explicit::state>> spot::tgba_explicit::sn_map` [protected]

12.93.2.3 `typedef std::list<transition*> spot::tgba_explicit::state`

12.93.3 Constructor & Destructor Documentation

12.93.3.1 `spot::tgba_explicit::tgba_explicit (bdd_dict * dict)`

12.93.3.2 `virtual spot::tgba_explicit::~~tgba_explicit ()` [virtual]

12.93.3.3 `spot::tgba_explicit::tgba_explicit (const tgba_explicit & other)` [private]

12.93.4 Member Function Documentation

12.93.4.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)`

12.93.4.2 `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)`

This assumes that all acceptance conditions in f are known from dict.

12.93.4.3 `void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f)`

12.93.4.4 `void spot::tgba_explicit::add_conditions (transition * t, bdd f)`

This assumes that all variables in f are known from dict.

12.93.4.5 `state* spot::tgba_explicit::add_state (const std::string & name)`

Return the `tgba_explicit::state` for $name$, creating the state if it does not exist.

12.93.4.6 `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`.

12.93.4.7 `void spot::tgba_explicit::complement_all_acceptance_conditions ()`

12.93.4.8 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.

12.93.4.9 virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state) const
[protected, virtual]

12.93.4.10 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.

12.93.4.11 virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state) const
[protected, virtual]

12.93.4.12 void spot::tgba_explicit::copy_acceptance_conditions_of (const tgba * a)

Copy the acceptance conditions of a tgba.

If used, this function should be called before creating any transition.

12.93.4.13 transition* spot::tgba_explicit::create_transition (state * source, const state * dest)

12.93.4.14 transition* spot::tgba_explicit::create_transition (const std::string & source, const std::string & dest)

12.93.4.15 void spot::tgba_explicit::declare_acceptance_condition (const ltl::formula * f)

12.93.4.16 virtual std::string spot::tgba::format_state (const state * state) const [pure virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in spot::tgba_bdd_concrete, spot::tgba_product, and spot::tgba_tba_proxy.

12.93.4.17 virtual std::string spot::tgba_explicit::format_state (const spot::state * state) const
[virtual]

Reimplemented in spot::tgba_reduc.

12.93.4.18 bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)
[protected]

12.93.4.19 virtual [bdd_dict*](#) spot::tgba_explicit::get_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

12.93.4.20 virtual [spot::state*](#) spot::tgba_explicit::get_init_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

12.93.4.21 bool spot::tgba_explicit::has_acceptance_condition (const [ltl::formula](#) * f) const**12.93.4.22 void spot::tgba_explicit::merge_transitions ()****12.93.4.23 virtual [bdd](#) spot::tgba_explicit::neg_acceptance_conditions () const** [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg_acceptance_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

12.93.4.24 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

12.93.4.25 virtual [state*](#) spot::tgba::project_state (const [state](#) * s, const [tgba](#) * t) const [virtual, inherited]

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

Returns:

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba_product](#), and [spot::tgba_tba_proxy](#).

12.93.4.26 `state* spot::tgba_explicit::set_init_state (const std::string & state)`**12.93.4.27** `virtual tgba_succ_iterator* spot::tgba::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const` [pure virtual, inherited]

Get an iterator over the successors of *local_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global_automaton* designate the root `spot::tgba`, and *global_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, *global_state* is not adopted by `succ_iter`.

global_automaton In a product, the global product automaton. Otherwise, 0.

Implemented in `spot::tgba_bdd_concrete`, `spot::tgba_product`, and `spot::tgba_tba_proxy`.

12.93.4.28 `virtual tgba_succ_iterator* spot::tgba_explicit::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const` [virtual]**12.93.4.29** `bdd spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

12.93.4.30 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

12.93.4.31 `tgba_explicit& spot::tgba_explicit::operator= (const tgba_explicit & other)` [private]

12.93.4.32 `virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const` `[virtual, inherited]`

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters:

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented in [spot::tgba_product](#), and [spot::tgba_tba_proxy](#).

12.93.5 Member Data Documentation

12.93.5.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` `[mutable, protected]`

12.93.5.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` `[mutable, protected]`

12.93.5.3 `bdd_dict* spot::tgba_explicit::dict_` `[protected]`

12.93.5.4 `tgba_explicit::state* spot::tgba_explicit::init_` `[protected]`

12.93.5.5 `ns_map spot::tgba_explicit::name_state_map_` `[protected]`

12.93.5.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_` `[protected]`

12.93.5.7 `sn_map spot::tgba_explicit::state_name_map_` `[protected]`

The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

12.94 spot::tgba_explicit::transition Struct Reference

Explicit transitions (used by [spot::tgba_explicit](#)).

```
#include <tgba/tgbaexplicit.hh>
```

Public Attributes

- bdd [condition](#)
- bdd [acceptance_conditions](#)
- const [state](#) * [dest](#)

12.94.1 Detailed Description

Explicit transitions (used by [spot::tgba_explicit](#)).

12.94.2 Member Data Documentation

12.94.2.1 bdd [spot::tgba_explicit::transition::acceptance_conditions](#)

12.94.2.2 bdd [spot::tgba_explicit::transition::condition](#)

12.94.2.3 const [state*](#) [spot::tgba_explicit::transition::dest](#)

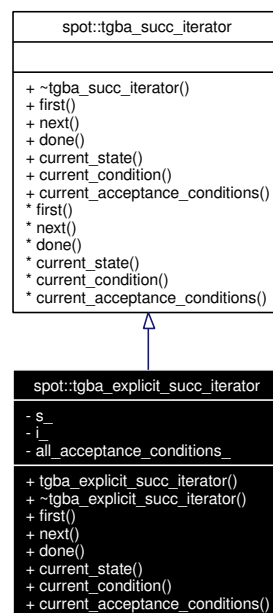
The documentation for this struct was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

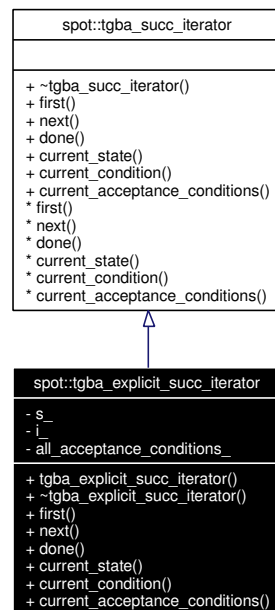
12.95 spot::tgba_explicit_succ_iterator Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for spot::tgba_explicit_succ_iterator:



Collaboration diagram for spot::tgba_explicit_succ_iterator:



Public Member Functions

- `tgba_explicit_succ_iterator` (const `tgba_explicit::state` *s, bdd all_acc)
- virtual `~tgba_explicit_succ_iterator` ()
- virtual void `first` ()
Position the iterator on the first successor (if any).
- virtual void `next` ()
Jump to the next successor (if any).
- virtual bool `done` () const
Check whether the iteration is finished.
- virtual `state_explicit` * `current_state` () const
Get the state of the current successor.
- virtual bdd `current_condition` () const
Get the condition on the transition leading to this successor.
- virtual bdd `current_acceptance_conditions` () const
Get the acceptance conditions on the transition leading to this successor.

Private Attributes

- const `tgba_explicit::state` * `s_`
- `tgba_explicit::state::const_iterator` `i_`
- bdd `all_acceptance_conditions_`

12.95.1 Detailed Description

Successor iterators used by [spot::tgba_explicit](#).

12.95.2 Constructor & Destructor Documentation

12.95.2.1 `spot::tgba_explicit_succ_iterator::tgba_explicit_succ_iterator (const tgba_explicit::state * s, bdd all_acc)`

12.95.2.2 `virtual spot::tgba_explicit_succ_iterator::~tgba_explicit_succ_iterator ()` `[inline, virtual]`

12.95.3 Member Function Documentation

12.95.3.1 `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](#).

12.95.3.2 `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](#).

12.95.3.3 `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](#).

12.95.3.4 `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](#).

12.95.3.5 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](#).

12.95.3.6 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](#).

12.95.4 Member Data Documentation**12.95.4.1 bdd spot::tgba_explicit_succ_iterator::all_acceptance_conditions_ [private]****12.95.4.2 tgba_explicit::state::const_iterator spot::tgba_explicit_succ_iterator::i_ [private]****12.95.4.3 const tgba_explicit::state* spot::tgba_explicit_succ_iterator::s_ [private]**

The documentation for this class was generated from the following file:

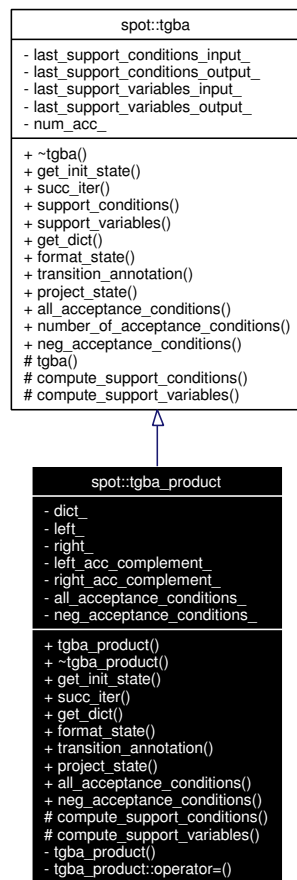
- [tgba/tgbaexplicit.hh](#)

12.96 spot::tgba_product Class Reference

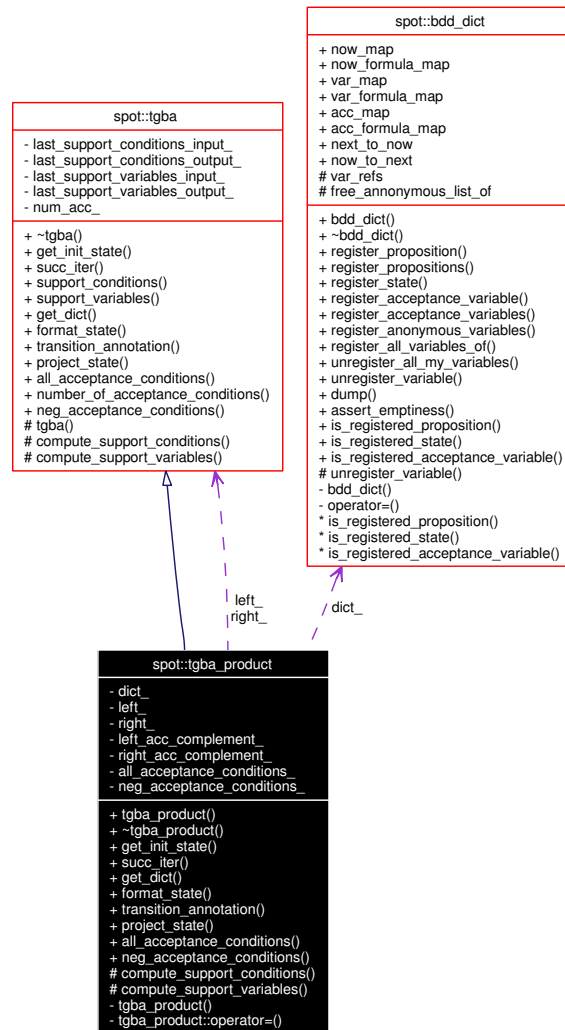
A lazy product. (States are computed on the fly.).

```
#include <tgba/tgbaproduct.hh>
```

Inheritance diagram for `spot::tgba_product`:



Collaboration diagram for `spot::tgba_product`:



Public Member Functions

- `tgba_product` (const `tgba` *left, const `tgba` *right)
Constructor.
- virtual `~tgba_product` ()
- virtual `state` * `get_init_state` () const
Get the initial state of the automaton.
- virtual `tgba_succ_iterator_product` * `succ_iter` (const `state` *local_state, const `state` *global_state=0, const `tgba` *global_automaton=0) const
Get an iterator over the successors of local_state.
- virtual `bdd_dict` * `get_dict` () const
Get the dictionary associated to the automaton.
- virtual std::string `format_state` (const `state` *state) const
Format the state as a string for printing.

- virtual std::string [transition_annotation](#) (const [tgba_succ_iterator](#) *t) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual [state](#) * [project_state](#) (const [state](#) *s, const [tgba](#) *t) const
Project a state on an automaton.
- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual unsigned int [number_of_acceptance_conditions](#) () const
The number of acceptance conditions.

Protected Member Functions

- virtual bdd [compute_support_conditions](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_conditions\(\)](#).
- virtual bdd [compute_support_variables](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_variables\(\)](#).

Private Member Functions

- [tgba_product](#) (const [tgba_product](#) &)
- [tgba_product](#) & [tgba_product::operator=](#) (const [tgba_product](#) &)

Private Attributes

- bdd_dict * [dict_](#)
- const [tgba](#) * [left_](#)
- const [tgba](#) * [right_](#)
- bdd [left_acc_complement_](#)
- bdd [right_acc_complement_](#)
- bdd [all_acceptance_conditions_](#)
- bdd [neg_acceptance_conditions_](#)

12.96.1 Detailed Description

A lazy product. (States are computed on the fly.).

12.96.2 Constructor & Destructor Documentation

12.96.2.1 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.

12.96.2.2 virtual spot::tgba_product::~~tgba_product () [virtual]

12.96.2.3 spot::tgba_product::tgba_product (const [tgba_product](#) &) [private]

12.96.3 Member Function Documentation

12.96.3.1 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](#).

12.96.3.2 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](#).

12.96.3.3 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](#).

12.96.3.4 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](#).

12.96.3.5 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](#).

12.96.3.6 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](#).

12.96.3.7 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](#).

12.96.3.8 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

12.96.3.9 virtual [state*](#) spot::tgba_product::project_state (const [state](#) * s, const [tgba](#) * t) const [virtual]

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

Returns:

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

12.96.3.10 virtual [tgba_succ_iterator_product*](#) spot::tgba_product::succ_iter (const [state](#) * local_state, const [state](#) * global_state = 0, const [tgba](#) * global_automaton = 0) const [virtual]

Get an iterator over the successors of `local_state`.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand.

global_automaton designate the root [spot::tgba](#), and *global_state* its state. This two objects can be used by [succ_iter\(\)](#) to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by [succ_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, *global_state* is not adopted by [succ_iter](#).

global_automaton In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

12.96.3.11 bdd spot::tgba::support_conditions (const [state](#) * state) const [inherited]

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by [succ_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute_support_conditions\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

12.96.3.12 bdd spot::tgba::support_variables (const [state](#) * state) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some [succ_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute_support_variables\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

12.96.3.13 [tgba_product&](#) spot::tgba_product::tgba_product::operator= (const [tgba_product](#) &) [private]

12.96.3.14 virtual std::string spot::tgba_product::transition_annotation (const [tgba_succ_iterator](#) * t) const [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

Parameters:

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented from [spot::tgba](#).

12.96.4 Member Data Documentation

12.96.4.1 bdd spot::tgba_product::all_acceptance_conditions_ [private]

12.96.4.2 `bdd_dict* spot::tgba_product::dict_` [private]

12.96.4.3 `const tgba* spot::tgba_product::left_` [private]

12.96.4.4 `bdd spot::tgba_product::left_acc_complement_` [private]

12.96.4.5 `bdd spot::tgba_product::neg_acceptance_conditions_` [private]

12.96.4.6 `const tgba* spot::tgba_product::right_` [private]

12.96.4.7 `bdd spot::tgba_product::right_acc_complement_` [private]

The documentation for this class was generated from the following file:

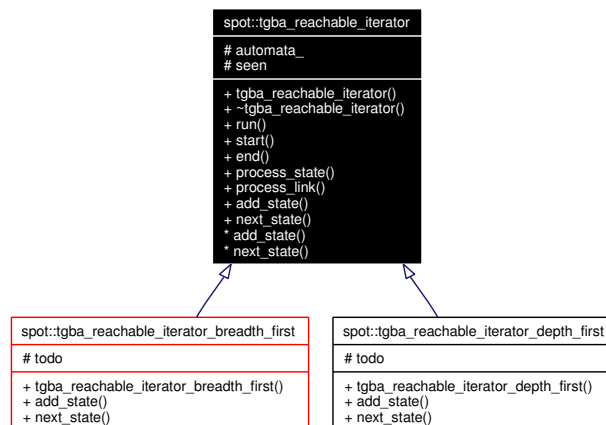
- [tgba/tgbaproduct.hh](#)

12.97 spot::tgba_reachable_iterator Class Reference

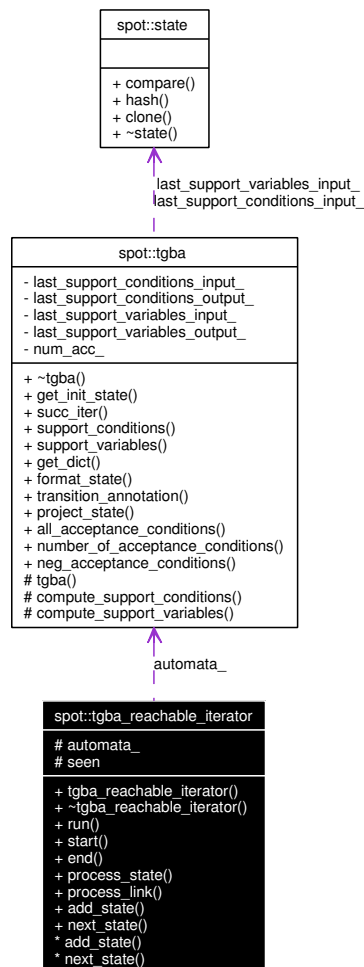
Iterate over all reachable states of a [spot::tgba](#).

```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::tgba_reachable_iterator`:



Collaboration diagram for `spot::tgba_reachable_iterator`:



Public Member Functions

- [tgba_reachable_iterator](#) (const [tgba](#) *a)
- virtual [~tgba_reachable_iterator](#) ()
- void [run](#) ()
Iterate over all reachable states of a [spot::tgba](#).
- virtual void [start](#) ()
Called by [run\(\)](#) before starting its iteration.
- virtual void [end](#) ()
Called by [run\(\)](#) once all states have been explored.
- virtual void [process_state](#) (const [state](#) *s, int n, [tgba_succ_iterator](#) *si)
- virtual void [process_link](#) (const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si)

Todo list management.

Called by [run\(\)](#) to register newly discovered states.

spot::tgba_reachable_iterator_depth_first and *spot::tgba_reachable_iterator_breadth_first* offer two precanned implementations for these functions.

- virtual void `add_state` (const `state` *s)=0
- virtual const `state` * `next_state` ()=0

Called by `run()` to obtain the.

Protected Types

- typedef Sgi::hash_map< const `state` *, int, `state_ptr_hash`, `state_ptr_equal` > `seen_map`

Protected Attributes

- const `tgba` * `automata_`
The `spot::tgba` to explore.
- `seen_map` `seen`
States already seen.

12.97.1 Detailed Description

Iterate over all reachable states of a `spot::tgba`.

12.97.2 Member Typedef Documentation

12.97.2.1 typedef Sgi::hash_map<const `state`*, int, `state_ptr_hash`, `state_ptr_equal`> `spot::tgba_reachable_iterator::seen_map` [protected]

Reimplemented in `spot::tgba_reduc`.

12.97.3 Constructor & Destructor Documentation

12.97.3.1 `spot::tgba_reachable_iterator::tgba_reachable_iterator` (const `tgba` * a)

12.97.3.2 virtual `spot::tgba_reachable_iterator::~~tgba_reachable_iterator` () [virtual]

12.97.4 Member Function Documentation

12.97.4.1 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`.

12.97.4.2 virtual void `spot::tgba_reachable_iterator::end` () [virtual]

Called by `run()` once all states have been explored.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

12.97.4.3 `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`.

12.97.4.4 `virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba_succ_iterator * si)` [virtual]

Called by `run()` to process a transition.

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The `spot::tgba_succ_iterator` positionned on the current transition.

The *in_s* and *out_s* states are owned by the `spot::tgba_reachable_iterator` instance and destroyed when the instance is destroyed.

12.97.4.5 `virtual void spot::tgba_reachable_iterator::process_state (const state * s, int n, tgba_succ_iterator * si)` [virtual]

Called by `run()` to process a state.

Parameters:

s The current state.

n A unique number assigned to *s*.

si The `spot::tgba_succ_iterator` for *s*.

Reimplemented in `spot::parity_game_graph`.

12.97.4.6 `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.

12.97.4.7 `virtual void spot::tgba_reachable_iterator::start ()` [virtual]

Called by `run()` before starting its iteration.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

12.97.5 Member Data Documentation**12.97.5.1** `const tgba* spot::tgba_reachable_iterator::automata_` [protected]

The `spot::tgba` to explore.

12.97.5.2 `seen_map` `spot::tgba_reachable_iterator::seen` [protected]

States already seen.

The documentation for this class was generated from the following file:

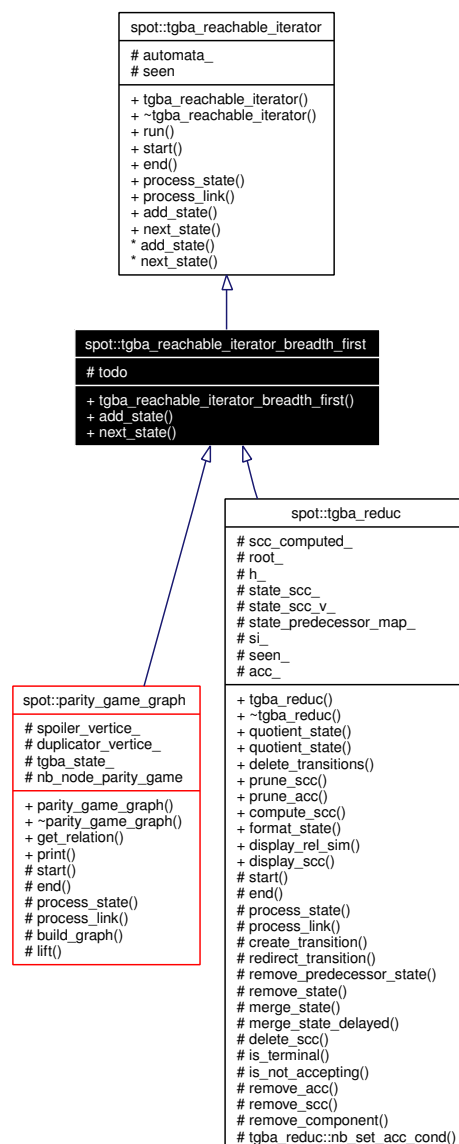
- `tgbaalgos/reachiter.hh`

12.98 `spot::tgba_reachable_iterator_breadth_first` Class Reference

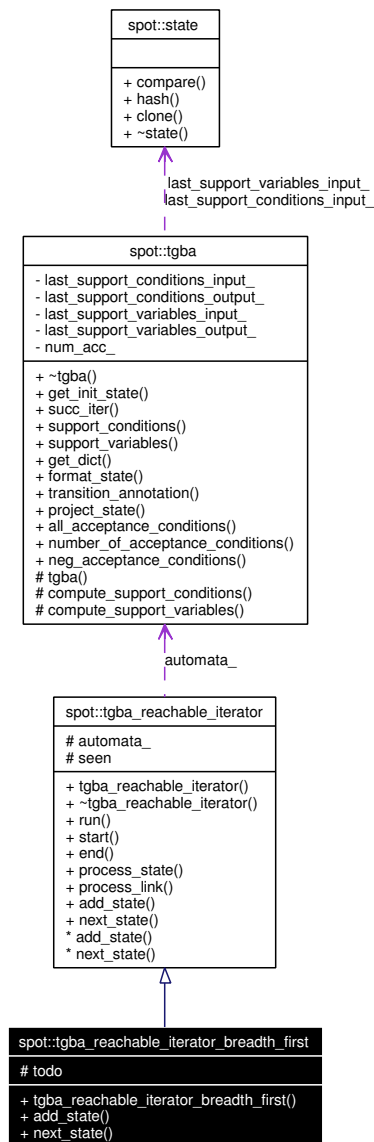
An implementation of `spot::tgba_reachable_iterator` that browses states breadth first.

```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::tgba_reachable_iterator_breadth_first`:



Collaboration diagram for `spot::tgba_reachable_iterator_breadth_first`:



Public Member Functions

- `tgba_reachable_iterator_breadth_first` (const `tgba` *a)
- virtual void `add_state` (const `state` *s)
- virtual const `state` * `next_state` ()
Called by `run()` to obtain the.
- void `run` ()
Iterate over all reachable states of a `spot::tgba`.
- virtual void `start` ()
Called by `run()` before starting its iteration.
- virtual void `end` ()

Called by [run\(\)](#) once all states have been explored.

- virtual void [process_state](#) (const [state](#) *s, int n, [tgba_succ_iterator](#) *si)
- virtual void [process_link](#) (const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si)

Protected Types

- typedef Sgi::hash_map< const [state](#) *, int, [state_ptr_hash](#), [state_ptr_equal](#) > [seen_map](#)

Protected Attributes

- std::deque< const [state](#) * > [todo](#)
A queue of states yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

12.98.1 Detailed Description

An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.

12.98.2 Member Typedef Documentation

12.98.2.1 typedef Sgi::hash_map<const [state](#)*, int, [state_ptr_hash](#), [state_ptr_equal](#)> [spot::tgba_reachable_iterator::seen_map](#) [protected, inherited]

Reimplemented in [spot::tgba_reduc](#).

12.98.3 Constructor & Destructor Documentation

12.98.3.1 [spot::tgba_reachable_iterator_breadth_first::tgba_reachable_iterator_breadth_first](#) (const [tgba](#) * a)

12.98.4 Member Function Documentation

12.98.4.1 virtual void [spot::tgba_reachable_iterator_breadth_first::add_state](#) (const [state](#) * s) [virtual]

Implements [spot::tgba_reachable_iterator](#).

12.98.4.2 virtual void [spot::tgba_reachable_iterator::end](#) () [virtual, inherited]

Called by [run\(\)](#) once all states have been explored.

Reimplemented in [spot::tgba_reduc](#), and [spot::parity_game_graph](#).

12.98.4.3 virtual const [state](#)* spot::tgba_reachable_iterator_breadth_first::next_state ()
[virtual]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba_reachable_iterator](#).

12.98.4.4 virtual void spot::tgba_reachable_iterator::process_link (const [state](#) * *in_s*, int *in*, const [state](#) * *out_s*, int *out*, const [tgba_succ_iterator](#) * *si*) [virtual, inherited]

Called by [run\(\)](#) to process a transition.

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The [spot::tgba_succ_iterator](#) positionned on the current transition.

The *in_s* and *out_s* states are owned by the [spot::tgba_reachable_iterator](#) instance and destroyed when the instance is destroyed.

12.98.4.5 virtual void spot::tgba_reachable_iterator::process_state (const [state](#) * *s*, int *n*, [tgba_succ_iterator](#) * *si*) [virtual, inherited]

Called by [run\(\)](#) to process a state.

Parameters:

s The current state.

n A unique number assigned to *s*.

si The [spot::tgba_succ_iterator](#) for *s*.

Reimplemented in [spot::parity_game_graph](#).

12.98.4.6 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.

12.98.4.7 virtual void spot::tgba_reachable_iterator::start () [virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented in [spot::tgba_reduc](#), and [spot::parity_game_graph](#).

12.98.5 Member Data Documentation

12.98.5.1 const [tgba](#)* spot::tgba_reachable_iterator::automata_ [protected, inherited]

The [spot::tgba](#) to explore.

12.98.5.2 `seen_map` `spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.98.5.3 `std::deque<const state*>` `spot::tgba_reachable_iterator_breadth_first::todo` [protected]

A queue of states yet to explore.

The documentation for this class was generated from the following file:

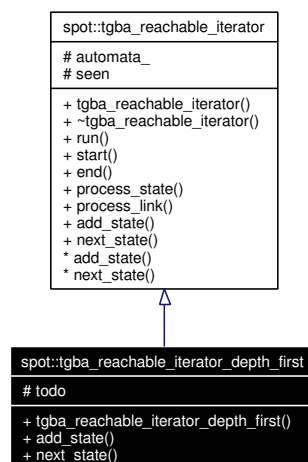
- `tgbaalgos/reachiter.hh`

12.99 `spot::tgba_reachable_iterator_depth_first` Class Reference

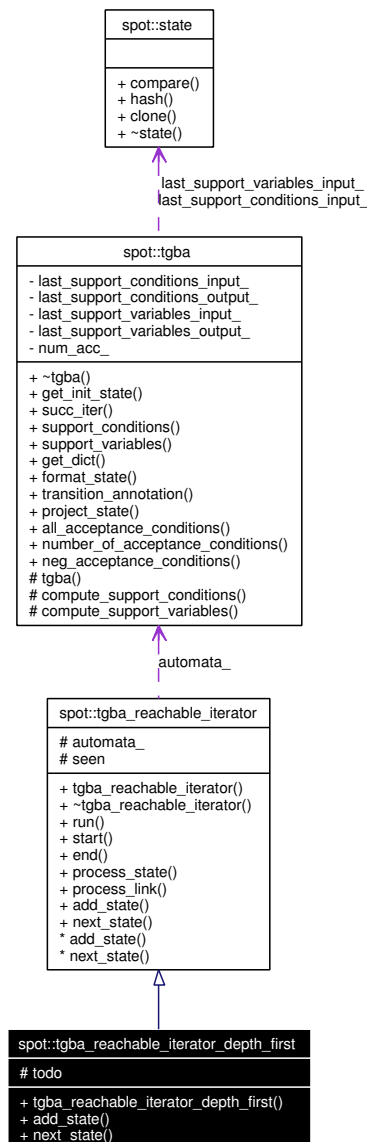
An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

```
#include <tgbaalgos/reachiter.hh>
```

Inheritance diagram for `spot::tgba_reachable_iterator_depth_first`:



Collaboration diagram for `spot::tgba_reachable_iterator_depth_first`:



Public Member Functions

- [tgba_reachable_iterator_depth_first](#) (const [tgba](#) *a)
- virtual void [add_state](#) (const [state](#) *s)
- virtual const [state](#) * [next_state](#) ()
Called by [run\(\)](#) to obtain the.
- void [run](#) ()
Iterate over all reachable states of a [spot::tgba](#).
- virtual void [start](#) ()
Called by [run\(\)](#) before starting its iteration.
- virtual void [end](#) ()

Called by `run()` once all states have been explored.

- virtual void `process_state` (const `state` *s, int n, `tgba_succ_iterator` *si)
- virtual void `process_link` (const `state` *in_s, int in, const `state` *out_s, int out, const `tgba_succ_iterator` *si)

Protected Types

- typedef `Sgi::hash_map< const state *, int, state_ptr_hash, state_ptr_equal >` `seen_map`

Protected Attributes

- `std::stack< const state * >` `todo`
A stack of states yet to explore.
- const `tgba` * `automata_`
The `spot::tgba` to explore.
- `seen_map` `seen`
States already seen.

12.99.1 Detailed Description

An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

12.99.2 Member Typedef Documentation

12.99.2.1 `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal> spot::tgba_reachable_iterator::seen_map` [protected, inherited]

Reimplemented in `spot::tgba_reduc`.

12.99.3 Constructor & Destructor Documentation

12.99.3.1 `spot::tgba_reachable_iterator_depth_first::tgba_reachable_iterator_depth_first (const tgba * a)`

12.99.4 Member Function Documentation

12.99.4.1 `virtual void spot::tgba_reachable_iterator_depth_first::add_state (const state * s)` [virtual]

Implements `spot::tgba_reachable_iterator`.

12.99.4.2 `virtual void spot::tgba_reachable_iterator::end ()` [virtual, inherited]

Called by `run()` once all states have been explored.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

12.99.4.3 virtual const [state](#)* spot::tgba_reachable_iterator_depth_first::next_state ()
[virtual]

Called by [run\(\)](#) to obtain the.

Implements [spot::tgba_reachable_iterator](#).

12.99.4.4 virtual void spot::tgba_reachable_iterator::process_link (const [state](#) * *in_s*, int *in*, const [state](#) * *out_s*, int *out*, const [tgba_succ_iterator](#) * *si*) [virtual, inherited]

Called by [run\(\)](#) to process a transition.

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The [spot::tgba_succ_iterator](#) positionned on the current transition.

The *in_s* and *out_s* states are owned by the [spot::tgba_reachable_iterator](#) instance and destroyed when the instance is destroyed.

12.99.4.5 virtual void spot::tgba_reachable_iterator::process_state (const [state](#) * *s*, int *n*, [tgba_succ_iterator](#) * *si*) [virtual, inherited]

Called by [run\(\)](#) to process a state.

Parameters:

s The current state.

n A unique number assigned to *s*.

si The [spot::tgba_succ_iterator](#) for *s*.

Reimplemented in [spot::parity_game_graph](#).

12.99.4.6 void spot::tgba_reachable_iterator::run () [inherited]

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterate over state.

12.99.4.7 virtual void spot::tgba_reachable_iterator::start () [virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented in [spot::tgba_reduc](#), and [spot::parity_game_graph](#).

12.99.5 Member Data Documentation

12.99.5.1 const [tgba](#)* spot::tgba_reachable_iterator::automata_ [protected, inherited]

The [spot::tgba](#) to explore.

12.99.5.2 `seen_map` `spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

12.99.5.3 `std::stack<const state*>` `spot::tgba_reachable_iterator_depth_first::todo` [protected]

A stack of states yet to explore.

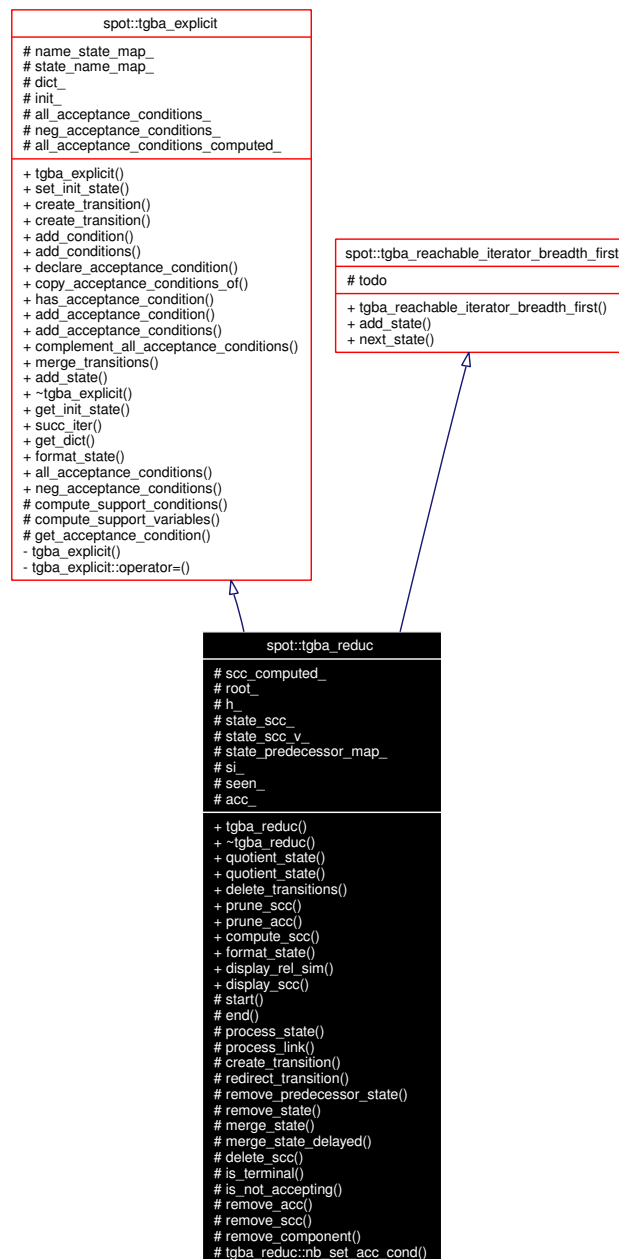
The documentation for this class was generated from the following file:

- `tgbaalgos/reachiter.hh`

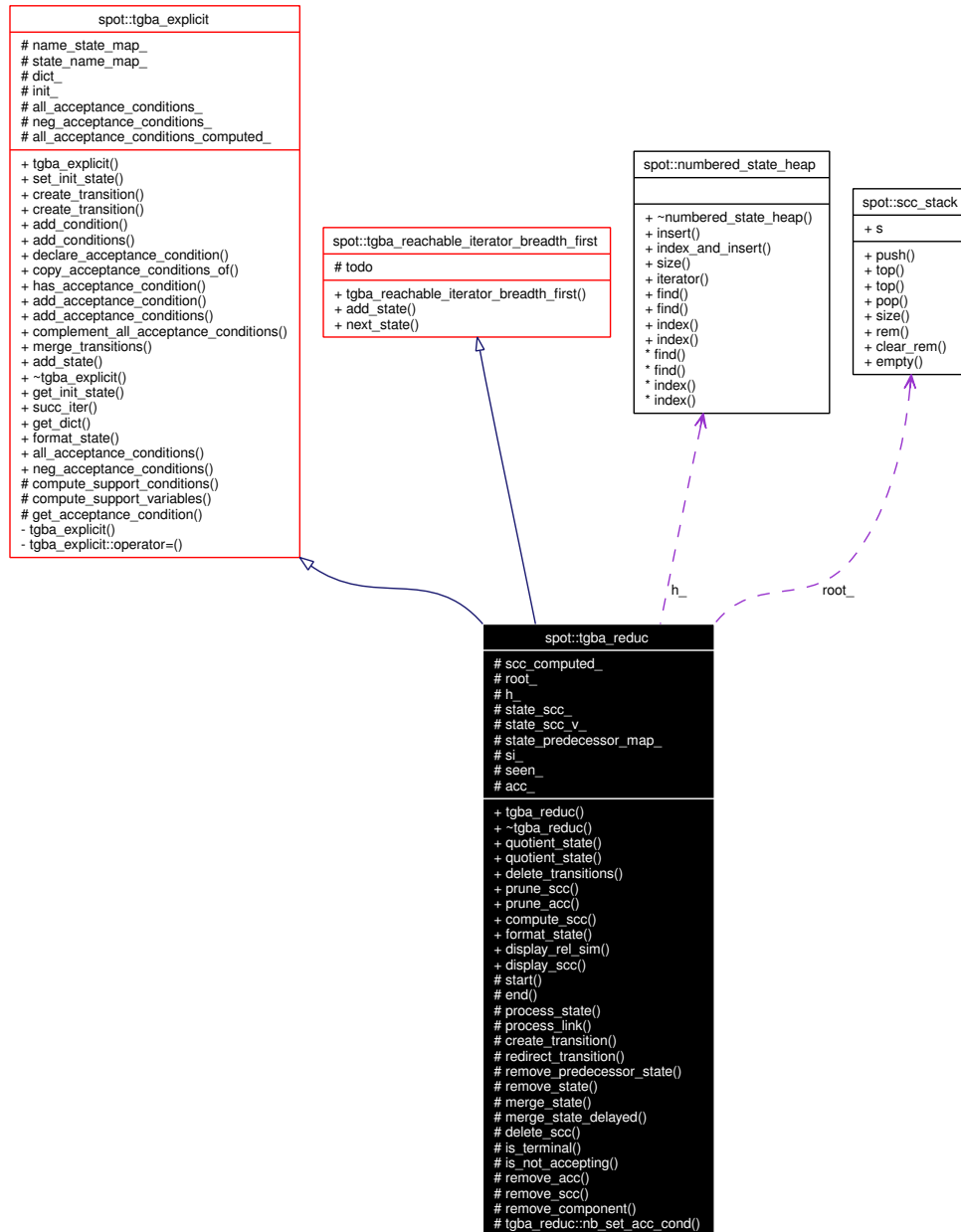
12.100 `spot::tgba_reduc` Class Reference

```
#include <tgba/tgbareduc.hh>
```

Inheritance diagram for `spot::tgba_reduc`:



Collaboration diagram for spot::tgba_reduc:



Public Types

- typedef std::list< transition * > [state](#)

Public Member Functions

- [tgba_reduc](#) (const [tgba](#) *a, const [numbered_state_heap_factory](#) *nshf=numbered_state_heap_hash_map_factory::instance())
- [~tgba_reduc](#) ()
- void [quotient_state](#) ([direct_simulation_relation](#) *rel)
- void [quotient_state](#) ([delayed_simulation_relation](#) *rel)

- void [delete_transitions](#) ([simulation_relation](#) *rel)
Delete some transitions with help of a simulation relation.
- void [prune_scc](#) ()
Remove all state which not lead to an accepting cycle.
- void [prune_acc](#) ()
Remove some useless acceptance condition.
- void [compute_scc](#) ()
Compute the maximal SCC of the automata.
- virtual std::string [format_state](#) (const [spot::state](#) *state) const
Add the SCC index to the display of the state state.
- void [display_rel_sim](#) ([simulation_relation](#) *rel, std::ostream &os)
- void [display_scc](#) (std::ostream &os)
- [state](#) * [set_init_state](#) (const std::string &state)
- transition * [create_transition](#) (const std::string &source, const std::string &dest)
- transition * [create_transition](#) ([state](#) *source, const [state](#) *dest)
- void [add_condition](#) (transition *t, const [ltl::formula](#) *f)
- void [add_conditions](#) (transition *t, bdd f)
This assumes that all variables in f are known from dict.
- void [declare_acceptance_condition](#) (const [ltl::formula](#) *f)
- void [copy_acceptance_conditions_of](#) (const [tgba](#) *a)
Copy the acceptance conditions of a tgba.
- bool [has_acceptance_condition](#) (const [ltl::formula](#) *f) const
- void [add_acceptance_condition](#) (transition *t, const [ltl::formula](#) *f)
- void [add_acceptance_conditions](#) (transition *t, bdd f)
This assumes that all acceptance conditions in f are known from dict.
- void [complement_all_acceptance_conditions](#) ()
- void [merge_transitions](#) ()
- [state](#) * [add_state](#) (const std::string &name)
- virtual [spot::state](#) * [get_init_state](#) () const
Get the initial state of the automaton.
- virtual [tgba_succ_iterator](#) * [succ_iter](#) (const [spot::state](#) *local_state, const [spot::state](#) *global_state=0, const [tgba](#) *global_automaton=0) const
- virtual [tgba_succ_iterator](#) * [succ_iter](#) (const [state](#) *local_state, const [state](#) *global_state=0, const [tgba](#) *global_automaton=0) const =0
Get an iterator over the successors of local_state.
- virtual [bdd_dict](#) * [get_dict](#) () const
Get the dictionary associated to the automaton.
- virtual std::string [format_state](#) (const [state](#) *state) const =0
Format the state as a string for printing.

- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual std::string [transition_annotation](#) (const [tgba_succ_iterator](#) *t) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual [state](#) * [project_state](#) (const [state](#) *s, const [tgba](#) *t) const
Project a state on an automaton.
- virtual unsigned int [number_of_acceptance_conditions](#) () const
The number of acceptance conditions.
- virtual void [add_state](#) (const [state](#) *s)
- virtual const [state](#) * [next_state](#) ()
Called by [run\(\)](#) to obtain the.
- void [run](#) ()
Iterate over all reachable states of a [spot::tgba](#).
- virtual void [process_state](#) (const [state](#) *s, int n, [tgba_succ_iterator](#) *si)
- virtual void [process_link](#) (const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si)

Protected Types

- typedef Sgi::hash_map< const [tgba_explicit::state](#) *, std::list< [state](#) * > *, ptr_hash< [tgba_explicit::state](#) > > [sp_map](#)
- typedef Sgi::hash_map< const [spot::state](#) *, int, state_ptr_hash, state_ptr_equal > [seen_map](#)
- typedef Sgi::hash_map< const std::string, [tgba_explicit::state](#) *, string_hash > [ns_map](#)
- typedef Sgi::hash_map< const [tgba_explicit::state](#) *, std::string, ptr_hash< [tgba_explicit::state](#) > > [sn_map](#)

Protected Member Functions

- void [start](#) ()
Called by [run\(\)](#) before starting its iteration.
- void [end](#) ()
Called by [run\(\)](#) once all states have been explored.

- void [process_state](#) (const [spot::state](#) *s, int n, [tgba_succ_iterator](#) *si)
- void [process_link](#) (int in, int out, const [tgba_succ_iterator](#) *si)
- transition * [create_transition](#) (const [spot::state](#) *source, const [spot::state](#) *dest)

Create a transition using two state of a TGBA.

- void [redirect_transition](#) (const [spot::state](#) *s, const [spot::state](#) *simul)
- void [remove_predecessor_state](#) (const [state](#) *s, const [state](#) *p)

Remove p of the predecessor of s.

- void [remove_state](#) (const [spot::state](#) *s)
- void [merge_state](#) (const [spot::state](#) *s1, const [spot::state](#) *s2)
- void [merge_state_delayed](#) (const [spot::state](#) *s1, const [spot::state](#) *s2)
- void [delete_scc](#) ()
- bool [is_terminal](#) (const [spot::state](#) *s, int n=-1)
- bool [is_not_accepting](#) (const [spot::state](#) *s, int n=-1)
- void [remove_acc](#) (const [spot::state](#) *s)
- void [remove_scc](#) ([spot::state](#) *s)

Remove all the state which belong to the same scc that s.

- void [remove_component](#) (const [spot::state](#) *from)

For compute_scc.

- int [tgba_reduc::nb_set_acc_cond](#) () const
- virtual bdd [compute_support_conditions](#) (const [spot::state](#) *state) const
- virtual bdd [compute_support_conditions](#) (const [state](#) *state) const =0

Do the actual computation of tgba::support_conditions().

- virtual bdd [compute_support_variables](#) (const [spot::state](#) *state) const
- virtual bdd [compute_support_variables](#) (const [state](#) *state) const =0

Do the actual computation of tgba::support_variables().

- bdd [get_acceptance_condition](#) (const [ltl::formula](#) *f)

Protected Attributes

- bool [scc_computed_](#)
- [scc_stack](#) root_
- [numbered_state_heap](#) * h_
- std::stack< const [spot::state](#) * > [state_scc_](#)
- Sgi::hash_map< int, const [spot::state](#) * > [state_scc_v_](#)
- [sp_map](#) [state_predecessor_map_](#)
- [seen_map](#) si_
- [seen_map](#) * seen_
- bdd [acc_](#)
- [ns_map](#) [name_state_map_](#)
- [sn_map](#) [state_name_map_](#)
- bdd_dict * dict_
- [tgba_explicit::state](#) * init_
- bdd [all_acceptance_conditions_](#)
- bdd [neg_acceptance_conditions_](#)

- bool [all_acceptance_conditions_computed_](#)
- `std::deque< const state * >` [todo](#)
A queue of states yet to explore.
- const [tgba](#) * [automata_](#)
The [spot::tgba](#) to explore.
- [seen_map](#) [seen](#)
States already seen.

12.100.1 Detailed Description

Explicit automata used in reductions.

12.100.2 Member Typedef Documentation

12.100.2.1 `typedef Sgi::hash_map<const std::string, tgba_explicit::state*, string_hash>`
[spot::tgba_explicit::ns_map](#) [protected, inherited]

12.100.2.2 `typedef Sgi::hash_map<const spot::state*, int, state_ptr_hash, state_ptr_equal>`
[spot::tgba_reduc::seen_map](#) [protected]

Reimplemented from [spot::tgba_reachable_iterator](#).

12.100.2.3 `typedef Sgi::hash_map<const tgba_explicit::state*, std::string, ptr_hash<tgba_explicit::state>>`
[spot::tgba_explicit::sn_map](#) [protected, inherited]

12.100.2.4 `typedef Sgi::hash_map<const tgba_explicit::state*, std::list<state*>*, ptr_hash<tgba_explicit::state>>`
[spot::tgba_reduc::sp_map](#) [protected]

12.100.2.5 `typedef std::list<transition*> spot::tgba_explicit::state` [inherited]

12.100.3 Constructor & Destructor Documentation

12.100.3.1 `spot::tgba_reduc::tgba_reduc (const tgba * a, const numbered_state_heap_factory * nshf = numbered_state_heap_hash_map_factory::instance\(\))`

12.100.3.2 `spot::tgba_reduc::~~tgba_reduc ()`

12.100.4 Member Function Documentation

12.100.4.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)` [inherited]

12.100.4.2 void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]

This assumes that all acceptance conditions in f are known from dict.

12.100.4.3 void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f) [inherited]

12.100.4.4 void spot::tgba_explicit::add_conditions (transition * t, bdd f) [inherited]

This assumes that all variables in f are known from dict.

12.100.4.5 virtual void spot::tgba_reachable_iterator_breadth_first::add_state (const state * s) [virtual, inherited]

Implements [spot::tgba_reachable_iterator](#).

12.100.4.6 state* spot::tgba_explicit::add_state (const std::string & name) [inherited]

Return the [tgba_explicit::state](#) for $name$, creating the state if it does not exist.

12.100.4.7 virtual bdd spot::tgba_explicit::all_acceptance_conditions () const [virtual, inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

12.100.4.8 void spot::tgba_explicit::complement_all_acceptance_conditions () [inherited]

12.100.4.9 void spot::tgba_reduc::compute_scc ()

Compute the maximal SCC of the automata.

12.100.4.10 virtual bdd spot::tgba::compute_support_conditions (const state * state) const [protected, pure virtual, inherited]

Do the actual computation of [tgba::support_conditions\(\)](#).

Implemented in [spot::tgba_bdd_concrete](#), [spot::tgba_product](#), and [spot::tgba_tba_proxy](#).

12.100.4.11 virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state) const [protected, virtual, inherited]

12.100.4.12 virtual bdd spot::tgba::compute_support_variables (const state * state) const [protected, pure virtual, inherited]

Do the actual computation of [tgba::support_variables\(\)](#).

Implemented in [spot::tgba_bdd_concrete](#), [spot::tgba_product](#), and [spot::tgba_tba_proxy](#).

12.100.4.13 virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state)
const [protected, virtual, inherited]

12.100.4.14 void spot::tgba_explicit::copy_acceptance_conditions_of (const tgba * a)
[inherited]

Copy the acceptance conditions of a tgba.

If used, this function should be called before creating any transition.

12.100.4.15 transition* spot::tgba_explicit::create_transition (state * source, const state * dest)
[inherited]

12.100.4.16 transition* spot::tgba_explicit::create_transition (const std::string & source, const std::string & dest) [inherited]

12.100.4.17 transition* spot::tgba_reduc::create_transition (const spot::state * source, const spot::state * dest) [protected]

Create a transition using two state of a TGBA.

12.100.4.18 void spot::tgba_explicit::declare_acceptance_condition (const ltl::formula * f)
[inherited]

12.100.4.19 void spot::tgba_reduc::delete_scc () [protected]

Remove all the scc which are terminal and doesn't contains all the acceptance conditions.

12.100.4.20 void spot::tgba_reduc::delete_transitions (simulation_relation * rel)

Delete some transitions with help of a simulation relation.

12.100.4.21 void spot::tgba_reduc::display_rel_sim (simulation_relation * rel, std::ostream & os)

12.100.4.22 void spot::tgba_reduc::display_scc (std::ostream & os)

12.100.4.23 void spot::tgba_reduc::end () [protected, virtual]

Called by run() once all states have been explored.

Reimplemented from spot::tgba_reachable_iterator.

12.100.4.24 virtual std::string spot::tgba::format_state (const state * state) const [pure virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implemented in spot::tgba_bdd_concrete, spot::tgba_product, and spot::tgba_tba_proxy.

12.100.4.25 `virtual std::string spot::tgba_reduc::format_state (const spot::state * state) const` [virtual]

Add the SCC index to the display of the state *state*.

Reimplemented from [spot::tgba_explicit](#).

12.100.4.26 `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)` [protected, inherited]

12.100.4.27 `virtual bdd_dict* spot::tgba_explicit::get_dict () const` [virtual, inherited]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

12.100.4.28 `virtual spot::state* spot::tgba_explicit::get_init_state () const` [virtual, inherited]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

12.100.4.29 `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const` [inherited]

12.100.4.30 `bool spot::tgba_reduc::is_not_accepting (const spot::state * s, int n = -1)` [protected]

12.100.4.31 `bool spot::tgba_reduc::is_terminal (const spot::state * s, int n = -1)` [protected]

Return true if the scc which contains *s* is an fixed-formula alpha-ball. this is explain in

```
@InProceedings{ etessami.00.concur,
  author    = {Kousha Etessami and Gerard J. Holzmann},
  title     = {Optimizing {B}\u}chi Automata},
  booktitle = {Proceedings of the 11th International Conference on
    Concurrency Theory (Concur'2000)},
  pages     = {153--167},
  year      = {2000},
  editor    = {C. Palamidessi},
  volume    = {1877},
  series    = {Lecture Notes in Computer Science},
  publisher = {Springer-Verlag}
}
```

12.100.4.32 void spot::tgba_reduc::merge_state (const spot::state * s1, const spot::state * s2) [protected]

Redirect all transition leading to s1 to s2. Note that we can do the reverse because s1 and s2 belong to a co-simulate relation.

12.100.4.33 void spot::tgba_reduc::merge_state_delayed (const spot::state * s1, const spot::state * s2) [protected]

Redirect all transition leading to s1 to s2. Note that we can do the reverse because s1 and s2 belong to a co-simulate relation.

12.100.4.34 void spot::tgba_explicit::merge_transitions () [inherited]

12.100.4.35 virtual bdd spot::tgba_explicit::neg_acceptance_conditions () const [virtual, inherited]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the `neg_acceptance_conditions()` of the other operand.

Implements `spot::tgba`.

12.100.4.36 virtual const state* spot::tgba_reachable_iterator_breadth_first::next_state () [virtual, inherited]

Called by `run()` to obtain the.

Implements `spot::tgba_reachable_iterator`.

12.100.4.37 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

12.100.4.38 virtual void spot::tgba_reachable_iterator::process_link (const state * in_s, int in, const state * out_s, int out, const tgba_succ_iterator * si) [virtual, inherited]

Called by `run()` to process a transition.

Parameters:

in_s The source state

in The source state number.

out_s The destination state

out The destination state number.

si The `spot::tgba_succ_iterator` positionned on the current transition.

The `in_s` and `out_s` states are owned by the `spot::tgba_reachable_iterator` instance and destroyed when the instance is destroyed.

12.100.4.39 void spot::tgba_reduc::process_link (int *in*, int *out*, const tgba_succ_iterator * *si*) [protected]

12.100.4.40 virtual void spot::tgba_reachable_iterator::process_state (const state * *s*, int *n*, tgba_succ_iterator * *si*) [virtual, inherited]

Called by run() to process a state.

Parameters:

- s* The current state.
- n* A unique number assigned to *s*.
- si* The spot::tgba_succ_iterator for *s*.

Reimplemented in spot::parity_game_graph.

12.100.4.41 void spot::tgba_reduc::process_state (const spot::state * *s*, int *n*, tgba_succ_iterator * *si*) [protected]

12.100.4.42 virtual state* spot::tgba::project_state (const state * *s*, const tgba * *t*) const [virtual, inherited]

Project a state on an automaton.

This converts *s*, into that corresponding spot::state for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata 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.

12.100.4.43 void spot::tgba_reduc::prune_acc ()

Remove some useless acceptance condition.

12.100.4.44 void spot::tgba_reduc::prune_scc ()

Remove all state which not lead to an accepting cycle.

12.100.4.45 void spot::tgba_reduc::quotient_state (delayed_simulation_relation * *rel*)

Build the quotient automata. Call this method when use to a delayed simulation relation.

12.100.4.46 void spot::tgba_reduc::quotient_state (direct_simulation_relation * *rel*)

Reduce the automata using a relation simulation Do not call this method with a delayed simulation relation.

12.100.4.47 void spot::tgba_reduc::redirect_transition (const spot::state * s, const spot::state * simul) [protected]

Remove all the transition from the state q, predecessor of both s and simul, which can be removed.

12.100.4.48 void spot::tgba_reduc::remove_acc (const spot::state * s) [protected]

If a scc maximal do not contains all the acceptance conditions we can remove all the acceptance conditions in this scc.

12.100.4.49 void spot::tgba_reduc::remove_component (const spot::state * from) [protected]

For compute_scc.

12.100.4.50 void spot::tgba_reduc::remove_predecessor_state (const state * s, const state * p) [protected]

Remove p of the predecessor of s.

12.100.4.51 void spot::tgba_reduc::remove_scc (spot::state * s) [protected]

Remove all the state which belong to the same scc that s.

12.100.4.52 void spot::tgba_reduc::remove_state (const spot::state * s) [protected]

Remove all the transition leading to s. s is then unreachable and can be consider as remove.

12.100.4.53 void spot::tgba_reachable_iterator::run () [inherited]

Iterate over all reachable states of a spot::tgba.

This is a template method that will call add_state(), next_state(), start(), end(), process_state(), and process_link(), while it iterate over state.

12.100.4.54 state* spot::tgba_explicit::set_init_state (const std::string & state) [inherited]

12.100.4.55 void spot::tgba_reduc::start () [protected, virtual]

Called by run() before starting its iteration.

Reimplemented from spot::tgba_reachable_iterator.

12.100.4.56 virtual tgba_succ_iterator* spot::tgba::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const [pure virtual, inherited]

Get an iterator over the successors of local_state.

The iterator has been allocated with new. It is the responsibility of the caller to delete it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of spot::tgba where most values are computed on demand. global_automaton designate the root spot::tgba, and global_state its state. This two objects can be used by succ_iter() to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like `local_state`, `global_state` is not adopted by `succ_iter`.

global_automaton In a product, the global product automaton. Otherwise, 0.

Implemented in `spot::tgba_bdd_concrete`, `spot::tgba_product`, and `spot::tgba_tba_proxy`.

12.100.4.57 `virtual tgba_succ_iterator* spot::tgba_explicit::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const` [virtual, inherited]

12.100.4.58 `bdd spot::tgba::support_conditions (const state * state) const` [inherited]

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of `state`.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

12.100.4.59 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of `state`.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

12.100.4.60 `int spot::tgba_reduc::tgba_reduc::nb_set_acc_cond () const` [protected]

12.100.4.61 `virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const` [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters:

t a non-done `tgba_succ_iterator` for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_tba_proxy`.

12.100.5 Member Data Documentation

12.100.5.1 `bdd spot::tgba_reduc::acc_` [protected]

12.100.5.2 `bdd spot::tgba_explicit::all_acceptance_conditions_` [mutable, protected, inherited]

12.100.5.3 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` [mutable, protected, inherited]

12.100.5.4 `const tgba* spot::tgba_reachable_iterator::automata_` [protected, inherited]

The `spot::tgba` to explore.

12.100.5.5 `bdd_dict* spot::tgba_explicit::dict_` [protected, inherited]

12.100.5.6 `numbered_state_heap* spot::tgba_reduc::h_` [protected]

12.100.5.7 `tgba_explicit::state* spot::tgba_explicit::init_` [protected, inherited]

12.100.5.8 `ns_map spot::tgba_explicit::name_state_map_` [protected, inherited]

12.100.5.9 `bdd spot::tgba_explicit::neg_acceptance_conditions_` [protected, inherited]

12.100.5.10 `scc_stack spot::tgba_reduc::root_` [protected]

12.100.5.11 `bool spot::tgba_reduc::scc_computed_` [protected]

12.100.5.12 `seen_map spot::tgba_reachable_iterator::seen_` [protected, inherited]

States already seen.

12.100.5.13 `seen_map* spot::tgba_reduc::seen_` [protected]

12.100.5.14 `seen_map spot::tgba_reduc::si_` [protected]

12.100.5.15 `sn_map spot::tgba_explicit::state_name_map_` [protected, inherited]

12.100.5.16 `sp_map spot::tgba_reduc::state_predecessor_map_` [protected]

12.100.5.17 `std::stack<const spot::state*> spot::tgba_reduc::state_scc_` [protected]

12.100.5.18 Sgi::hash_map<int, const spot::state*> spot::tgba_reduc::state_scc_v_- [protected]

12.100.5.19 std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo [protected, inherited]

A queue of states yet to explore.

The documentation for this class was generated from the following file:

- [tgba/tgbareduc.hh](#)

12.101 spot::tgba_run Struct Reference

An accepted run, for a tgba.

```
#include <tgbaalgos/emptiness.hh>
```

Public Types

- typedef std::list< [step](#) > [steps](#)

Public Member Functions

- [~tgba_run](#) ()
- [tgba_run](#) ()
- [tgba_run](#) (const [tgba_run](#) &run)
- [tgba_run](#) & [operator=](#) (const [tgba_run](#) &run)

Public Attributes

- [steps](#) prefix
- [steps](#) cycle

Classes

- struct [step](#)

12.101.1 Detailed Description

An accepted run, for a tgba.

12.101.2 Member Typedef Documentation

12.101.2.1 typedef std::list<[step](#)> [spot::tgba_run::steps](#)

12.101.3 Constructor & Destructor Documentation

12.101.3.1 [spot::tgba_run::~~tgba_run](#) ()

12.101.3.2 spot::tgba_run::tgba_run () [inline]

12.101.3.3 spot::tgba_run::tgba_run (const tgba_run & run)

12.101.4 Member Function Documentation

12.101.4.1 tgba_run& spot::tgba_run::operator= (const tgba_run & run)

12.101.5 Member Data Documentation

12.101.5.1 steps spot::tgba_run::cycle

12.101.5.2 steps spot::tgba_run::prefix

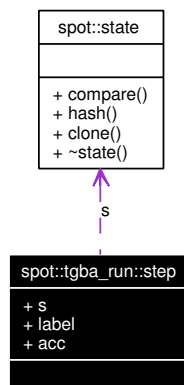
The documentation for this struct was generated from the following file:

- tgbaalgos/emptiness.hh

12.102 spot::tgba_run::step Struct Reference

```
#include <tgbaalgos/emptiness.hh>
```

Collaboration diagram for spot::tgba_run::step:



Public Attributes

- const state * s
- bdd label
- bdd acc

12.102.1 Member Data Documentation

12.102.1.1 bdd spot::tgba_run::step::acc

12.102.1.2 bdd spot::tgba_run::step::label

12.102.1.3 const state* spot::tgba_run::step::s

The documentation for this struct was generated from the following file:

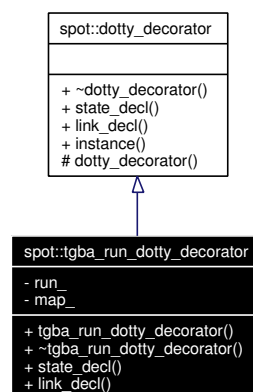
- [tgbaalgos/emptiness.hh](#)

12.103 spot::tgba_run_dotty_decorator Class Reference

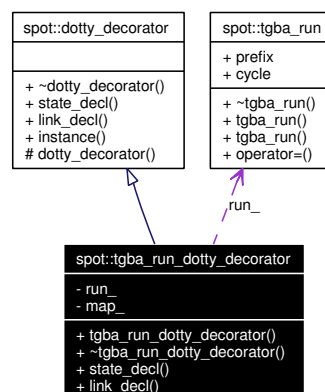
Highlight a [spot::tgba_run](#) on a [spot::tgba](#).

```
#include <tgbaalgos/rundotdec.hh>
```

Inheritance diagram for spot::tgba_run_dotty_decorator:



Collaboration diagram for spot::tgba_run_dotty_decorator:



Public Member Functions

- [tgba_run_dotty_decorator](#) (const [tgba_run](#) *run)
- virtual `~tgba_run_dotty_decorator` ()
- virtual std::string `state_decl` (const [tgba](#) *a, const [state](#) *s, int n, [tgba_succ_iterator](#) *si, const std::string &label)

Compute the style of a state.

- virtual std::string [link_decl](#) (const [tgba](#) *a, const [state](#) *in_s, int in, const [state](#) *out_s, int out, const [tgba_succ_iterator](#) *si, const std::string &label)

Compute the style of a link.

Static Public Member Functions

- static [dotty_decorator](#) * [instance](#) ()
Get the unique instance of the default [dotty_decorator](#).

Private Types

- typedef std::pair< [tgba_run::steps::const_iterator](#), int > [step_num](#)
- typedef std::list< [step_num](#) > [step_set](#)
- typedef std::map< const [state](#) *, std::pair< [step_set](#), [step_set](#) >, [spot::state_ptr_less_than](#) > [step_map](#)

Private Attributes

- const [tgba_run](#) * [run_](#)
- [step_map](#) [map_](#)

12.103.1 Detailed Description

Highlight a [spot::tgba_run](#) on a [spot::tgba](#).

An instance of this class can be passed to [spot::dotty_reachable](#).

12.103.2 Member Typedef Documentation

12.103.2.1 typedef std::map<const [state](#)*, std::pair<[step_set](#), [step_set](#)>, [spot::state_ptr_less_than](#)> [spot::tgba_run_dotty_decorator::step_map](#) [private]

12.103.2.2 typedef std::pair<[tgba_run::steps::const_iterator](#), int> [spot::tgba_run_dotty_decorator::step_num](#) [private]

12.103.2.3 typedef std::list<[step_num](#)> [spot::tgba_run_dotty_decorator::step_set](#) [private]

12.103.3 Constructor & Destructor Documentation

12.103.3.1 [spot::tgba_run_dotty_decorator::tgba_run_dotty_decorator](#) (const [tgba_run](#) * [run](#))

12.103.3.2 virtual [spot::tgba_run_dotty_decorator::~~tgba_run_dotty_decorator](#) () [virtual]

12.103.4 Member Function Documentation

12.103.4.1 static [dotty_decorator](#)* spot::dotty_decorator::instance () [static, inherited]

Get the unique instance of the default [dotty_decorator](#).

12.103.4.2 virtual std::string spot::tgba_run_dotty_decorator::link_decl (const [tgba](#) * *a*, const [state](#) * *in_s*, int *in*, const [state](#) * *out_s*, int *out*, const [tgba_succ_iterator](#) * *si*, const std::string & *label*) [virtual]

Compute the style of a link.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL"] with LABEL replaced by the value of *label*.

Parameters:

- a* the automaton being drawn
- in_s* the source state of the transition being drawn (owned by the caller)
- in* the unique number associated to *in_s*
- out_s* the destination state of the transition being drawn (owned by the caller)
- out* the unique number associated to *out_s*
- si* an iterator over the successors of *in_s*, pointing to the current transition (owned by the caller and cannot be iterated)
- label* the computed name of this state

Reimplemented from [spot::dotty_decorator](#).

12.103.4.3 virtual std::string spot::tgba_run_dotty_decorator::state_decl (const [tgba](#) * *a*, const [state](#) * *s*, int *n*, [tgba_succ_iterator](#) * *si*, const std::string & *label*) [virtual]

Compute the style of a state.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL"] with LABEL replaced by the value of *label*.

Parameters:

- a* the automaton being drawn
- s* the state being drawn (owned by the caller)
- n* a unique number for this state
- si* an iterator over the successors of this state (owned by the caller, but can be freely iterated)
- label* the computed name of this state

Reimplemented from [spot::dotty_decorator](#).

12.103.5 Member Data Documentation

12.103.5.1 [step_map](#) spot::tgba_run_dotty_decorator::map_ [private]

12.103.5.2 `const tgba_run* spot::tgba_run_dotty_decorator::run_` [private]

The documentation for this class was generated from the following file:

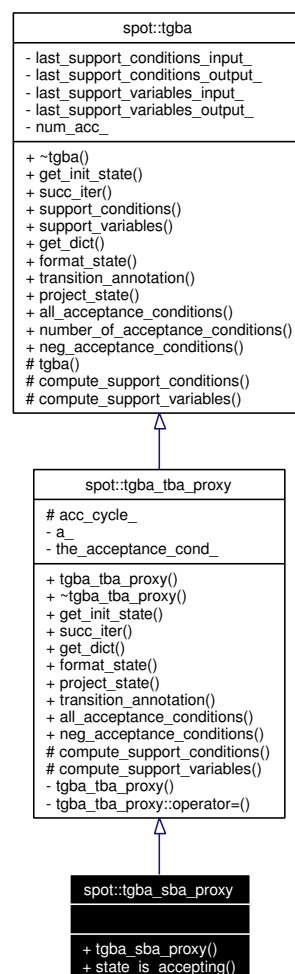
- `tgbaalgos/rundotdec.hh`

12.104 spot::tgba_sba_proxy Class Reference

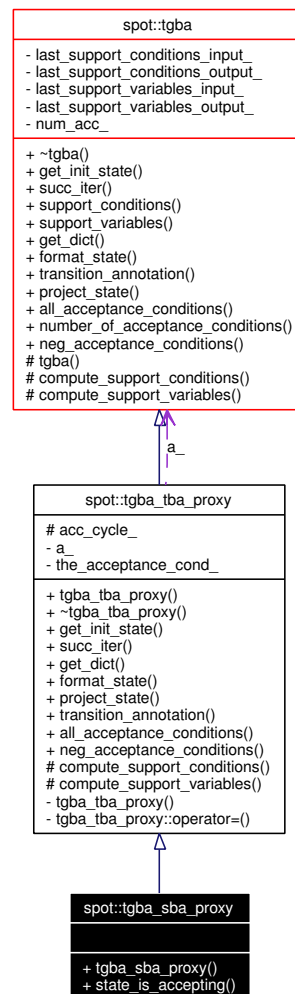
Degeneralize a `spot::tgba` on the fly, producing an SBA.

```
#include <tgba/tgbatba.hh>
```

Inheritance diagram for `spot::tgba_sba_proxy`:



Collaboration diagram for `spot::tgba_sba_proxy`:



Public Types

- `typedef std::list< bdd > cycle_list`

Public Member Functions

- `tgba_sba_proxy` (const `tgba` *a)
- `bool state_is_accepting` (const `state` *state) const
Whether the state is accepting.
- virtual `state` * `get_init_state` () const
Get the initial state of the automaton.
- virtual `tgba_succ_iterator` * `succ_iter` (const `state` *local_state, const `state` *global_state=0, const `tgba` *global_automaton=0) const
Get an iterator over the successors of local_state.
- virtual `bdd_dict` * `get_dict` () const

Get the dictionary associated to the automaton.

- virtual std::string [format_state](#) (const [state](#) *state) const
Format the state as a string for printing.
- virtual [state](#) * [project_state](#) (const [state](#) *s, const [tgba](#) *t) const
Project a state on an automaton.
- virtual std::string [transition_annotation](#) (const [tgba_succ_iterator](#) *t) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual unsigned int [number_of_acceptance_conditions](#) () const
The number of acceptance conditions.

Protected Member Functions

- virtual bdd [compute_support_conditions](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_conditions\(\)](#).
- virtual bdd [compute_support_variables](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_variables\(\)](#).

Protected Attributes

- [cycle_list](#) [acc_cycle_](#)

12.104.1 Detailed Description

Degeneralize a [spot::tgba](#) on the fly, producing an SBA.

This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly.

This is similar to [tgba_tba_proxy](#), except that automata produced with this algorithms can also be seen as State-based Büchi Automata (SBA). See [tgba_sba_proxy::state_is_accepting\(\)](#). (An SBA is a TBA, and a TBA is a TGBA.)

This extra property has a small cost in size: if the input automaton uses N acceptance conditions, the output automaton can have at most max(N,1)+1 times more states and transitions. (This is only max(N,1) for [tgba_tba_proxy](#).)

12.104.2 Member Typedef Documentation

12.104.2.1 typedef std::list<bdd> [spot::tgba_tba_proxy::cycle_list](#) [inherited]

12.104.3 Constructor & Destructor Documentation

12.104.3.1 spot::tgba_sba_proxy::tgba_sba_proxy (const [tgba](#) * a)

12.104.4 Member Function Documentation

12.104.4.1 virtual bdd spot::tgba_tba_proxy::all_acceptance_conditions () const [virtual, inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

12.104.4.2 virtual bdd spot::tgba_tba_proxy::compute_support_conditions (const [state](#) * state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support_conditions\(\)](#).

Implements [spot::tgba](#).

12.104.4.3 virtual bdd spot::tgba_tba_proxy::compute_support_variables (const [state](#) * state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support_variables\(\)](#).

Implements [spot::tgba](#).

12.104.4.4 virtual std::string spot::tgba_tba_proxy::format_state (const [state](#) * state) const [virtual, inherited]

Format the state as a string for printing.

This formatting is the responsibility of the automata who owns the state.

Implements [spot::tgba](#).

12.104.4.5 virtual [bdd_dict](#)* spot::tgba_tba_proxy::get_dict () const [virtual, inherited]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

12.104.4.6 `virtual state* spot::tgba_tba_proxy::get_init_state () const [virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

12.104.4.7 `virtual bdd spot::tgba_tba_proxy::neg_acceptance_conditions () const [virtual, inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg_acceptance_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

12.104.4.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

12.104.4.9 `virtual state* spot::tgba_tba_proxy::project_state (const state * s, const tgba * t) const [virtual, inherited]`

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

Returns:

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

12.104.4.10 `bool spot::tgba_sba_proxy::state_is_accepting (const state * state) const`

Whether the state is accepting.

A particularity of a `spot::tgba_sba_proxy` automaton is that when a state has an outgoing accepting arc, all its outgoing arcs are accepting. The state itself can therefore be considered accepting. This is useful in algorithms working on degeneralized automata with state acceptance conditions.

12.104.4.11 `virtual tgba_succ_iterator* spot::tgba_tba_proxy::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const [virtual, inherited]`

Get an iterator over the successors of `local_state`.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. `global_automaton` designate the root `spot::tgba`, and `global_state` its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, `global_state` is not adopted by `succ_iter`.

global_automaton In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

12.104.4.12 bdd spot::tgba::support_conditions (const state * state) const [inherited]

Get a formula that must hold whatever successor is taken.

Returns:

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

12.104.4.13 bdd spot::tgba::support_variables (const state * state) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

12.104.4.14 virtual std::string spot::tgba_tba_proxy::transition_annotation (const tgba_succ_iterator * t) const [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

Parameters:

t a non-done `tgba_succ_iterator` for this automata

Reimplemented from `spot::tgba`.

12.104.5 Member Data Documentation

12.104.5.1 [cycle_list](#) [spot::tgba_tba_proxy::acc_cycle_](#) [protected, inherited]

The documentation for this class was generated from the following file:

- [tgba/tgbatba.hh](#)

12.105 spot::tgba_statistics Struct Reference

```
#include <tgbaalgos/stats.hh>
```

Public Attributes

- unsigned [transitions](#)
- unsigned [states](#)

12.105.1 Member Data Documentation

12.105.1.1 unsigned [spot::tgba_statistics::states](#)

12.105.1.2 unsigned [spot::tgba_statistics::transitions](#)

The documentation for this struct was generated from the following file:

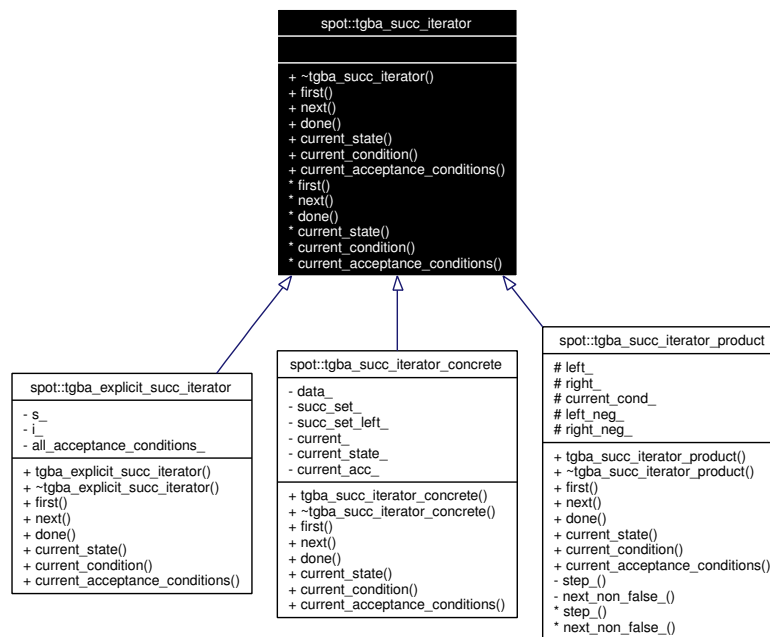
- [tgbaalgos/stats.hh](#)

12.106 spot::tgba_succ_iterator Class Reference

Iterate over the successors of a state.

```
#include <tgba/succiter.hh>
```

Inheritance diagram for `spot::tgba_succ_iterator`:



Public Member Functions

- virtual `~tgba_succ_iterator()`

Iteration

- virtual void `first()` = 0
Position the iterator on the first successor (if any).
- virtual void `next()` = 0
Jump to the next successor (if any).
- virtual bool `done()` const = 0
Check whether the iteration is finished.

Inspection

- virtual `state * current_state()` const = 0
Get the state of the current successor.
- virtual bdd `current_condition()` const = 0
Get the condition on the transition leading to this successor.
- virtual bdd `current_acceptance_conditions()` const = 0
Get the acceptance conditions on the transition leading to this successor.

12.106.1 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.

12.106.2 Constructor & Destructor Documentation

12.106.2.1 `virtual spot::tgba_succ_iterator::~tgba_succ_iterator () [inline, virtual]`

12.106.3 Member Function Documentation

12.106.3.1 `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](#).

12.106.3.2 `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](#).

12.106.3.3 `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](#).

12.106.3.4 `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](#).

12.106.3.5 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`.

12.106.3.6 virtual void spot::tgba_succ_iterator::next () [pure virtual]

Jump to the next successor (if any).

Warning:

Again, one should always call `done ()` to ensure there is a successor.

Implemented in `spot::tgba_succ_iterator_concrete`, `spot::tgba_explicit_succ_iterator`, and `spot::tgba_succ_iterator_product`.

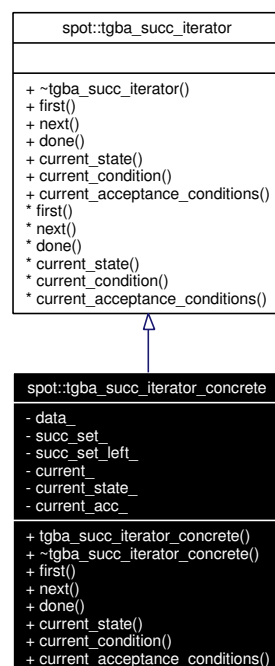
The documentation for this class was generated from the following file:

- [tgba/succiter.hh](#)

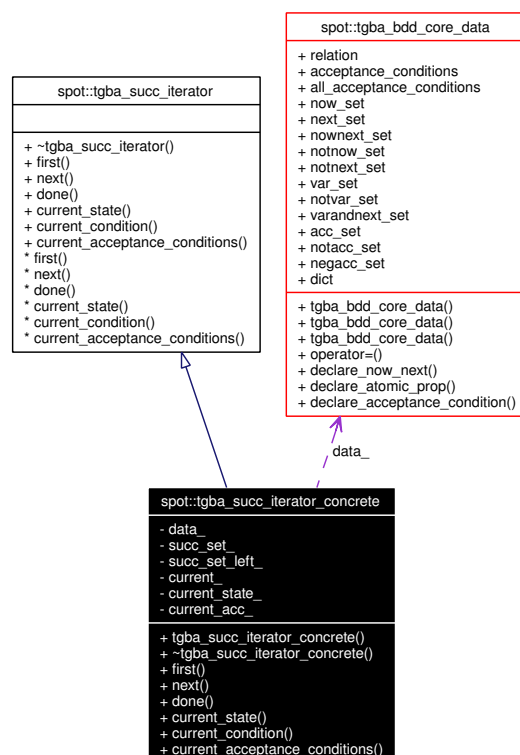
12.107 spot::tgba_succ_iterator_concrete Class Reference

```
#include <tgba/succiterconcrete.hh>
```

Inheritance diagram for `spot::tgba_succ_iterator_concrete`:



Collaboration diagram for spot::tgba_succ_iterator_concrete:



Public Member Functions

- [tgba_succ_iterator_concrete](#) (const [tgba_bdd_core_data](#) &d, bdd successors)
Build a `spot::tgba_succ_iterator_concrete`.
- virtual `~tgba_succ_iterator_concrete ()`
- void [first](#) ()
Position the iterator on the first successor (if any).
- void [next](#) ()
Jump to the next successor (if any).
- bool [done](#) () const
Check whether the iteration is finished.
- `state_bdd * current_state ()` const
Get the state of the current successor.
- bdd [current_condition](#) () const
Get the condition on the transition leading to this successor.
- bdd [current_acceptance_conditions](#) () const
Get the acceptance conditions on the transition leading to this successor.

Private Attributes

- const [tgba_bdd_core_data](#) & [data_](#)
Core data of the automaton.
- bdd [succ_set_](#)
The set of successors.
- bdd [succ_set_left_](#)
Unexplored successors (including current_).
- bdd [current_](#)
Current successor, as a conjunction of atomic proposition and Next variables.
- bdd [current_state_](#)
Current successor, as a conjunction of Now variables.
- bdd [current_acc_](#)
Acceptance conditions for the current transition.

12.107.1 Detailed Description

A concrete iterator over successors of a TGBA state.

12.107.2 Constructor & Destructor Documentation

12.107.2.1 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.

12.107.2.2 virtual spot::tgba_succ_iterator_concrete::~~tgba_succ_iterator_concrete () [virtual]

12.107.3 Member Function Documentation

12.107.3.1 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](#).

12.107.3.2 `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](#).

12.107.3.3 `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](#).

12.107.3.4 `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](#).

12.107.3.5 `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](#).

12.107.3.6 `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](#).

12.107.4 Member Data Documentation**12.107.4.1** `bdd spot::tgba_succ_iterator_concrete::current_` [private]

Current successor, as a conjunction of atomic proposition and Next variables.

12.107.4.2 bdd [spot::tgba_succ_iterator_concrete::current_acc_](#) [private]

Acceptance conditions for the current transition.

12.107.4.3 bdd [spot::tgba_succ_iterator_concrete::current_state_](#) [private]

Current successor, as a conjunction of Now variables.

12.107.4.4 const [tgba_bdd_core_data&](#) [spot::tgba_succ_iterator_concrete::data_](#) [private]

Core data of the automaton.

12.107.4.5 bdd [spot::tgba_succ_iterator_concrete::succ_set_](#) [private]

The set of successors.

12.107.4.6 bdd [spot::tgba_succ_iterator_concrete::succ_set_left_](#) [private]

Unexplored successors (including current_).

The documentation for this class was generated from the following file:

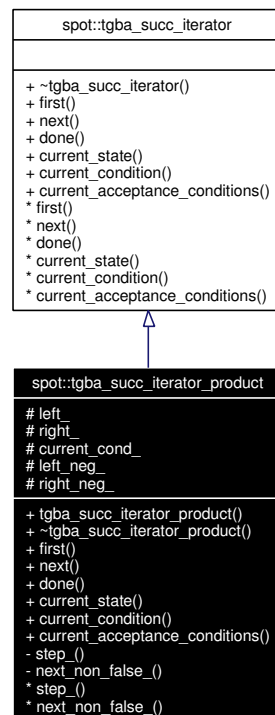
- [tgba/succiterconcrete.hh](#)

12.108 spot::tgba_succ_iterator_product Class Reference

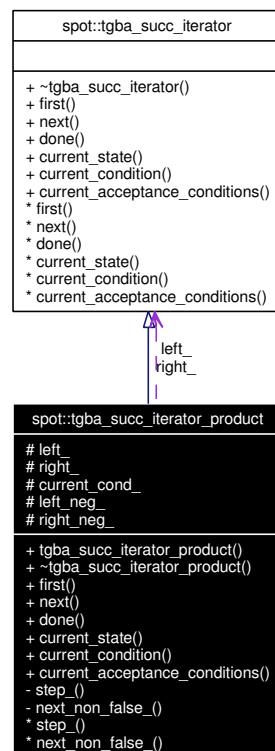
Iterate over the successors of a product computed on the fly.

```
#include <tgba/tgbaproduct.hh>
```

Inheritance diagram for spot::tgba_succ_iterator_product:



Collaboration diagram for `spot::tgba_succ_iterator_product`:



Public Member Functions

- [tgba_succ_iterator_product](#) ([tgba_succ_iterator](#) *left, [tgba_succ_iterator](#) *right, bdd left_neg, bdd right_neg)
- virtual [~tgba_succ_iterator_product](#) ()
- void [first](#) ()
Position the iterator on the first successor (if any).
- void [next](#) ()
Jump to the next successor (if any).
- bool [done](#) () const
Check whether the iteration is finished.
- [state_product](#) * [current_state](#) () const
Get the state of the current successor.
- bdd [current_condition](#) () const
Get the condition on the transition leading to this successor.
- bdd [current_acceptance_conditions](#) () const
Get the acceptance conditions on the transition leading to this successor.

Protected Attributes

- [tgba_succ_iterator](#) * left_
- [tgba_succ_iterator](#) * right_
- bdd [current_cond_](#)
- bdd [left_neg_](#)
- bdd [right_neg_](#)

Private Member Functions

- void [step_](#) ()
Internal routines to advance to the next successor.
- void [next_non_false_](#) ()

Friends

- class [tgba_product](#)

12.108.1 Detailed Description

Iterate over the successors of a product computed on the fly.

12.108.2 Constructor & Destructor Documentation

12.108.2.1 `spot::tgba_succ_iterator_product::tgba_succ_iterator_product (tgba_succ_iterator * left, tgba_succ_iterator * right, bdd left_neg, bdd right_neg)`

12.108.2.2 `virtual spot::tgba_succ_iterator_product::~tgba_succ_iterator_product ()` [virtual]

12.108.3 Member Function Documentation

12.108.3.1 `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](#).

12.108.3.2 `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](#).

12.108.3.3 `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](#).

12.108.3.4 `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](#).

12.108.3.5 `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`.

12.108.3.6 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`.

12.108.3.7 void spot::tgba_succ_iterator_product::next_non_false() [private]**12.108.3.8 void spot::tgba_succ_iterator_product::step() [private]**

Internal routines to advance to the next successor.

12.108.4 Friends And Related Function Documentation**12.108.4.1 friend class tgba_product [friend]****12.108.5 Member Data Documentation****12.108.5.1 bdd spot::tgba_succ_iterator_product::current_cond_ [protected]****12.108.5.2 tgba_succ_iterator* spot::tgba_succ_iterator_product::left_ [protected]****12.108.5.3 bdd spot::tgba_succ_iterator_product::left_neg_ [protected]****12.108.5.4 tgba_succ_iterator* spot::tgba_succ_iterator_product::right_ [protected]****12.108.5.5 bdd spot::tgba_succ_iterator_product::right_neg_ [protected]**

The documentation for this class was generated from the following file:

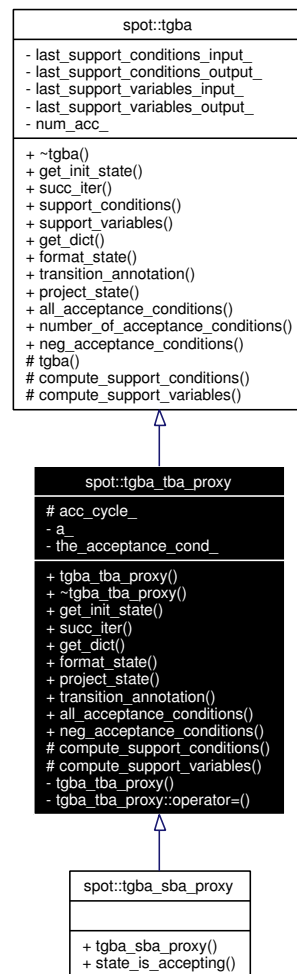
- `tgba/tgbaproduct.hh`

12.109 spot::tgba_tba_proxy Class Reference

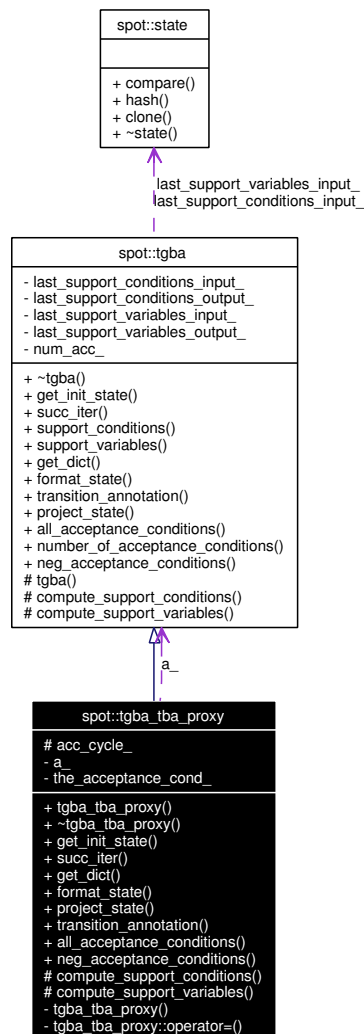
Degeneralize a `spot::tgba` on the fly, producing a TBA.

```
#include <tgba/tgbatba.hh>
```

Inheritance diagram for `spot::tgba_tba_proxy`:



Collaboration diagram for spot::tgba_tba_proxy:



Public Types

- `typedef std::list< bdd > cycle_list`

Public Member Functions

- `tgba_tba_proxy (const tgba *a)`
- `virtual ~tgba_tba_proxy ()`
- `virtual state * get_init_state () const`
Get the initial state of the automaton.
- `virtual tgba_succ_iterator * succ_iter (const state *local_state, const state *global_state=0, const tgba *global_automaton=0) const`
Get an iterator over the successors of local_state.
- `virtual bdd_dict * get_dict () const`
Get the dictionary associated to the automaton.

- virtual std::string [format_state](#) (const [state](#) *state) const
Format the state as a string for printing.
- virtual [state](#) * [project_state](#) (const [state](#) *s, const [tgba](#) *t) const
Project a state on an automaton.
- virtual std::string [transition_annotation](#) (const [tgba_succ_iterator](#) *t) const
Return a possible annotation for the transition pointed to by the iterator.
- virtual bdd [all_acceptance_conditions](#) () const
Return the set of all acceptance conditions used by this automaton.
- virtual bdd [neg_acceptance_conditions](#) () const
Return the conjunction of all negated acceptance variables.
- bdd [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- bdd [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- virtual unsigned int [number_of_acceptance_conditions](#) () const
The number of acceptance conditions.

Protected Member Functions

- virtual bdd [compute_support_conditions](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_conditions\(\)](#).
- virtual bdd [compute_support_variables](#) (const [state](#) *state) const
Do the actual computation of [tgba::support_variables\(\)](#).

Protected Attributes

- [cycle_list](#) [acc_cycle_](#)

Private Member Functions

- [tgba_tba_proxy](#) (const [tgba_tba_proxy](#) &)
- [tgba_tba_proxy](#) & [tgba_tba_proxy::operator=](#) (const [tgba_tba_proxy](#) &)

Private Attributes

- const [tgba](#) * [a_](#)
- bdd [the_acceptance_cond_](#)

12.109.1 Detailed Description

Degeneralize a [spot::tgba](#) on the fly, producing a TBA.

This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly. The result is still a [spot::tgba](#), but it will always have exactly one acceptance condition so it could be called TBA (without the G).

The degeneralization is done by synchronizing the input automaton with a "counter" automaton such as the one shown in "On-the-fly Verification of Linear Temporal Logic" (Jean-Michel Couvreur, FME99).

If the input automaton uses N acceptance conditions, the output automaton can have at most max(N,1) times more states and transitions.

See also:

[tgba_sba_proxy](#)

12.109.2 Member Typedef Documentation

12.109.2.1 `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list`

12.109.3 Constructor & Destructor Documentation

12.109.3.1 `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba * a)`

12.109.3.2 `virtual spot::tgba_tba_proxy::~tgba_tba_proxy ()` [virtual]

12.109.3.3 `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba_tba_proxy &)` [private]

12.109.4 Member Function Documentation

12.109.4.1 `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](#).

12.109.4.2 `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](#).

12.109.4.3 `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](#).

12.109.4.4 `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](#).

12.109.4.5 `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](#).

12.109.4.6 `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](#).

12.109.4.7 `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](#).

12.109.4.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

12.109.4.9 `virtual state* spot::tgba_tba_proxy::project_state (const state * s, const tgba * t) const [virtual]`

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occurring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

Returns:

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

12.109.4.10 virtual [tgba_succ_iterator](#)* [spot::tgba_tba_proxy::succ_iter](#) (const [state](#) * *local_state*, const [state](#) * *global_state* = 0, const [tgba](#) * *global_automaton* = 0) const [virtual]

Get an iterator over the successors of *local_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global_automaton* designate the root [spot::tgba](#), and *global_state* its state. This two objects can be used by [succ_iter\(\)](#) to restrict the set of successors to compute.

Parameters:

local_state The state whose successors are to be explored. This pointer is not adopted in any way by [succ_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *locale_state*, *global_state* is not adopted by [succ_iter](#).

global_automaton In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

12.109.4.11 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.

12.109.4.12 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.

12.109.4.13 [tgba_tba_proxy](#)& [spot::tgba_tba_proxy::tgba_tba_proxy::operator=](#) (const [tgba_tba_proxy](#) &) [private]

12.109.4.14 virtual std::string [spot::tgba_tba_proxy::transition_annotation](#) (const [tgba_succ_iterator](#) * *t*) const [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

Parameters:

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented from [spot::tgba](#).

12.109.5 Member Data Documentation

12.109.5.1 `const tgba* spot::tgba_tba_proxy::a_` [private]

12.109.5.2 `cycle_list spot::tgba_tba_proxy::acc_cycle_` [protected]

12.109.5.3 `bdd spot::tgba_tba_proxy::the_acceptance_cond_` [private]

The documentation for this class was generated from the following file:

- [tgba/tgbatba.hh](#)

12.110 `spot::time_info` Struct Reference

A structure to record elapsed time in clock ticks.

```
#include <misc/timer.hh>
```

Public Member Functions

- [time_info](#) ()

Public Attributes

- `clock_t` [utime](#)
- `clock_t` [stime](#)

12.110.1 Detailed Description

A structure to record elapsed time in clock ticks.

12.110.2 Constructor & Destructor Documentation

12.110.2.1 `spot::time_info::time_info ()` [inline]

12.110.3 Member Data Documentation

12.110.3.1 `clock_t spot::time_info::stime`

12.110.3.2 `clock_t spot::time_info::utime`

The documentation for this struct was generated from the following file:

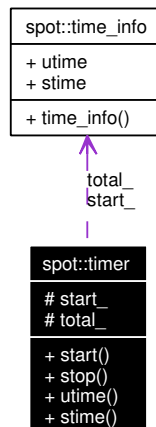
- [misc/timer.hh](#)

12.111 spot::timer Class Reference

A timekeeper that accumulate interval of time.

```
#include <misc/timer.hh>
```

Collaboration diagram for spot::timer:



Public Member Functions

- void [start](#) ()
Start a time interval.
- void [stop](#) ()
Stop a time interval and update the sum of all intervals.
- clock_t [utime](#) () const
Return the user time of all accumulated interval.
- clock_t [stime](#) () const
Return the system time of all accumulated interval.

Protected Attributes

- [time_info](#) [start_](#)
- [time_info](#) [total_](#)

12.111.1 Detailed Description

A timekeeper that accumulate interval of time.

12.111.2 Member Function Documentation

12.111.2.1 void spot::timer::start () [inline]

Start a time interval.

12.111.2.2 `clock_t spot::timer::time () const` [inline]

Return the system time of all accumulated interval.

Any time interval that has been `start()`ed but not `stop()`ed will not be accounted for.

12.111.2.3 `void spot::timer::stop ()` [inline]

Stop a time interval and update the sum of all intervals.

12.111.2.4 `clock_t spot::timer::utime () const` [inline]

Return the user time of all accumulated interval.

Any time interval that has been `start()`ed but not `stop()`ed will not be accounted for.

12.111.3 Member Data Documentation**12.111.3.1** `time_info spot::timer::start_` [protected]**12.111.3.2** `time_info spot::timer::total_` [protected]

The documentation for this class was generated from the following file:

- `misc/timer.hh`

12.112 `spot::timer_map` Class Reference

A map of timer, where each timer has a name.

```
#include <misc/timer.hh>
```

Public Member Functions

- `void start (const std::string &name)`
Start a timer with name name.
- `void stop (const std::string &name)`
Stop timer name.
- `void cancel (const std::string &name)`
Cancel timer name.
- `const spot::timer & timer (const std::string &name) const`
Return the timer name.
- `spot::timer & timer (const std::string &name)`
Return the timer name.
- `bool empty () const`
Whether there is no timer in the map.

- `std::ostream & print (std::ostream &os) const`

Format information about all timers in a table.

Protected Types

- `typedef std::pair< spot::timer, int > item_type`
- `typedef std::map< std::string, item_type > tm_type`

Protected Attributes

- `tm_type tm`

12.112.1 Detailed Description

A map of timer, where each timer has a name.

Timer_map also keeps track of the number of measures each timer has performed.

12.112.2 Member Typedef Documentation

12.112.2.1 `typedef std::pair<spot::timer, int> spot::timer_map::item_type [protected]`

12.112.2.2 `typedef std::map<std::string, item_type> spot::timer_map::tm_type [protected]`

12.112.3 Member Function Documentation

12.112.3.1 `void spot::timer_map::cancel (const std::string & name) [inline]`

Cancel timer *name*.

The timer must have been previously started with [start\(\)](#).

This cancel only the current measure. (Previous measures recorded by the timer are preserved.) When a timer that has not done any measure is canceled, it is removed from the map.

12.112.3.2 `bool spot::timer_map::empty () const [inline]`

Whether there is no timer in the map.

If [empty\(\)](#) return true, then either no timer where ever started, or all started timers were canceled without completing any measure.

12.112.3.3 `std::ostream& spot::timer_map::print (std::ostream & os) const`

Format information about all timers in a table.

12.112.3.4 `void spot::timer_map::start (const std::string & name)` `[inline]`

Start a timer with name *name*.

The timer is created if it did not exist already. Once started, a timer should be either `stop()`ed or `cancel()`ed.

12.112.3.5 `void spot::timer_map::stop (const std::string & name)` `[inline]`

Stop timer *name*.

The timer must have been previously started with `start()`.

12.112.3.6 `spot::timer& spot::timer_map::timer (const std::string & name)` `[inline]`

Return the timer *name*.

12.112.3.7 `const spot::timer& spot::timer_map::timer (const std::string & name) const` `[inline]`

Return the timer *name*.

12.112.4 Member Data Documentation**12.112.4.1** `tm_type spot::timer_map::tm` `[protected]`

The documentation for this class was generated from the following file:

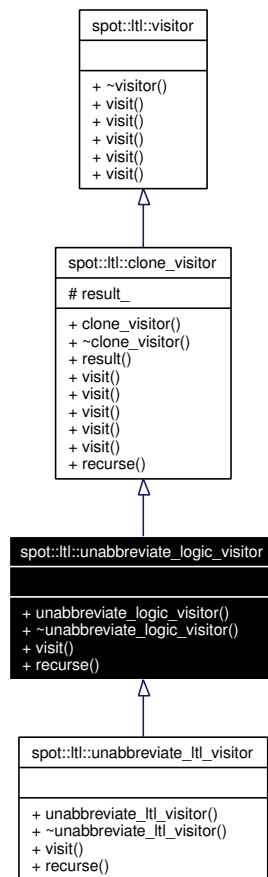
- `misc/timer.hh`

12.113 `spot::ltl::unabbreviate_logic_visitor` Class Reference

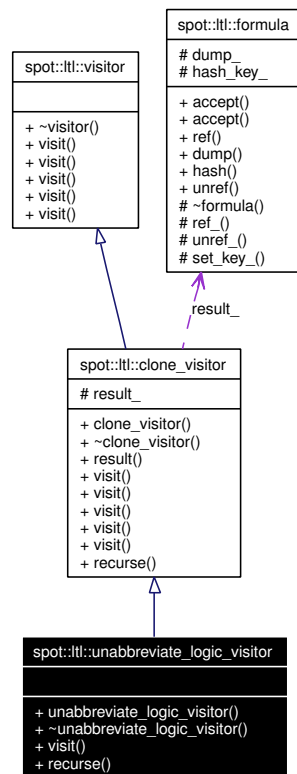
Clone and rewrite a formula to remove most of the abbreviated logical operators.

```
#include <ltlvisit/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)`
- `formula * result() const`
- `void visit(atomic_prop *ap)`
- `void visit(unop *uo)`
- `void visit(multop *mo)`
- `void visit(constant *c)`

Protected Attributes

- `formula * result_`

Private Types

- `typedef clone_visitor super`

12.113.1 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.

12.113.2 Member Typedef Documentation

12.113.2.1 typedef [clone_visitor](#) [spot::ltl::unabbreviate_logic_visitor::super](#) [private]

Reimplemented in [spot::ltl::unabbreviate_ltl_visitor](#).

12.113.3 Constructor & Destructor Documentation

12.113.3.1 [spot::ltl::unabbreviate_logic_visitor::unabbreviate_logic_visitor](#) ()

12.113.3.2 virtual [spot::ltl::unabbreviate_logic_visitor::~~unabbreviate_logic_visitor](#) ()
[virtual]

12.113.4 Member Function Documentation

12.113.4.1 virtual [formula*](#) [spot::ltl::unabbreviate_logic_visitor::recurse](#) ([formula](#) * *f*)
[virtual]

Reimplemented from [spot::ltl::clone_visitor](#).

Reimplemented in [spot::ltl::unabbreviate_ltl_visitor](#).

12.113.4.2 [formula*](#) [spot::ltl::clone_visitor::result](#) () const [inherited]

12.113.4.3 void [spot::ltl::clone_visitor::visit](#) ([constant](#) * *c*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

12.113.4.4 void [spot::ltl::clone_visitor::visit](#) ([multop](#) * *mo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

12.113.4.5 void [spot::ltl::clone_visitor::visit](#) ([unop](#) * *uo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

Reimplemented in [spot::ltl::unabbreviate_ltl_visitor](#).

12.113.4.6 void [spot::ltl::clone_visitor::visit](#) ([atomic_prop](#) * *ap*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

12.113.4.7 void [spot::ltl::unabbreviate_logic_visitor::visit](#) ([binop](#) * *bo*) [virtual]

Reimplemented from [spot::ltl::clone_visitor](#).

12.113.5 Member Data Documentation

12.113.5.1 formula* spot::ltl::clone_visitor::result_ [protected, inherited]

The documentation for this class was generated from the following file:

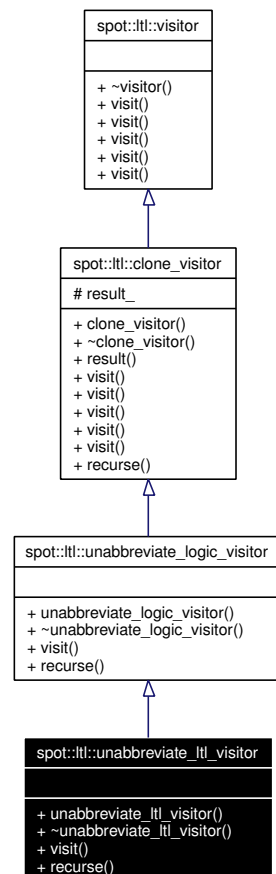
- [ltlvisit/unabbrev.hh](#)

12.114 spot::ltl::unabbreviate_ltl_visitor Class Reference

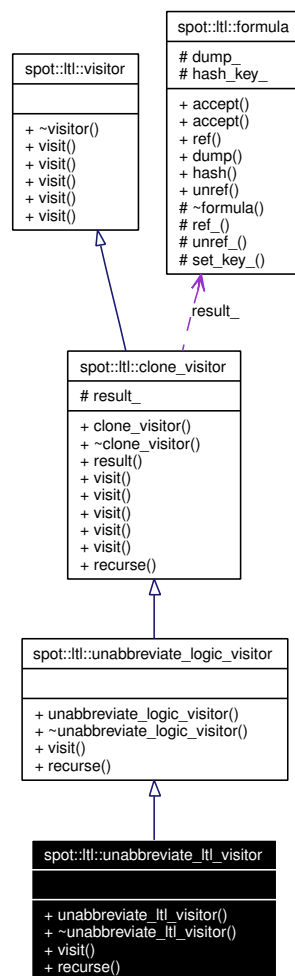
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

```
#include <ltlvisit/tunabbrev.hh>
```

Inheritance diagram for spot::ltl::unabbreviate_ltl_visitor:



Collaboration diagram for spot::ltl::unabbreviate_ltl_visitor:



Public Member Functions

- `unabbreviate_ltl_visitor()`
- `virtual ~unabbreviate_ltl_visitor()`
- `void visit (unop *uo)`
- `formula * recurse (formula *f)`
- `void visit (binop *bo)`
- `void visit (atomic_prop *ap)`
- `void visit (multop *mo)`
- `void visit (constant *c)`
- `formula * result () const`

Protected Attributes

- `formula * result_`

Private Types

- `typedef unabbreviate_logic_visitor super`

12.114.1 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.

12.114.2 Member Typedef Documentation

12.114.2.1 `typedef unabbreviate_logic_visitor spot::ltl::unabbreviate_ltl_visitor::super [private]`

Reimplemented from [spot::ltl::unabbreviate_logic_visitor](#).

12.114.3 Constructor & Destructor Documentation

12.114.3.1 `spot::ltl::unabbreviate_ltl_visitor::unabbreviate_ltl_visitor ()`

12.114.3.2 `virtual spot::ltl::unabbreviate_ltl_visitor::~~unabbreviate_ltl_visitor () [virtual]`

12.114.4 Member Function Documentation

12.114.4.1 `formula* spot::ltl::unabbreviate_ltl_visitor::recurse (formula *f) [virtual]`

Reimplemented from [spot::ltl::unabbreviate_logic_visitor](#).

12.114.4.2 `formula* spot::ltl::clone_visitor::result () const [inherited]`

12.114.4.3 `void spot::ltl::clone_visitor::visit (constant *c) [virtual, inherited]`

Implements [spot::ltl::visitor](#).

12.114.4.4 `void spot::ltl::clone_visitor::visit (multop *mo) [virtual, inherited]`

Implements [spot::ltl::visitor](#).

12.114.4.5 `void spot::ltl::clone_visitor::visit (atomic_prop *ap) [virtual, inherited]`

Implements [spot::ltl::visitor](#).

12.114.4.6 `void spot::ltl::unabbreviate_logic_visitor::visit (binop *bo) [virtual, inherited]`

Reimplemented from [spot::ltl::clone_visitor](#).

12.114.4.7 void spot::ltl::unabbreviate_ltl_visitor::visit ([unop](#) * *uo*) [virtual]

Reimplemented from [spot::ltl::clone_visitor](#).

12.114.5 Member Data Documentation

12.114.5.1 formula* spot::ltl::clone_visitor::result_ [protected, inherited]

The documentation for this class was generated from the following file:

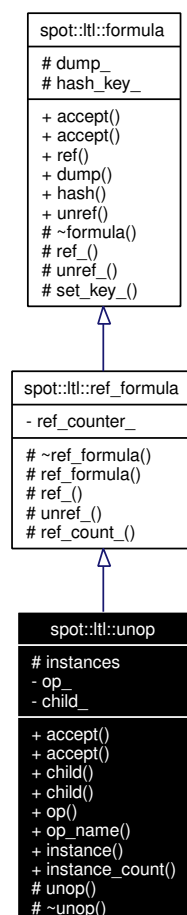
- ltlvisit/[tunabbrev.hh](#)

12.115 spot::ltl::unop Class Reference

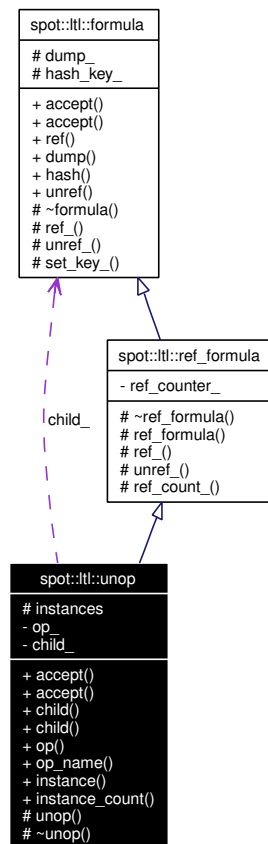
Unary operators.

```
#include <ltlast/unop.hh>
```

Inheritance diagram for spot::ltl::unop:



Collaboration diagram for spot::ltl::unop:



Public Types

- enum `type` { `Not`, `X`, `F`, `G` }

Public Member Functions

- virtual void `accept` (`visitor` &`v`)
Entry point for `vspot::ltl::visitor` instances.
- virtual void `accept` (`const_visitor` &`v`) const
Entry point for `vspot::ltl::const_visitor` instances.
- const `formula` * `child` () const
Get the sole operand of this operator.
- `formula` * `child` ()
Get the sole operand of this operator.
- `type` `op` () const
Get the type of this operator.
- const char * `op_name` () const
Get the type of this operator, as a string.

- `formula * ref ()`
clone this node
- `const std::string & dump () const`
Return a canonic representation of the formula.
- `const size_t hash () const`
Return a hash_key for the formula.

Static Public Member Functions

- static `unop * instance (type op, formula *child)`
- static unsigned `instance_count ()`
Number of instantiated unary operators. For debugging.
- static void `unref (formula *f)`
release this node

Protected Types

- `typedef std::pair< type, formula * > pair`
- `typedef std::map< pair, formula * > map`

Protected Member Functions

- `unop (type op, formula *child)`
- virtual `~unop ()`
- void `ref_ ()`
increment reference counter if any
- bool `unref_ ()`
decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).
- unsigned `ref_count_ ()`
Number of references to this formula.
- void `set_key_ ()`
Compute key_ from dump_.

Protected Attributes

- std::string [dump_](#)
The canonic representation of the formula.
- size_t [hash_key_](#)
The hash key of this formula.

Static Protected Attributes

- static [map instances](#)

Private Attributes

- [type op_](#)
- [formula * child_](#)

12.115.1 Detailed Description

Unary operators.

12.115.2 Member Typedef Documentation

12.115.2.1 typedef std::map<[pair](#), [formula*](#)> [spot::ltl::unop::map](#) [protected]

12.115.2.2 typedef std::pair<[type](#), [formula*](#)> [spot::ltl::unop::pair](#) [protected]

12.115.3 Member Enumeration Documentation

12.115.3.1 enum [spot::ltl::unop::type](#)

Enumeration values:

Not

X

F

G

12.115.4 Constructor & Destructor Documentation

12.115.4.1 [spot::ltl::unop::unop](#) ([type op](#), [formula * child](#)) [protected]

12.115.4.2 virtual [spot::ltl::unop::~~unop](#) () [protected, virtual]

12.115.5 Member Function Documentation

12.115.5.1 virtual void spot::ltl::unop::accept (const_visitor & v) const [virtual]

Entry point for vspot::ltl::const_visitor instances.

Implements [spot::ltl::formula](#).

12.115.5.2 virtual void spot::ltl::unop::accept (visitor & v) [virtual]

Entry point for vspot::ltl::visitor instances.

Implements [spot::ltl::formula](#).

12.115.5.3 formula* spot::ltl::unop::child ()

Get the sole operand of this operator.

12.115.5.4 const formula* spot::ltl::unop::child () const

Get the sole operand of this operator.

12.115.5.5 const std::string& spot::ltl::formula::dump () const [inherited]

Return a canonic representation of the formula.

12.115.5.6 const size_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash_key for the formula.

12.115.5.7 static unop* spot::ltl::unop::instance (type op, formula * child) [static]

Build an unary operator with operation *op* and child *child*.

12.115.5.8 static unsigned spot::ltl::unop::instance_count () [static]

Number of instantiated unary operators. For debugging.

12.115.5.9 type spot::ltl::unop::op () const

Get the type of this operator.

12.115.5.10 const char* spot::ltl::unop::op_name () const

Get the type of this operator, as a string.

12.115.5.11 formula* spot::ltl::formula::ref () [inherited]

clone this node

This increments the reference counter of this node (if one is used). You should almost never use this method directly as it doesn't touch the children. If you want to clone a whole formula, use [spot::ltl::clone\(\)](#) instead.

12.115.5.12 `void spot::ltl::ref_formula::ref_()` [protected, virtual, inherited]

increment reference counter if any

Reimplemented from [spot::ltl::formula](#).

12.115.5.13 `unsigned spot::ltl::ref_formula::ref_count_()` [protected, inherited]

Number of references to this formula.

12.115.5.14 `void spot::ltl::formula::set_key_()` [protected, inherited]

Compute `key_` from `dump_`.

Should be called once in each object, after `dump_` has been set.

12.115.5.15 `static void spot::ltl::formula::unref(formula *f)` [static, inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object. You should almost never use this method directly as it doesn't touch the children. If you want to release a whole formula, use [spot::ltl::destroy\(\)](#) instead.

12.115.5.16 `bool spot::ltl::ref_formula::unref_()` [protected, virtual, inherited]

decrement reference counter if any, return true when the instance must be deleted (usually when the counter hits 0).

Reimplemented from [spot::ltl::formula](#).

12.115.6 Member Data Documentation**12.115.6.1** `formula* spot::ltl::unop::child_` [private]**12.115.6.2** `std::string spot::ltl::formula::dump_` [protected, inherited]

The canonic representation of the formula.

12.115.6.3 `size_t spot::ltl::formula::hash_key_` [protected, inherited]

The hash key of this formula.

Initialized by [set_key_\(\)](#).

12.115.6.4 `map spot::ltl::unop::instances` [static, protected]**12.115.6.5** `type spot::ltl::unop::op_` [private]

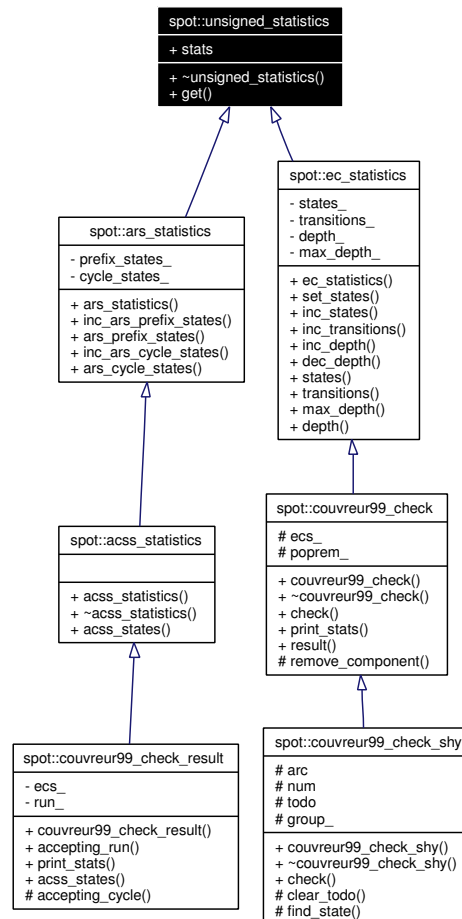
The documentation for this class was generated from the following file:

- [ltlast/unop.hh](#)

12.116 spot::unsigned_statistics Struct Reference

```
#include <tgbaalgos/emptiness_stats.hh>
```

Inheritance diagram for spot::unsigned_statistics:



Public Types

- typedef unsigned(unsigned_statistics::* [unsigned_fun](#))() const
- typedef std::map< const char *, [unsigned_fun](#), [char_ptr_less_than](#) > [stats_map](#)

Public Member Functions

- virtual [~unsigned_statistics](#) ()
- unsigned [get](#) (const char *str) const

Public Attributes

- [stats_map](#) stats

12.116.1 Member Typedef Documentation

12.116.1.1 `typedef std::map<const char*, unsigned_fun, char_ptr_less_than> spot::unsigned_statistics::stats_map`

12.116.1.2 `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const`

12.116.2 Constructor & Destructor Documentation

12.116.2.1 `virtual spot::unsigned_statistics::~~unsigned_statistics () [inline, virtual]`

12.116.3 Member Function Documentation

12.116.3.1 `unsigned spot::unsigned_statistics::get (const char * str) const [inline]`

12.116.4 Member Data Documentation

12.116.4.1 `stats_map spot::unsigned_statistics::stats`

The documentation for this struct was generated from the following file:

- `tgbaalgos/emptiness_stats.hh`

12.117 `spot::unsigned_statistics_copy` Class Reference

comparable statistics

```
#include <tgbaalgos/emptiness_stats.hh>
```

Public Types

- `typedef std::map< const char *, unsigned, char_ptr_less_than > stats_map`

Public Member Functions

- `unsigned_statistics_copy ()`
- `unsigned_statistics_copy (const unsigned_statistics &o)`
- `bool seteq (const unsigned_statistics &o)`
- `bool operator== (const unsigned_statistics_copy &o) const`
- `bool operator!= (const unsigned_statistics_copy &o) const`

Public Attributes

- `stats_map stats`
- `bool set`

12.117.1 Detailed Description

comparable statistics

This must be built from a [spot::unsigned_statistics](#). But unlike [spot::unsigned_statistics](#), it supports equality and inequality tests. (It's the only operations it supports, BTW.)

12.117.2 Member Typedef Documentation

12.117.2.1 `typedef std::map<const char*, unsigned, char_ptr_less_than> spot::unsigned_statistics_copy::stats_map`

12.117.3 Constructor & Destructor Documentation

12.117.3.1 `spot::unsigned_statistics_copy::unsigned_statistics_copy () [inline]`

12.117.3.2 `spot::unsigned_statistics_copy::unsigned_statistics_copy (const unsigned_statistics & o) [inline]`

12.117.4 Member Function Documentation

12.117.4.1 `bool spot::unsigned_statistics_copy::operator!= (const unsigned_statistics_copy & o) const [inline]`

12.117.4.2 `bool spot::unsigned_statistics_copy::operator== (const unsigned_statistics_copy & o) const [inline]`

12.117.4.3 `bool spot::unsigned_statistics_copy::seteq (const unsigned_statistics & o) [inline]`

12.117.5 Member Data Documentation

12.117.5.1 `bool spot::unsigned_statistics_copy::set`

12.117.5.2 `stats_map spot::unsigned_statistics_copy::stats`

The documentation for this class was generated from the following file:

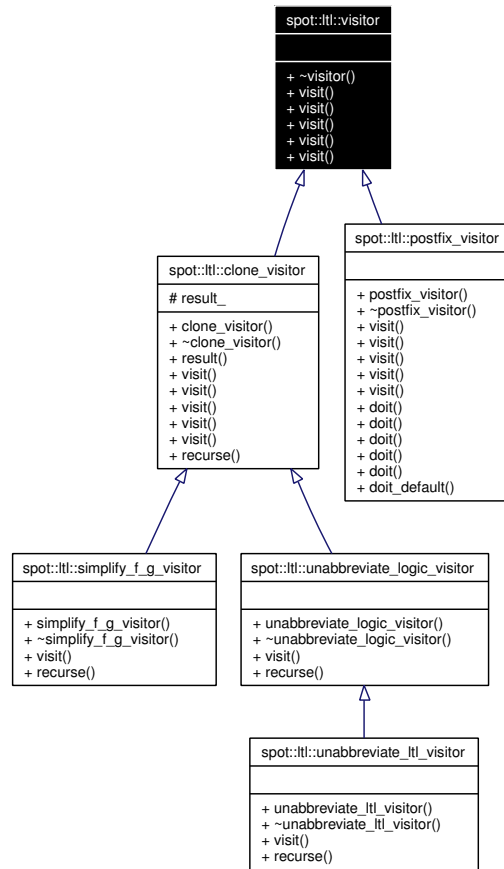
- [tgbaalgos/emptiness_stats.hh](#)

12.118 spot::ltl::visitor Struct Reference

Formula visitor that can modify the formula.

```
#include <ltlast/visitor.hh>
```

Inheritance diagram for `spot::ltl::visitor`:



Public Member Functions

- virtual `~visitor()`
- 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`

12.118.1 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.

12.118.2 Constructor & Destructor Documentation

12.118.2.1 virtual `spot::ltl::visitor::~~visitor()` `[inline, virtual]`

12.118.3 Member Function Documentation

12.118.3.1 virtual void spot::ltl::visitor::visit (multop * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), and [spot::ltl::postfix_visitor](#).

12.118.3.2 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](#).

12.118.3.3 virtual void spot::ltl::visitor::visit (binop * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), [spot::ltl::unabbreviate_logic_visitor](#), [spot::ltl::postfix_visitor](#), and [spot::ltl::simplify_f_g_visitor](#).

12.118.3.4 virtual void spot::ltl::visitor::visit (constant * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), and [spot::ltl::postfix_visitor](#).

12.118.3.5 virtual void spot::ltl::visitor::visit (atomic_prop * node) [pure virtual]

Implemented in [spot::ltl::clone_visitor](#), and [spot::ltl::postfix_visitor](#).

The documentation for this struct was generated from the following file:

- [ltlast/visitor.hh](#)

12.119 spot::weight Class Reference

Manage for a given automaton a vector of counter indexed by its acceptance condition.

```
#include <tgbaaalgos/weight.hh>
```

Public Member Functions

- [weight](#) (const bdd &neg_all_cond)
- [weight](#) & [operator+=](#) (const bdd &acc)
Increment by one the counters of each acceptance condition in acc.
- [weight](#) & [operator-=](#) (const bdd &acc)
Decrement by one the counters of each acceptance condition in acc.
- bdd [operator-](#) (const [weight](#) &w) const

Private Types

- typedef std::map< int, int > [weight_vector](#)

Static Private Member Functions

- static void [inc_weight_handler](#) (char *varset, int size)
- static void [dec_weight_handler](#) (char *varset, int size)

Private Attributes

- [weight_vector](#) *m*
- bdd [neg_all_acc](#)

Static Private Attributes

- static [weight_vector](#) * *pm*

Friends

- `std::ostream & operator<< (std::ostream &os, const weight &w)`

12.119.1 Detailed Description

Manage for a given automaton a vector of counter indexed by its acceptance condition.

12.119.2 Member Typedef Documentation

12.119.2.1 `typedef std::map<int, int> spot::weight::weight_vector [private]`

12.119.3 Constructor & Destructor Documentation

12.119.3.1 `spot::weight::weight (const bdd & neg_all_cond)`

Construct a empty vector (all counters set to zero).

Parameters:

neg_all_cond : negation of all the acceptance conditions of the automaton (the bdd returned by [tgba::neg_acceptance_conditions\(\)](#)).

12.119.4 Member Function Documentation

12.119.4.1 `static void spot::weight::dec_weight_handler (char * varset, int size) [static, private]`

12.119.4.2 `static void spot::weight::inc_weight_handler (char * varset, int size) [static, private]`

12.119.4.3 `weight& spot::weight::operator+= (const bdd & acc)`

Increment by one the counters of each acceptance condition in *acc*.

12.119.4.4 `bdd spot::weight::operator- (const weight & w) const`

Return the set of each acceptance condition such that its counter is strictly greatest than the corresponding counter in *w*.

Precondition:

For each acceptance condition, its counter is greatest or equal to the corresponding counter in *w*.

12.119.4.5 `weight` & `spot::weight::operator-= (const bdd & acc)`

Decrement by one the counters of each acceptance condition in *acc*.

12.119.5 Friends And Related Function Documentation**12.119.5.1** `std::ostream& operator<< (std::ostream & os, const weight & w)` [*friend*]**12.119.6** Member Data Documentation**12.119.6.1** `weight_vector spot::weight::m` [*private*]**12.119.6.2** `bdd spot::weight::neg_all_acc` [*private*]**12.119.6.3** `weight_vector* spot::weight::pm` [*static, private*]

The documentation for this class was generated from the following file:

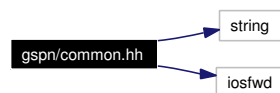
- `tgbaalgorithms/weight.hh`

13 spot File Documentation**13.1** `mainpage.dox` File Reference**13.2** `gspn/common.hh` File Reference

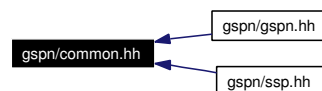
```
#include <string>
```

```
#include <iosfwd>
```

Include dependency graph for `common.hh`:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `spot`

Functions

- `std::ostream & operator<< (std::ostream &os, const gspn_exception &e)`

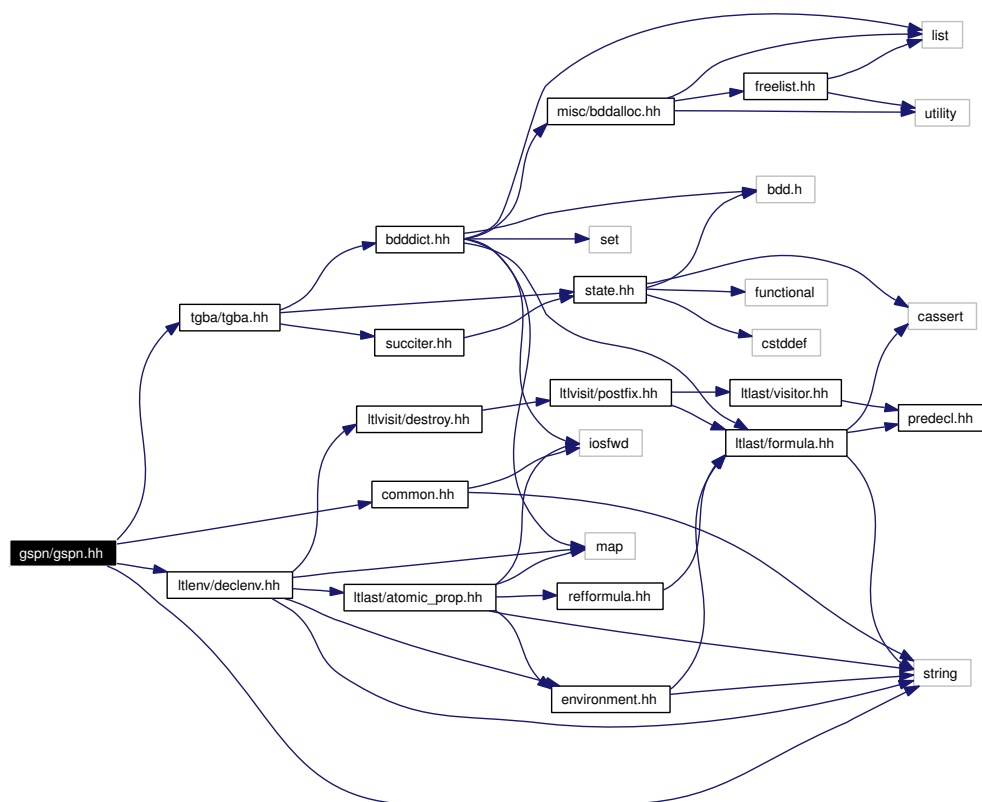
13.2.1 Function Documentation

13.2.1.1 `std::ostream& operator<< (std::ostream &os, const gspn_exception &e)`

13.3 gspn/gspn.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "ltlenv/declenv.hh"
```

Include dependency graph for gspn.hh:



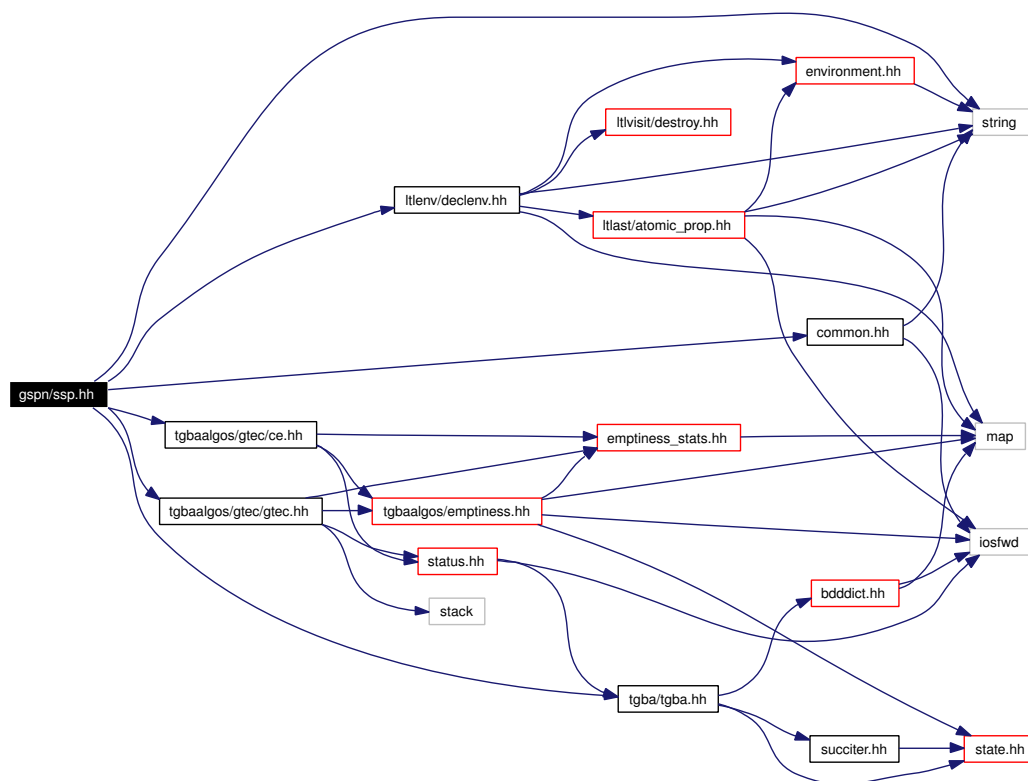
Namespaces

- namespace `spot`

13.4 gspn/ssp.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "tgbaalgos/gtec/gtec.hh"
#include "tgbaalgos/gtec/ce.hh"
#include "ltlenv/declenv.hh"
```

Include dependency graph for ssp.hh:



Namespaces

- namespace [spot](#)

Functions

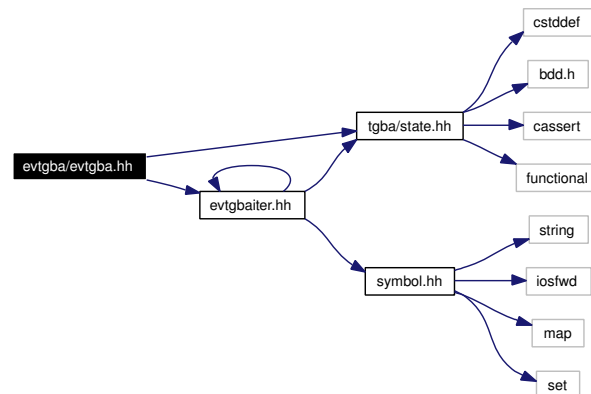
- `couvreur99_check * couvreur99_check_ssp_semi` (const tgba *ssp_automata)
- `couvreur99_check * couvreur99_check_ssp_shy_semi` (const tgba *ssp_automata)
- `couvreur99_check * couvreur99_check_ssp_shy` (const tgba *ssp_automata)

13.5 evtgba/evtgba.hh File Reference

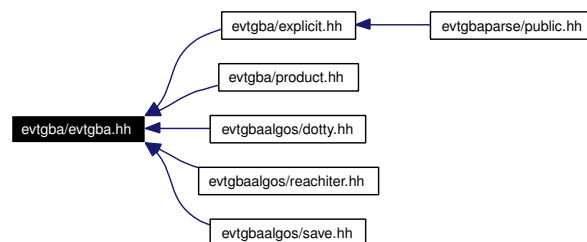
```
#include "tgba/state.hh"
```

```
#include "evtgbaiter.hh"
```

Include dependency graph for evtgba.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`

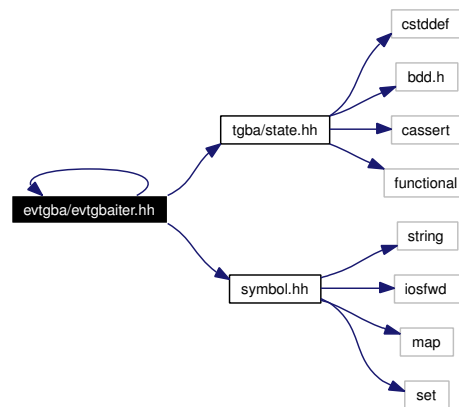
13.6 evtgba/evtgbaiter.hh File Reference

```
#include "tgba/state.hh"
```

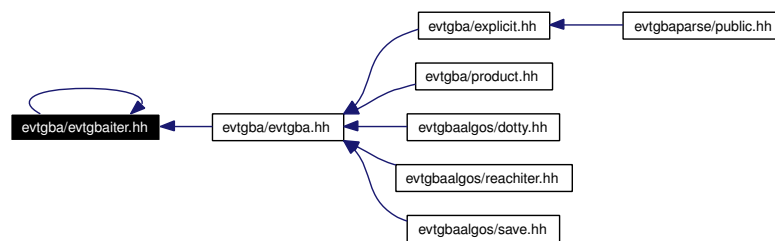
```
#include "symbol.hh"
```

```
#include "evtgbaiter.hh"
```

Include dependency graph for evtgbaiter.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`

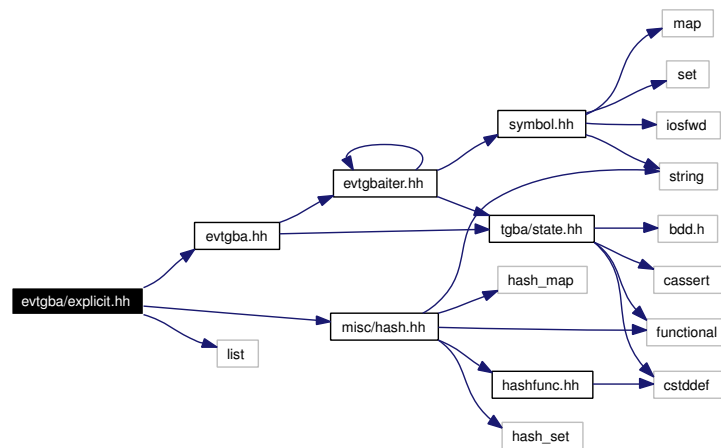
13.7 evtgba/explicit.hh File Reference

```
#include "evtgba.hh"
```

```
#include <list>
```

```
#include "misc/hash.hh"
```

Include dependency graph for `explicit.hh`:



This graph shows which files directly or indirectly include this file:



Namespaces

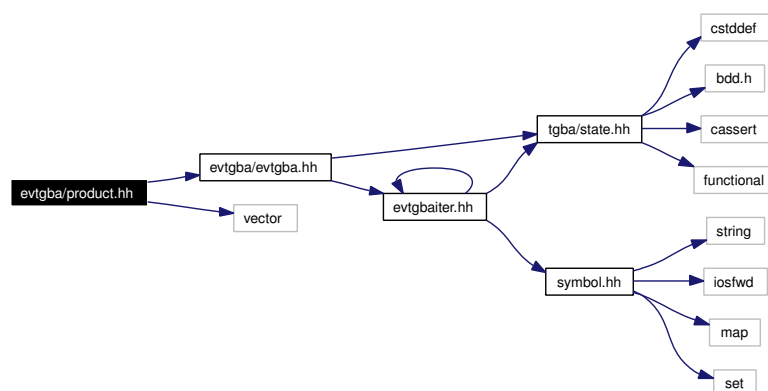
- namespace [spot](#)

13.8 evtgba/product.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <vector>
```

Include dependency graph for product.hh:



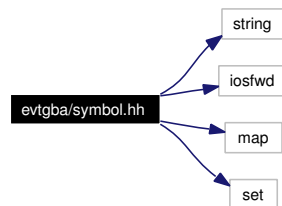
Namespaces

- namespace [spot](#)

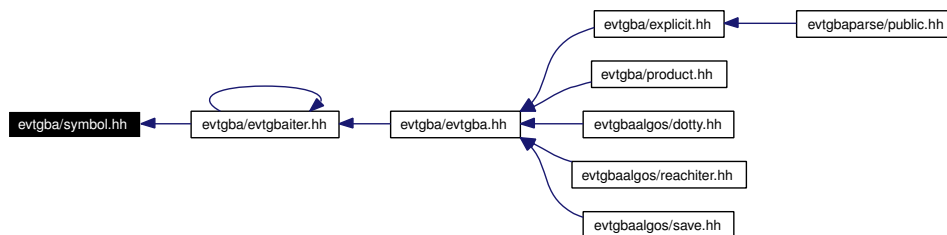
13.9 evtgba/symbol.hh File Reference

```
#include <string>
#include <iosfwd>
#include <map>
#include <set>
```

Include dependency graph for symbol.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

Typedefs

- typedef std::set< const symbol * > [symbol_set](#)
- typedef std::set< rsymbol > [rsymbol_set](#)

13.9.1 Typedef Documentation

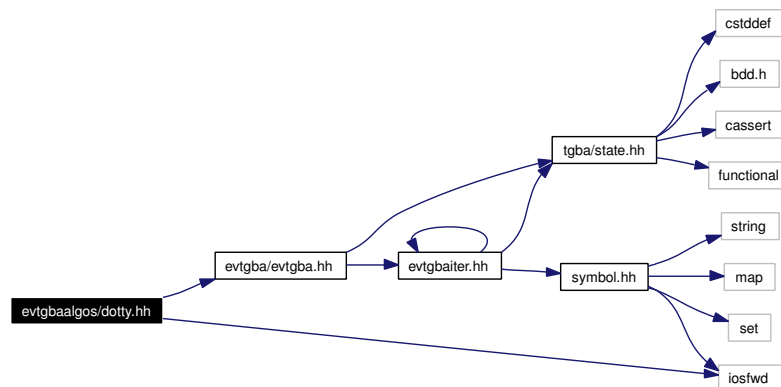
13.9.1.1 typedef std::set<rsymbol> [spot::rsymbol_set](#)

13.9.1.2 typedef std::set<const symbol*> [spot::symbol_set](#)

13.10 evtgbalgos/dotty.hh File Reference

```
#include "evtgba/evtgba.hh"
#include <iosfwd>
```

Include dependency graph for dotty.hh:



Namespaces

- namespace [spot](#)

Functions

- `std::ostream & dotty_reachable (std::ostream &os, const evtgba *g)`
Print reachable states in dot format.

13.10.1 Function Documentation

13.10.1.1 `std::ostream& dotty_reachable (std::ostream &os, const evtgba *g)`

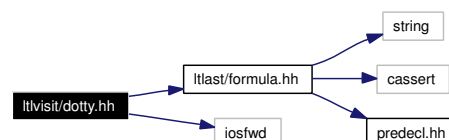
Print reachable states in dot format.

13.11 Itlvisit/dotty.hh File Reference

```
#include <ltlast/formula.hh>
```

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Functions

- `std::ostream & dotty` (`std::ostream &os`, `const formula *f`)

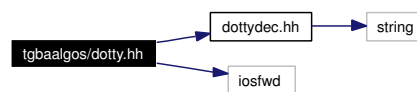
Write a formula tree using dot's syntax.

13.12 tgbaalgos/dotty.hh File Reference

```
#include "dottydec.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:

**Namespaces**

- namespace `spot`

Functions

- `std::ostream & dotty_reachable` (`std::ostream &os`, `const tgba *g`, `dotty_decorator *dd=dotty_decorator::instance()`)

Print reachable states in dot format.

13.13 evtgbaalgos/reachiter.hh File Reference

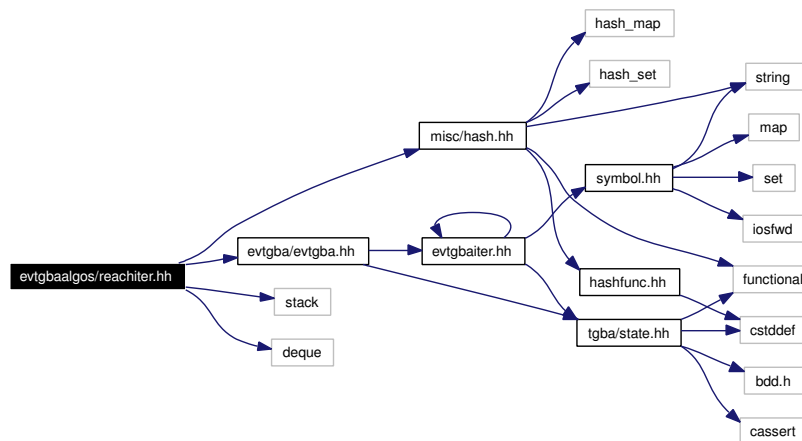
```
#include "misc/hash.hh"
```

```
#include "evtgba/evtgba.hh"
```

```
#include <stack>
```

```
#include <deque>
```

Include dependency graph for reachiter.hh:



Namespaces

- namespace [spot](#)

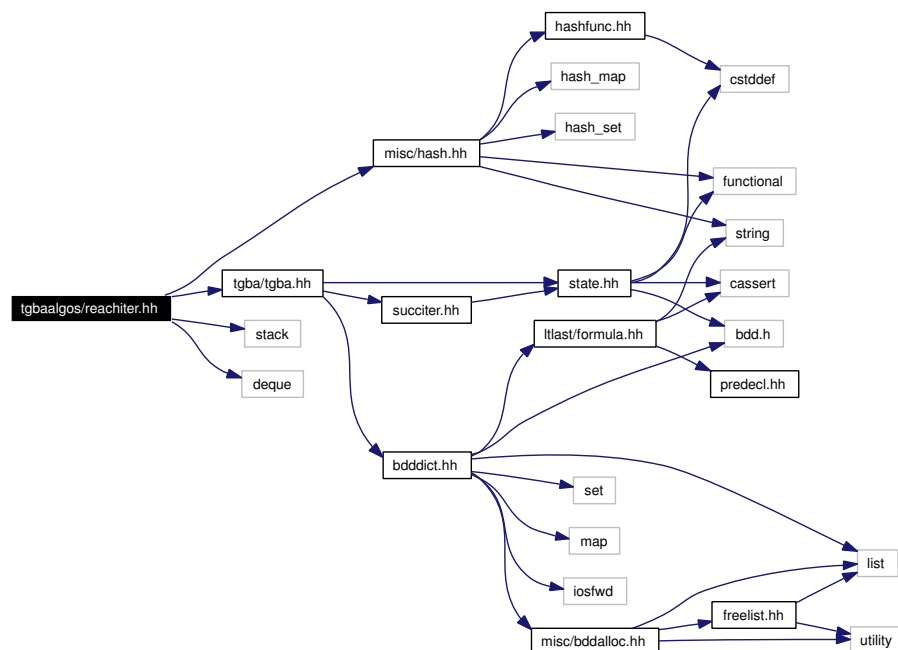
13.14 tgbaalgos/reachiter.hh File Reference

```

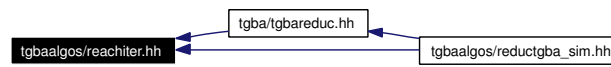
#include "misc/hash.hh"
#include "tgba/tgba.hh"
#include <stack>
#include <deque>

```

Include dependency graph for reachiter.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

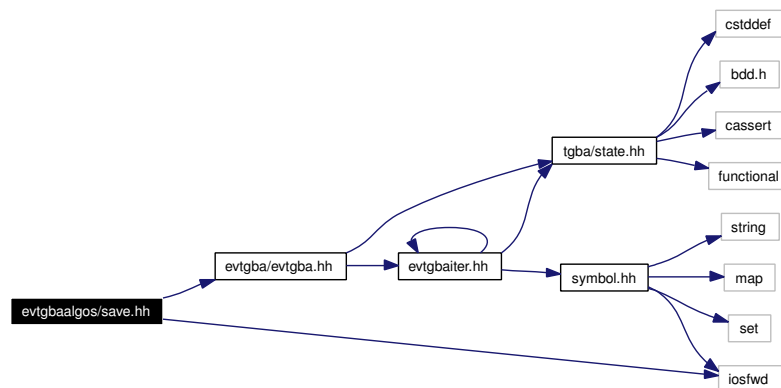
- namespace [spot](#)

13.15 evtgbaalgorithms/save.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



Namespaces

- namespace [spot](#)

Functions

- `std::ostream & evtgba_save_reachable (std::ostream &os, const evtgba *g)`
Save reachable states in text format.

13.15.1 Function Documentation

13.15.1.1 `std::ostream& evtgba_save_reachable (std::ostream &os, const evtgba *g)`

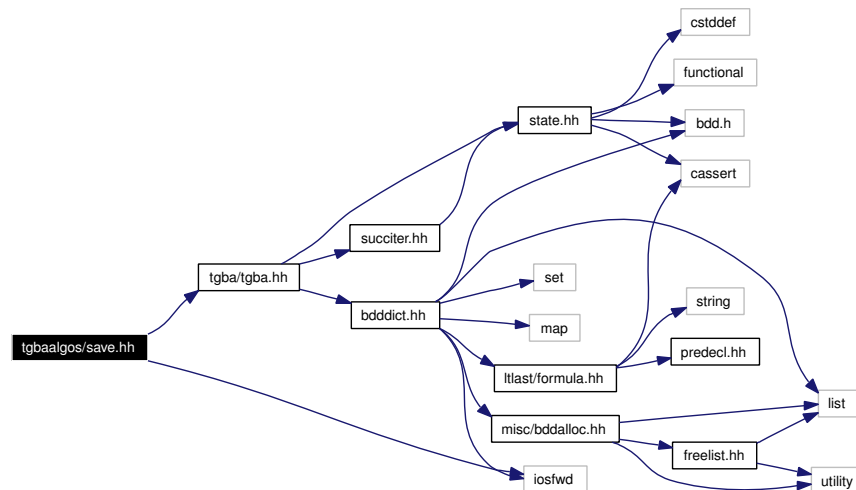
Save reachable states in text format.

13.16 tgbaalgos/save.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



Namespaces

- namespace [spot](#)

Functions

- `std::ostream & tgba_save_reachable (std::ostream &os, const tgba *g)`
Save reachable states in text format.

13.17 evtgbaalgos/tgba2evtgba.hh File Reference

Namespaces

- namespace [spot](#)

Functions

- `evtgba_explicit * tgba_to_evtgba (const tgba *a)`
Convert a tgba into an evtgba.

13.17.1 Function Documentation

13.17.1.1 evtgba_explicit* tgba_to_evtgba (const tgba * a)

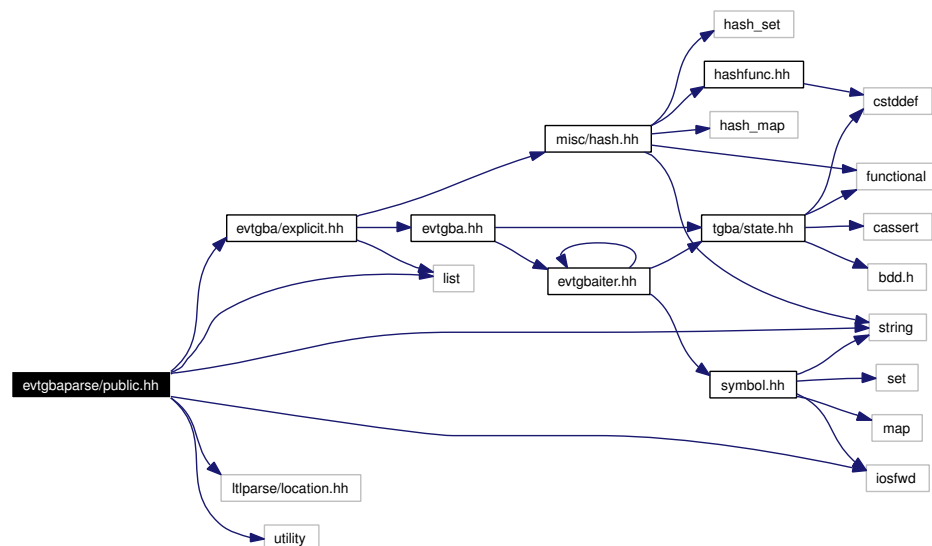
Convert a tgba into an evtgba.

(This cannot be done on-the-fly because the alphabet of a tgba is unknown beforehand.)

13.18 evtgba/parse/public.hh File Reference

```
#include "evtgba/explicit.hh"
#include "ltlparse/location.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



Namespaces

- namespace [spot](#)

Typedefs

- typedef std::pair< yy::location, std::string > [evtgba_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [evtgba_parse_error](#) > [evtgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.

Functions

- `evtgba_explicit * evtgba_parse` (`const std::string &filename`, `evtgba_parse_error_list &error_list`, `bool debug=false`)

Build a `spot::evtgba_explicit` from a text file.

- `bool format_evtgba_parse_errors` (`std::ostream &os`, `const std::string &filename`, `evtgba_parse_error_list &error_list` &`error_list`)

Format diagnostics produced by `spot::evtgba_parse`.

13.18.1 Typedef Documentation

13.18.1.1 `typedef std::pair<yy::location, std::string> spot::evtgba_parse_error`

A parse diagnostic with its location.

13.18.1.2 `typedef std::list<evtgba_parse_error> spot::evtgba_parse_error_list`

A list of parser diagnostics, as filled by parse.

13.18.2 Function Documentation

13.18.2.1 `evtgba_explicit* evtgba_parse` (`const std::string &filename`, `evtgba_parse_error_list &error_list`, `bool debug = false`)

Build a `spot::evtgba_explicit` from a text file.

Parameters:

filename The name of the file to parse.

error_list A list that will be filled with parse errors that occurred during parsing.

debug When true, causes the parser to trace its execution.

Returns:

A pointer to the `evtgba` built from *filename*, or 0 if the file could not be opened.

Note that the parser usually tries to recover from errors. It can return a non zero value even if it encountered error during the parsing of *filename*. If you want to make sure *filename* was parsed successfully, check *error_list* for emptiness.

Warning:

This function is not reentrant.

13.18.2.2 `bool format_evtgba_parse_errors` (`std::ostream & os`, `const std::string &filename`, `evtgba_parse_error_list &error_list` &`error_list`)

Format diagnostics produced by `spot::evtgba_parse`.

Parameters:

os Where diagnostics should be output.

filename The filename that should appear in the diagnostics.

error_list The error list filled by `spot::ltl::parse` while parsing *ltl_string*.

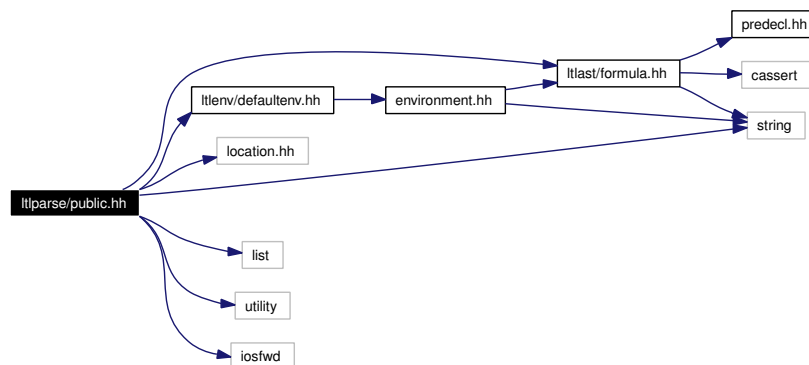
Returns:

`true` iff any diagnostic was output.

13.19 Itlparse/public.hh File Reference

```
#include "ltlast/formula.hh"
#include "location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

Typedefs

- typedef `std::pair< yy::location, std::string >` `parse_error`
A parse diagnostic with its location.
- typedef `std::list< parse_error >` `parse_error_list`
A list of parser diagnostics, as filled by `parse`.

Functions

- formula * [parse](#) (const std::string <l_string, [parse_error_list](#) &error_list, environment &env=default_environment::instance(), bool debug=false)

Build a formula from an LTL string.

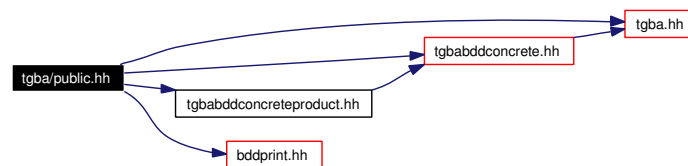
- bool [format_parse_errors](#) (std::ostream &os, const std::string <l_string, [parse_error_list](#) &error_list)

Format diagnostics produced by [spot::ltl::parse](#).

13.20 tgba/public.hh File Reference

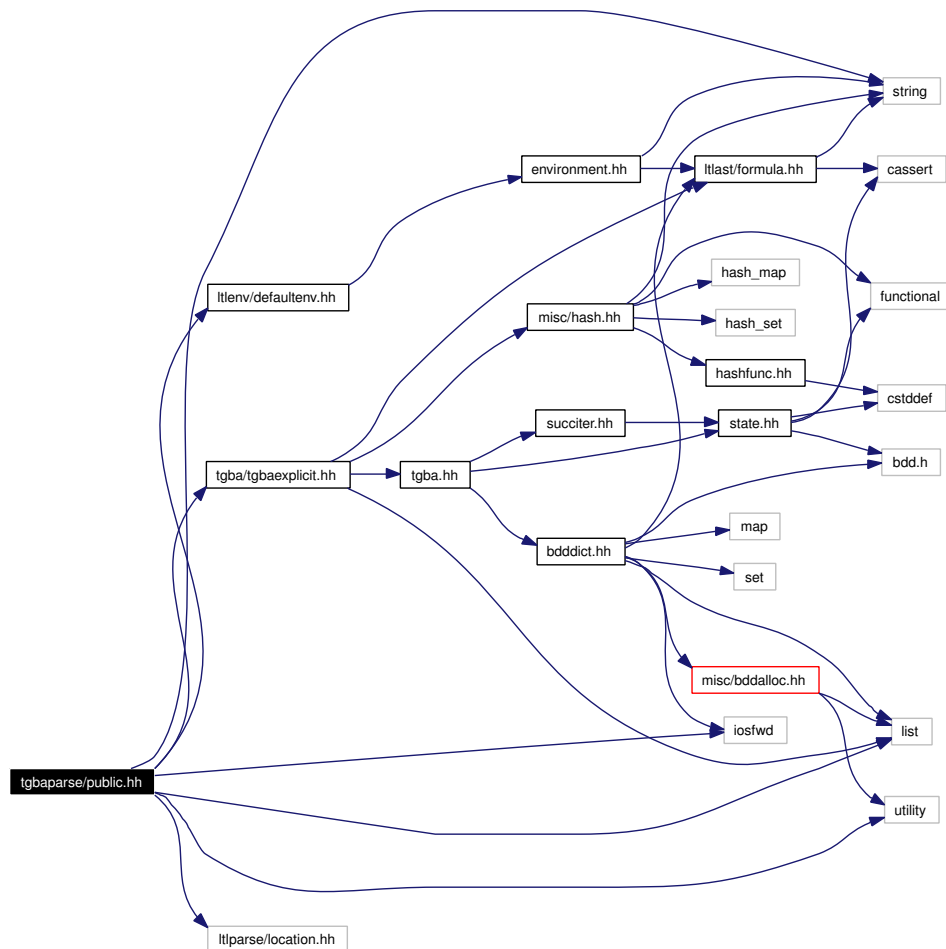
```
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include "tgbabddconcreteproduct.hh"
#include "bddprint.hh"
```

Include dependency graph for public.hh:

**13.21 tgbaparse/public.hh File Reference**

```
#include "tgba/tgbaexplicit.hh"
#include "ltlparse/location.hh"
#include "ltlenv/defaultenv.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



Namespaces

- namespace [spot](#)

Typedefs

- typedef std::pair< yy::location, std::string > [tgba_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [tgba_parse_error](#) > [tgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.

Functions

- tgba_explicit * [tgba_parse](#) (const std::string &filename, [tgba_parse_error_list](#) &error_list, bdd_dict *dict, ltl::environment &env=ltl::default_environment::instance(), bool debug=false)
Build a [spot::tgba_explicit](#) from a text file.

- bool `format_tgba_parse_errors` (std::ostream &os, const std::string &filename, `tgba_parse_error_list` &error_list)

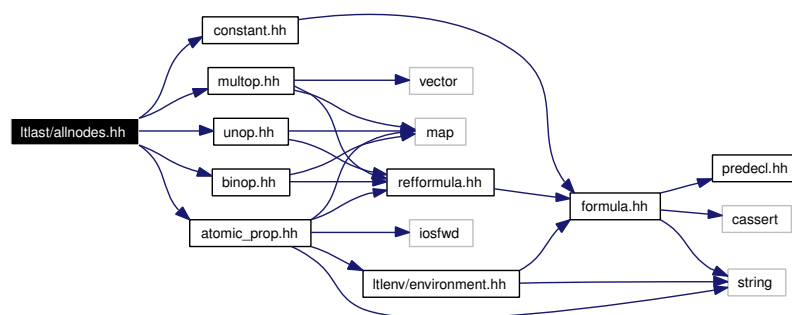
Format diagnostics produced by `spot::tgba_parse`.

13.22 Itlast/allnodes.hh File Reference

Define all LTL node types.

```
#include "binop.hh"
#include "unop.hh"
#include "multop.hh"
#include "atomic_prop.hh"
#include "constant.hh"
```

Include dependency graph for allnodes.hh:



13.22.1 Detailed Description

Define all LTL node types.

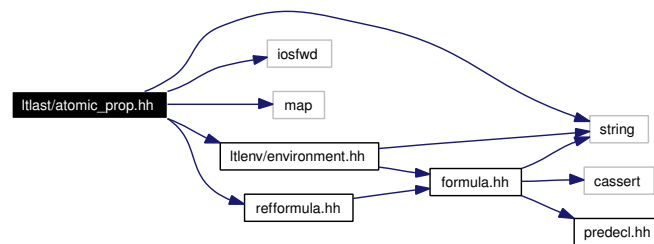
This file is usually needed when **defining** a visitor. Prefer `ltlast/predecl.hh` when only **declaring** methods and functions over LTL nodes.

13.23 Itlast/atomic_prop.hh File Reference

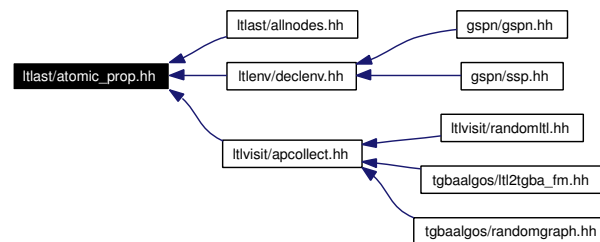
LTL atomic propositions.

```
#include <string>
#include <iosfwd>
#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](#)

13.23.1 Detailed Description

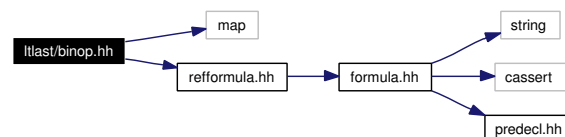
LTL atomic propositions.

13.24 Itlast/binop.hh File Reference

LTL binary operators.

```
#include <map>
#include "reformula.hh"
```

Include dependency graph for binop.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

13.24.1 Detailed Description

LTL binary operators.

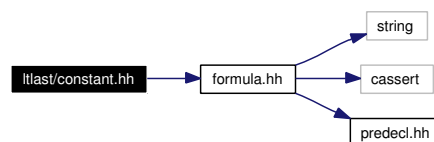
This does not include AND and OR operators. These are considered to be multi-operand operators (see [spot::ltl::multop](#)).

13.25 Itlast/constant.hh File Reference

LTL constants.

```
#include "formula.hh"
```

Include dependency graph for constant.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

13.25.1 Detailed Description

LTL constants.

13.26 Itlast/formula.hh File Reference

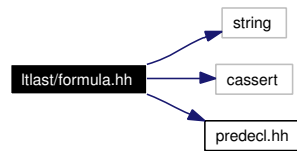
LTL formula interface.

```
#include <string>
```

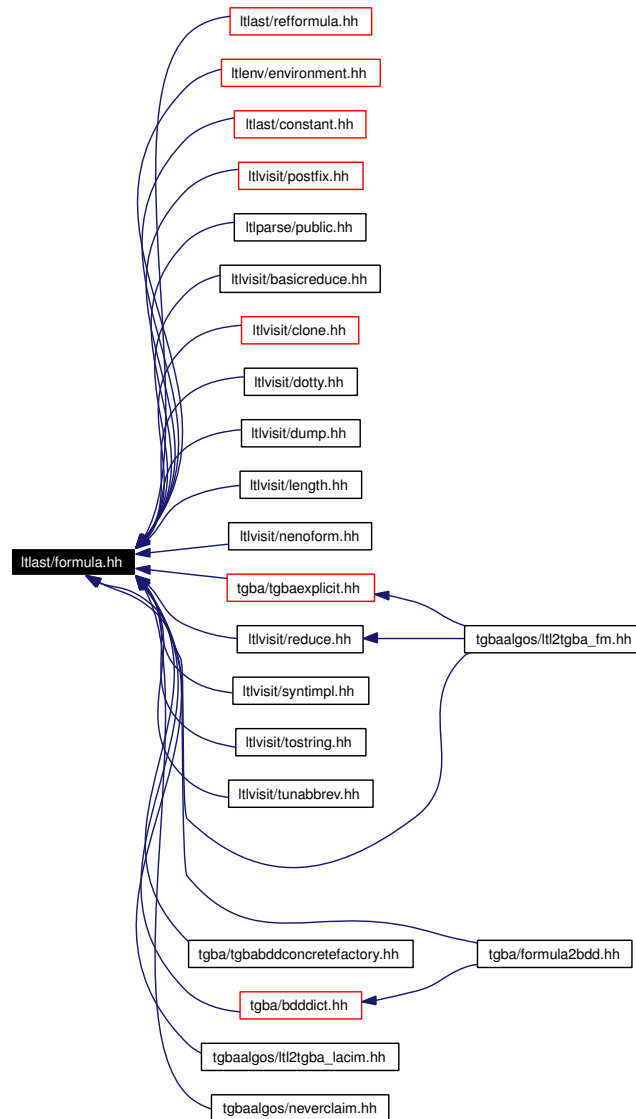
```
#include <cassert>
```

```
#include "predecl.hh"
```

Include dependency graph for formula.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltrl`

13.26.1 Detailed Description

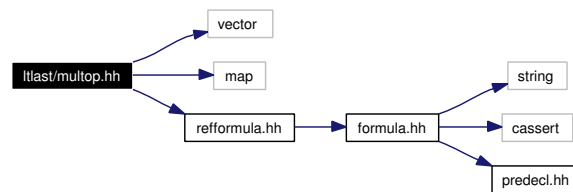
LTL formula interface.

13.27 Itlast/multop.hh File Reference

LTL multi-operand operators.

```
#include <vector>
#include <map>
#include "reformula.hh"
```

Include dependency graph for multop.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

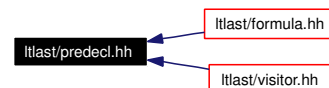
13.27.1 Detailed Description

LTL multi-operand operators.

13.28 Itlast/predecl.hh File Reference

Predeclare all LTL node types.

This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

13.28.1 Detailed Description

Predeclare all LTL node types.

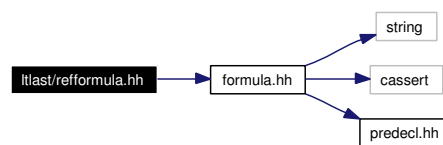
This file is usually used when **declaring** methods and functions over LTL nodes. Use `Itlast/allnodes.hh` or an individual header when the definition of the node is actually needed.

13.29 Itlast/reformula.hh File Reference

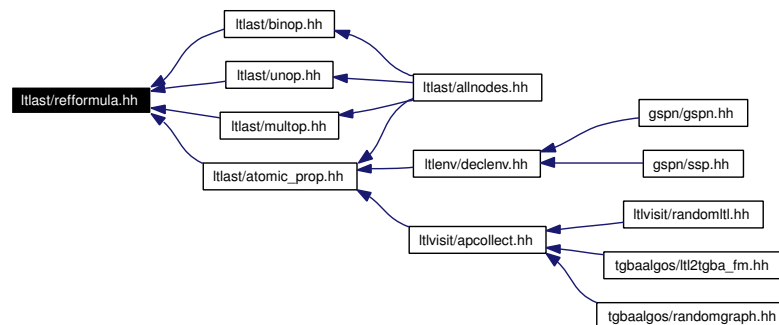
Reference-counted LTL formulae.

```
#include "formula.hh"
```

Include dependency graph for `reformula.hh`:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

13.29.1 Detailed Description

Reference-counted LTL formulae.

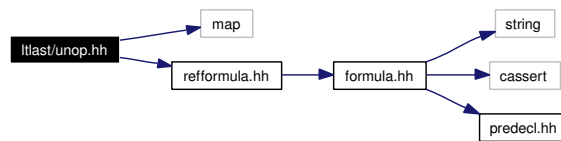
13.30 Itlast/unop.hh File Reference

LTL unary operators.

```
#include <map>
```

```
#include "reformula.hh"
```

Include dependency graph for `unop.hh`:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

13.30.1 Detailed Description

LTL unary operators.

13.31 Itlast/visitor.hh File Reference

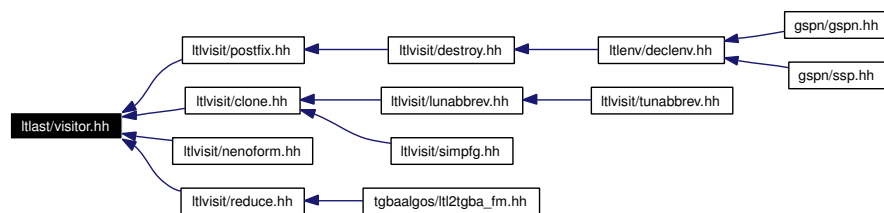
LTL visitor interface.

```
#include "predecl.hh"
```

Include dependency graph for visitor.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

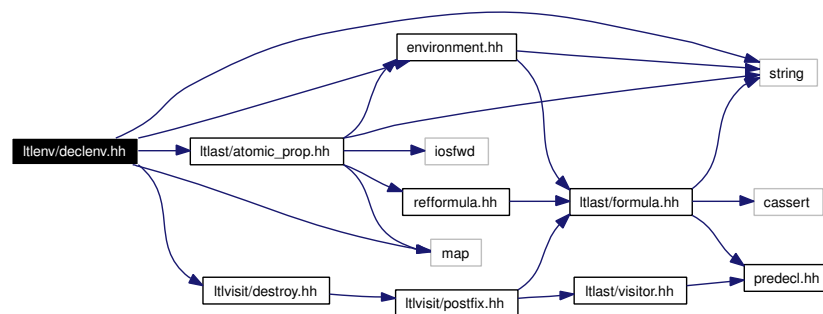
13.31.1 Detailed Description

LTL visitor interface.

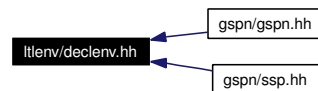
13.32 Itlenv/declenv.hh File Reference

```
#include "environment.hh"
#include <string>
#include <map>
#include "ltlvisit/destroy.hh"
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for declenv.hh:



This graph shows which files directly or indirectly include this file:



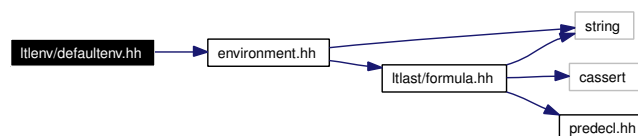
Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

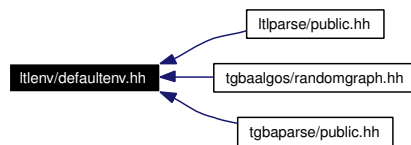
13.33 Itlenv/defaultenv.hh File Reference

```
#include "environment.hh"
```

Include dependency graph for defaultenv.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

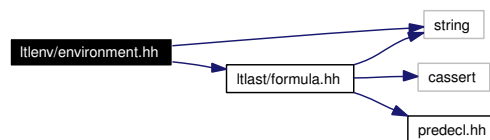
- namespace [spot](#)
- namespace [spot::ltl](#)

13.34 Itlenv/environment.hh File Reference

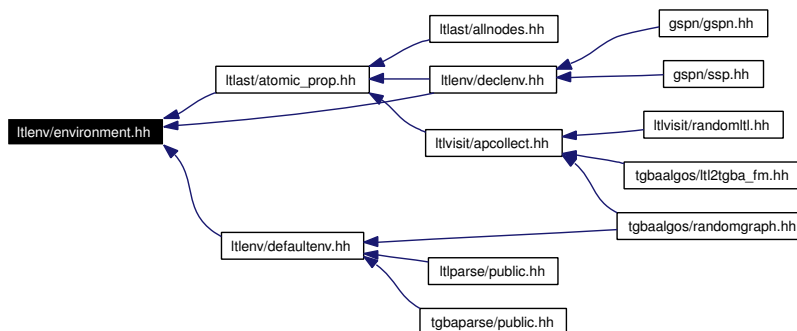
```
#include "ltlast/formula.hh"
```

```
#include <string>
```

Include dependency graph for environment.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

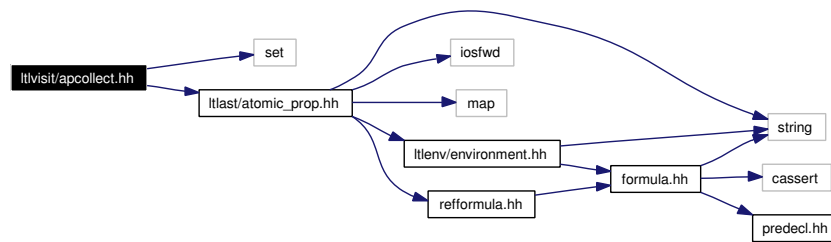
- namespace [spot](#)
- namespace [spot::ltl](#)

13.35 Itlvisit/apcollect.hh File Reference

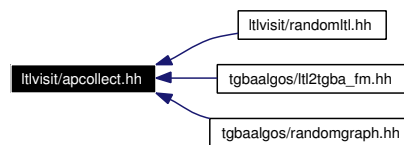
```
#include <set>
```

```
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for apcollect.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Typedefs

- `typedef std::set< atomic_prop *, formula_ptr_less_than > atomic_prop_set`
Set of atomic propositions.

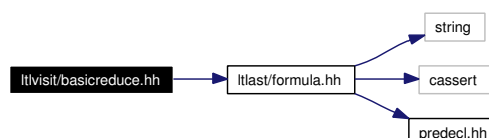
Functions

- `atomic_prop_set * atomic_prop_collect (const formula *f, atomic_prop_set *s=0)`
Return the set of atomic propositions occurring in a formula.

13.36 Itlvisit/basicreduce.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for basicreduce.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Functions

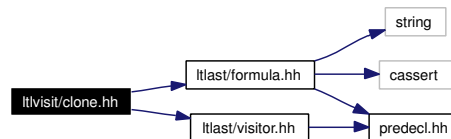
- formula * [basic_reduce](#) (const formula *f)
Basic rewritings.
- bool [is_GF](#) (const formula *f)
Whether a formula starts with GF.
- bool [is_FG](#) (const formula *f)
Whether a formula starts with FG.

13.37 Itlvisit/clone.hh File Reference

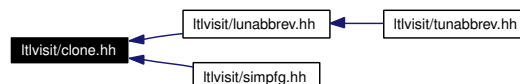
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for clone.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

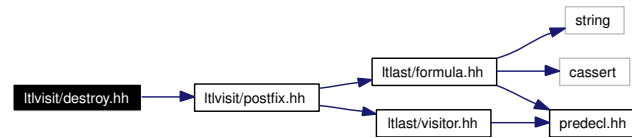
Functions

- formula * [clone](#) (const formula *f)
Clone a formula.

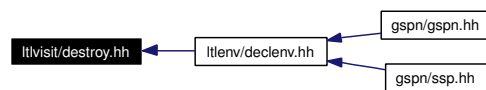
13.38 ltlvisit/destroy.hh File Reference

```
#include "ltlvisit/postfix.hh"
```

Include dependency graph for destroy.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

Functions

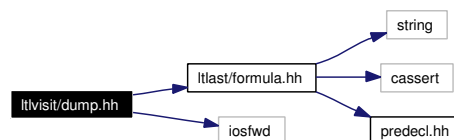
- void `destroy` (const formula *f)
Destroys a formula.

13.39 ltlvisit/dump.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dump.hh:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

Functions

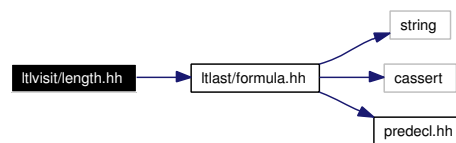
- `std::ostream & dump` (`std::ostream &os`, `const formula *f`)

Dump a formula tree.

13.40 Itlvisit/length.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for length.hh:

**Namespaces**

- namespace `spot`
- namespace `spot::ltl`

Functions

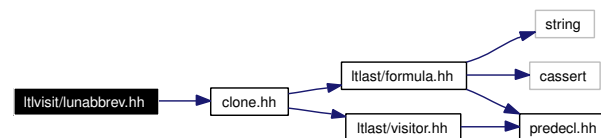
- `int length` (`const formula *f`)

Compute the length of a formula.

13.41 Itlvisit/lunabbrev.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for lunabbrev.hh:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `spot`
- namespace `spot::ltl`

Functions

- formula * [unabbreviate_logic](#) (const formula *f)

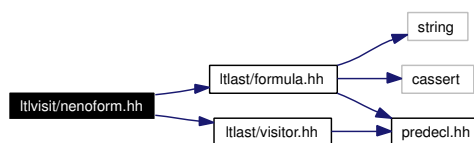
Clone and rewrite a formula to remove most of the abbreviated logical operators.

13.42 Itlvisit/venoform.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for venoform.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Functions

- formula * [negative_normal_form](#) (const formula *f, bool negated=false)

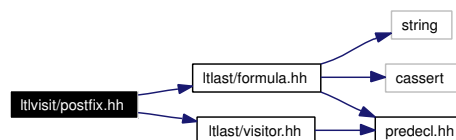
Build the negative normal form of f.

13.43 Itlvisit/postfix.hh File Reference

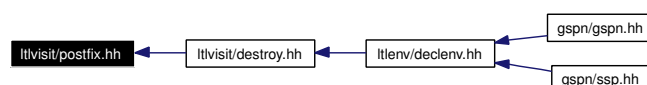
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for postfix.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

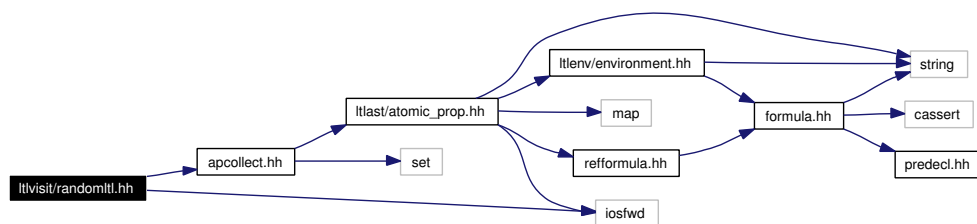
- namespace [spot](#)
- namespace [spot::ltl](#)

13.44 Itlvisit/randomltl.hh File Reference

```
#include "apcollect.hh"
```

```
#include <iosfwd>
```

Include dependency graph for randomltl.hh:



Namespaces

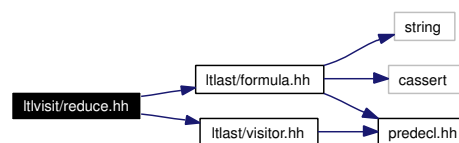
- namespace [spot](#)
- namespace [spot::ltl](#)

13.45 Itlvisit/reduce.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for reduce.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Enumerations

- enum [reduce_options](#) {
[Reduce_None](#) = 0, [Reduce_Basics](#) = 1, [Reduce_Syntactic_Implications](#) = 2, [Reduce_Eventuality_And_Universality](#) = 4,
[Reduce_All](#) = -1U }
Options for [spot::ltl::reduce](#).

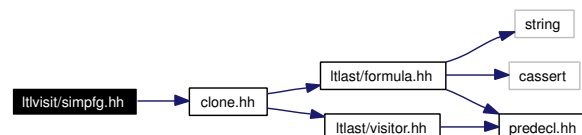
Functions

- formula * [reduce](#) (const formula *f, int opt=[Reduce_All](#))
Reduce a formula f.
- bool [is_eventual](#) (const formula *f)
Check whether a formula is a pure eventuality.
- bool [is_universal](#) (const formula *f)
Check whether a formula is purely universal.

13.46 Itlvisit/simpfg.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for simpfg.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

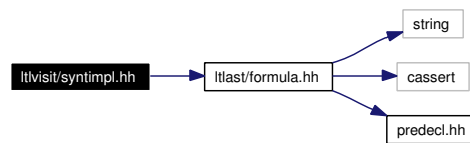
Functions

- formula * [simplify_f_g](#) (const formula *f)
Replace true U f and false R g by F f and G g.

13.47 Itlvisit/syntimpl.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for syntimpl.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Functions

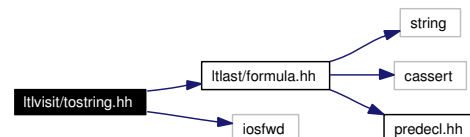
- bool [syntactic_implication](#) (const formula *f1, const formula *f2)
Syntactic implication.
- bool [syntactic_implication_neg](#) (const formula *f1, const formula *f2, bool right)
Syntactic implication.

13.48 ltlvisit/tostring.hh File Reference

```
#include <ltlast/formula.hh>
```

```
#include <iosfwd>
```

Include dependency graph for tostring.hh:



Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

Functions

- std::ostream & [to_string](#) (const formula *f, std::ostream &os)
Output a formula as a (parsable) string.
- std::string [to_string](#) (const formula *f)
Convert a formula into a (parsable) string.
- std::ostream & [to_spin_string](#) (const formula *f, std::ostream &os)
Output a formula as a (parsable by Spin) string.

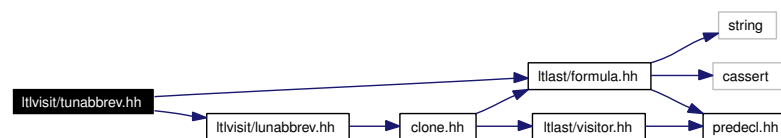
- `std::string to_spin_string (const formula *f)`
Convert a formula into a (parsable by Spin) string.

13.49 Itlvisit/tunabbrev.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlvisit/lunabbrev.hh"
```

Include dependency graph for tunabbrev.hh:



Namespaces

- namespace `spot`
- namespace `spot::ltl`

Functions

- `formula * unabbreviate_ltl (const formula *f)`
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

13.49.1 Function Documentation

13.49.1.1 `formula* unabbreviate_ltl (const formula *f)`

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic`.

This will also rewrite unary operators such as `unop::F`, and `unop::G`, using only `binop::U`, and `binop::R`.

13.50 misc/bareword.hh File Reference

```
#include <string>
```

Include dependency graph for bareword.hh:



Namespaces

- namespace `spot`

Functions

- bool `is_bare_word` (const char *str)
- std::string `quote_unless_bare_word` (const std::string &str)

Double-quote words that are not bare.

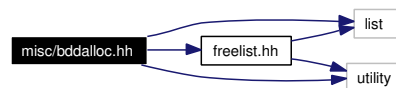
13.51 misc/bddalloc.hh File Reference

```
#include "freelist.hh"
```

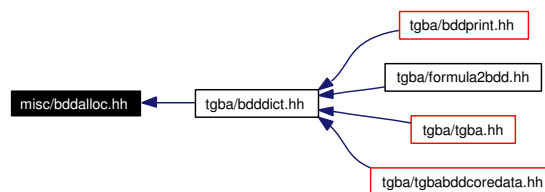
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for bddalloc.hh:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `spot`

13.52 misc/bddlt.hh File Reference

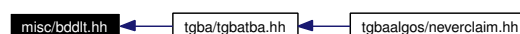
```
#include <bdd.h>
```

```
#include <functional>
```

Include dependency graph for bddlt.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

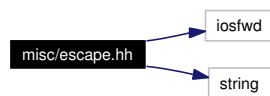
- namespace [spot](#)

13.53 misc/escape.hh File Reference

```
#include <iosfwd>
```

```
#include <string>
```

Include dependency graph for escape.hh:

**Namespaces**

- namespace [spot](#)

Functions

- `std::ostream & escape_str (std::ostream &os, const std::string &str)`
Escape " and \ characters in str.
- `std::string escape_str (const std::string &str)`
Escape " and \ characters in str.

13.54 misc/freelist.hh File Reference

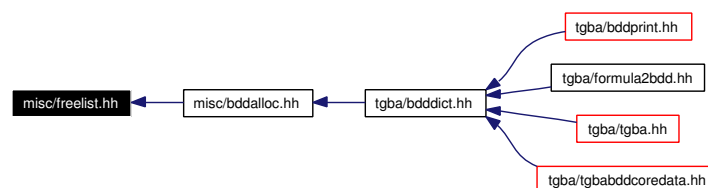
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for freelist.hh:



This graph shows which files directly or indirectly include this file:



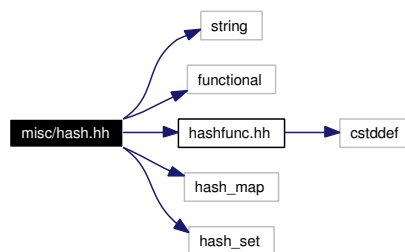
Namespaces

- namespace **spot**

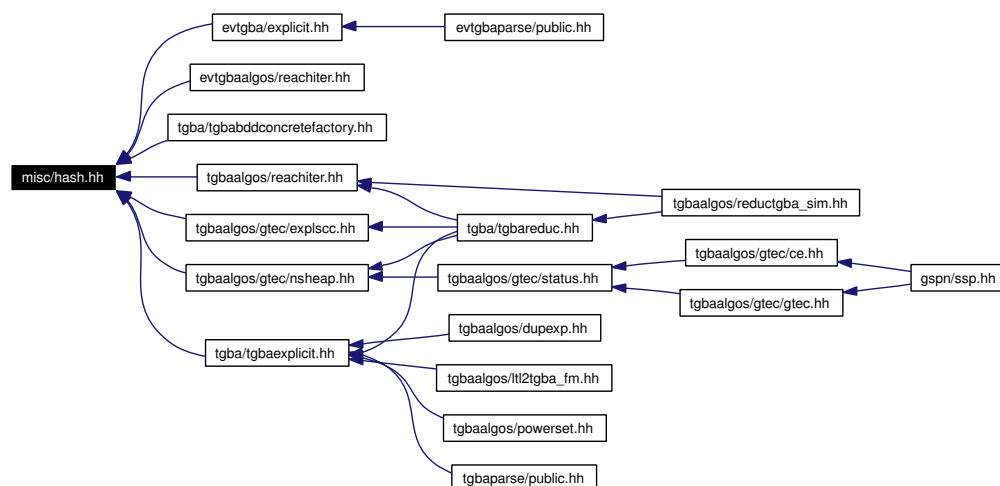
13.55 misc/hash.hh File Reference

```
#include <string>
#include <functional>
#include "hashfunc.hh"
#include <hash_map>
#include <hash_set>
```

Include dependency graph for hash.hh:



This graph shows which files directly or indirectly include this file:



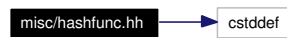
Namespaces

- namespace **spot**

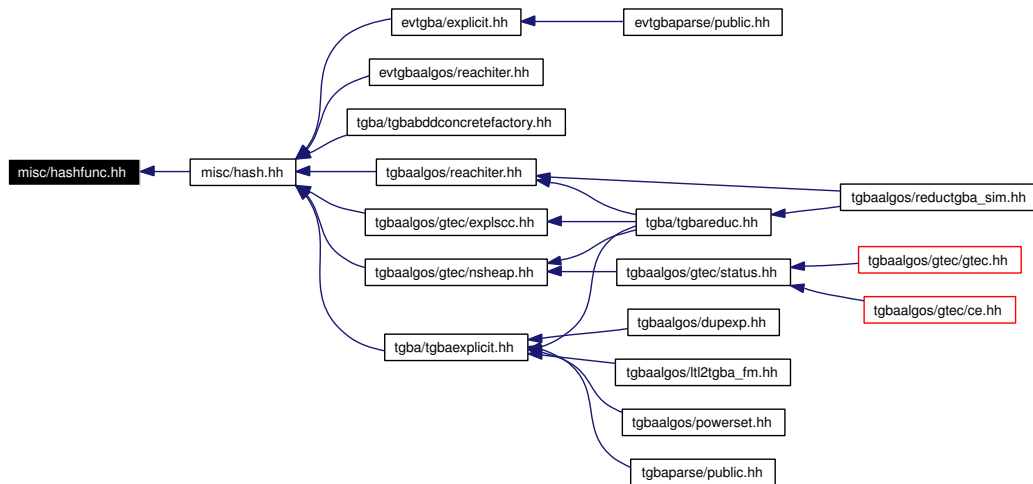
13.56 misc/hashfunc.hh File Reference

```
#include <cstdint>
```

Include dependency graph for hashfunc.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

Functions

- size_t [wang32_hash](#) (size_t key)
Thomas Wang's 32 bit hash function.
- size_t [knuth32_hash](#) (size_t key)
Knuth's Multiplicative hash function.

13.57 misc/ltstr.hh File Reference

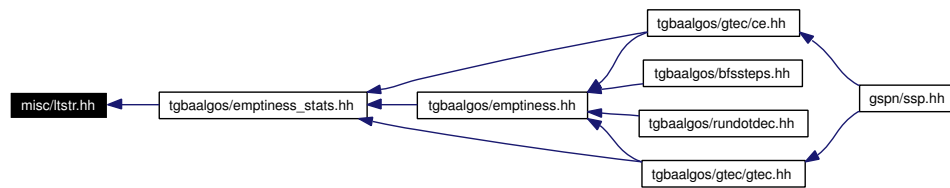
```
#include <cstring>
```

```
#include <functional>
```

Include dependency graph for ltstr.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

13.58 misc/minato.hh File Reference

```
#include <bdd.h>
```

```
#include <stack>
```

Include dependency graph for minato.hh:



Namespaces

- namespace [spot](#)

13.59 misc/modgray.hh File Reference

Namespaces

- namespace [spot](#)

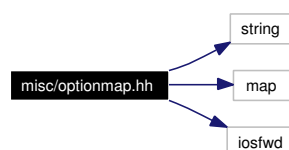
13.60 misc/optionmap.hh File Reference

```
#include <string>
```

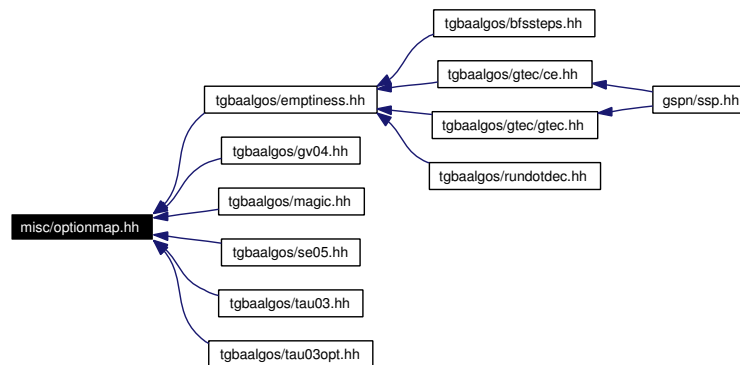
```
#include <map>
```

```
#include <iosfwd>
```

Include dependency graph for optionmap.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

13.61 misc/random.hh File Reference

```
#include <cmath>
```

Include dependency graph for random.hh:



Namespaces

- namespace [spot](#)

Functions

- void [srand](#) (unsigned int seed)
Reset the seed of the pseudo-random number generator.
- int [rrand](#) (int min, int max)
Compute a pseudo-random integer value between min and max included.
- int [mrand](#) (int max)
Compute a pseudo-random integer value between 0 and max-1 included.
- double [drand](#) ()
Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).
- double [nrand](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).
- double [bmrnd](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).

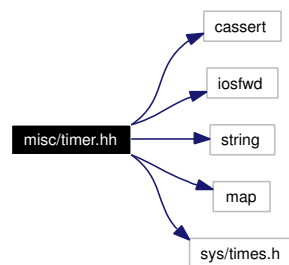
- int `prand` (double p)

Return a pseudo-random positive integer value following a Poisson distribution with parameter p.

13.62 misc/timer.hh File Reference

```
#include <cassert>
#include <iosfwd>
#include <string>
#include <map>
#include <sys/times.h>
```

Include dependency graph for timer.hh:



Namespaces

- namespace `spot`

13.63 misc/version.hh File Reference

Namespaces

- namespace `spot`

Functions

- const char * `version` ()

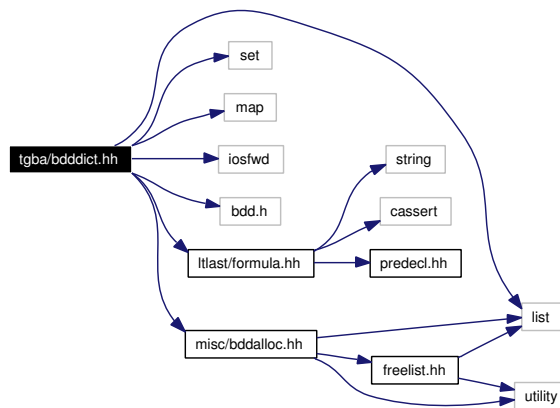
Return Spot's version.

13.64 tgba/bdddict.hh File Reference

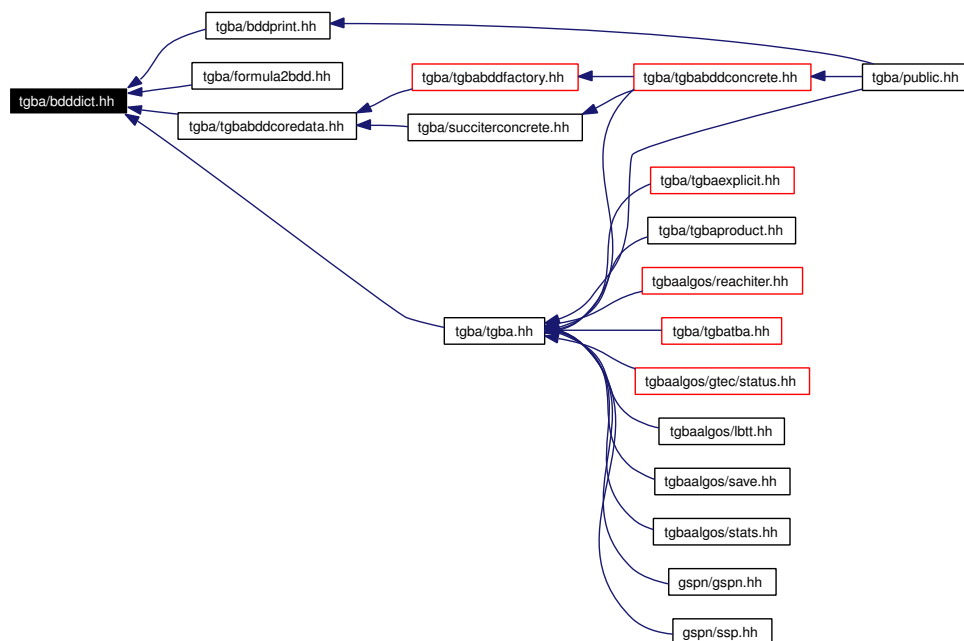
```
#include <list>
#include <set>
#include <map>
#include <iosfwd>
```

```
#include <bdd.h>
#include "ltlast/formula.hh"
#include "misc/bddalloc.hh"
```

Include dependency graph for bdddict.hh:



This graph shows which files directly or indirectly include this file:



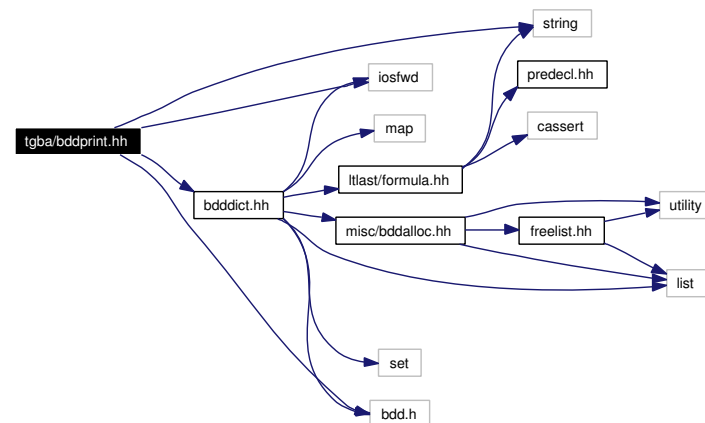
Namespaces

- namespace [spot](#)

13.65 tgba/bddprint.hh File Reference

```
#include <string>
#include <iosfwd>
#include "bdddict.hh"
#include <bdd.h>
```

Include dependency graph for bddprint.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

Functions

- `std::ostream & bdd_print_sat (std::ostream &os, const bdd_dict *dict, bdd b)`
Print a BDD as a list of literals.
- `std::string bdd_format_sat (const bdd_dict *dict, bdd b)`
Format a BDD as a list of literals.
- `std::ostream & bdd_print_acc (std::ostream &os, const bdd_dict *dict, bdd b)`
Print a BDD as a list of acceptance conditions.
- `std::ostream & bdd_print_accset (std::ostream &os, const bdd_dict *dict, bdd b)`
Print a BDD as a set of acceptance conditions.
- `std::string bdd_format_accset (const bdd_dict *dict, bdd b)`
Format a BDD as a set of acceptance conditions.

- `std::ostream & bdd_print_set` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a set.
- `std::string bdd_format_set` (`const bdd_dict *dict`, `bdd b`)
Format a BDD as a set.
- `std::ostream & bdd_print_formula` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a formula.
- `std::string bdd_format_formula` (`const bdd_dict *dict`, `bdd b`)
Format a BDD as a formula.
- `std::ostream & bdd_print_dot` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a diagram in dotty format.
- `std::ostream & bdd_print_table` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a table.

13.65.1 Function Documentation

13.65.1.1 `std::string bdd_format_accset` (`const bdd_dict * dict`, `bdd b`)

Format a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.2 `std::string bdd_format_formula` (`const bdd_dict * dict`, `bdd b`)

Format a BDD as a formula.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.3 std::string bdd_format_sat (const bdd_dict * *dict*, bdd *b*)

Format a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.4 std::string bdd_format_set (const bdd_dict * *dict*, bdd *b*)

Format a BDD as a set.

Parameters:

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.5 std::ostream& bdd_print_acc (std::ostream & *os*, const bdd_dict * *dict*, bdd *b*)

Print a BDD as a list of acceptance conditions.

This is used when saving a TGBA.

Parameters:

os The output stream.

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.6 std::ostream& bdd_print_accset (std::ostream & *os*, const bdd_dict * *dict*, bdd *b*)

Print a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

Parameters:

os The output stream.

dict The dictionary to use, to lookup variables.

b The BDD to print.

Returns:

The BDD formatted as a string.

13.65.1.7 `std::ostream& bdd_print_dot (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a diagram in dotty format.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

13.65.1.8 `std::ostream& bdd_print_formula (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a formula.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

13.65.1.9 `std::ostream& bdd_print_sat (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a list of literals.

This assumes that *b* is a conjunction of literals.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

13.65.1.10 `std::ostream& bdd_print_set (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a set.

Parameters:

- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

13.65.1.11 `std::ostream& bdd_print_table (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a table.

Parameters:

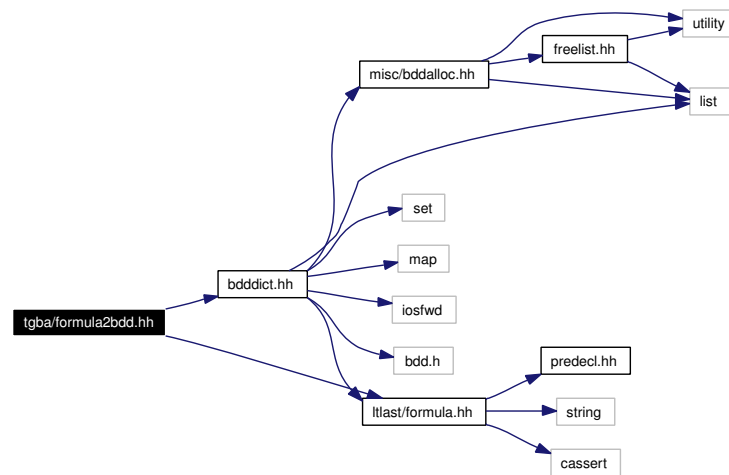
- os* The output stream.
- dict* The dictionary to use, to lookup variables.
- b* The BDD to print.

13.66 tgba/formula2bdd.hh File Reference

```
#include "bdddict.hh"
```

```
#include "ltlast/formula.hh"
```

Include dependency graph for formula2bdd.hh:



Namespaces

- namespace [spot](#)

Functions

- bdd [formula_to_bdd](#) (const ltl::formula *f, bdd_dict *d, void *for_me)
- const ltl::formula * [bdd_to_formula](#) (bdd f, const bdd_dict *d)

13.66.1 Function Documentation

13.66.1.1 const ltl::formula* [bdd_to_formula](#) (bdd f, const bdd_dict *d)

13.66.1.2 bdd [formula_to_bdd](#) (const ltl::formula *f, bdd_dict *d, void *for_me)

13.67 tgba/state.hh File Reference

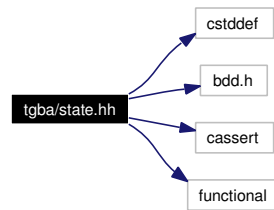
```
#include <cstdint>
```

```
#include <bdd.h>
```

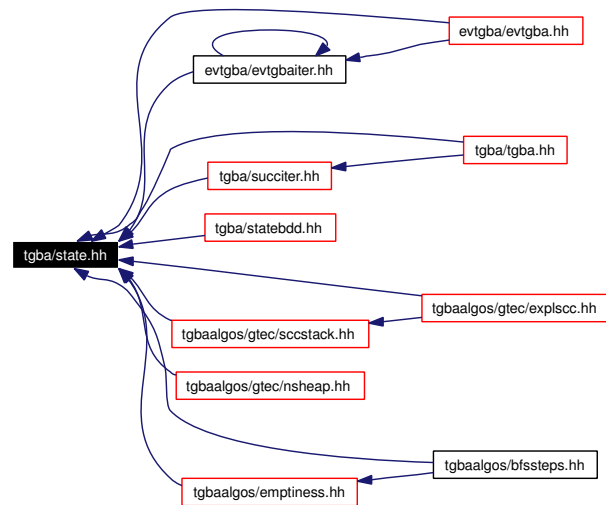
```
#include <cassert>
```

```
#include <functional>
```

Include dependency graph for state.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

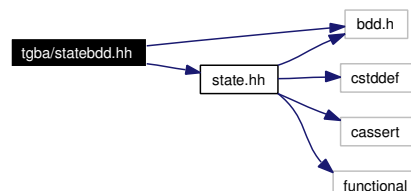
- namespace `spot`

13.68 tgba/statebdd.hh File Reference

```
#include <bdd.h>
```

```
#include "state.hh"
```

Include dependency graph for `statebdd.hh`:



This graph shows which files directly or indirectly include this file:



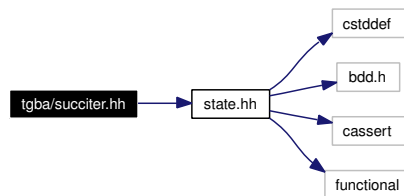
Namespaces

- namespace [spot](#)

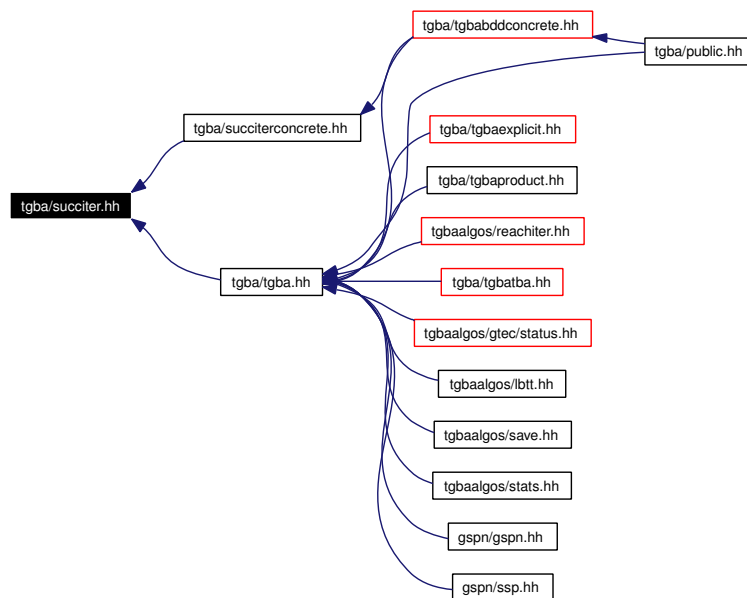
13.69 tgba/succiter.hh File Reference

```
#include "state.hh"
```

Include dependency graph for succiter.hh:



This graph shows which files directly or indirectly include this file:



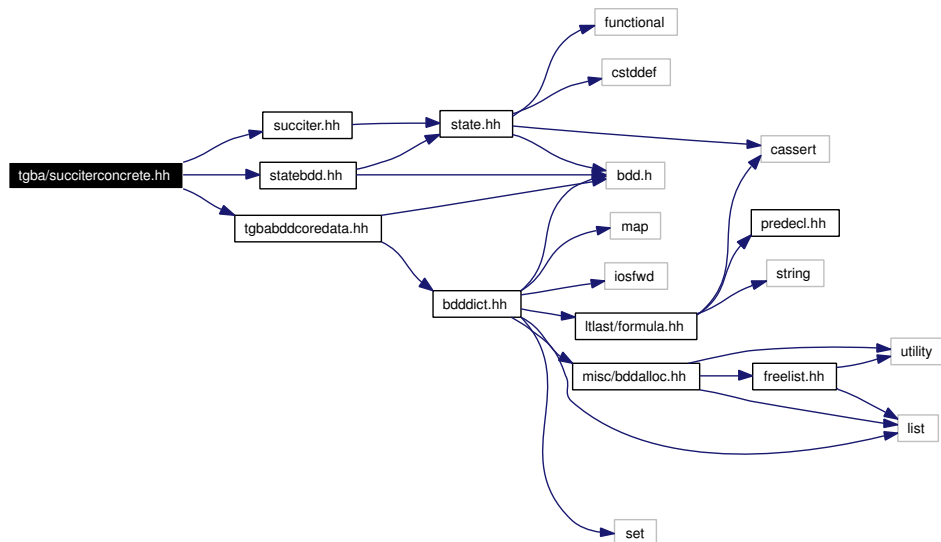
Namespaces

- namespace [spot](#)

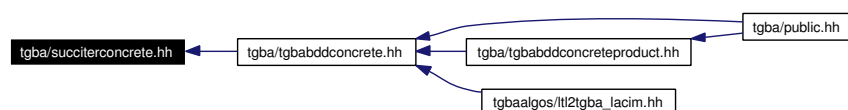
13.70 tgba/succiterconcrete.hh File Reference

```
#include "statebdd.hh"
#include "succiter.hh"
#include "tgbabddcoredata.hh"
```

Include dependency graph for succiterconcrete.hh:



This graph shows which files directly or indirectly include this file:



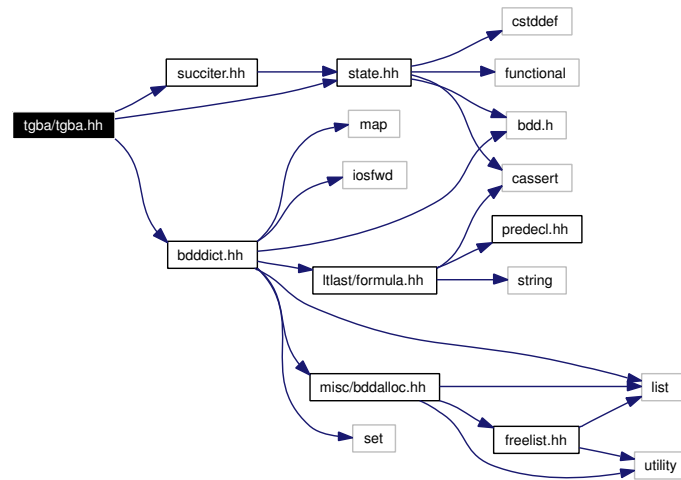
Namespaces

- namespace [spot](#)

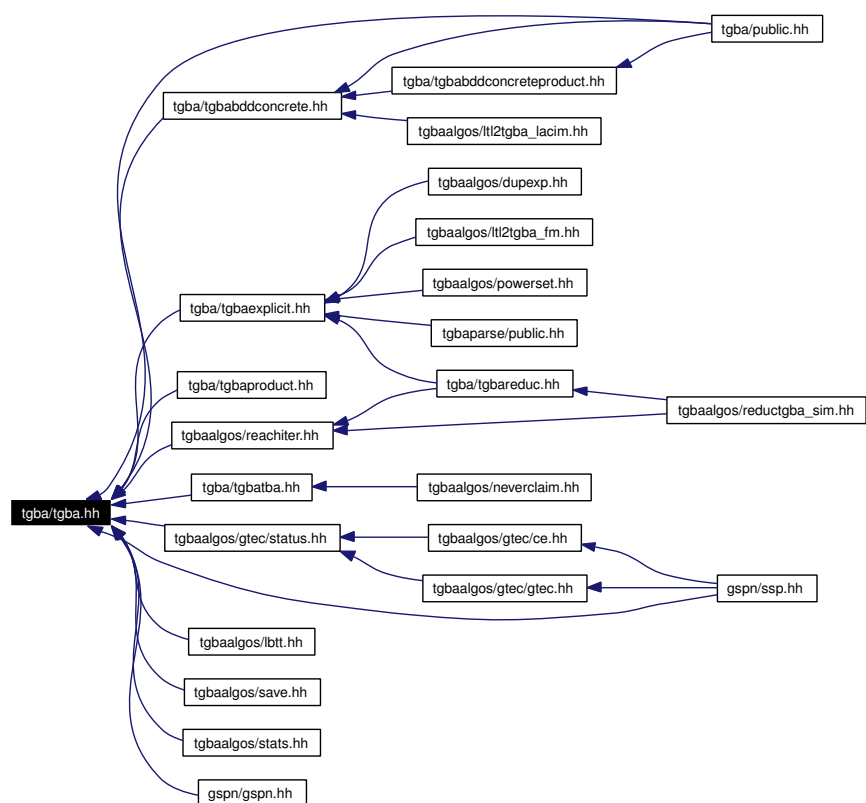
13.71 tgba/tgba.hh File Reference

```
#include "state.hh"
#include "succiter.hh"
#include "bdddict.hh"
```

Include dependency graph for tgba.hh:



This graph shows which files directly or indirectly include this file:



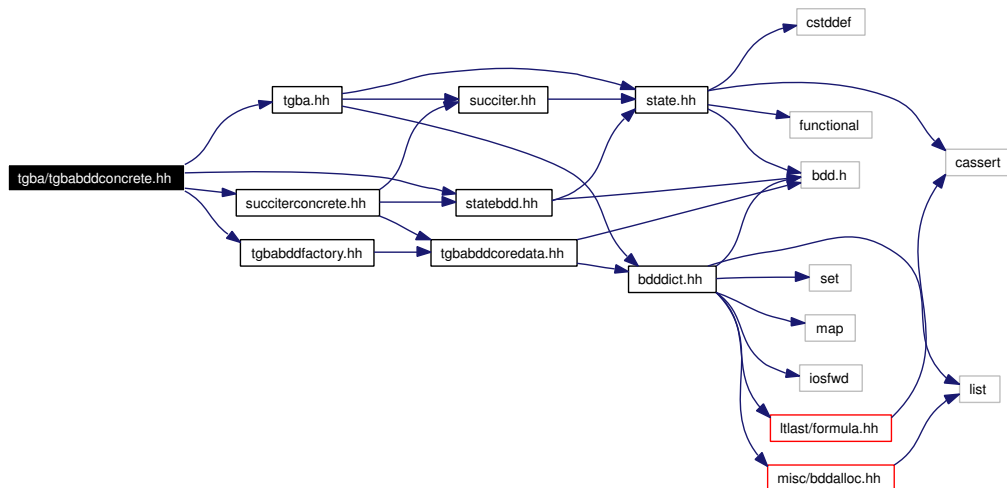
Namespaces

- namespace `spot`

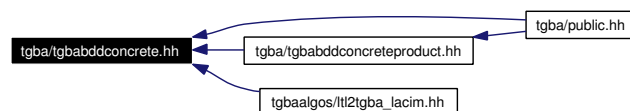
13.72 tgba/tgbabddconcrete.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
#include "tgbabddfactory.hh"
#include "succiterconcrete.hh"
```

Include dependency graph for tgbabddconcrete.hh:



This graph shows which files directly or indirectly include this file:



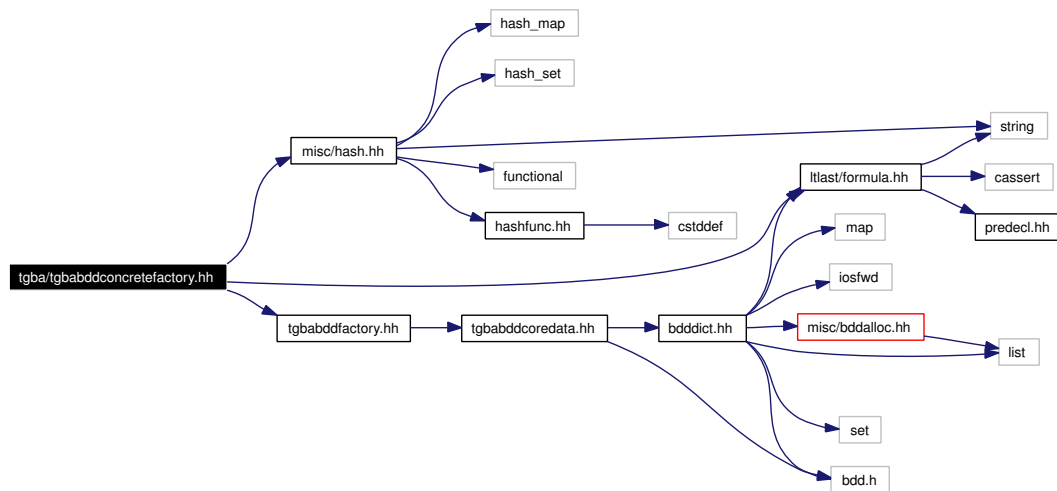
Namespaces

- namespace [spot](#)

13.73 tgba/tgbabddconcretefactory.hh File Reference

```
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "tgbabddfactory.hh"
```

Include dependency graph for tgbabddconcretefactory.hh:



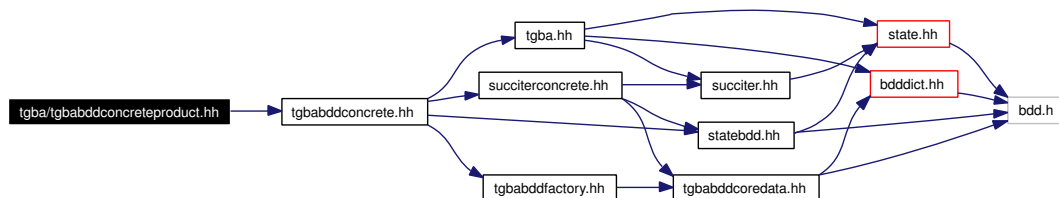
Namespaces

- namespace [spot](#)

13.74 tgba/tgbabddconcreteproduct.hh File Reference

```
#include "tgbabddconcrete.hh"
```

Include dependency graph for tgbabddconcreteproduct.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

Functions

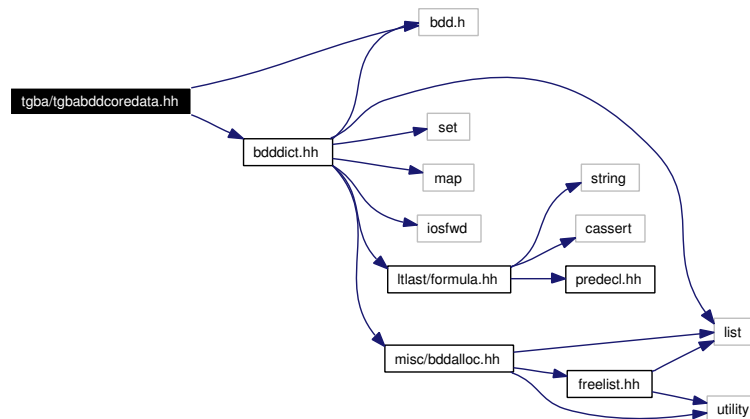
- `tgba_bdd_concrete * product (const tgba_bdd_concrete *left, const tgba_bdd_concrete *right)`
Multiplies two [spot::tgba_bdd_concrete](#) automata.

13.75 tgba/tgbabddcoredata.hh File Reference

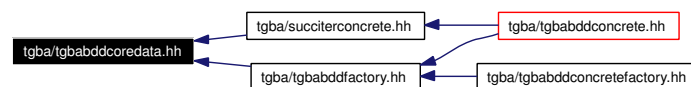
```
#include <bdd.h>
```

```
#include "bdddict.hh"
```

Include dependency graph for tgbabddcoredata.hh:



This graph shows which files directly or indirectly include this file:



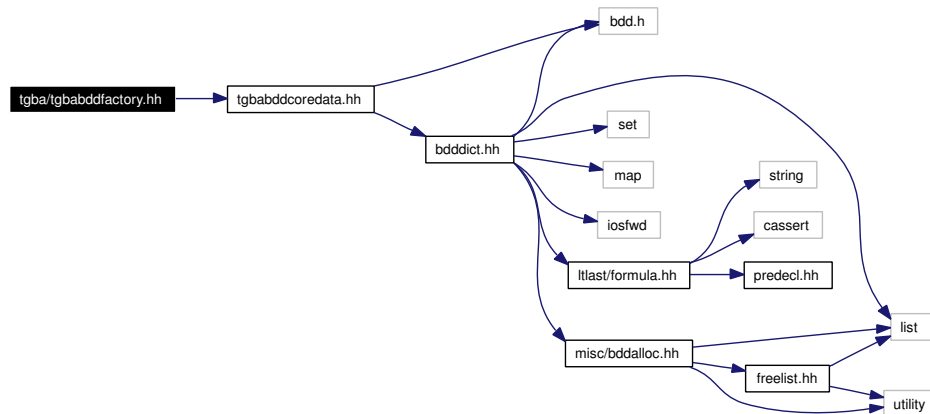
Namespaces

- namespace `spot`

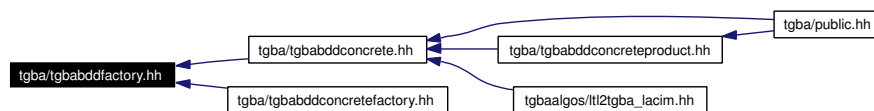
13.76 tgba/tgbabddfactory.hh File Reference

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for tgbabddfactory.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

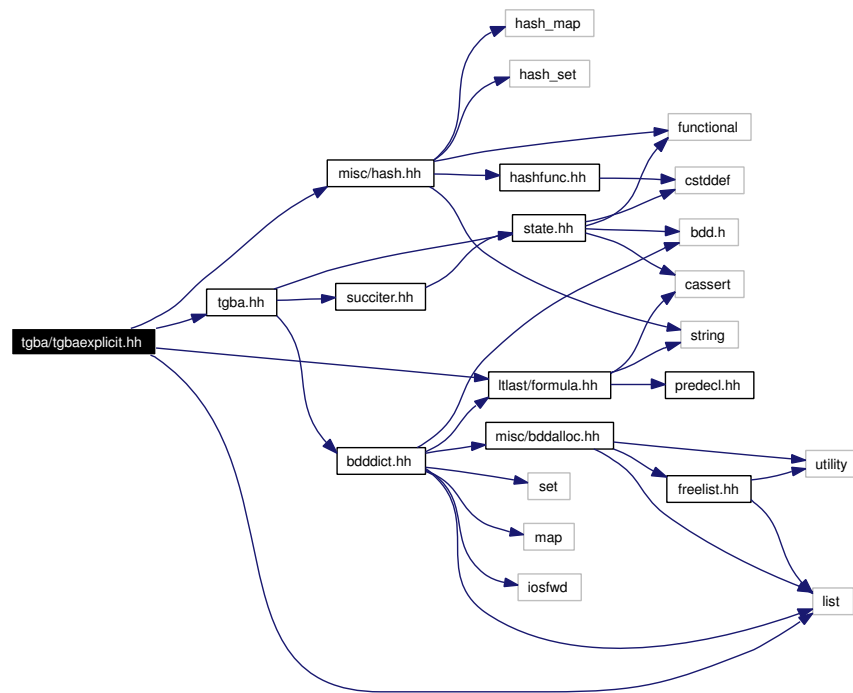
13.77 tgba/tgbaexplicit.hh File Reference

```

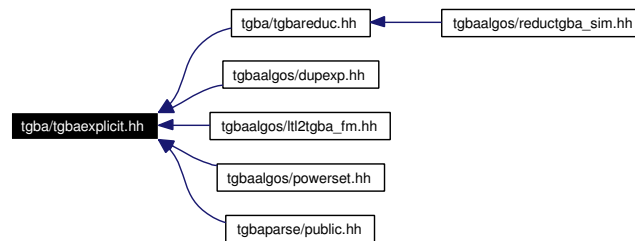
#include "misc/hash.hh"
#include <list>
#include "tgba.hh"
#include "ltlast/formula.hh"

```

Include dependency graph for tgbaexplicit.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

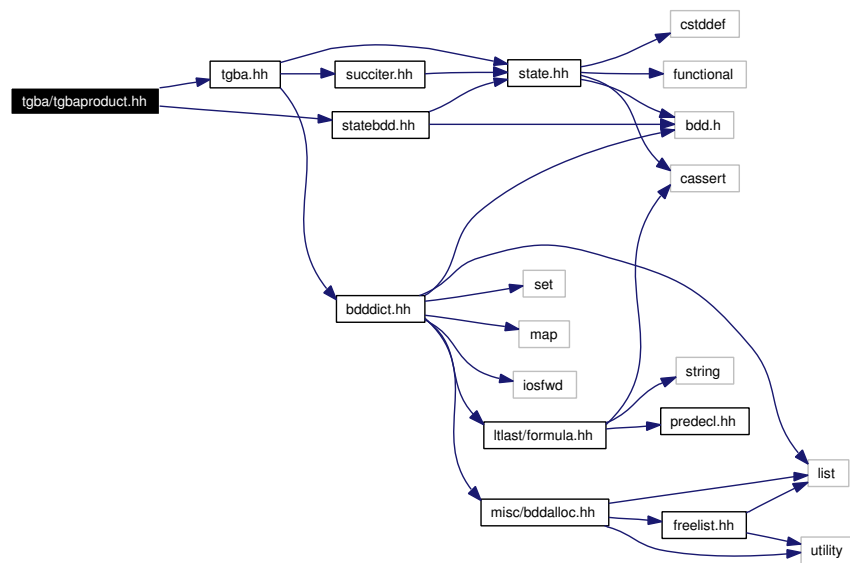
- namespace [spot](#)

13.78 tgba/tgbaproduct.hh File Reference

```
#include "tgba.hh"
```

```
#include "statebdd.hh"
```

Include dependency graph for tgbaproduct.hh:



Namespaces

- namespace [spot](#)

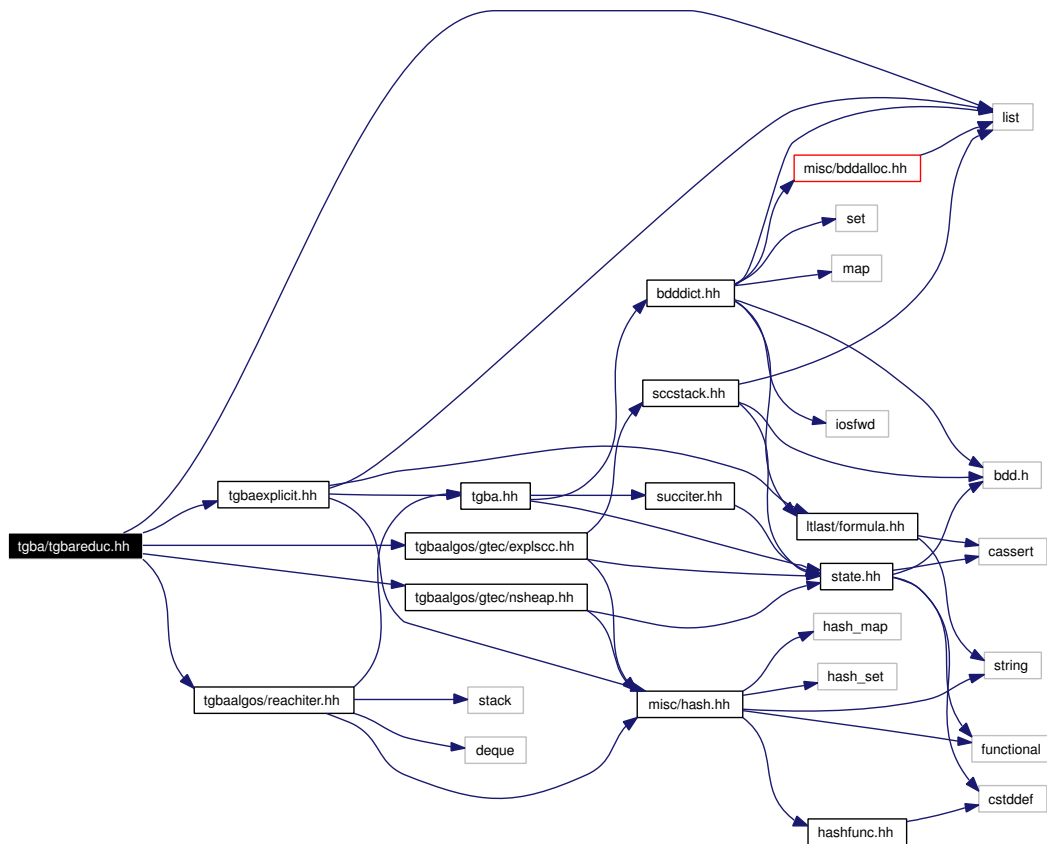
13.79 tgba/tgbareduc.hh File Reference

```

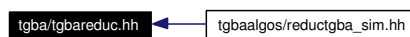
#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#include "tgbaalgos/gtec/explsc.c.hh"
#include "tgbaalgos/gtec/nsheap.hh"
#include <list>

```

Include dependency graph for tgbaeduc.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace **spot**

Typedefs

- typedef std::pair< const spot::state *, const spot::state * > state_couple
- typedef std::vector< state_couple * > simulation_relation

13.79.1 Typedef Documentation

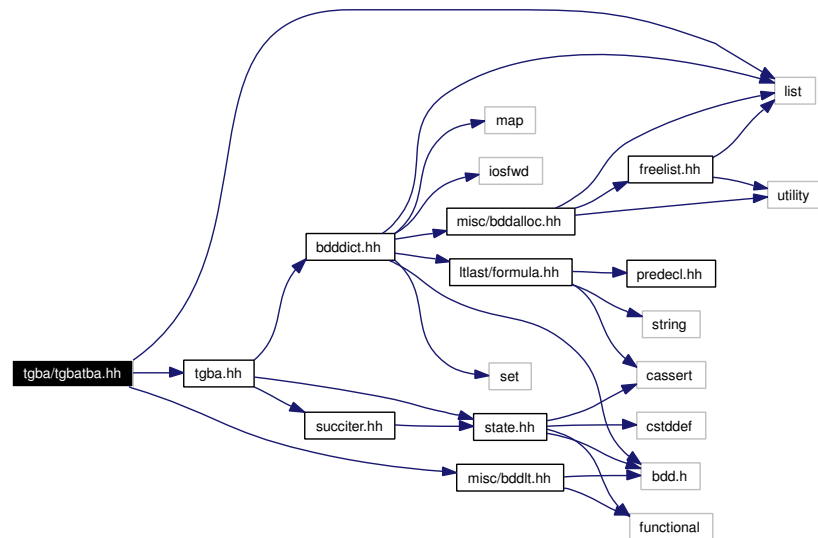
13.79.1.1 `typedef std::vector<state_couple*> spot::simulation_relation`

```
13.79.1.2 typedef std::pair<const spot::state*, const spot::state*> spot::state_couple
```


13.80 tgba/tgbatba.hh File Reference

```
#include <list>
#include "tgba.hh"
#include "misc/bddlt.hh"
```

Include dependency graph for tgbatba.hh:



This graph shows which files directly or indirectly include this file:



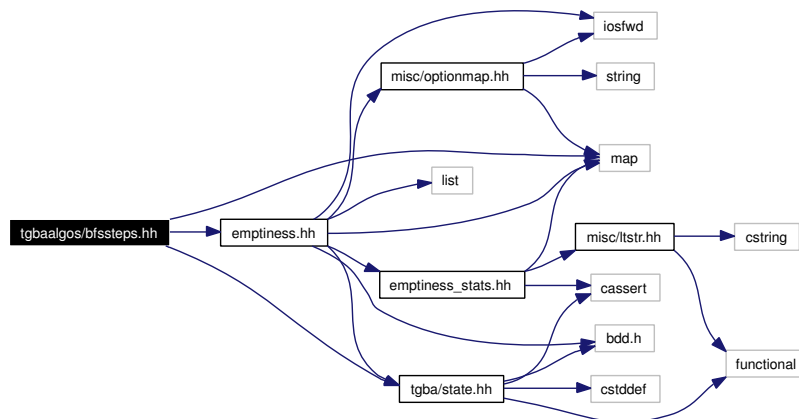
Namespaces

- namespace [spot](#)

13.81 tgbaalgos/bfssteps.hh File Reference

```
#include <map>
#include "tgba/state.hh"
#include "emptiness.hh"
```

Include dependency graph for bfssteps.hh:



Namespaces

- namespace [spot](#)

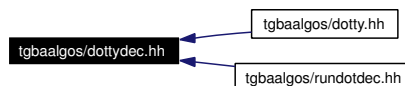
13.82 tgbaaalgos/dottydec.hh File Reference

```
#include <string>
```

Include dependency graph for dottydec.hh:



This graph shows which files directly or indirectly include this file:



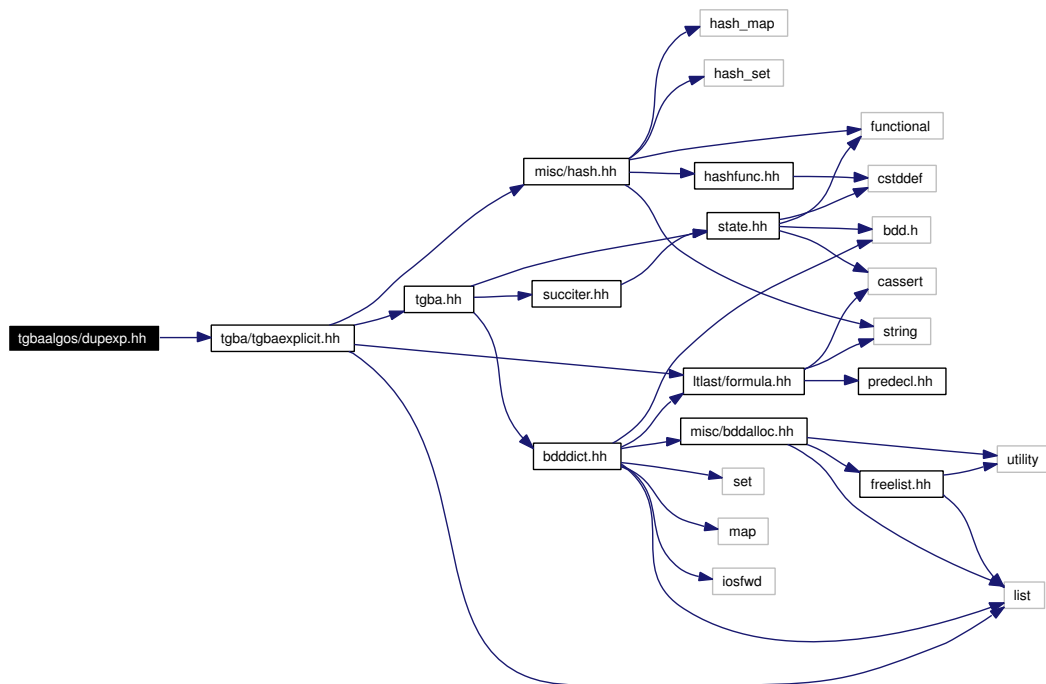
Namespaces

- namespace [spot](#)

13.83 tgbaaalgos/dupeexp.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for dupeexp.hh:



Namespaces

- namespace [spot](#)

Functions

- `tgba_explicit * tgba_dupexp_bfs (const tgba *aut)`
Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.
- `tgba_explicit * tgba_dupexp_dfs (const tgba *aut)`
Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.

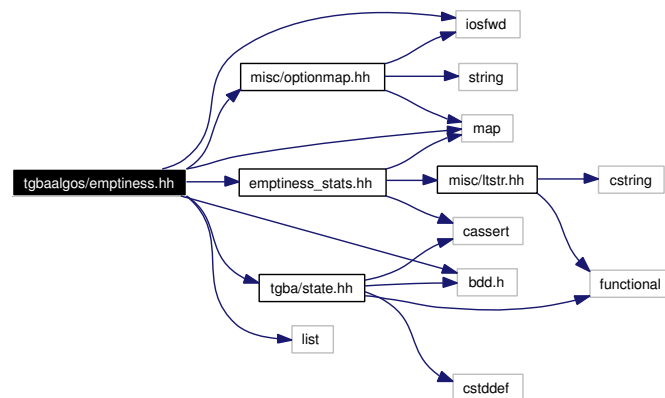
13.84 tgbaalgos/emptiness.hh File Reference

```

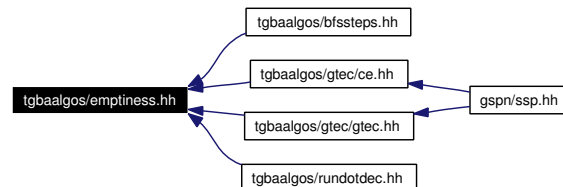
#include <map>
#include <list>
#include <iosfwd>
#include <bdd.h>
#include "misc/optionmap.hh"
#include "tgba/state.hh"
#include "emptiness_stats.hh"

```

Include dependency graph for emptiness.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

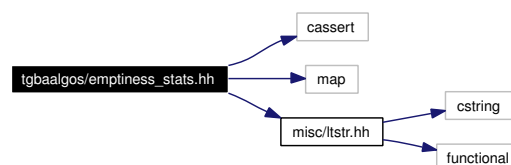
Functions

- `std::ostream & print_tgba_run` (`std::ostream &os`, `const tgba *a`, `const tgba_run *run`)
Display a [tgba_run](#).
- `tgba * tgba_run_to_tgba` (`const tgba *a`, `const tgba_run *run`)
Return an `explicit_tgba` corresponding to run (i.e. comparable states are merged).

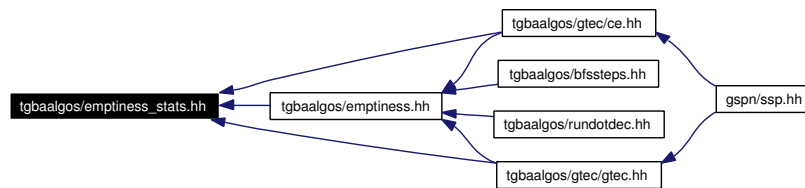
13.85 tgbaalgos/emptiness_stats.hh File Reference

```
#include <cassert>
#include <map>
#include "misc/ltstr.hh"
```

Include dependency graph for emptiness_stats.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

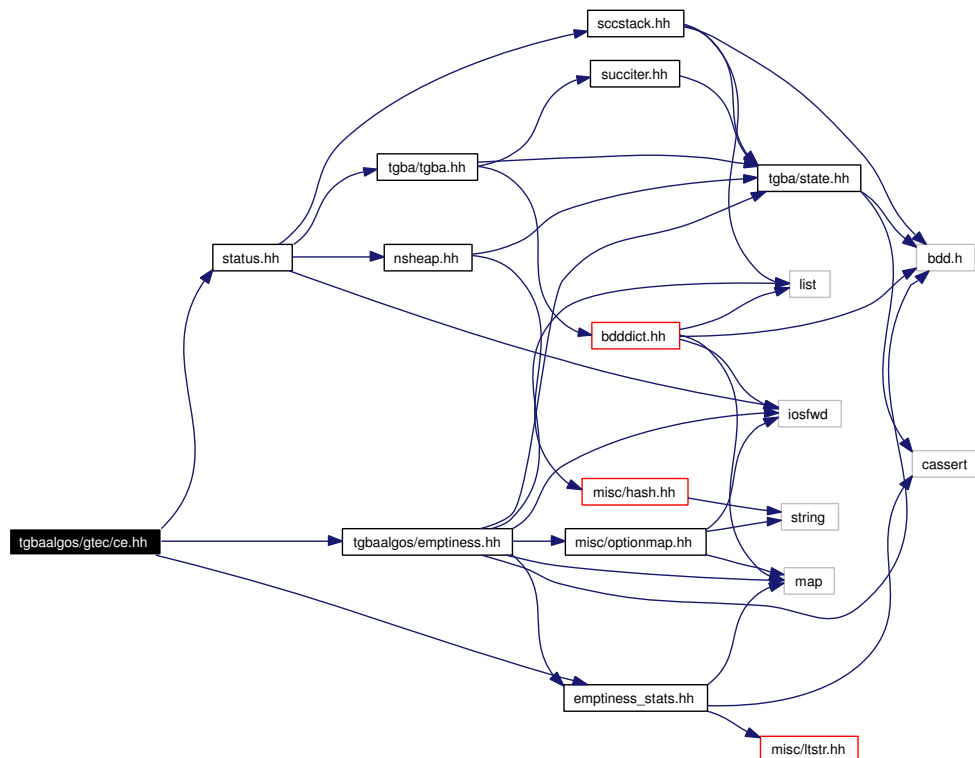
13.86 tgbaalgos/gtec/ce.hh File Reference

```
#include "status.hh"
```

```
#include "tgbaalgos/emptiness.hh"
```

```
#include "tgbaalgos/emptiness_stats.hh"
```

Include dependency graph for ce.hh:



This graph shows which files directly or indirectly include this file:



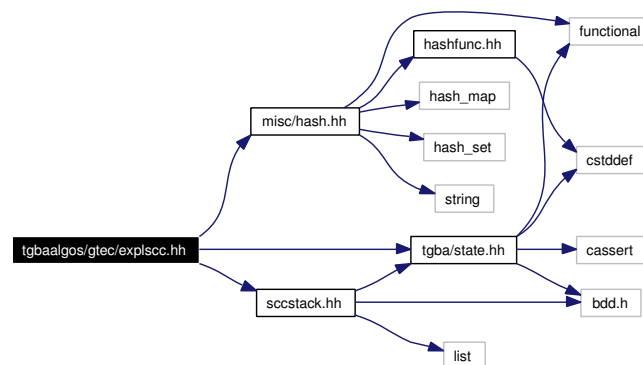
Namespaces

- namespace [spot](#)

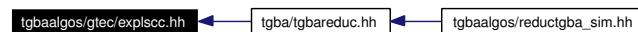
13.87 tgbaalgos/gtec/explsc.h File Reference

```
#include "misc/hash.hh"
#include "tgba/state.hh"
#include "sccstack.hh"
```

Include dependency graph for explsc.h:



This graph shows which files directly or indirectly include this file:



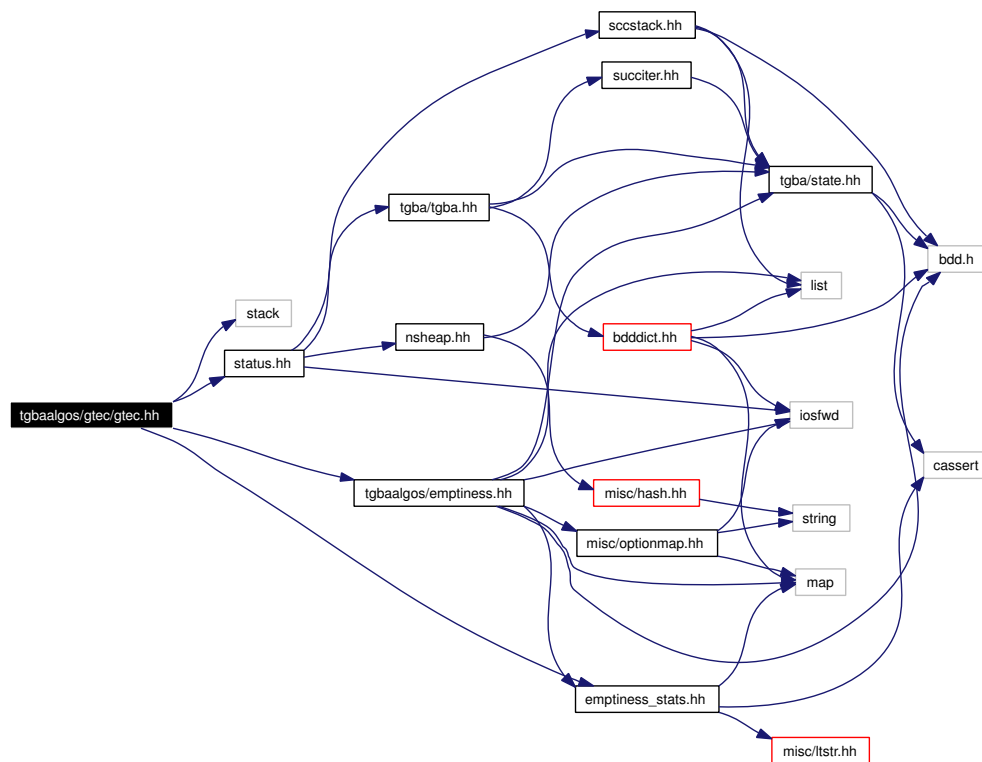
Namespaces

- namespace [spot](#)

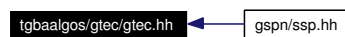
13.88 tgbaalgos/gtec/gtec.h File Reference

```
#include <stack>
#include "status.hh"
#include "tgbaalgos/emptiness.hh"
#include "tgbaalgos/emptiness_stats.hh"
```

Include dependency graph for gtec.h:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

Functions

- emptiness_check * [couveur99](#) (const tgba *a, option_map options=option_map(), const numbered_state_heap_factory *nshf=numbered_state_heap_hash_map_factory::instance())

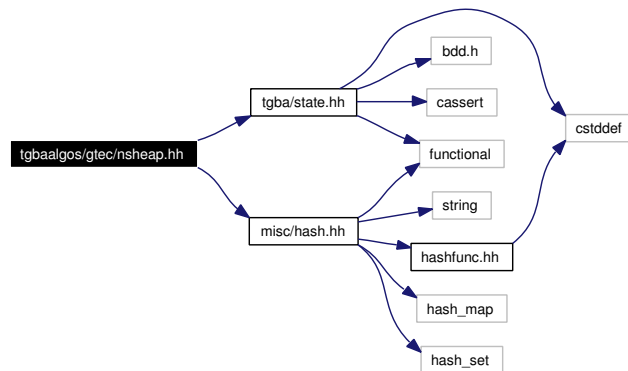
Check whether the language of an automata is empty.

13.89 tgbaalgos/gtec/nsheap.hh File Reference

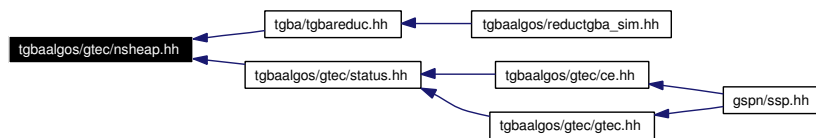
```
#include "tgba/state.hh"
```

```
#include "misc/hash.hh"
```

Include dependency graph for nsheap.hh:



This graph shows which files directly or indirectly include this file:



Namespaces

- namespace [spot](#)

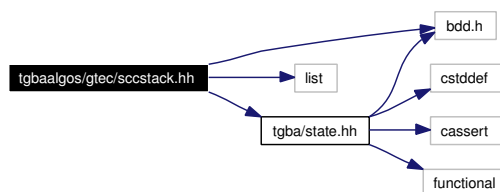
13.90 tgbaalgos/gtec/sccteststack.hh File Reference

```
#include <bdd.h>
```

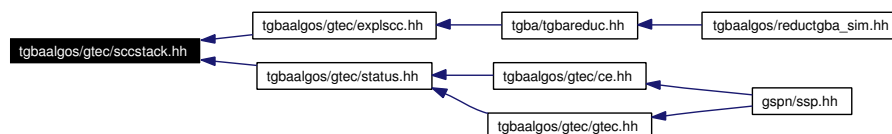
```
#include <list>
```

```
#include <tgba/state.hh>
```

Include dependency graph for sccteststack.hh:



This graph shows which files directly or indirectly include this file:



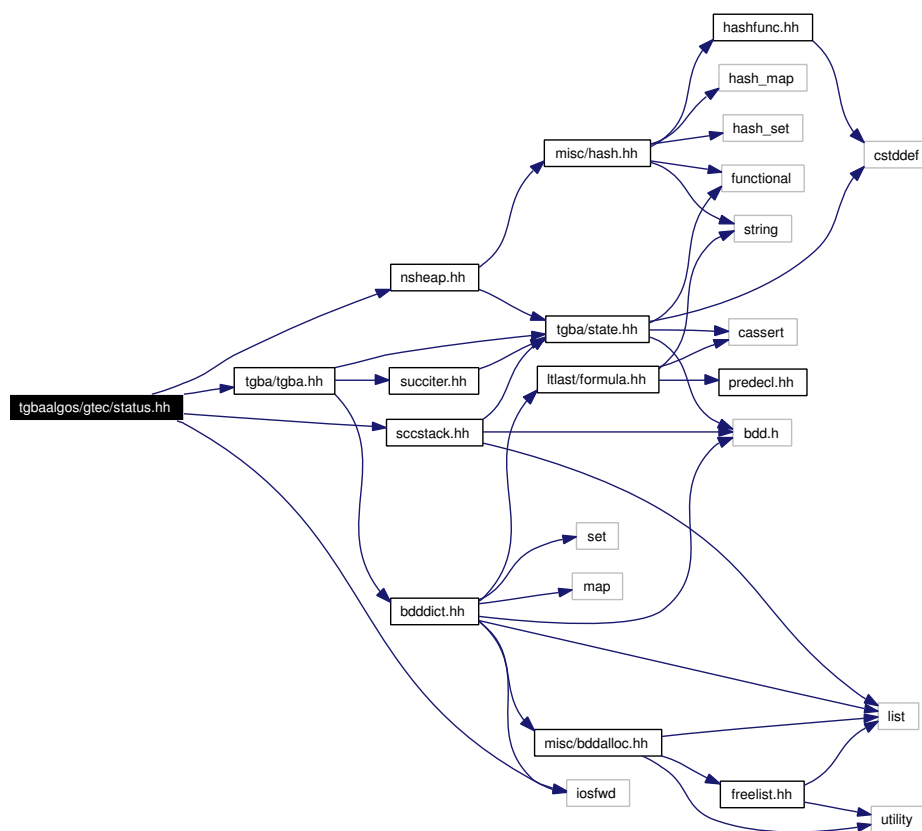
Namespaces

- namespace [spot](#)

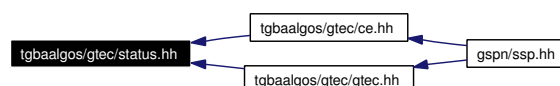
13.91 tgbaalgos/gtec/status.hh File Reference

```
#include "sccstack.hh"
#include "nsheap.hh"
#include "tgba/tgba.hh"
#include <iosfwd>
```

Include dependency graph for status.hh:



This graph shows which files directly or indirectly include this file:



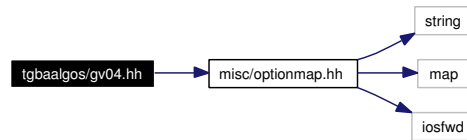
Namespaces

- namespace [spot](#)

13.92 tgbaalgos/gv04.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for gv04.hh:



Namespaces

- namespace `spot`

Functions

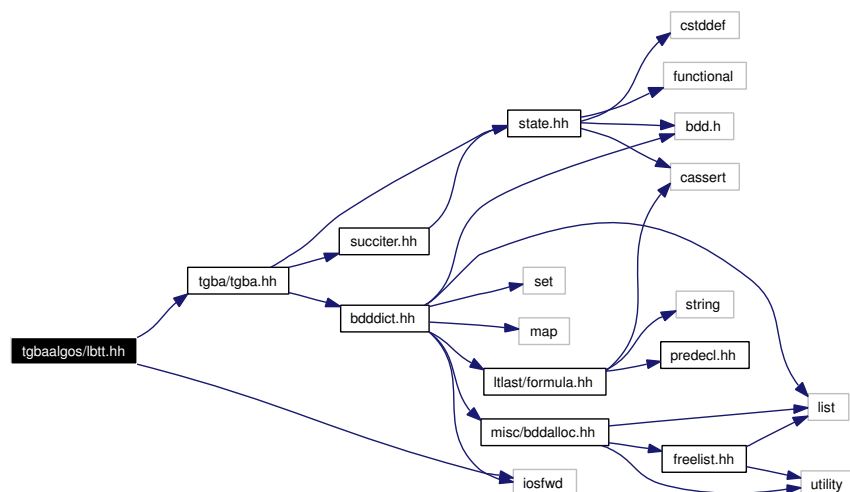
- `emptiness_check` * `explicit_gv04_check` (const tgba *a, option_map o=option_map())
Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.

13.93 tgbaalgos/lbtt.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for lbtt.hh:



Namespaces

- namespace `spot`

Functions

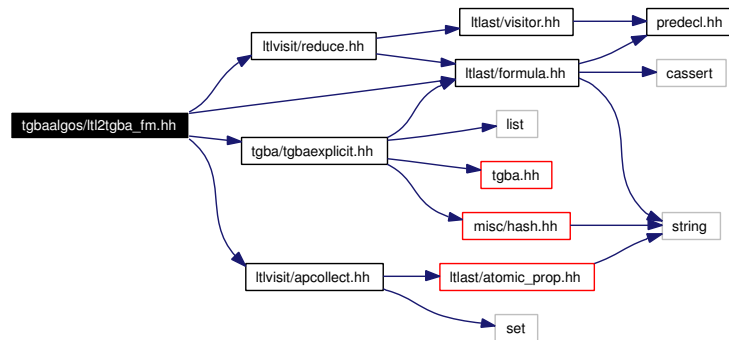
- `std::ostream & lbttr_reachable` (`std::ostream &os`, `const tgba *g`)

Print reachable states in LBTT format.

13.94 tgbaalgos/ltl2tgba_fm.hh File Reference

```
#include "ltlast/formula.hh"
#include "tgba/tgbaexplicit.hh"
#include "ltlvisit/apcollect.hh"
#include "ltlvisit/reduce.hh"
```

Include dependency graph for `ltl2tgba_fm.hh`:



Namespaces

- namespace `spot`

Functions

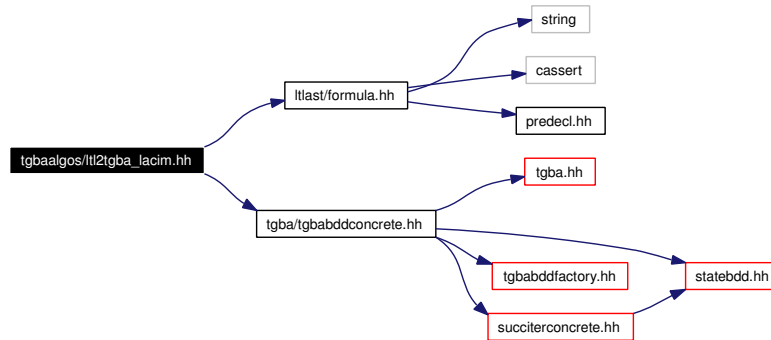
- `tgba_explicit * ltl_to_tgba_fm` (`const ltl::formula *f`, `bdd_dict *dict`, `bool exprop=false`, `bool symb_merge=true`, `bool branching_postponement=false`, `bool fair_loop_approx=false`, `const ltl::atomic_prop_set *unobs=0`, `int reduce_ltl=ltl::Reduce_None`)

Build a `spot::tgba_explicit` from an LTL formula.

13.95 tgbaalgos/ltl2tgba_lacim.hh File Reference

```
#include "ltlast/formula.hh"
#include "tgba/tgbabddconcrete.hh"
```

Include dependency graph for `ltl2tgba_lacim.hh`:



Namespaces

- namespace [spot](#)

Functions

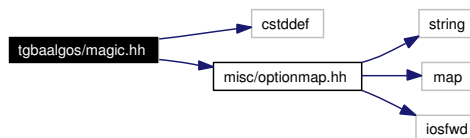
- `tgba_bdd_concrete * ltl_to_tgba_lacim` (const ltl::formula *f, bdd_dict *dict)
Build a [spot::tgba_bdd_concrete](#) from an LTL formula.

13.96 tgbaalgos/magic.hh File Reference

```
#include <cstdint>
```

```
#include "misc/optionmap.hh"
```

Include dependency graph for magic.hh:



Namespaces

- namespace [spot](#)

Functions

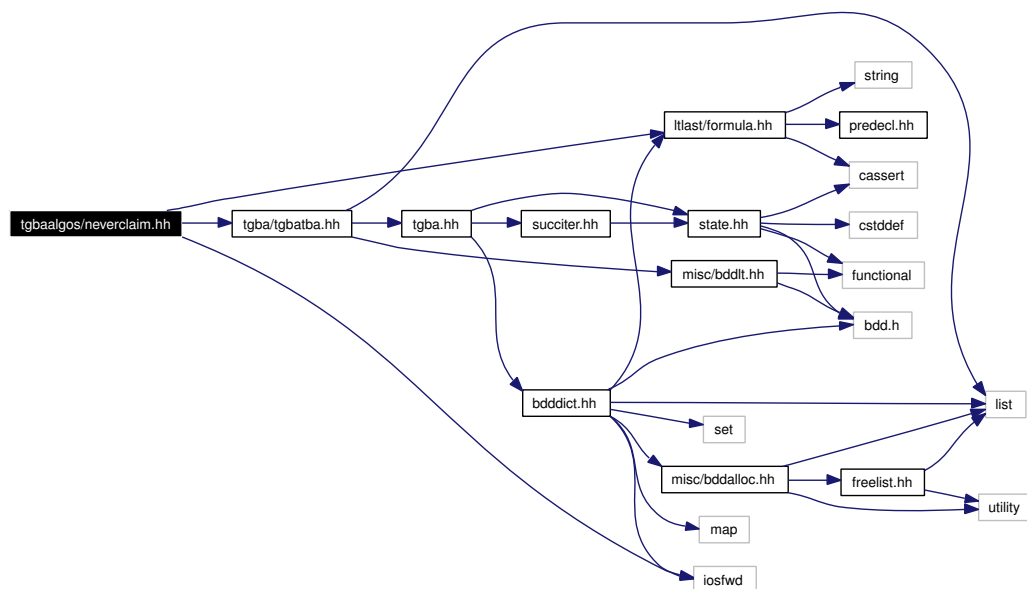
- `emptiness_check * explicit_magic_search` (const tgba *a, option_map o=option_map())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- `emptiness_check * bit_state_hashing_magic_search` (const tgba *a, size_t size, option_map o=option_map())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- `emptiness_check * magic_search` (const tgba *a, option_map o=option_map())

Wrapper for the two magic_search implementations.

13.97 tgbaalgos/neverclaim.hh File Reference

```
#include <iosfwd>
#include "ltlast/formula.hh"
#include "tgba/tgbatba.hh"
```

Include dependency graph for neverclaim.hh:



Namespaces

- namespace [spot](#)

Functions

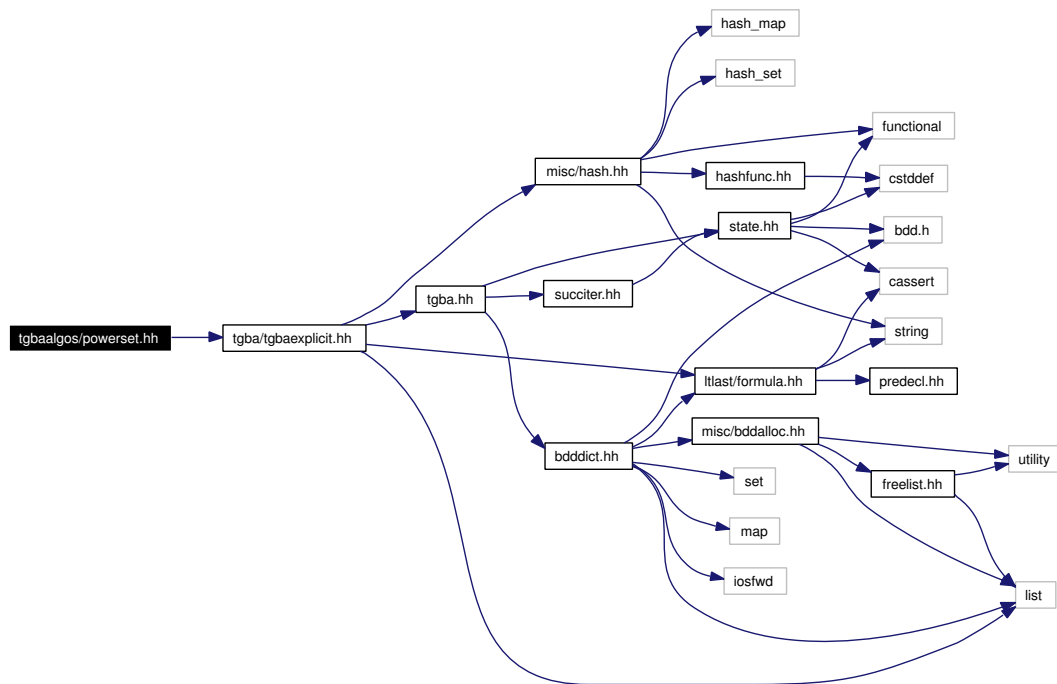
- `std::ostream & never_claim_reachable (std::ostream &os, const tgba_sba_proxy *g, const ltl::formula *f=0)`

Print reachable states in Spin never claim format.

13.98 tgbaalgos/powerset.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for powerset.hh:



Namespaces

- namespace [spot](#)

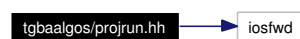
Functions

- `tgba_explicit * tgba_powerset` (const tgba *aut)
Build a deterministic automaton, ignoring acceptance conditions.

13.99 tgbaalgos/projrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for projrun.hh:



Namespaces

- namespace [spot](#)

Functions

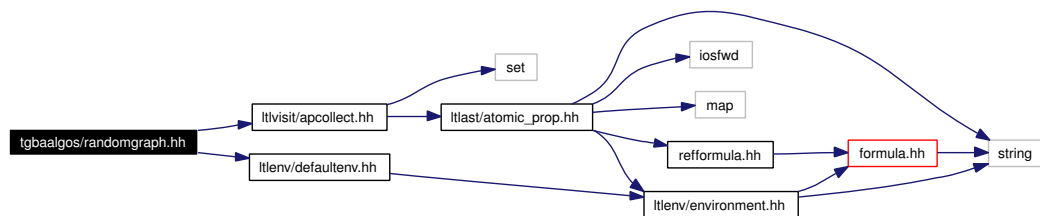
- `tgba_run * project_tgba_run` (const tgba *a_run, const tgba *a_proj, const tgba_run *run)
Project a [tgba_run](#) on a tgba.

13.100 tgbaalgos/randomgraph.hh File Reference

```
#include "ltlvisit/apcollect.hh"
```

```
#include "ltlenv/defaultenv.hh"
```

Include dependency graph for randomgraph.hh:



Namespaces

- namespace [spot](#)

Functions

- tgba * [random_graph](#) (int n, float d, const [ltl::atomic_prop_set](#) *ap, bdd_dict *dict, int n_acc=0, float a=0.1, float t=0.5, ltl::environment *env=<l::default_environment::instance())

Construct a tgba randomly.

13.101 tgbaalgos/reducerun.hh File Reference

Namespaces

- namespace [spot](#)

Functions

- tgba_run * [reduce_run](#) (const tgba *a, const tgba_run *org)

Reduce an accepting run.

13.102 tgbaalgos/reductgba_sim.hh File Reference

```
#include "tgba/tgbareduc.hh"
```

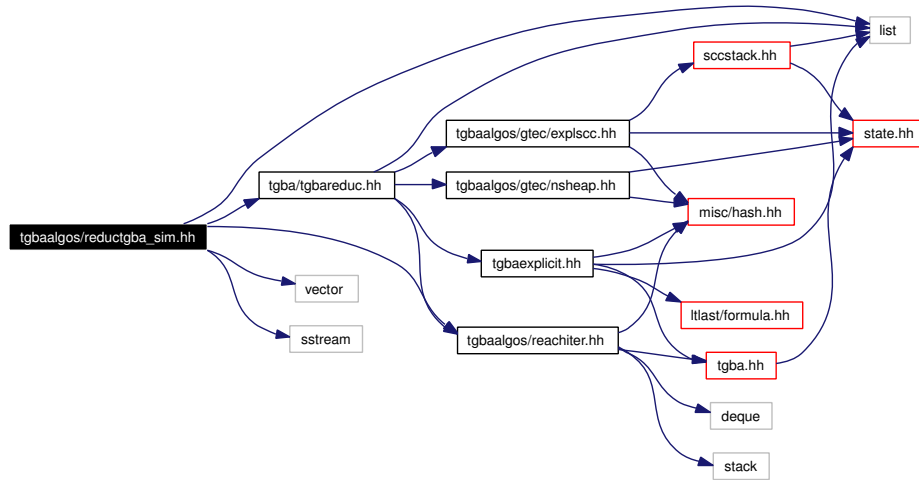
```
#include "tgbaalgos/reachiter.hh"
```

```
#include <vector>
```

```
#include <list>
```

```
#include <sstream>
```

Include dependency graph for reductgba_sim.hh:



Namespaces

- namespace [spot](#)

Typedefs

- typedef std::vector< spoiler_node * > [sn_v](#)
- typedef std::vector< duplicator_node * > [dn_v](#)
- typedef std::vector< const state * > [s_v](#)

Enumerations

- enum [reduce_tgba_options](#) {
[Reduce_None](#) = 0, [Reduce_quotient_Dir_Sim](#) = 1, [Reduce_transition_Dir_Sim](#) = 2, [Reduce_](#)
[quotient_Del_Sim](#) = 4,
[Reduce_transition_Del_Sim](#) = 8, [Reduce_Scc](#) = 16, [Reduce_All](#) = -1U }
Options for reduce.

Functions

- tgba * [reduc_tgba_sim](#) (const tgba *a, int opt=Reduce_All)
Remove some node of the automata using a simulation relation.
- direct_simulation_relation * [get_direct_relation_simulation](#) (const tgba *a, std::ostream &os, int opt=-1)
Compute a direct simulation relation on state of tgba f.
- delayed_simulation_relation * [get_delayed_relation_simulation](#) (const tgba *a, std::ostream &os, int opt=-1)
- void [free_relation_simulation](#) (direct_simulation_relation *rel)

To free a simulation relation.

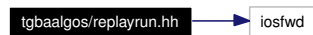
- void [free_relation_simulation](#) (delayed_simulation_relation *rel)

To free a simulation relation.

13.103 tgbaalgos/replayrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for replayrun.hh:



Namespaces

- namespace [spot](#)

Functions

- bool [replay_tgba_run](#) (std::ostream &os, const tgba *a, const tgba_run *run, bool debug=false)

Replay a [tgba_run](#) on a [tgba](#).

13.104 tgbaalgos/rundotdec.hh File Reference

```
#include <map>
```

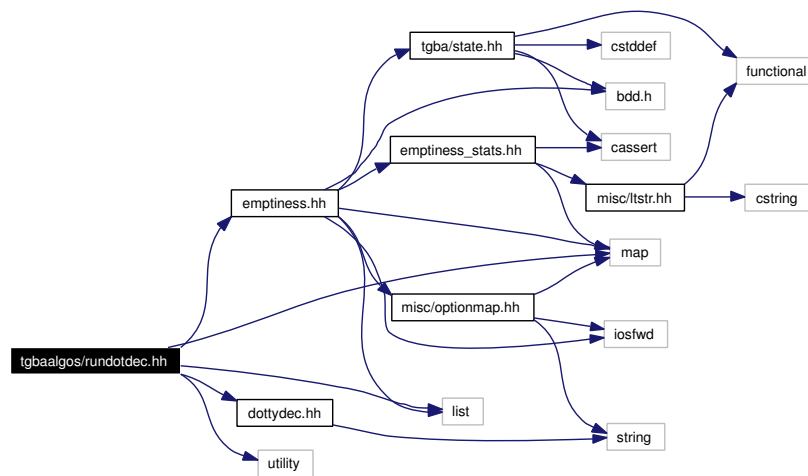
```
#include <utility>
```

```
#include <list>
```

```
#include "dottydec.hh"
```

```
#include "emptiness.hh"
```

Include dependency graph for rundotdec.hh:



Namespaces

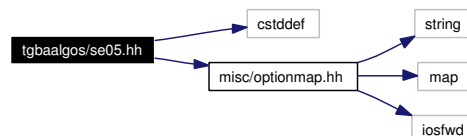
- namespace [spot](#)

13.105 tgbaalgos/se05.hh File Reference

```
#include <cstddef>
```

```
#include "misc/optionmap.hh"
```

Include dependency graph for se05.hh:



Namespaces

- namespace [spot](#)

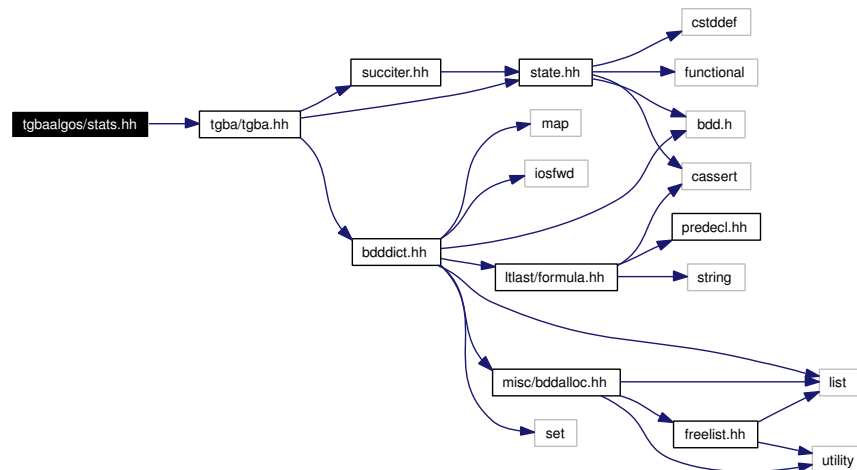
Functions

- emptiness_check * [explicit_se05_search](#) (const tgba *a, option_map o=option_map())
Returns an emptiness check on the [spot::tgba](#) automaton a.
- emptiness_check * [bit_state_hashing_se05_search](#) (const tgba *a, size_t size, option_map o=option_map())
Returns an emptiness checker on the [spot::tgba](#) automaton a.
- emptiness_check * [se05](#) (const tgba *a, option_map o)
Wrapper for the two se05 implementations.

13.106 tgbaalgos/stats.hh File Reference

```
#include "tgba/tgba.hh"
```

Include dependency graph for stats.hh:



Namespaces

- namespace [spot](#)

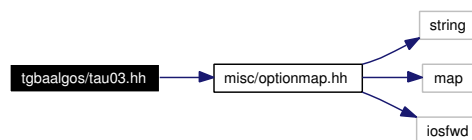
Functions

- `tgba_statistics` [stats_reachable](#) (const tgba *g)
Compute statistics for an automaton.

13.107 tgbaalgos/tau03.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for tau03.hh:



Namespaces

- namespace [spot](#)

Functions

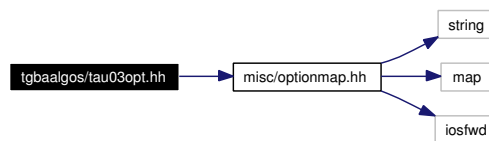
- emptiness_check * [explicit_tau03_search](#) (const tgba *a, option_map o=option_map())

Returns an emptiness checker on the *spot::tgba* automaton a.

13.108 tgbaalgos/tau03opt.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for tau03opt.hh:



Namespaces

- namespace [spot](#)

Functions

- emptiness_check * [explicit_tau03_opt_search](#) (const tgba *a, option_map o=option_map())

Returns an emptiness checker on the *spot::tgba* automaton a.

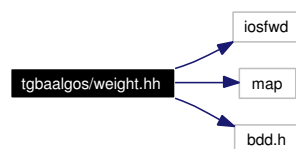
13.109 tgbaalgos/weight.hh File Reference

```
#include <iosfwd>
```

```
#include <map>
```

```
#include <bdd.h>
```

Include dependency graph for weight.hh:



Namespaces

- namespace [spot](#)

14 spot Page Documentation

14.1 Bug List

Member [spot::get_delayed_relation_simulation](#)(const tgba *a, std::ostream &os, int opt=-1) Does not work for generalized automata.

Member [spot::explicit_magic_search](#)(const tgba *a, option_map o=option_map()) The name is misleading. Magic-search is the algorithm from [godefroid.93.pstv](#), not [courcoubet.92.fmsd](#).

Index

- ~acss_statistics
 - spot::acss_statistics, [80](#)
- ~atomic_prop
 - spot::ltl::atomic_prop, [87](#)
- ~bdd_dict
 - spot::bdd_dict, [100](#)
- ~bfs_steps
 - spot::bfs_steps, [109](#)
- ~binop
 - spot::ltl::binop, [115](#)
- ~clone_visitor
 - spot::ltl::clone_visitor, [119](#)
- ~connected_component_hash_set
 - spot::connected_component_hash_set, [122](#)
- ~connected_component_hash_set_factory
 - spot::connected_component_hash_set_factory, [124](#)
- ~const_visitor
 - spot::ltl::const_visitor, [125](#)
- ~constant
 - spot::ltl::constant, [128](#)
- ~couvreur99_check
 - spot::couvreur99_check, [133](#)
- ~couvreur99_check_shy
 - spot::couvreur99_check_shy, [143](#)
- ~couvreur99_check_status
 - spot::couvreur99_check_status, [149](#)
- ~declarative_environment
 - spot::ltl::declarative_environment, [151](#)
- ~default_environment
 - spot::ltl::default_environment, [153](#)
- ~dotty_decorator
 - spot::dotty_decorator, [155](#)
- ~duplicator_node
 - spot::duplicator_node, [159](#)
- ~duplicator_node_delayed
 - spot::duplicator_node_delayed, [163](#)
- ~emptiness_check
 - spot::emptiness_check, [171](#)
- ~emptiness_check_result
 - spot::emptiness_check_result, [176](#)
- ~environment
 - spot::ltl::environment, [178](#)
- ~evtgba
 - spot::evtgba, [180](#)
- ~evtgba_explicit
 - spot::evtgba_explicit, [183](#)
- ~evtgba_iterator
 - spot::evtgba_iterator, [187](#)
- ~evtgba_product
 - spot::evtgba_product, [189](#)
- ~evtgba_reachable_iterator
 - spot::evtgba_reachable_iterator, [192](#)
- ~explicit_connected_component
 - spot::explicit_connected_component, [202](#)
- ~explicit_connected_component_factory
 - spot::explicit_connected_component_factory, [204](#)
- ~formula
 - spot::ltl::formula, [206](#)
- ~free_list
 - spot::free_list, [210](#)
- ~gspn_interface
 - spot::gspn_interface, [213](#)
- ~gspn_ssp_interface
 - spot::gspn_ssp_interface, [215](#)
- ~loopless_modular_mixed_radix_gray_code
 - spot::loopless_modular_mixed_radix_gray_code, [217](#)
- ~multop
 - spot::ltl::multop, [227](#)
- ~numbered_state_heap
 - spot::numbered_state_heap, [231](#)
- ~numbered_state_heap_const_iterator
 - spot::numbered_state_heap_const_iterator, [233](#)
- ~numbered_state_heap_factory
 - spot::numbered_state_heap_factory, [234](#)
- ~numbered_state_heap_hash_map
 - spot::numbered_state_heap_hash_map, [237](#)
- ~numbered_state_heap_hash_map_factory
 - spot::numbered_state_heap_hash_map_factory, [239](#)
- ~parity_game_graph
 - spot::parity_game_graph, [244](#)
- ~parity_game_graph_delayed
 - spot::parity_game_graph_delayed, [250](#)
- ~parity_game_graph_direct
 - spot::parity_game_graph_direct, [257](#)
- ~postfix_visitor
 - spot::ltl::postfix_visitor, [260](#)
- ~random_ltl
 - spot::ltl::random_ltl, [263](#)
- ~ref_formula
 - spot::ltl::ref_formula, [268](#)
- ~rsymbol
 - spot::rsymbol, [270](#)
- ~simplify_f_g_visitor
 - spot::ltl::simplify_f_g_visitor, [276](#)
- ~spoiler_node

- spot::spoiler_node, 278
- ~spoiler_node_delayed
 - spot::spoiler_node_delayed, 281
- ~state
 - spot::state, 284
- ~state_evtgba_explicit
 - spot::state_evtgba_explicit, 289
- ~state_explicit
 - spot::state_explicit, 291
- ~state_product
 - spot::state_product, 294
- ~symbol
 - spot::symbol, 298
- ~tgba
 - spot::tgba, 302
- ~tgba_bdd_concrete
 - spot::tgba_bdd_concrete, 309
- ~tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, 314
- ~tgba_bdd_factory
 - spot::tgba_bdd_factory, 321
- ~tgba_explicit
 - spot::tgba_explicit, 327
- ~tgba_explicit_succ_iterator
 - spot::tgba_explicit_succ_iterator, 334
- ~tgba_product
 - spot::tgba_product, 339
- ~tgba_reachable_iterator
 - spot::tgba_reachable_iterator, 344
- ~tgba_reduc
 - spot::tgba_reduc, 360
- ~tgba_run
 - spot::tgba_run, 369
- ~tgba_run_dotty_decorator
 - spot::tgba_run_dotty_decorator, 372
- ~tgba_succ_iterator
 - spot::tgba_succ_iterator, 382
- ~tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, 385
- ~tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, 390
- ~tgba_tba_proxy
 - spot::tgba_tba_proxy, 395
- ~unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, 405
- ~unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, 408
- ~unop
 - spot::ltl::unop, 412
- ~unsigned_statistics
 - spot::unsigned_statistics, 416
- ~visitor
 - spot::ltl::visitor, 418
- a_
 - spot::bfs_steps, 110
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 145
 - spot::emptiness_check, 172
 - spot::emptiness_check_result, 177
 - spot::loopless_modular_mixed_radix_gray_code, 218
 - spot::tgba_tba_proxy, 398
- a_first
 - spot::loopless_modular_mixed_radix_gray_code, 218
- a_last
 - spot::loopless_modular_mixed_radix_gray_code, 218
- a_next
 - spot::loopless_modular_mixed_radix_gray_code, 218
- acc
 - spot::couvreur99_check_shy::successor, 147
 - spot::tgba_run::step, 370
- acc_
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::tgba_bdd_concrete_factory, 315
 - spot::tgba_reduc, 368
- acc_cycle_
 - spot::tgba_sba_proxy, 380
 - spot::tgba_tba_proxy, 398
- acc_formula_map
 - spot::bdd_dict, 103
- acc_map
 - spot::bdd_dict, 103
- acc_map_
 - spot::tgba_bdd_concrete_factory, 314
- acc_set
 - spot::tgba_bdd_core_data, 319
- acc_set_
 - spot::evtgba_explicit, 185
- accept
 - spot::ltl::atomic_prop, 87
 - spot::ltl::binop, 115
 - spot::ltl::constant, 128
 - spot::ltl::formula, 206
 - spot::ltl::multop, 227
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 413
- acceptance_condition_visited_
 - spot::spoiler_node_delayed, 283
- acceptance_conditions
 - spot::evtgba_explicit::transition, 186
 - spot::tgba_bdd_core_data, 319
 - spot::tgba_explicit::transition, 332

- accepting_cycle
 - spot::couvreur99_check_result, 138
- accepting_run
 - spot::couvreur99_check_result, 138
 - spot::emptiness_check_result, 177
- acss_states
 - spot::acss_statistics, 80
 - spot::couvreur99_check_result, 138
- acss_statistics
 - spot::acss_statistics, 80
- add_acceptance_condition
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 360
- add_acceptance_conditions
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 360
- add_condition
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 361
- add_conditions
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 361
- add_duplicator_node_delayed
 - spot::parity_game_graph_delayed, 250
- add_pred
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 163
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- add_spoiler_node_delayed
 - spot::parity_game_graph_delayed, 250
- add_state
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 196
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
 - spot::tgba_explicit, 327
 - spot::tgba_reachable_iterator, 344
 - spot::tgba_reachable_iterator_breadth_first, 348
 - spot::tgba_reachable_iterator_depth_first, 352
 - spot::tgba_reduc, 361
- add_succ
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 163
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- add_transition
 - spot::evtgba_explicit, 184
- Algorithm patterns, 37
- Algorithms for LTL formulae, 19
- all_acc_
 - spot::evtgba_product, 190
- all_acceptance_conditions
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 189
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 309
 - spot::tgba_bdd_core_data, 319
 - spot::tgba_explicit, 327
 - spot::tgba_product, 339
 - spot::tgba_reduc, 361
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 395
- all_acceptance_conditions_
 - spot::tgba_explicit, 331
 - spot::tgba_explicit_succ_iterator, 335
 - spot::tgba_product, 341
 - spot::tgba_reduc, 368
- all_acceptance_conditions_computed_
 - spot::tgba_explicit, 331
 - spot::tgba_reduc, 368
- allocate_variables
 - spot::bdd_allocator, 94
 - spot::bdd_dict, 101
- alphabet
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 189
- alphabet_
 - spot::evtgba_explicit, 185
 - spot::evtgba_product, 190
- And
 - spot::ltl::multop, 227
- annon_free_list
 - spot::bdd_dict::annon_free_list, 106
- ap
 - spot::ltl::random_ltl, 263
- ap_
 - spot::ltl::random_ltl, 264
- arc
 - spot::couvreur99_check_shy, 145
- ars_cycle_states
 - spot::acss_statistics, 80
 - spot::ars_statistics, 83
 - spot::couvreur99_check_result, 138
- ars_prefix_states
 - spot::acss_statistics, 80
 - spot::ars_statistics, 83
 - spot::couvreur99_check_result, 138
- ars_statistics
 - spot::ars_statistics, 83
- as_bdd

- spot::state_bdd, 286
- assert_emptiness
 - spot::bdd_dict, 101
- atomic_prop
 - spot::ltl::atomic_prop, 87
- atomic_prop_collect
 - ltl_misc, 25
- atomic_prop_set
 - ltl_misc, 25
- aut
 - spot::couvreur99_check_status, 149
- automata_
 - spot::evtgba_reachable_iterator, 194
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 201
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
 - spot::tgba_reachable_iterator, 345
 - spot::tgba_reachable_iterator_breadth_first, 349
 - spot::tgba_reachable_iterator_depth_first, 353
 - spot::tgba_reduc, 368
- automaton
 - spot::couvreur99_check, 133
 - spot::couvreur99_check_result, 138
 - spot::couvreur99_check_shy, 143
 - spot::emptiness_check, 171
 - spot::emptiness_check_result, 177
 - spot::gspn_interface, 214
 - spot::gspn_ssp_interface, 216
- barand
 - spot::barand, 90
- basic_reduce
 - ltl_rewriting, 24
- bdd_allocator
 - spot::bdd_allocator, 94
- bdd_dict
 - spot::bdd_dict, 100
- bdd_format_accset
 - bddprint.hh, 465
 - spot, 71
- bdd_format_formula
 - bddprint.hh, 465
 - spot, 71
- bdd_format_sat
 - bddprint.hh, 465
 - spot, 71
- bdd_format_set
 - bddprint.hh, 466
 - spot, 71
- bdd_print_acc
 - bddprint.hh, 466
 - spot, 72
- bdd_print_accset
 - bddprint.hh, 466
 - spot, 72
- bdd_print_dot
 - bddprint.hh, 466
 - spot, 72
- bdd_print_formula
 - bddprint.hh, 467
 - spot, 72
- bdd_print_sat
 - bddprint.hh, 467
 - spot, 73
- bdd_print_set
 - bddprint.hh, 467
 - spot, 73
- bdd_print_table
 - bddprint.hh, 467
 - spot, 73
- bdd_to_formula
 - formula2bdd.hh, 468
 - spot, 73
- bdd_v
 - spot::parity_game_graph_delayed, 250
- bddprint.hh
 - bdd_format_accset, 465
 - bdd_format_formula, 465
 - bdd_format_sat, 465
 - bdd_format_set, 466
 - bdd_print_acc, 466
 - bdd_print_accset, 466
 - bdd_print_dot, 466
 - bdd_print_formula, 467
 - bdd_print_sat, 467
 - bdd_print_set, 467
 - bdd_print_table, 467
- bfs_steps
 - spot::bfs_steps, 109
- binop
 - spot::ltl::binop, 115
- bit_state_hashing_magic_search
 - emptiness_check_algorithms, 44
- bit_state_hashing_se05_search
 - emptiness_check_algorithms, 44
- bmrand
 - random, 30
- build
 - spot::connected_component_hash_set_factory, 124
 - spot::explicit_connected_component_factory, 204
 - spot::ltl::random_ltl::op_proba, 265

- spot::numbered_state_heap_factory, 234
- spot::numbered_state_heap_hash_map_factory, 239
- build_graph
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
- build_link
 - spot::parity_game_graph_direct, 257
- build_recurse_successor_duplicator
 - spot::parity_game_graph_delayed, 251
- build_recurse_successor_spoiler
 - spot::parity_game_graph_delayed, 251
- builder
 - spot::ltl::random_ltl::op_proba, 265
- cancel
 - spot::timer_map, 401
- check
 - spot::couvreur99_check, 133
 - spot::couvreur99_check_shy, 143
 - spot::emptiness_check, 171
- child
 - spot::ltl::unop, 413
- child_
 - spot::ltl::unop, 414
- children_
 - spot::ltl::multop, 229
- clear_rem
 - spot::scc_stack, 272
- clear_todo
 - spot::couvreur99_check_shy, 144
- clone
 - ltl_essential, 18
 - spot::state, 284
 - spot::state_bdd, 286
 - spot::state_evtgba_explicit, 289
 - spot::state_explicit, 291
 - spot::state_product, 294
- clone_visitor
 - spot::ltl::clone_visitor, 119
- common.hh
 - operator<<, 422
- common_symbol_table
 - spot::evtgba_product, 189
- common_symbols_
 - spot::evtgba_product, 190
- compare
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 163
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
 - spot::state, 284
 - spot::state_bdd, 287
- spot::state_evtgba_explicit, 289
- spot::state_explicit, 291
- spot::state_product, 294
- complement_all_acceptance_conditions
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 361
- compute_scc
 - spot::tgba_reduc, 361
- compute_support_conditions
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 309
 - spot::tgba_explicit, 327, 328
 - spot::tgba_product, 339
 - spot::tgba_reduc, 361
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 395
- compute_support_variables
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 309
 - spot::tgba_explicit, 328
 - spot::tgba_product, 339
 - spot::tgba_reduc, 361
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 395
- condition
 - spot::connected_component_hash_set, 122
 - spot::explicit_connected_component, 203
 - spot::scc_stack::connected_component, 274
 - spot::tgba_explicit::transition, 332
- connected_component
 - spot::scc_stack::connected_component, 274
- connected_component_hash_set_factory
 - spot::connected_component_hash_set_factory, 124
- constant
 - spot::ltl::constant, 128
- constrain_relation
 - spot::tgba_bdd_concrete_factory, 314
- construct
 - spot::emptiness_check_instantiator, 174
- copy_acceptance_conditions_of
 - spot::tgba_explicit, 328
 - spot::tgba_reduc, 362
- couvreur99
 - emptiness_check_algorithms, 45
- couvreur99_check
 - spot::couvreur99_check, 133
- couvreur99_check_result
 - spot::couvreur99_check_result, 138
- couvreur99_check_shy
 - spot::couvreur99_check_shy, 143
- couvreur99_check_ssp_semi
 - emptiness_check_ssp, 17
- couvreur99_check_ssp_shy

- emptiness_check_ssp, 17
- couvreur99_check_ssp_shy_semi
 - emptiness_check_ssp, 18
- couvreur99_check_status
 - spot::couvreur99_check_status, 149
- create_atomic_prop
 - spot::tgba_bdd_concrete_factory, 314
- create_state
 - spot::tgba_bdd_concrete_factory, 315
- create_transition
 - spot::tgba_explicit, 328
 - spot::tgba_reduc, 362
- cube_
 - spot::minato_isop, 220
- current_
 - spot::tgba_succ_iterator_concrete, 386
- current_acc_
 - spot::tgba_succ_iterator_concrete, 386
- current_acceptance_conditions
 - spot::evtgba_iterator, 187
 - spot::tgba_explicit_succ_iterator, 334
 - spot::tgba_succ_iterator, 382
 - spot::tgba_succ_iterator_concrete, 385
 - spot::tgba_succ_iterator_product, 390
- current_cond_
 - spot::tgba_succ_iterator_product, 391
- current_condition
 - spot::tgba_explicit_succ_iterator, 334
 - spot::tgba_succ_iterator, 382
 - spot::tgba_succ_iterator_concrete, 385
 - spot::tgba_succ_iterator_product, 390
- current_label
 - spot::evtgba_iterator, 187
- current_state
 - spot::evtgba_iterator, 187
 - spot::tgba_explicit_succ_iterator, 334
 - spot::tgba_succ_iterator, 382
 - spot::tgba_succ_iterator_concrete, 386
 - spot::tgba_succ_iterator_product, 390
- current_state_
 - spot::tgba_succ_iterator_concrete, 387
- cycle
 - spot::tgba_run, 370
- cycle_list
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 395
- cycle_seed
 - spot::couvreur99_check_status, 149
- cycle_states_
 - spot::ars_statistics, 83
- data_
 - spot::tgba_bdd_concrete, 312
 - spot::tgba_bdd_concrete_factory, 315
 - spot::tgba_succ_iterator_concrete, 387
- dead_
 - spot::gspn_interface, 214
- dec_depth
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_shy, 144
 - spot::ec_statistics, 168
- dec_weight_handler
 - spot::weight, 420
- declarative_environment
 - spot::ltl::declarative_environment, 151
- declare
 - spot::ltl::declarative_environment, 151
- declare_acceptance_condition
 - spot::evtgba_explicit, 184
 - spot::tgba_bdd_concrete_factory, 315
 - spot::tgba_bdd_core_data, 318
 - spot::tgba_explicit, 328
 - spot::tgba_reduc, 362
- declare_atomic_prop
 - spot::tgba_bdd_core_data, 318
- declare_now_next
 - spot::tgba_bdd_core_data, 318
- declare_state
 - spot::evtgba_explicit, 184
- Decorating the dot output, 42
- default_environment
 - spot::ltl::default_environment, 153
- del_pred
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- del_succ
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- delete_scc
 - spot::tgba_reduc, 362
- delete_transitions
 - spot::tgba_reduc, 362
- depth
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_shy, 144
 - spot::ec_statistics, 168
- depth_
 - spot::ec_statistics, 168
- Derivable visitors, 22
- dest
 - spot::tgba_explicit::transition, 332
- destroy
 - ltl_essential, 18
- dict

- spot::tgba_bdd_core_data, 319
- dict_
 - spot::bdd_dict::annon_free_list, 107
 - spot::gspn_interface, 214
 - spot::gspn_ssp_interface, 216
 - spot::tgba_explicit, 331
 - spot::tgba_product, 341
 - spot::tgba_reduc, 368
- display_rel_sim
 - spot::tgba_reduc, 362
- display_scc
 - spot::tgba_reduc, 362
- dn_v
 - tgba_reduction, 39
- doit
 - spot::ltl::postfix_visitor, 260, 261
- doit_default
 - spot::ltl::postfix_visitor, 261
- done
 - spot::evtgba_iterator, 187
 - spot::loopless_modular_mixed_radix_gray_code, 218
 - spot::numbered_state_heap_const_iterator, 233
 - spot::tgba_explicit_succ_iterator, 334
 - spot::tgba_succ_iterator, 382
 - spot::tgba_succ_iterator_concrete, 386
 - spot::tgba_succ_iterator_product, 390
- done_
 - spot::loopless_modular_mixed_radix_gray_code, 218
- dotty
 - ltl_io, 21
- dotty_decorator
 - spot::dotty_decorator, 155
- dotty_reachable
 - evtgbalgorithms/dotty.hh, 428
 - spot, 73
 - tgba_io, 34
- drand
 - random, 30
- dump
 - ltl_io, 21
 - spot::bdd_dict, 101
 - spot::ltl::atomic_prop, 87
 - spot::ltl::binop, 115
 - spot::ltl::constant, 128
 - spot::ltl::formula, 206
 - spot::ltl::multop, 227
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 413
- dump_
 - spot::ltl::atomic_prop, 89
 - spot::ltl::binop, 116
 - spot::ltl::constant, 130
 - spot::ltl::formula, 207
 - spot::ltl::multop, 229
 - spot::ltl::ref_formula, 269
 - spot::ltl::unop, 414
- dump_free_list
 - spot::bdd_allocator, 94
 - spot::bdd_dict::annon_free_list, 106
 - spot::free_list, 210
- dump_instances
 - spot::ltl::atomic_prop, 87
 - spot::symbol, 298
- dump_priorities
 - spot::ltl::random_ltl, 263
- duplicator_node
 - spot::duplicator_node, 159
- duplicator_node_delayed
 - spot::duplicator_node_delayed, 163
- duplicator_vertice_
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
- ec_statistics
 - spot::ec_statistics, 168
- ecs_
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 145
- Emptiness-check algorithms, 43
- Emptiness-check algorithms for SSP, 17
- Emptiness-check statistics, 53
- Emptiness-checks, 42
- emptiness_check
 - spot::emptiness_check, 171
- emptiness_check_algorithms
 - bit_state_hashing_magic_search, 44
 - bit_state_hashing_se05_search, 44
 - couvreur99, 45
 - explicit_gv04_check, 46
 - explicit_magic_search, 47
 - explicit_se05_search, 48
 - explicit_tau03_opt_search, 49
 - explicit_tau03_search, 50
 - magic_search, 51
 - se05, 51
- emptiness_check_instantiator
 - spot::emptiness_check_instantiator, 173
- emptiness_check_result
 - spot::emptiness_check_result, 176
- emptiness_check_ssp
 - couvreur99_check_ssp_semi, 17
 - couvreur99_check_ssp_shy, 17
 - couvreur99_check_ssp_shy_semi, 18

- empty
 - spot::scc_stack, 272
 - spot::timer_map, 401
- end
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 196
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
 - spot::tgba_reachable_iterator, 344
 - spot::tgba_reachable_iterator_breadth_first, 348
 - spot::tgba_reachable_iterator_depth_first, 352
 - spot::tgba_reduc, 362
- env
 - spot::ltl::atomic_prop, 87
- env_
 - spot::gspn_interface, 214
 - spot::gspn_ssp_interface, 216
 - spot::ltl::atomic_prop, 89
- Equiv
 - spot::ltl::binop, 114
- err_
 - spot::gspn_exeption, 212
- escape_str
 - misc_tools, 28
- Essential LTL types, 18
- Essential TGBA types, 31
- evtgba
 - spot::evtgba, 180
- evtgba/ Directory Reference, 54
- evtgba/evtgba.hh, 424
- evtgba/evtgbaiter.hh, 424
- evtgba/explicit.hh, 425
- evtgba/product.hh, 426
- evtgba/symbol.hh, 427
- evtgba_explicit
 - spot::evtgba_explicit, 183
- evtgba_parse
 - evtgbaparse/public.hh, 434
 - spot, 73
- evtgba_parse_error
 - evtgbaparse/public.hh, 434
 - spot, 70
- evtgba_parse_error_list
 - evtgbaparse/public.hh, 434
 - spot, 70
- evtgba_product
 - spot::evtgba_product, 189
- evtgba_product_operands
 - spot::evtgba_product, 189
- evtgba_reachable_iterator
 - spot::evtgba_reachable_iterator, 192
- evtgba_reachable_iterator_breadth_first
 - spot::evtgba_reachable_iterator_breadth_first, 196
- evtgba_reachable_iterator_depth_first
 - spot::evtgba_reachable_iterator_depth_first, 199
- evtgba_save_reachable
 - evtgbaalgos/save.hh, 431
 - spot, 74
- evtgbaalgos/ Directory Reference, 54
- evtgbaalgos/dotty.hh, 427
 - dotty_reachable, 428
- evtgbaalgos/reachiter.hh, 429
- evtgbaalgos/save.hh, 431
 - evtgba_save_reachable, 431
- evtgbaalgos/tgba2evtgba.hh, 432
- evtgbaparse/ Directory Reference, 55
- evtgbaparse/public.hh, 433
 - evtgba_parse, 434
 - evtgba_parse_error, 434
 - evtgba_parse_error_list, 434
 - format_evtgba_parse_errors, 434
- explicit_gv04_check
 - emptiness_check_algorithms, 46
- explicit_magic_search
 - emptiness_check_algorithms, 47
- explicit_se05_search
 - emptiness_check_algorithms, 48
- explicit_tau03_opt_search
 - emptiness_check_algorithms, 49
- explicit_tau03_search
 - emptiness_check_algorithms, 50
- extend
 - spot::bdd_allocator, 94
 - spot::bdd_dict::annon_free_list, 106
 - spot::free_list, 210
- extvarnum
 - spot::bdd_allocator, 94
- F
 - spot::ltl::unop, 412
- f0_max
 - spot::minato_isop::local_vars, 221
- f0_min
 - spot::minato_isop::local_vars, 221
- f1_max
 - spot::minato_isop::local_vars, 222
- f1_min
 - spot::minato_isop::local_vars, 222
- f_
 - spot::loopless_modular_mixed_radix_gray_code, 218

- f_max
 - spot::minato_isop::local_vars, 222
- f_min
 - spot::minato_isop::local_vars, 222
- False
 - spot::ltl::constant, 128
- false_instance
 - spot::ltl::constant, 128
- filter
 - spot::bfs_steps, 109
- finalize
 - spot::bfs_steps, 109
- find
 - spot::numbered_state_heap, 231
 - spot::numbered_state_heap_hash_map, 237
- find_state
 - spot::couvreur99_check_shy, 144
- finish
 - spot::tgba_bdd_concrete_factory, 315
- first
 - spot::evtgba_iterator, 187
 - spot::loopless_modular_mixed_radix_gray_code, 218
 - spot::ltl::binop, 115
 - spot::numbered_state_heap_const_iterator, 233
 - spot::tgba_explicit_succ_iterator, 334
 - spot::tgba_succ_iterator, 382
 - spot::tgba_succ_iterator_concrete, 386
 - spot::tgba_succ_iterator_product, 390
- first_
 - spot::ltl::binop, 116
- FirstStep
 - spot::minato_isop::local_vars, 221
- fl
 - spot::bdd_allocator, 95
 - spot::bdd_dict::annon_free_list, 107
 - spot::free_list, 211
- format_acceptance_condition
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 190
- format_acceptance_conditions
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 190
- format_evtgba_parse_errors
 - evtgbaparse/public.hh, 434
 - spot, 74
- format_label
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 190
- format_parse_errors
 - ltl_io, 21
- format_state
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 190
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 309
 - spot::tgba_explicit, 328
 - spot::tgba_product, 339
 - spot::tgba_reduc, 362
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 395
- format_tgba_parse_errors
 - tgba_io, 34
- formula2bdd.hh
 - bdd_to_formula, 468
 - formula_to_bdd, 468
- formula_to_bdd
 - formula2bdd.hh, 468
 - spot, 74
- FourthStep
 - spot::minato_isop::local_vars, 221
- free_anonymous_list_of
 - spot::bdd_dict, 103
- free_anonymous_list_of_type
 - spot::bdd_dict, 100
- free_list_type
 - spot::bdd_allocator, 93
 - spot::bdd_dict::annon_free_list, 106
 - spot::free_list, 210
- free_relation_simulation
 - tgba_reduction, 39
- fv_map
 - spot::bdd_dict, 100
- G
 - spot::ltl::unop, 412
- g0
 - spot::minato_isop::local_vars, 222
- g1
 - spot::minato_isop::local_vars, 222
- generate
 - spot::ltl::random_ltl, 263
- get
 - spot::acss_statistics, 80
 - spot::ars_statistics, 83
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 144
 - spot::ec_statistics, 168
 - spot::option_map, 240
 - spot::unsigned_statistics, 416
- get_acc
 - spot::duplicator_node, 159

- spot::duplicator_node_delayed, 164
- get_acceptance_condition
 - spot::tgba_explicit, 328
 - spot::tgba_reduc, 363
- get_acceptance_condition_visited
 - spot::spoiler_node_delayed, 282
- get_core_data
 - spot::tgba_bdd_concrete, 310
 - spot::tgba_bdd_concrete_factory, 315
 - spot::tgba_bdd_factory, 321
- get_delayed_relation_simulation
 - tgba_reduction, 40
- get_dict
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 310
 - spot::tgba_bdd_concrete_factory, 315
 - spot::tgba_explicit, 328
 - spot::tgba_product, 339
 - spot::tgba_reduc, 363
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 396
- get_direct_relation_simulation
 - tgba_reduction, 40
- get_duplicator_node
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- get_err
 - spot::gspn_exeption, 212
- get_index
 - spot::numbered_state_heap_const_iterator, 233
- get_init_bdd
 - spot::tgba_bdd_concrete, 310
- get_init_state
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 310
 - spot::tgba_explicit, 329
 - spot::tgba_product, 340
 - spot::tgba_reduc, 363
 - spot::tgba_sba_proxy, 377
 - spot::tgba_tba_proxy, 396
- get_label
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 164
- get_lead_2_acc_all
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node_delayed, 282
- get_nb_succ
 - spot::duplicator_node, 159
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 278
 - spot::spoiler_node_delayed, 282
- get_pair
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 282
- get_progress_measure
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node_delayed, 282
- get_prop_map
 - spot::ltl::declarative_environment, 151
- get_relation
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
- get_spoiler_node
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 282
- get_state
 - spot::numbered_state_heap_const_iterator, 233
 - spot::state_evtgba_explicit, 289
 - spot::state_explicit, 292
- get_where
 - spot::gspn_exeption, 212
- group_
 - spot::couvereur99_check_shy, 145
- gspn/ Directory Reference, 55
- gspn/common.hh, 421
- gspn/gspn.hh, 422
- gspn/ssp.hh, 423
- gspn_exeption
 - spot::gspn_exeption, 212
- gspn_interface
 - spot::gspn_interface, 213
- gspn_ssp_interface
 - spot::gspn_ssp_interface, 215
- h
 - spot::couvereur99_check_status, 149
 - spot::numbered_state_heap_hash_map, 238
- h_
 - spot::tgba_reduc, 368
- has_acceptance_condition
 - spot::tgba_explicit, 329
 - spot::tgba_reduc, 363
- has_state
 - spot::connected_component_hash_set, 122
 - spot::explicit_connected_component, 202
- hash
 - spot::ltl::atomic_prop, 87
 - spot::ltl::binop, 115
 - spot::ltl::constant, 129

- spot::ltl::formula, 206
- spot::ltl::multop, 227
- spot::ltl::ref_formula, 268
- spot::ltl::unop, 413
- spot::state, 285
- spot::state_bdd, 287
- spot::state_evtgba_explicit, 289
- spot::state_explicit, 292
- spot::state_product, 294
- hash_funcs
 - knuth32_hash, 29
 - wang32_hash, 29
- hash_key_
 - spot::ltl::atomic_prop, 89
 - spot::ltl::binop, 117
 - spot::ltl::constant, 130
 - spot::ltl::formula, 207
 - spot::ltl::multop, 229
 - spot::ltl::ref_formula, 269
 - spot::ltl::unop, 414
- hash_type
 - spot::numbered_state_heap_hash_map, 236
- Hashing functions, 29
- i_
 - spot::tgba_explicit_succ_iterator, 335
- Implies
 - spot::ltl::binop, 114
- implies
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
- implies_acc
 - spot::duplicator_node_delayed, 164
- implies_label
 - spot::duplicator_node_delayed, 164
- in
 - spot::evtgba_explicit::state, 186
 - spot::evtgba_explicit::transition, 186
- inc_ars_cycle_states
 - spot::acss_statistics, 81
 - spot::ars_statistics, 83
 - spot::couvereur99_check_result, 139
- inc_ars_prefix_states
 - spot::acss_statistics, 81
 - spot::ars_statistics, 83
 - spot::couvereur99_check_result, 139
- inc_depth
 - spot::couvereur99_check, 134
 - spot::couvereur99_check_shy, 144
 - spot::ec_statistics, 168
- inc_states
 - spot::couvereur99_check, 134
 - spot::couvereur99_check_shy, 144
 - spot::ec_statistics, 168
- inc_transitions
 - spot::couvereur99_check, 134
 - spot::couvereur99_check_shy, 144
 - spot::ec_statistics, 168
- inc_weight_handler
 - spot::weight, 420
- index
 - spot::connected_component_hash_set, 122
 - spot::explicit_connected_component, 203
 - spot::numbered_state_heap, 232
 - spot::numbered_state_heap_hash_map, 237
 - spot::scc_stack::connected_component, 274
- index_and_insert
 - spot::numbered_state_heap, 232
 - spot::numbered_state_heap_hash_map, 237
- info_
 - spot::emptiness_check_instantiator, 174
- init_
 - spot::tgba_bdd_concrete, 312
 - spot::tgba_explicit, 331
 - spot::tgba_reduc, 368
- init_iter
 - spot::evtgba, 180
 - spot::evtgba_explicit, 184
 - spot::evtgba_product, 190
- init_states_
 - spot::evtgba_explicit, 185
- initialize
 - spot::bdd_allocator, 94
 - spot::bdd_dict, 101
- initialized
 - spot::bdd_allocator, 95
 - spot::bdd_dict, 103
- Input/Output of LTL formulae, 19
- Input/Output of TGBA, 33
- insert
 - spot::bdd_allocator, 94
 - spot::bdd_dict::annon_free_list, 106
 - spot::connected_component_hash_set, 122
 - spot::explicit_connected_component, 202
 - spot::free_list, 211
 - spot::numbered_state_heap, 232
 - spot::numbered_state_heap_hash_map, 237
- instance
 - spot::connected_component_hash_set_-factory, 124
 - spot::dotty_decorator, 156
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 115
 - spot::ltl::default_environment, 153
 - spot::ltl::multop, 227
 - spot::ltl::unop, 413
 - spot::numbered_state_heap_hash_map_-factory, 239

- spot::symbol, 298
- spot::tgba_run_dotty_decorator, 373
- instance_count
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 115
 - spot::ltl::multop, 228
 - spot::ltl::unop, 413
 - spot::symbol, 298
- instances
 - spot::ltl::atomic_prop, 89
 - spot::ltl::binop, 117
 - spot::ltl::multop, 229
 - spot::ltl::unop, 414
- instances_
 - spot::symbol, 299
- instantiate
 - spot::emptiness_check_instantiator, 174
- is_bare_word
 - misc_tools, 28
- is_eventual
 - ltl_misc, 25
- is_FG
 - ltl_misc, 26
- is_GF
 - ltl_misc, 26
- is_not_accepting
 - spot::tgba_reduc, 363
- is_registered_acceptance_variable
 - spot::bdd_dict, 101
- is_registered_proposition
 - spot::bdd_dict, 101
- is_registered_state
 - spot::bdd_dict, 101
- is_terminal
 - spot::tgba_reduc, 363
- is_universal
 - ltl_misc, 26
- item_type
 - spot::timer_map, 401
- iterator
 - spot::numbered_state_heap, 232
 - spot::numbered_state_heap_hash_map, 237
- knuth32_hash
 - hash_funcs, 29
- label
 - spot::evtgba_explicit::transition, 186
 - spot::tgba_run::step, 370
- label_
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
- last
 - spot::loopless_modular_mixed_radix_gray_code, 218
- last_support_conditions_input_
 - spot::tgba, 305
- last_support_conditions_output_
 - spot::tgba, 305
- last_support_variables_input_
 - spot::tgba, 305
- last_support_variables_output_
 - spot::tgba, 305
- lbt_reachable
 - tgba_io, 34
- lead_2_acc_all_
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node_delayed, 283
- left
 - spot::state_product, 295
- left_
 - spot::state_product, 295
 - spot::tgba_product, 342
 - spot::tgba_succ_iterator_product, 391
- left_acc_complement_
 - spot::tgba_product, 342
- left_neg_
 - spot::tgba_succ_iterator_product, 391
- length
 - ltl_misc, 26
- lift
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
- link_decl
 - spot::dotty_decorator, 156
 - spot::tgba_run_dotty_decorator, 373
- lnode_pred
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 283
- lnode_succ
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 283
- local_vars
 - spot::minato_isop::local_vars, 221
- loopless_modular_mixed_radix_gray_code
 - spot::loopless_modular_mixed_radix_gray_code, 217
- LTL Abstract Syntax Tree, 18
- LTL environments, 19
- LTL formulae, 17
- ltl_essential
 - clone, 18

- destroy, 18
- ltl_io
 - dotty, 21
 - dump, 21
 - format_parse_errors, 21
 - parse, 21
 - parse_error, 20
 - parse_error_list, 20
 - to_spin_string, 22
 - to_string, 22
- ltl_misc
 - atomic_prop_collect, 25
 - atomic_prop_set, 25
 - is_eventual, 25
 - is_FG, 26
 - is_GF, 26
 - is_universal, 26
 - length, 26
 - syntactic_implication, 26
 - syntactic_implication_neg, 27
- ltl_rewriting
 - Reduce_All, 23
 - Reduce_Basics, 23
 - Reduce_Eventuality_And_Universality, 23
 - Reduce_None, 23
 - Reduce_Syntactic_Implications, 23
- ltl_rewriting
 - basic_reduce, 24
 - negative_normal_form, 24
 - reduce, 24
 - reduce_options, 23
 - simplify_f_g, 24
 - unabbreviate_logic, 24
- ltl_to_tgba_fm
 - tgba_ltl, 36
- ltl_to_tgba_lacim
 - tgba_ltl, 37
- ltlast/ Directory Reference, 56
- ltlast/allnodes.hh, 438
- ltlast/atomic_prop.hh, 438
- ltlast/binop.hh, 439
- ltlast/constant.hh, 440
- ltlast/formula.hh, 440
- ltlast/multop.hh, 442
- ltlast/predecl.hh, 442
- ltlast/refformula.hh, 443
- ltlast/unop.hh, 443
- ltlast/visitor.hh, 444
- ltlenv/ Directory Reference, 57
- ltlenv/declenv.hh, 445
- ltlenv/defaultenv.hh, 445
- ltlenv/environment.hh, 446
- ltlparse/ Directory Reference, 57
- ltlparse/public.hh, 435
- ltlvisit/ Directory Reference, 57
- ltlvisit/apcollect.hh, 446
- ltlvisit/basicreduce.hh, 447
- ltlvisit/clone.hh, 448
- ltlvisit/destroy.hh, 449
- ltlvisit/dotty.hh, 428
- ltlvisit/dump.hh, 449
- ltlvisit/length.hh, 450
- ltlvisit/lunabbrev.hh, 450
- ltlvisit/nenoform.hh, 451
- ltlvisit/postfix.hh, 451
- ltlvisit/randomltl.hh, 452
- ltlvisit/reduce.hh, 452
- ltlvisit/simpfpg.hh, 453
- ltlvisit/syntimpl.hh, 453
- ltlvisit/tostring.hh, 454
- ltlvisit/tunabbrev.hh, 455
- lvarnum
 - spot::bdd_allocator, 95
 - spot::bdd_dict, 103
- m
 - spot::weight, 421
- m_
 - spot::barand, 90
 - spot::loopless_modular_mixed_radix_gray_code, 218
- magic_search
 - emptiness_check_algorithms, 51
- mainpage.dox, 421
- map
 - spot::ltl::atomic_prop, 87
 - spot::ltl::binop, 114
 - spot::ltl::multop, 226
 - spot::ltl::unop, 412
 - spot::symbol, 298
- map_
 - spot::tgba_run_dotty_decorator, 373
- match
 - spot::bfs_steps, 110
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
- max_acceptance_conditions
 - spot::emptiness_check_instantiator, 174
- max_depth
 - spot::couvereur99_check, 134
 - spot::couvereur99_check_shy, 144
 - spot::ec_statistics, 168
- max_depth_
 - spot::ec_statistics, 168
- merge_state
 - spot::tgba_reduc, 363
- merge_state_delayed
 - spot::tgba_reduc, 364

- merge_transitions
 - spot::tgba_explicit, 329
 - spot::tgba_reduc, 364
- min_acceptance_conditions
 - spot::emptiness_check_instantiator, 174
- min_n
 - spot::ltl::random_ltl::op_proba, 265
- minato_isop
 - spot::minato_isop, 220
- misc/ Directory Reference, 58
- misc/bareword.hh, 455
- misc/bddalloc.hh, 456
- misc/bddlt.hh, 456
- misc/escape.hh, 457
- misc/freelist.hh, 457
- misc/hash.hh, 458
- misc/hashfunc.hh, 458
- misc/ltstr.hh, 459
- misc/minato.hh, 460
- misc/modgray.hh, 460
- misc/optionmap.hh, 460
- misc/random.hh, 461
- misc/timer.hh, 462
- misc/version.hh, 462
- misc_tools
 - escape_str, 28
 - is_bare_word, 28
 - quote_unless_bare_word, 29
 - version, 29
- Miscellaneous algorithms for LTL formulae, 24
- Miscellaneous algorithms on TGBA, 40
- Miscellaneous helper algorithms, 27
- mrnd
 - random, 31
- multop
 - spot::ltl::multop, 227
- n
 - spot::couvreur99_check_shy::todo_item, 147
- n_
 - spot::barand, 90
 - spot::loopless_modular_mixed_radix_gray_code, 218
- name
 - spot::ltl::atomic_prop, 88
 - spot::ltl::declarative_environment, 151
 - spot::ltl::default_environment, 153
 - spot::ltl::environment, 178
 - spot::ltl::random_ltl::op_proba, 265
 - spot::symbol, 299
- name_
 - spot::ltl::atomic_prop, 89
 - spot::symbol, 299
- name_state_map_
 - spot::evtgba_explicit, 185
 - spot::tgba_explicit, 331
 - spot::tgba_reduc, 368
- nb_node_parity_game
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
- nb_set_acc_cond
 - spot::parity_game_graph_delayed, 251
- neg_acceptance_conditions
 - spot::tgba, 303
 - spot::tgba_bdd_concrete, 310
 - spot::tgba_explicit, 329
 - spot::tgba_product, 340
 - spot::tgba_reduc, 364
 - spot::tgba_sba_proxy, 378
 - spot::tgba_tba_proxy, 396
- neg_acceptance_conditions_
 - spot::tgba_explicit, 331
 - spot::tgba_product, 342
 - spot::tgba_reduc, 368
- neg_all_acc
 - spot::weight, 421
- negacc_set
 - spot::tgba_bdd_core_data, 319
- negative_normal_form
 - ltl_rewriting, 24
- never_claim_reachable
 - tgba_io, 34
- next
 - spot::evtgba_iterator, 187
 - spot::loopless_modular_mixed_radix_gray_code, 218
 - spot::minato_isop, 220
 - spot::numbered_state_heap_const_iterator, 233
 - spot::tgba_explicit_succ_iterator, 335
 - spot::tgba_succ_iterator, 383
 - spot::tgba_succ_iterator_concrete, 386
 - spot::tgba_succ_iterator_product, 391
- next_non_false_
 - spot::tgba_succ_iterator_product, 391
- next_set
 - spot::tgba_bdd_core_data, 319
- next_state
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 196
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
 - spot::tgba_reachable_iterator, 344

- spot::tgba_reachable_iterator_breadth_first, 348
- spot::tgba_reachable_iterator_depth_first, 352
- spot::tgba_reduc, 364
- next_to_now
 - spot::bdd_dict, 103
- non_one_radixes_
 - spot::loopless_modular_mixed_radix_gray_code, 219
- Not
 - spot::ltl::unop, 412
- not_win
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 283
- notacc_set
 - spot::tgba_bdd_core_data, 320
- notnext_set
 - spot::tgba_bdd_core_data, 320
- notnow_set
 - spot::tgba_bdd_core_data, 320
- notvar_set
 - spot::tgba_bdd_core_data, 320
- now_formula_map
 - spot::bdd_dict, 103
- now_map
 - spot::bdd_dict, 103
- now_set
 - spot::tgba_bdd_core_data, 320
- now_to_next
 - spot::bdd_dict, 103
- nownext_set
 - spot::tgba_bdd_core_data, 320
- nrand
 - random, 31
- ns_map
 - spot::evtgba_explicit, 183
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 360
- nth
 - spot::ltl::multop, 228
- num
 - spot::couvreur99_check_shy, 145
- num_
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 283
- num_acc_
 - spot::tgba, 305
- number_of_acceptance_conditions
 - spot::tgba, 304
 - spot::tgba_bdd_concrete, 310
- spot::tgba_explicit, 329
- spot::tgba_product, 340
- spot::tgba_reduc, 364
- spot::tgba_sba_proxy, 378
- spot::tgba_tba_proxy, 396
- numbered_state_heap_hash_map_factory
 - spot::numbered_state_heap_hash_map_factory, 239
- o_
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 146
 - spot::emptiness_check, 172
 - spot::emptiness_check_instantiator, 174
 - spot::emptiness_check_result, 177
- op
 - spot::ltl::binop, 115
 - spot::ltl::multop, 228
 - spot::ltl::unop, 413
- op_
 - spot::evtgba_product, 190
 - spot::ltl::binop, 117
 - spot::ltl::multop, 229
 - spot::ltl::unop, 414
- op_name
 - spot::ltl::binop, 115
 - spot::ltl::multop, 228
 - spot::ltl::unop, 413
- operator const symbol *
 - spot::rsymbol, 271
- operator!=
 - spot::rsymbol, 271
 - spot::unsigned_statistics_copy, 417
- operator()
 - spot::bdd_less_than, 108
 - spot::char_ptr_less_than, 117
 - spot::ltl::formula_ptr_hash, 208
 - spot::ltl::formula_ptr_less_than, 208
 - spot::ltl::multop::paircmp, 230
 - spot::ptr_hash, 262
 - spot::state_ptr_equal, 295
 - spot::state_ptr_hash, 296
 - spot::state_ptr_less_than, 297
 - spot::string_hash, 297
- operator+=
 - spot::weight, 420
- operator-
 - spot::weight, 420
- operator-=
 - spot::weight, 420
- operator<
 - spot::rsymbol, 271
- operator<<

- common.hh, 422
- spot, 74
- spot::option_map, 241
- spot::weight, 421
- operator=
 - spot::bdd_dict, 101
 - spot::rsymbol, 271
 - spot::tgba_bdd_core_data, 319
 - spot::tgba_run, 370
- operator==
 - spot::rsymbol, 271
 - spot::unsigned_statistics_copy, 417
- operator[]
 - spot::option_map, 240, 241
- options
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 144
 - spot::emptiness_check, 171
 - spot::emptiness_check_instantiator, 174
 - spot::emptiness_check_result, 177
- options_
 - spot::option_map, 241
- options_updated
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 144
 - spot::emptiness_check, 172
 - spot::emptiness_check_result, 177
- Or
 - spot::ltl::multop, 227
- out
 - spot::evtgba_explicit::state, 186
 - spot::evtgba_explicit::transition, 187
- pair
 - spot::ltl::atomic_prop, 87
 - spot::ltl::binop, 114
 - spot::ltl::multop, 226
 - spot::ltl::unop, 412
- pairf
 - spot::ltl::binop, 114
- parity_game_graph
 - spot::parity_game_graph, 244
- parity_game_graph_delayed
 - spot::parity_game_graph_delayed, 250
- parity_game_graph_direct
 - spot::parity_game_graph_direct, 257
- parse
 - ltl_io, 21
- parse_error
 - ltl_io, 20
- parse_error_list
 - ltl_io, 20
- parse_options
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 144
 - spot::emptiness_check, 172
 - spot::emptiness_check_result, 177
 - spot::ltl::random_ltl, 264
 - spot::option_map, 241
- pm
 - spot::weight, 421
- pop
 - spot::scc_stack, 272
- poprem_
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_shy, 146
- pos_lenght_pair
 - spot::bdd_allocator, 93
 - spot::bdd_dict::annon_free_list, 106
 - spot::free_list, 210
- postfix_visitor
 - spot::ltl::postfix_visitor, 260
- prand
 - random, 31
- pred_iter
 - spot::evtgba, 181
 - spot::evtgba_explicit, 184, 185
 - spot::evtgba_product, 190
- prefix
 - spot::tgba_run, 370
- prefix_states_
 - spot::ars_statistics, 83
- print
 - spot::parity_game_graph, 245
 - spot::parity_game_graph_delayed, 251
 - spot::parity_game_graph_direct, 257
 - spot::timer_map, 401
- print_stats
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 144
 - spot::couvreur99_check_status, 149
 - spot::emptiness_check, 172
- print_tgba_run
 - tgba_run, 52
- proba
 - spot::ltl::random_ltl::op_proba, 265
- proba_
 - spot::ltl::random_ltl, 264
- proba_2_
 - spot::ltl::random_ltl, 264
- process_link
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 196

- spot::evtgba_reachable_iterator_depth_first, 200
- spot::parity_game_graph, 245
- spot::parity_game_graph_delayed, 251, 252
- spot::parity_game_graph_direct, 257, 258
- spot::tgba_reachable_iterator, 345
- spot::tgba_reachable_iterator_breadth_first, 349
- spot::tgba_reachable_iterator_depth_first, 353
- spot::tgba_reduc, 364
- process_state
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
 - spot::tgba_reachable_iterator, 345
 - spot::tgba_reachable_iterator_breadth_first, 349
 - spot::tgba_reachable_iterator_depth_first, 353
 - spot::tgba_reduc, 365
- product
 - tgba_algorithms, 33
- progress_measure_
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node_delayed, 283
- project_state
 - spot::tgba, 304
 - spot::tgba_bdd_concrete, 311
 - spot::tgba_explicit, 329
 - spot::tgba_product, 340
 - spot::tgba_reduc, 365
 - spot::tgba_sba_proxy, 378
 - spot::tgba_tba_proxy, 396
- project_tgba_run
 - tgba_run, 52
- prop_map
 - spot::ltl::declarative_environment, 151
- props_
 - spot::ltl::declarative_environment, 152
- prune
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 282
- prune_acc
 - spot::tgba_reduc, 365
- prune_scc
 - spot::tgba_reduc, 365
- push
 - spot::scc_stack, 272
- q
 - spot::couvereur99_check_shy::todo_item, 147
- quote_unless_bare_word
 - misc_tools, 29
- quotient_state
 - spot::tgba_reduc, 365
- R
 - spot::ltl::binop, 115
- rand
 - spot::barand, 90
- random
 - bmrand, 30
 - drand, 30
 - mrnd, 31
 - nrnd, 31
 - prand, 31
 - rrand, 31
 - srnd, 31
- Random functions, 30
- random_graph
 - tgba_misc, 41
- random_ltl
 - spot::ltl::random_ltl, 263
- recurse
 - spot::ltl::clone_visitor, 120
 - spot::ltl::simplify_f_g_visitor, 276
 - spot::ltl::unabbreviate_logic_visitor, 405
 - spot::ltl::unabbreviate_ltl_visitor, 408
- redirect_transition
 - spot::tgba_reduc, 365
- reduc_tgba_sim
 - tgba_reduction, 40
- reduce
 - ltl_rewriting, 24
- Reduce_All
 - ltl_rewriting, 23
 - tgba_reduction, 39
- Reduce_Basics
 - ltl_rewriting, 23
- Reduce_Eventuality_And_Universality
 - ltl_rewriting, 23
- Reduce_None
 - ltl_rewriting, 23
 - tgba_reduction, 39
- reduce_options
 - ltl_rewriting, 23
- Reduce_quotient_Del_Sim
 - tgba_reduction, 39
- Reduce_quotient_Dir_Sim
 - tgba_reduction, 39
- reduce_run
 - tgba_run, 52
- Reduce_Scc

- tgba_reduction, 39
- Reduce_Syntactic_Implications
 - ltl_rewriting, 23
- reduce_tgba_options
 - tgba_reduction, 39
- Reduce_transition_Del_Sim
 - tgba_reduction, 39
- Reduce_transition_Dir_Sim
 - tgba_reduction, 39
- ref
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::constant, 129
 - spot::ltl::formula, 206
 - spot::ltl::multop, 228
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 413
 - spot::symbol, 299
- ref_
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::constant, 129
 - spot::ltl::formula, 206
 - spot::ltl::multop, 228
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 413
- ref_count_
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::multop, 228
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 414
 - spot::symbol, 299
- ref_counter_
 - spot::ltl::ref_formula, 269
- ref_formula
 - spot::ltl::ref_formula, 268
- ref_set
 - spot::bdd_dict, 100
- refs_
 - spot::symbol, 299
- register_acceptance_variable
 - spot::bdd_dict, 101
- register_acceptance_variables
 - spot::bdd_dict, 101
- register_all_variables_of
 - spot::bdd_dict, 102
- register_anonymous_variables
 - spot::bdd_dict, 102
- register_n
 - spot::bdd_allocator, 94
 - spot::bdd_dict::annon_free_list, 106
 - spot::free_list, 211
- register_proposition
 - spot::bdd_dict, 102
- register_propositions
 - spot::bdd_dict, 102
- register_state
 - spot::bdd_dict, 102
- relation
 - spot::tgba_bdd_core_data, 320
- release_n
 - spot::bdd_allocator, 94
 - spot::bdd_dict::annon_free_list, 107
 - spot::free_list, 211
- release_variables
 - spot::bdd_allocator, 95
 - spot::bdd_dict, 102
- rem
 - spot::connected_component_hash_set, 123
 - spot::explicit_connected_component, 203
 - spot::scc_stack, 272
 - spot::scc_stack::connected_component, 274
- remove
 - spot::bdd_allocator, 95
 - spot::bdd_dict::annon_free_list, 107
 - spot::free_list, 211
- remove_acc
 - spot::tgba_reduc, 366
- remove_component
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_shy, 145
 - spot::tgba_reduc, 366
- remove_predecessor_state
 - spot::tgba_reduc, 366
- remove_scc
 - spot::tgba_reduc, 366
- remove_state
 - spot::tgba_reduc, 366
- replay_tgba_run
 - tgba_run, 53
- require
 - spot::ltl::declarative_environment, 151
 - spot::ltl::default_environment, 153
 - spot::ltl::environment, 178
- result
 - spot::couvreur99_check, 134
 - spot::couvreur99_check_shy, 145
 - spot::ltl::clone_visitor, 120
 - spot::ltl::simplify_f_g_visitor, 276
 - spot::ltl::unabbreviate_logic_visitor, 405
 - spot::ltl::unabbreviate_ltl_visitor, 408
- result_
 - spot::ltl::clone_visitor, 120
 - spot::ltl::simplify_f_g_visitor, 276
 - spot::ltl::unabbreviate_logic_visitor, 406
 - spot::ltl::unabbreviate_ltl_visitor, 409
- ret_

- spot::minato_isop, 220
- Rewriting LTL formulae, 23
- right
 - spot::state_product, 295
- right_
 - spot::state_product, 295
 - spot::tgba_product, 342
 - spot::tgba_succ_iterator_product, 391
- right_acc_complement_
 - spot::tgba_product, 342
- right_neg_
 - spot::tgba_succ_iterator_product, 391
- root
 - spot::couvereur99_check_status, 149
- root_
 - spot::tgba_reduc, 368
- rrand
 - random, 31
- rsymbol
 - spot::rsymbol, 270
- rsymbol_set
 - spot, 70
 - symbol.hh, 427
- run
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
 - spot::tgba_reachable_iterator, 345
 - spot::tgba_reachable_iterator_breadth_first, 349
 - spot::tgba_reachable_iterator_depth_first, 353
 - spot::tgba_reduc, 366
- run_
 - spot::couvereur99_check_result, 139
 - spot::tgba_run_dotty_decorator, 373
- s
 - spot::couvereur99_check_shy::successor, 147
 - spot::couvereur99_check_shy::todo_item, 147
 - spot::scc_stack, 273
 - spot::tgba_run::step, 370
- s_
 - spot::barand, 90
 - spot::loopless_modular_mixed_radix_gray_code, 219
 - spot::rsymbol, 271
 - spot::tgba_explicit_succ_iterator, 335
- s_v
 - tgba_reduction, 39
- safe
 - spot::couvereur99_check, 135
 - spot::couvereur99_check_shy, 145
 - spot::emptiness_check, 172
- sc_
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 283
- scc_computed_
 - spot::tgba_reduc, 368
- se05
 - emptiness_check_algorithms, 51
- search
 - spot::bfs_steps, 110
- second
 - spot::ltl::binop, 116
- second_
 - spot::ltl::binop, 117
- SecondStep
 - spot::minato_isop::local_vars, 221
- seen
 - spot::evtgba_reachable_iterator, 194
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 201
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 253
 - spot::parity_game_graph_direct, 258
 - spot::tgba_reachable_iterator, 345
 - spot::tgba_reachable_iterator_breadth_first, 349
 - spot::tgba_reachable_iterator_depth_first, 353
 - spot::tgba_reduc, 368
- seen_
 - spot::duplicator_node_delayed, 165
 - spot::spoiler_node_delayed, 283
 - spot::tgba_reduc, 368
- seen_map
 - spot::evtgba_reachable_iterator, 192
 - spot::evtgba_reachable_iterator_breadth_first, 196
 - spot::evtgba_reachable_iterator_depth_first, 199
 - spot::parity_game_graph, 244
 - spot::parity_game_graph_delayed, 250
 - spot::parity_game_graph_direct, 256
 - spot::tgba_reachable_iterator, 344
 - spot::tgba_reachable_iterator_breadth_first, 348
 - spot::tgba_reachable_iterator_depth_first, 352
 - spot::tgba_reduc, 360
- set

- spot::option_map, 241
- spot::unsigned_statistics_copy, 417
- set_init_state
 - spot::evtgba_explicit, 185
 - spot::tgba_bdd_concrete, 311
 - spot::tgba_explicit, 329
 - spot::tgba_reduc, 366
- set_key_
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::constant, 129
 - spot::ltl::formula, 206
 - spot::ltl::multop, 228
 - spot::ltl::ref_formula, 268
 - spot::ltl::unop, 414
- set_lead_2_acc_all
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node_delayed, 282
- set_states
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_shy, 145
 - spot::ec_statistics, 168
- set_type
 - spot::connected_component_hash_set, 122
- set_win
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 282
- seteq
 - spot::unsigned_statistics_copy, 417
- setup
 - spot::ltl::random_ltl::op_proba, 265
- si_
 - spot::tgba_reduc, 368
- simplify_f_g
 - ltl_rewriting, 24
- simplify_f_g_visitor
 - spot::ltl::simplify_f_g_visitor, 276
- simulation_relation
 - spot, 71
 - tgbaeduc.hh, 479
- size
 - spot::ltl::multop, 228
 - spot::numbered_state_heap, 232
 - spot::numbered_state_heap_hash_map, 238
 - spot::scc_stack, 272
- sn_map
 - spot::evtgba_explicit, 183
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 360
- sn_v
 - tgba_reduction, 39
- sp_map
 - spot::tgba_reduc, 360
- spoiler_node
 - spot::spoiler_node, 278
- spoiler_node_delayed
 - spot::spoiler_node_delayed, 281
- spoiler_vertice_
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 253
 - spot::parity_game_graph_direct, 259
- spot, 61
 - bdd_format_accset, 71
 - bdd_format_formula, 71
 - bdd_format_sat, 71
 - bdd_format_set, 71
 - bdd_print_acc, 72
 - bdd_print_accset, 72
 - bdd_print_dot, 72
 - bdd_print_formula, 72
 - bdd_print_sat, 73
 - bdd_print_set, 73
 - bdd_print_table, 73
 - bdd_to_formula, 73
 - dotty_reachable, 73
 - evtgba_parse, 73
 - evtgba_parse_error, 70
 - evtgba_parse_error_list, 70
 - evtgba_save_reachable, 74
 - format_evtgba_parse_errors, 74
 - formula_to_bdd, 74
 - operator<<, 74
 - rsymbol_set, 70
 - simulation_relation, 71
 - state_couple, 71
 - symbol_set, 71
 - tgba_to_evtgba, 74
- spot::acss_statistics, 78
 - ~acss_statistics, 80
 - acss_states, 80
 - acss_statistics, 80
 - ars_cycle_states, 80
 - ars_prefix_states, 80
 - get, 80
 - inc_ars_cycle_states, 81
 - inc_ars_prefix_states, 81
 - stats, 81
 - stats_map, 80
 - unsigned_fun, 80
- spot::ars_statistics, 81
 - ars_cycle_states, 83
 - ars_prefix_states, 83
 - ars_statistics, 83
 - cycle_states_, 83
 - get, 83
 - inc_ars_cycle_states, 83

- inc_ars_prefix_states, 83
- prefix_states_, 83
- stats, 83
- stats_map, 82
- unsigned_fun, 82
- spot::barand, 89
 - barand, 90
 - m_, 90
 - n_, 90
 - rand, 90
 - s_, 90
- spot::bdd_allocator, 90
 - allocate_variables, 94
 - bdd_allocator, 94
 - dump_free_list, 94
 - extend, 94
 - extvarnum, 94
 - fl, 95
 - free_list_type, 93
 - initialize, 94
 - initialized, 95
 - insert, 94
 - lvarnum, 95
 - pos_lenght_pair, 93
 - register_n, 94
 - release_n, 94
 - release_variables, 95
 - remove, 95
- spot::bdd_dict, 95
 - ~bdd_dict, 100
 - acc_formula_map, 103
 - acc_map, 103
 - allocate_variables, 101
 - assert_emptiness, 101
 - bdd_dict, 100
 - dump, 101
 - free_anonymous_list_of, 103
 - free_anonymous_list_of_type, 100
 - fv_map, 100
 - initialize, 101
 - initialized, 103
 - is_registered_acceptance_variable, 101
 - is_registered_proposition, 101
 - is_registered_state, 101
 - lvarnum, 103
 - next_to_now, 103
 - now_formula_map, 103
 - now_map, 103
 - now_to_next, 103
 - operator=, 101
 - ref_set, 100
 - register_acceptance_variable, 101
 - register_acceptance_variables, 101
 - register_all_variables_of, 102
 - register_anonymous_variables, 102
 - register_proposition, 102
 - register_propositions, 102
 - register_state, 102
 - release_variables, 102
 - unregister_all_my_variables, 103
 - unregister_variable, 103
 - var_formula_map, 104
 - var_map, 104
 - var_refs, 104
 - vf_map, 100
 - vr_map, 100
- spot::bdd_dict::annon_free_list, 104
 - annon_free_list, 106
 - dict_, 107
 - dump_free_list, 106
 - extend, 106
 - fl, 107
 - free_list_type, 106
 - insert, 106
 - pos_lenght_pair, 106
 - register_n, 106
 - release_n, 107
 - remove, 107
- spot::bdd_less_than, 107
 - operator(), 108
- spot::bfs_steps, 108
 - ~bfs_steps, 109
 - a_, 110
 - bfs_steps, 109
 - filter, 109
 - finalize, 109
 - match, 110
 - search, 110
- spot::char_ptr_less_than, 117
 - operator(), 117
- spot::connected_component_hash_set, 120
 - ~connected_component_hash_set, 122
 - condition, 122
 - has_state, 122
 - index, 122
 - insert, 122
 - rem, 123
 - set_type, 122
 - states, 123
- spot::connected_component_hash_set_factory, 123
 - ~connected_component_hash_set_factory, 124
 - build, 124
 - connected_component_hash_set_factory, 124
 - instance, 124
- spot::couvreur99_check, 130
 - ~couvreur99_check, 133
 - a_, 135

- automaton, [133](#)
 - check, [133](#)
 - couvreur99_check, [133](#)
 - dec_depth, [134](#)
 - depth, [134](#)
 - ecs_, [135](#)
 - get, [134](#)
 - inc_depth, [134](#)
 - inc_states, [134](#)
 - inc_transitions, [134](#)
 - max_depth, [134](#)
 - o_, [135](#)
 - options, [134](#)
 - options_updated, [134](#)
 - parse_options, [134](#)
 - poprem_, [135](#)
 - print_stats, [134](#)
 - remove_component, [134](#)
 - result, [134](#)
 - safe, [135](#)
 - set_states, [135](#)
 - states, [135](#)
 - statistics, [135](#)
 - stats, [135](#)
 - stats_map, [133](#)
 - transitions, [135](#)
 - unsigned_fun, [133](#)
- spot::couvreur99_check_result, [135](#)
 - a_, [139](#)
 - accepting_cycle, [138](#)
 - accepting_run, [138](#)
 - acss_states, [138](#)
 - ars_cycle_states, [138](#)
 - ars_prefix_states, [138](#)
 - automaton, [138](#)
 - couvreur99_check_result, [138](#)
 - ecs_, [139](#)
 - get, [139](#)
 - inc_ars_cycle_states, [139](#)
 - inc_ars_prefix_states, [139](#)
 - o_, [139](#)
 - options, [139](#)
 - options_updated, [139](#)
 - parse_options, [139](#)
 - print_stats, [139](#)
 - run_, [139](#)
 - statistics, [139](#)
 - stats, [139](#)
 - stats_map, [138](#)
 - unsigned_fun, [138](#)
- spot::couvreur99_check_shy, [140](#)
 - ~couvreur99_check_shy, [143](#)
 - a_, [145](#)
 - arc, [145](#)
 - automaton, [143](#)
 - check, [143](#)
 - clear_todo, [144](#)
 - couvreur99_check_shy, [143](#)
 - dec_depth, [144](#)
 - depth, [144](#)
 - ecs_, [145](#)
 - find_state, [144](#)
 - get, [144](#)
 - group_, [145](#)
 - inc_depth, [144](#)
 - inc_states, [144](#)
 - inc_transitions, [144](#)
 - max_depth, [144](#)
 - num, [145](#)
 - o_, [146](#)
 - options, [144](#)
 - options_updated, [144](#)
 - parse_options, [144](#)
 - poprem_, [146](#)
 - print_stats, [144](#)
 - remove_component, [145](#)
 - result, [145](#)
 - safe, [145](#)
 - set_states, [145](#)
 - states, [145](#)
 - statistics, [145](#)
 - stats, [146](#)
 - stats_map, [143](#)
 - succ_queue, [143](#)
 - todo, [146](#)
 - todo_list, [143](#)
 - transitions, [145](#)
 - unsigned_fun, [143](#)
- spot::couvreur99_check_shy::successor, [146](#)
 - acc, [147](#)
 - s, [147](#)
 - successor, [146](#)
- spot::couvreur99_check_shy::todo_item, [147](#)
 - n, [147](#)
 - q, [147](#)
 - s, [147](#)
 - todo_item, [147](#)
- spot::couvreur99_check_status, [148](#)
 - ~couvreur99_check_status, [149](#)
 - aut, [149](#)
 - couvreur99_check_status, [149](#)
 - cycle_seed, [149](#)
 - h, [149](#)
 - print_stats, [149](#)
 - root, [149](#)
 - states, [149](#)
- spot::delayed_simulation_relation, [154](#)
- spot::direct_simulation_relation, [154](#)

- spot::dotty_decorator, 154
 - ~dotty_decorator, 155
 - dotty_decorator, 155
 - instance, 156
 - link_decl, 156
 - state_decl, 156
- spot::duplicator_node, 157
 - ~duplicator_node, 159
 - acc_, 160
 - add_pred, 159
 - add_succ, 159
 - compare, 159
 - del_pred, 159
 - del_succ, 159
 - duplicator_node, 159
 - get_acc, 159
 - get_duplicator_node, 159
 - get_label, 159
 - get_nb_succ, 159
 - get_pair, 160
 - get_spoiler_node, 160
 - implies, 160
 - label_, 160
 - lnode_pred, 160
 - lnode_succ, 160
 - match, 160
 - not_win, 160
 - num_, 160
 - prune, 160
 - sc_, 160
 - set_win, 160
 - succ_to_string, 160
 - to_string, 160
- spot::duplicator_node_delayed, 161
 - ~duplicator_node_delayed, 163
 - acc_, 165
 - add_pred, 163
 - add_succ, 163
 - compare, 163
 - del_pred, 164
 - del_succ, 164
 - duplicator_node_delayed, 163
 - get_acc, 164
 - get_duplicator_node, 164
 - get_label, 164
 - get_lead_2_acc_all, 164
 - get_nb_succ, 164
 - get_pair, 164
 - get_progress_measure, 164
 - get_spoiler_node, 164
 - implies, 164
 - implies_acc, 164
 - implies_label, 164
 - label_, 165
 - lead_2_acc_all_, 165
 - lnode_pred, 165
 - lnode_succ, 165
 - match, 164
 - not_win, 165
 - num_, 165
 - progress_measure_, 165
 - prune, 164
 - sc_, 165
 - seen_, 165
 - set_lead_2_acc_all, 164
 - set_win, 164
 - succ_to_string, 164
 - to_string, 165
- spot::ec_statistics, 165
 - dec_depth, 168
 - depth, 168
 - depth_, 168
 - ec_statistics, 168
 - get, 168
 - inc_depth, 168
 - inc_states, 168
 - inc_transitions, 168
 - max_depth, 168
 - max_depth_, 168
 - set_states, 168
 - states, 168
 - states_, 168
 - stats, 168
 - stats_map, 167
 - transitions, 168
 - transitions_, 168
 - unsigned_fun, 167
- spot::emptiness_check, 169
 - ~emptiness_check, 171
 - a_, 172
 - automaton, 171
 - check, 171
 - emptiness_check, 171
 - o_, 172
 - options, 171
 - options_updated, 172
 - parse_options, 172
 - print_stats, 172
 - safe, 172
 - statistics, 172
- spot::emptiness_check_instantiator, 172
 - construct, 174
 - emptiness_check_instantiator, 173
 - info_, 174
 - instantiate, 174
 - max_acceptance_conditions, 174
 - min_acceptance_conditions, 174
 - o_, 174

- options, 174
- spot::emptiness_check_result, 174
 - ~emptiness_check_result, 176
 - a_, 177
 - accepting_run, 177
 - automaton, 177
 - emptiness_check_result, 176
 - o_, 177
 - options, 177
 - options_updated, 177
 - parse_options, 177
 - statistics, 177
- spot::evtgba, 179
 - ~evtgba, 180
 - all_acceptance_conditions, 180
 - alphabet, 180
 - evtgba, 180
 - format_acceptance_condition, 180
 - format_acceptance_conditions, 180
 - format_label, 180
 - format_state, 180
 - init_iter, 180
 - pred_iter, 181
 - succ_iter, 181
- spot::evtgba_explicit, 181
 - ~evtgba_explicit, 183
 - acc_set_, 185
 - add_transition, 184
 - all_acceptance_conditions, 184
 - alphabet, 184
 - alphabet_, 185
 - declare_acceptance_condition, 184
 - declare_state, 184
 - evtgba_explicit, 183
 - format_acceptance_condition, 184
 - format_acceptance_conditions, 184
 - format_label, 184
 - format_state, 184
 - init_iter, 184
 - init_states_, 185
 - name_state_map_, 185
 - ns_map, 183
 - pred_iter, 184, 185
 - set_init_state, 185
 - sn_map, 183
 - state_name_map_, 185
 - succ_iter, 185
 - transition_list, 183
- spot::evtgba_explicit::state, 185
 - in, 186
 - out, 186
- spot::evtgba_explicit::transition, 186
 - acceptance_conditions, 186
 - in, 186
 - label, 186
 - out, 187
- spot::evtgba_iterator, 187
 - ~evtgba_iterator, 187
 - current_acceptance_conditions, 187
 - current_label, 187
 - current_state, 187
 - done, 187
 - first, 187
 - next, 187
- spot::evtgba_product, 188
 - ~evtgba_product, 189
 - all_acc_, 190
 - all_acceptance_conditions, 189
 - alphabet, 189
 - alphabet_, 190
 - common_symbol_table, 189
 - common_symbols_, 190
 - evtgba_product, 189
 - evtgba_product_operands, 189
 - format_acceptance_condition, 190
 - format_acceptance_conditions, 190
 - format_label, 190
 - format_state, 190
 - init_iter, 190
 - op_, 190
 - pred_iter, 190
 - succ_iter, 190
- spot::evtgba_reachable_iterator, 191
 - ~evtgba_reachable_iterator, 192
 - add_state, 193
 - automata_, 194
 - end, 193
 - evtgba_reachable_iterator, 192
 - next_state, 193
 - process_link, 193
 - process_state, 193
 - run, 193
 - seen, 194
 - seen_map, 192
 - start, 193
- spot::evtgba_reachable_iterator_breadth_first, 194
 - add_state, 196
 - automata_, 197
 - end, 196
 - evtgba_reachable_iterator_breadth_first, 196
 - next_state, 196
 - process_link, 196
 - process_state, 197
 - run, 197
 - seen, 197
 - seen_map, 196
 - start, 197
 - todo, 197

- spot::evtgba_reachable_iterator_depth_first, 198
 - add_state, 200
 - automata_, 201
 - end, 200
 - evtgba_reachable_iterator_depth_first, 199
 - next_state, 200
 - process_link, 200
 - process_state, 200
 - run, 200
 - seen, 201
 - seen_map, 199
 - start, 200
 - todo, 201
- spot::explicit_connected_component, 201
 - ~explicit_connected_component, 202
 - condition, 203
 - has_state, 202
 - index, 203
 - insert, 202
 - rem, 203
- spot::explicit_connected_component_factory, 203
 - ~explicit_connected_component_factory, 204
 - build, 204
- spot::free_list, 208
 - ~free_list, 210
 - dump_free_list, 210
 - extend, 210
 - fl, 211
 - free_list_type, 210
 - insert, 211
 - pos_lenght_pair, 210
 - register_n, 211
 - release_n, 211
 - remove, 211
- spot::gspn_exeption, 211
 - err_, 212
 - get_err, 212
 - get_where, 212
 - gspn_exeption, 212
 - where_, 212
- spot::gspn_interface, 212
 - ~gspn_interface, 213
 - automaton, 214
 - dead_, 214
 - dict_, 214
 - env_, 214
 - gspn_interface, 213
- spot::gspn_ssp_interface, 214
 - ~gspn_ssp_interface, 215
 - automaton, 216
 - dict_, 216
 - env_, 216
 - gspn_ssp_interface, 215
- spot::loopless_modular_mixed_radix_gray_code, 216
 - ~loopless_modular_mixed_radix_gray_code, 217
 - a_, 218
 - a_first, 218
 - a_last, 218
 - a_next, 218
 - done, 218
 - done_, 218
 - f_, 218
 - first, 218
 - last, 218
 - loopless_modular_mixed_radix_gray_code, 217
 - m_, 218
 - n_, 218
 - next, 218
 - non_one_radixes_, 219
 - s_, 219
- spot::ltl, 75
 - unabbreviate_ltl, 78
- spot::ltl::atomic_prop, 83
 - ~atomic_prop, 87
 - accept, 87
 - atomic_prop, 87
 - dump, 87
 - dump_, 89
 - dump_instances, 87
 - env, 87
 - env_, 89
 - hash, 87
 - hash_key_, 89
 - instance, 88
 - instance_count, 88
 - instances, 89
 - map, 87
 - name, 88
 - name_, 89
 - pair, 87
 - ref, 88
 - ref_, 88
 - ref_count_, 88
 - set_key_, 88
 - unref, 88
 - unref_, 88
- spot::ltl::binop, 110
 - ~binop, 115
 - accept, 115
 - binop, 115
 - dump, 115
 - dump_, 116
 - Equiv, 114
 - first, 115

- first_, 116
- hash, 115
- hash_key_, 117
- Implies, 114
- instance, 115
- instance_count, 115
- instances, 117
- map, 114
- op, 115
- op_, 117
- op_name, 115
- pair, 114
- pairf, 114
- R, 115
- ref, 116
- ref_, 116
- ref_count_, 116
- second, 116
- second_, 117
- set_key_, 116
- type, 114
- U, 115
- unref, 116
- unref_, 116
- Xor, 114
- spot::ltl::clone_visitor, 118
 - ~clone_visitor, 119
 - clone_visitor, 119
 - recurse, 120
 - result, 120
 - result_, 120
 - visit, 120
- spot::ltl::const_visitor, 124
 - ~const_visitor, 125
 - visit, 125
- spot::ltl::constant, 125
 - ~constant, 128
 - accept, 128
 - constant, 128
 - dump, 128
 - dump_, 130
 - False, 128
 - false_instance, 128
 - hash, 129
 - hash_key_, 130
 - ref, 129
 - ref_, 129
 - set_key_, 129
 - True, 128
 - true_instance, 129
 - type, 128
 - unref, 129
 - unref_, 129
 - val, 129
 - val_, 130
 - val_name, 129
- spot::ltl::declarative_environment, 150
 - ~declarative_environment, 151
 - declarative_environment, 151
 - declare, 151
 - get_prop_map, 151
 - name, 151
 - prop_map, 151
 - props_, 152
 - require, 151
- spot::ltl::default_environment, 152
 - ~default_environment, 153
 - default_environment, 153
 - instance, 153
 - name, 153
 - require, 153
- spot::ltl::environment, 177
 - ~environment, 178
 - name, 178
 - require, 178
- spot::ltl::formula, 204
 - ~formula, 206
 - accept, 206
 - dump, 206
 - dump_, 207
 - hash, 206
 - hash_key_, 207
 - ref, 206
 - ref_, 206
 - set_key_, 206
 - unref, 206
 - unref_, 207
- spot::ltl::formula_ptr_hash, 207
 - operator(), 208
- spot::ltl::formula_ptr_less_than, 208
 - operator(), 208
- spot::ltl::multop, 222
 - ~multop, 227
 - accept, 227
 - And, 227
 - children_, 229
 - dump, 227
 - dump_, 229
 - hash, 227
 - hash_key_, 229
 - instance, 227
 - instance_count, 228
 - instances, 229
 - map, 226
 - multop, 227
 - nth, 228
 - op, 228
 - op_, 229

- op_name, 228
- Or, 227
- pair, 226
- ref, 228
- ref_, 228
- ref_count_, 228
- set_key_, 228
- size, 228
- type, 227
- unref, 229
- unref_, 229
- vec, 226
- spot::ltl::multop::pairemp, 229
 - operator(), 230
- spot::ltl::postfix_visitor, 259
 - ~postfix_visitor, 260
 - doit, 260, 261
 - doit_default, 261
 - postfix_visitor, 260
 - visit, 261
- spot::ltl::random_ltl, 262
 - ~random_ltl, 263
 - ap, 263
 - ap_, 264
 - dump_priorities, 263
 - generate, 263
 - parse_options, 264
 - proba_, 264
 - proba_2_, 264
 - random_ltl, 263
 - total_1_, 264
 - total_2_, 264
 - total_2_and_more_, 264
 - update_sums, 264
- spot::ltl::random_ltl::op_proba, 265
 - build, 265
 - builder, 265
 - min_n, 265
 - name, 265
 - proba, 265
 - setup, 265
- spot::ltl::ref_formula, 265
 - ~ref_formula, 268
 - accept, 268
 - dump, 268
 - dump_, 269
 - hash, 268
 - hash_key_, 269
 - ref, 268
 - ref_, 268
 - ref_count_, 268
 - ref_counter_, 269
 - ref_formula, 268
 - set_key_, 268
 - unref, 269
 - unref_, 269
- spot::ltl::simplify_f_g_visitor, 274
 - ~simplify_f_g_visitor, 276
 - recurse, 276
 - result, 276
 - result_, 276
 - simplify_f_g_visitor, 276
 - super, 276
 - visit, 276
- spot::ltl::unabbreviate_logic_visitor, 402
 - ~unabbreviate_logic_visitor, 405
 - recurse, 405
 - result, 405
 - result_, 406
 - super, 405
 - unabbreviate_logic_visitor, 405
 - visit, 405
- spot::ltl::unabbreviate_ltl_visitor, 406
 - ~unabbreviate_ltl_visitor, 408
 - recurse, 408
 - result, 408
 - result_, 409
 - super, 408
 - unabbreviate_ltl_visitor, 408
 - visit, 408
- spot::ltl::unop, 409
 - ~unop, 412
 - accept, 413
 - child, 413
 - child_, 414
 - dump, 413
 - dump_, 414
 - F, 412
 - G, 412
 - hash, 413
 - hash_key_, 414
 - instance, 413
 - instance_count, 413
 - instances, 414
 - map, 412
 - Not, 412
 - op, 413
 - op_, 414
 - op_name, 413
 - pair, 412
 - ref, 413
 - ref_, 413
 - ref_count_, 414
 - set_key_, 414
 - type, 412
 - unop, 412
 - unref, 414
 - unref_, 414

- X, 412
- spot::ltl::visitor, 417
 - ~visitor, 418
 - visit, 419
- spot::minato_isop, 219
 - cube_, 220
 - minato_isop, 220
 - next, 220
 - ret_, 220
 - todo_, 220
- spot::minato_isop::local_vars
 - FirstStep, 221
 - FourthStep, 221
 - SecondStep, 221
 - ThirdStep, 221
- spot::minato_isop::local_vars, 221
 - f0_max, 221
 - f0_min, 221
 - f1_max, 222
 - f1_min, 222
 - f_max, 222
 - f_min, 222
 - g0, 222
 - g1, 222
 - local_vars, 221
 - step, 222
 - v1, 222
 - vars, 222
- spot::numbered_state_heap, 230
 - ~numbered_state_heap, 231
 - find, 231
 - index, 232
 - index_and_insert, 232
 - insert, 232
 - iterator, 232
 - size, 232
 - state_index, 231
 - state_index_p, 231
- spot::numbered_state_heap_const_iterator, 233
 - ~numbered_state_heap_const_iterator, 233
 - done, 233
 - first, 233
 - get_index, 233
 - get_state, 233
 - next, 233
- spot::numbered_state_heap_factory, 234
 - ~numbered_state_heap_factory, 234
 - build, 234
- spot::numbered_state_heap_hash_map, 235
 - ~numbered_state_heap_hash_map, 237
 - find, 237
 - h, 238
 - hash_type, 236
 - index, 237
 - index_and_insert, 237
 - insert, 237
 - iterator, 237
 - size, 238
 - state_index, 236
 - state_index_p, 236
- spot::numbered_state_heap_hash_map_factory, 238
 - ~numbered_state_heap_hash_map_factory, 239
 - build, 239
 - instance, 239
 - numbered_state_heap_hash_map_factory, 239
- spot::option_map, 239
 - get, 240
 - operator<<, 241
 - operator[], 240, 241
 - options_, 241
 - parse_options, 241
 - set, 241
- spot::parity_game_graph, 242
 - ~parity_game_graph, 244
 - add_state, 245
 - automata_, 246
 - build_graph, 245
 - duplicator_vertice_, 246
 - end, 245
 - get_relation, 245
 - lift, 245
 - nb_node_parity_game, 246
 - next_state, 245
 - parity_game_graph, 244
 - print, 245
 - process_link, 245
 - process_state, 246
 - run, 246
 - seen, 246
 - seen_map, 244
 - spoiler_vertice_, 246
 - start, 246
 - tgba_state_, 246
 - todo, 246
- spot::parity_game_graph_delayed, 247
 - ~parity_game_graph_delayed, 250
 - add_duplicator_node_delayed, 250
 - add_spoiler_node_delayed, 250
 - add_state, 251
 - automata_, 252
 - bdd_v, 250
 - build_graph, 251
 - build_recurse_successor_duplicator, 251
 - build_recurse_successor_spoiler, 251
 - duplicator_vertice_, 252
 - end, 251

- get_relation, 251
- lift, 251
- nb_node_parity_game, 252
- nb_set_acc_cond, 251
- next_state, 251
- parity_game_graph_delayed, 250
- print, 251
- process_link, 251, 252
- process_state, 252
- run, 252
- seen, 253
- seen_map, 250
- spoiler_vertice_, 253
- start, 252
- sub_set_acc_cond_, 253
- tgba_state_, 253
- todo, 253
- spot::parity_game_graph_direct, 253
 - ~parity_game_graph_direct, 257
 - add_state, 257
 - automata_, 258
 - build_graph, 257
 - build_link, 257
 - duplicator_vertice_, 258
 - end, 257
 - get_relation, 257
 - lift, 257
 - nb_node_parity_game, 258
 - next_state, 257
 - parity_game_graph_direct, 257
 - print, 257
 - process_link, 257, 258
 - process_state, 258
 - run, 258
 - seen, 258
 - seen_map, 256
 - spoiler_vertice_, 259
 - start, 258
 - tgba_state_, 259
 - todo, 259
- spot::ptr_hash, 261
 - operator(), 262
- spot::rsymbol, 269
 - ~rsymbol, 270
 - operator const symbol *, 271
 - operator!=, 271
 - operator<, 271
 - operator=, 271
 - operator==, 271
 - rsymbol, 270
 - s_, 271
- spot::scc_stack, 271
 - clear_rem, 272
 - empty, 272
 - pop, 272
 - push, 272
 - rem, 272
 - s, 273
 - size, 272
 - stack_type, 272
 - top, 272, 273
- spot::scc_stack::connected_component, 273
 - condition, 274
 - connected_component, 274
 - index, 274
 - rem, 274
- spot::spoiler_node, 277
 - ~spoiler_node, 278
 - add_pred, 278
 - add_succ, 278
 - compare, 278
 - del_pred, 278
 - del_succ, 278
 - get_duplicator_node, 278
 - get_nb_succ, 278
 - get_pair, 279
 - get_spoiler_node, 279
 - lnode_pred, 279
 - lnode_succ, 279
 - not_win, 279
 - num_, 279
 - prune, 279
 - sc_, 279
 - set_win, 279
 - spoiler_node, 278
 - succ_to_string, 279
 - to_string, 279
- spot::spoiler_node_delayed, 279
 - ~spoiler_node_delayed, 281
 - acceptance_condition_visited_, 283
 - add_pred, 282
 - add_succ, 282
 - compare, 282
 - del_pred, 282
 - del_succ, 282
 - get_acceptance_condition_visited, 282
 - get_duplicator_node, 282
 - get_lead_2_acc_all, 282
 - get_nb_succ, 282
 - get_pair, 282
 - get_progress_measure, 282
 - get_spoiler_node, 282
 - lead_2_acc_all_, 283
 - lnode_pred, 283
 - lnode_succ, 283
 - not_win, 283
 - num_, 283
 - progress_measure_, 283

- prune, 282
- sc_, 283
- seen_, 283
- set_lead_2_acc_all, 282
- set_win, 282
- spoiler_node_delayed, 281
- succ_to_string, 282
- to_string, 282
- spot::state, 283
 - ~state, 284
 - clone, 284
 - compare, 284
 - hash, 285
- spot::state_bdd, 285
 - as_bdd, 286
 - clone, 286
 - compare, 287
 - hash, 287
 - state_, 287
 - state_bdd, 286
- spot::state_evtgba_explicit, 287
 - ~state_evtgba_explicit, 289
 - clone, 289
 - compare, 289
 - get_state, 289
 - hash, 289
 - state_, 290
 - state_evtgba_explicit, 289
- spot::state_explicit, 290
 - ~state_explicit, 291
 - clone, 291
 - compare, 291
 - get_state, 292
 - hash, 292
 - state_, 292
 - state_explicit, 291
- spot::state_product, 292
 - ~state_product, 294
 - clone, 294
 - compare, 294
 - hash, 294
 - left, 295
 - left_, 295
 - right, 295
 - right_, 295
 - state_product, 294
- spot::state_ptr_equal, 295
 - operator(), 295
- spot::state_ptr_hash, 296
 - operator(), 296
- spot::state_ptr_less_than, 296
 - operator(), 297
- spot::string_hash, 297
 - operator(), 297
- spot::symbol, 297
 - ~symbol, 298
 - dump_instances, 298
 - instance, 298
 - instance_count, 298
 - instances_, 299
 - map, 298
 - name, 299
 - name_, 299
 - ref, 299
 - ref_count_, 299
 - refs_, 299
 - symbol, 298
 - unref, 299
- spot::tgba, 299
 - ~tgba, 302
 - all_acceptance_conditions, 303
 - compute_support_conditions, 303
 - compute_support_variables, 303
 - format_state, 303
 - get_dict, 303
 - get_init_state, 303
 - last_support_conditions_input_, 305
 - last_support_conditions_output_, 305
 - last_support_variables_input_, 305
 - last_support_variables_output_, 305
 - neg_acceptance_conditions, 303
 - num_acc_, 305
 - number_of_acceptance_conditions, 304
 - project_state, 304
 - succ_iter, 304
 - support_conditions, 304
 - support_variables, 305
 - tgba, 302
 - transition_annotation, 305
- spot::tgba_bdd_concrete, 306
 - ~tgba_bdd_concrete, 309
 - all_acceptance_conditions, 309
 - compute_support_conditions, 309
 - compute_support_variables, 309
 - data_, 312
 - format_state, 309
 - get_core_data, 310
 - get_dict, 310
 - get_init_bdd, 310
 - get_init_state, 310
 - init_, 312
 - neg_acceptance_conditions, 310
 - number_of_acceptance_conditions, 310
 - project_state, 311
 - set_init_state, 311
 - succ_iter, 311
 - support_conditions, 311
 - support_variables, 312

- tgba_bdd_concrete, 309
- tgba_bdd_concrete::operator=, 312
- transition_annotation, 312
- spot::tgba_bdd_concrete_factory, 312
 - ~tgba_bdd_concrete_factory, 314
 - acc_, 315
 - acc_map_, 314
 - constrain_relation, 314
 - create_atomic_prop, 314
 - create_state, 315
 - data_, 315
 - declare_acceptance_condition, 315
 - finish, 315
 - get_core_data, 315
 - get_dict, 315
 - tgba_bdd_concrete_factory, 314
- spot::tgba_bdd_core_data, 316
 - acc_set, 319
 - acceptance_conditions, 319
 - all_acceptance_conditions, 319
 - declare_acceptance_condition, 318
 - declare_atomic_prop, 318
 - declare_now_next, 318
 - dict, 319
 - negacc_set, 319
 - next_set, 319
 - notacc_set, 320
 - notnext_set, 320
 - notnow_set, 320
 - notvar_set, 320
 - now_set, 320
 - nownext_set, 320
 - operator=, 319
 - relation, 320
 - tgba_bdd_core_data, 318
 - var_set, 320
 - varandnext_set, 320
- spot::tgba_bdd_factory, 321
 - ~tgba_bdd_factory, 321
 - get_core_data, 321
- spot::tgba_explicit, 322
 - ~tgba_explicit, 327
 - add_acceptance_condition, 327
 - add_acceptance_conditions, 327
 - add_condition, 327
 - add_conditions, 327
 - add_state, 327
 - all_acceptance_conditions, 327
 - all_acceptance_conditions_, 331
 - all_acceptance_conditions_computed_, 331
 - complement_all_acceptance_conditions, 327
 - compute_support_conditions, 327, 328
 - compute_support_variables, 328
 - copy_acceptance_conditions_of, 328
 - create_transition, 328
 - declare_acceptance_condition, 328
 - dict_, 331
 - format_state, 328
 - get_acceptance_condition, 328
 - get_dict, 328
 - get_init_state, 329
 - has_acceptance_condition, 329
 - init_, 331
 - merge_transitions, 329
 - name_state_map_, 331
 - neg_acceptance_conditions, 329
 - neg_acceptance_conditions_, 331
 - ns_map, 327
 - number_of_acceptance_conditions, 329
 - project_state, 329
 - set_init_state, 329
 - sn_map, 327
 - state, 327
 - state_name_map_, 331
 - succ_iter, 330
 - support_conditions, 330
 - support_variables, 330
 - tgba_explicit, 327
 - tgba_explicit::operator=, 330
 - transition_annotation, 330
- spot::tgba_explicit::transition, 331
 - acceptance_conditions, 332
 - condition, 332
 - dest, 332
- spot::tgba_explicit_succ_iterator, 332
 - ~tgba_explicit_succ_iterator, 334
 - all_acceptance_conditions_, 335
 - current_acceptance_conditions, 334
 - current_condition, 334
 - current_state, 334
 - done, 334
 - first, 334
 - i_, 335
 - next, 335
 - s_, 335
 - tgba_explicit_succ_iterator, 334
- spot::tgba_product, 335
 - ~tgba_product, 339
 - all_acceptance_conditions, 339
 - all_acceptance_conditions_, 341
 - compute_support_conditions, 339
 - compute_support_variables, 339
 - dict_, 341
 - format_state, 339
 - get_dict, 339
 - get_init_state, 340
 - left_, 342
 - left_acc_complement_, 342

- neg_acceptance_conditions, 340
- neg_acceptance_conditions_, 342
- number_of_acceptance_conditions, 340
- project_state, 340
- right_, 342
- right_acc_complement_, 342
- succ_iter, 340
- support_conditions, 341
- support_variables, 341
- tgba_product, 339
- tgba_product::operator=, 341
- transition_annotation, 341
- spot::tgba_reachable_iterator, 342
 - ~tgba_reachable_iterator, 344
 - add_state, 344
 - automata_, 345
 - end, 344
 - next_state, 344
 - process_link, 345
 - process_state, 345
 - run, 345
 - seen, 345
 - seen_map, 344
 - start, 345
 - tgba_reachable_iterator, 344
- spot::tgba_reachable_iterator_breadth_first, 346
 - add_state, 348
 - automata_, 349
 - end, 348
 - next_state, 348
 - process_link, 349
 - process_state, 349
 - run, 349
 - seen, 349
 - seen_map, 348
 - start, 349
 - tgba_reachable_iterator_breadth_first, 348
 - todo, 350
- spot::tgba_reachable_iterator_depth_first, 350
 - add_state, 352
 - automata_, 353
 - end, 352
 - next_state, 352
 - process_link, 353
 - process_state, 353
 - run, 353
 - seen, 353
 - seen_map, 352
 - start, 353
 - tgba_reachable_iterator_depth_first, 352
 - todo, 354
- spot::tgba_reduc, 354
 - ~tgba_reduc, 360
 - acc_, 368
 - add_acceptance_condition, 360
 - add_acceptance_conditions, 360
 - add_condition, 361
 - add_conditions, 361
 - add_state, 361
 - all_acceptance_conditions, 361
 - all_acceptance_conditions_, 368
 - all_acceptance_conditions_computed_, 368
 - automata_, 368
 - complement_all_acceptance_conditions, 361
 - compute_scc, 361
 - compute_support_conditions, 361
 - compute_support_variables, 361
 - copy_acceptance_conditions_of, 362
 - create_transition, 362
 - declare_acceptance_condition, 362
 - delete_scc, 362
 - delete_transitions, 362
 - dict_, 368
 - display_rel_sim, 362
 - display_scc, 362
 - end, 362
 - format_state, 362
 - get_acceptance_condition, 363
 - get_dict, 363
 - get_init_state, 363
 - h_, 368
 - has_acceptance_condition, 363
 - init_, 368
 - is_not_accepting, 363
 - is_terminal, 363
 - merge_state, 363
 - merge_state_delayed, 364
 - merge_transitions, 364
 - name_state_map_, 368
 - neg_acceptance_conditions, 364
 - neg_acceptance_conditions_, 368
 - next_state, 364
 - ns_map, 360
 - number_of_acceptance_conditions, 364
 - process_link, 364
 - process_state, 365
 - project_state, 365
 - prune_acc, 365
 - prune_scc, 365
 - quotient_state, 365
 - redirect_transition, 365
 - remove_acc, 366
 - remove_component, 366
 - remove_predecessor_state, 366
 - remove_scc, 366
 - remove_state, 366
 - root_, 368
 - run, 366

- scc_computed_, 368
- seen, 368
- seen_, 368
- seen_map, 360
- set_init_state, 366
- si_, 368
- sn_map, 360
- sp_map, 360
- start, 366
- state, 360
- state_name_map_, 368
- state_predecessor_map_, 368
- state_scc_, 368
- state_scc_v_, 368
- succ_iter, 366, 367
- support_conditions, 367
- support_variables, 367
- tgba_reduc, 360
- tgba_reduc::nb_set_acc_cond, 367
- todo, 369
- transition_annotation, 367
- spot::tgba_run, 369
 - ~tgba_run, 369
 - cycle, 370
 - operator=, 370
 - prefix, 370
 - steps, 369
 - tgba_run, 369, 370
- spot::tgba_run::step, 370
 - acc, 370
 - label, 370
 - s, 370
- spot::tgba_run_dotty_decorator, 371
 - ~tgba_run_dotty_decorator, 372
 - instance, 373
 - link_decl, 373
 - map_, 373
 - run_, 373
 - state_decl, 373
 - step_map, 372
 - step_num, 372
 - step_set, 372
 - tgba_run_dotty_decorator, 372
- spot::tgba_sba_proxy, 374
 - acc_cycle_, 380
 - all_acceptance_conditions, 377
 - compute_support_conditions, 377
 - compute_support_variables, 377
 - cycle_list, 377
 - format_state, 377
 - get_dict, 377
 - get_init_state, 377
 - neg_acceptance_conditions, 378
 - number_of_acceptance_conditions, 378
 - project_state, 378
 - state_is_accepting, 378
 - succ_iter, 378
 - support_conditions, 379
 - support_variables, 379
 - tgba_sba_proxy, 377
 - transition_annotation, 379
- spot::tgba_statistics, 380
 - states, 380
 - transitions, 380
- spot::tgba_succ_iterator, 380
 - ~tgba_succ_iterator, 382
 - current_acceptance_conditions, 382
 - current_condition, 382
 - current_state, 382
 - done, 382
 - first, 382
 - next, 383
- spot::tgba_succ_iterator_concrete, 383
 - ~tgba_succ_iterator_concrete, 385
 - current_, 386
 - current_acc_, 386
 - current_acceptance_conditions, 385
 - current_condition, 385
 - current_state, 386
 - current_state_, 387
 - data_, 387
 - done, 386
 - first, 386
 - next, 386
 - succ_set_, 387
 - succ_set_left_, 387
 - tgba_succ_iterator_concrete, 385
- spot::tgba_succ_iterator_product, 387
 - ~tgba_succ_iterator_product, 390
 - current_acceptance_conditions, 390
 - current_cond_, 391
 - current_condition, 390
 - current_state, 390
 - done, 390
 - first, 390
 - left_, 391
 - left_neg_, 391
 - next, 391
 - next_non_false_, 391
 - right_, 391
 - right_neg_, 391
 - step_, 391
 - tgba_product, 391
 - tgba_succ_iterator_product, 390
- spot::tgba_tba_proxy, 391
 - ~tgba_tba_proxy, 395
 - a_, 398
 - acc_cycle_, 398

- all_acceptance_conditions, 395
- compute_support_conditions, 395
- compute_support_variables, 395
- cycle_list, 395
- format_state, 395
- get_dict, 396
- get_init_state, 396
- neg_acceptance_conditions, 396
- number_of_acceptance_conditions, 396
- project_state, 396
- succ_iter, 396
- support_conditions, 397
- support_variables, 397
- tgba_tba_proxy, 395
- tgba_tba_proxy::operator=, 397
- the_acceptance_cond_, 398
- transition_annotation, 397
- spot::time_info, 398
 - stime, 398
 - time_info, 398
 - utime, 398
- spot::timer, 399
 - start, 399
 - start_, 400
 - stime, 399
 - stop, 400
 - total_, 400
 - utime, 400
- spot::timer_map, 400
 - cancel, 401
 - empty, 401
 - item_type, 401
 - print, 401
 - start, 401
 - stop, 402
 - timer, 402
 - tm, 402
 - tm_type, 401
- spot::unsigned_statistics, 415
 - ~unsigned_statistics, 416
 - get, 416
 - stats, 416
 - stats_map, 416
 - unsigned_fun, 416
- spot::unsigned_statistics_copy, 416
 - operator!=, 417
 - operator==, 417
 - set, 417
 - seteq, 417
 - stats, 417
 - stats_map, 417
 - unsigned_statistics_copy, 417
- spot::weight, 419
 - dec_weight_handler, 420
 - inc_weight_handler, 420
 - m, 421
 - neg_all_acc, 421
 - operator+=, 420
 - operator-, 420
 - operator=, 420
 - operator<<, 421
 - pm, 421
 - weight, 420
 - weight_vector, 420
- srand
 - random, 31
- stack_type
 - spot::scc_stack, 272
- start
 - spot::evtgba_reachable_iterator, 193
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 200
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 252
 - spot::parity_game_graph_direct, 258
 - spot::tgba_reachable_iterator, 345
 - spot::tgba_reachable_iterator_breadth_first, 349
 - spot::tgba_reachable_iterator_depth_first, 353
 - spot::tgba_reduc, 366
 - spot::timer, 399
 - spot::timer_map, 401
- start_
 - spot::timer, 400
- state
 - spot::tgba_explicit, 327
 - spot::tgba_reduc, 360
- state_
 - spot::state_bdd, 287
 - spot::state_evtgba_explicit, 290
 - spot::state_explicit, 292
- state_bdd
 - spot::state_bdd, 286
- state_couple
 - spot, 71
 - tgbaeduc.hh, 479
- state_decl
 - spot::dotty_decorator, 156
 - spot::tgba_run_dotty_decorator, 373
- state_evtgba_explicit
 - spot::state_evtgba_explicit, 289
- state_explicit
 - spot::state_explicit, 291
- state_index
 - spot::numbered_state_heap, 231
 - spot::numbered_state_heap_hash_map, 236

- state_index_p
 - spot::numbered_state_heap, 231
 - spot::numbered_state_heap_hash_map, 236
- state_is_accepting
 - spot::tgba_sba_proxy, 378
- state_name_map_
 - spot::evtgba_explicit, 185
 - spot::tgba_explicit, 331
 - spot::tgba_reduc, 368
- state_predecessor_map_
 - spot::tgba_reduc, 368
- state_product
 - spot::state_product, 294
- state_scc_
 - spot::tgba_reduc, 368
- state_scc_v_
 - spot::tgba_reduc, 368
- states
 - spot::connected_component_hash_set, 123
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_shy, 145
 - spot::couvreur99_check_status, 149
 - spot::ec_statistics, 168
 - spot::tgba_statistics, 380
- states_
 - spot::ec_statistics, 168
- statistics
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 145
 - spot::emptiness_check, 172
 - spot::emptiness_check_result, 177
- stats
 - spot::acss_statistics, 81
 - spot::ars_statistics, 83
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_result, 139
 - spot::couvreur99_check_shy, 146
 - spot::ec_statistics, 168
 - spot::unsigned_statistics, 416
 - spot::unsigned_statistics_copy, 417
- stats_map
 - spot::acss_statistics, 80
 - spot::ars_statistics, 82
 - spot::couvreur99_check, 133
 - spot::couvreur99_check_result, 138
 - spot::couvreur99_check_shy, 143
 - spot::ec_statistics, 167
 - spot::unsigned_statistics, 416
 - spot::unsigned_statistics_copy, 417
- stats_reachable
 - tgba_misc, 41
- step
 - spot::minato_isop::local_vars, 222
- step_
 - spot::tgba_succ_iterator_product, 391
- step_map
 - spot::tgba_run_dotty_decorator, 372
- step_num
 - spot::tgba_run_dotty_decorator, 372
- step_set
 - spot::tgba_run_dotty_decorator, 372
- steps
 - spot::tgba_run, 369
- stime
 - spot::time_info, 398
 - spot::timer, 399
- stop
 - spot::timer, 400
 - spot::timer_map, 402
- sub_set_acc_cond_
 - spot::parity_game_graph_delayed, 253
- succ_iter
 - spot::evtgba, 181
 - spot::evtgba_explicit, 185
 - spot::evtgba_product, 190
 - spot::tgba, 304
 - spot::tgba_bdd_concrete, 311
 - spot::tgba_explicit, 330
 - spot::tgba_product, 340
 - spot::tgba_reduc, 366, 367
 - spot::tgba_sba_proxy, 378
 - spot::tgba_tba_proxy, 396
- succ_queue
 - spot::couvreur99_check_shy, 143
- succ_set_
 - spot::tgba_succ_iterator_concrete, 387
- succ_set_left_
 - spot::tgba_succ_iterator_concrete, 387
- succ_to_string
 - spot::duplicator_node, 160
 - spot::duplicator_node_delayed, 164
 - spot::spoiler_node, 279
 - spot::spoiler_node_delayed, 282
- successor
 - spot::couvreur99_check_shy::successor, 146
- super
 - spot::ltl::simplify_f_g_visitor, 276
 - spot::ltl::unabbreviate_logic_visitor, 405
 - spot::ltl::unabbreviate_ltl_visitor, 408
- support_conditions
 - spot::tgba, 304
 - spot::tgba_bdd_concrete, 311
 - spot::tgba_explicit, 330
 - spot::tgba_product, 341
 - spot::tgba_reduc, 367
 - spot::tgba_sba_proxy, 379
 - spot::tgba_tba_proxy, 397

- support_variables
 - spot::tgba, 305
 - spot::tgba_bdd_concrete, 312
 - spot::tgba_explicit, 330
 - spot::tgba_product, 341
 - spot::tgba_reduc, 367
 - spot::tgba_sba_proxy, 379
 - spot::tgba_tba_proxy, 397
- symbol
 - spot::symbol, 298
- symbol.hh
 - rsymbol_set, 427
 - symbol_set, 427
- symbol_set
 - spot, 71
 - symbol.hh, 427
- syntactic_implication
 - ltl_misc, 26
- syntactic_implication_neg
 - ltl_misc, 27
- tgba
 - spot::tgba, 302
- TGBA (Transition-based Generalized Büchi Automata), 17
- TGBA algorithms, 32
- TGBA on-the-fly algorithms, 33
- TGBA representations, 32
- TGBA runs and supporting functions, 51
- TGBA simplifications, 38
- tgba/ Directory Reference, 59
- tgba/bdddict.hh, 462
- tgba/bddprint.hh, 464
- tgba/formula2bdd.hh, 468
- tgba/public.hh, 436
- tgba/state.hh, 468
- tgba/statebdd.hh, 469
- tgba/succiter.hh, 470
- tgba/succiterconcrete.hh, 471
- tgba/tgba.hh, 471
- tgba/tgababddconcrete.hh, 473
- tgba/tgababddconcretefactory.hh, 473
- tgba/tgababddconcreteproduct.hh, 474
- tgba/tgababddcoredata.hh, 475
- tgba/tgababddfactory.hh, 475
- tgba/tgbaexplicit.hh, 476
- tgba/tgabproduct.hh, 477
- tgba/tgbareduc.hh, 478
- tgba/tgbatba.hh, 480
- tgba2evtgba.hh
 - tgba_to_evtgba, 432
- tgba_algorithms
 - product, 33
- tgba_bdd_concrete
 - spot::tgba_bdd_concrete, 309
- tgba_bdd_concrete::operator=
 - spot::tgba_bdd_concrete, 312
- tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, 314
- tgba_bdd_core_data
 - spot::tgba_bdd_core_data, 318
- tgba_dupexp_bfs
 - tgba_misc, 41
- tgba_dupexp_dfs
 - tgba_misc, 42
- tgba_explicit
 - spot::tgba_explicit, 327
- tgba_explicit::operator=
 - spot::tgba_explicit, 330
- tgba_explicit_succ_iterator
 - spot::tgba_explicit_succ_iterator, 334
- tgba_io
 - dotty_reachable, 34
 - format_tgba_parse_errors, 34
 - lbt_reachable, 34
 - never_claim_reachable, 34
 - tgba_parse, 35
 - tgba_parse_error, 34
 - tgba_parse_error_list, 34
 - tgba_save_reachable, 35
- tgba_ltl
 - ltl_to_tgba_fm, 36
 - ltl_to_tgba_lacim, 37
- tgba_misc
 - random_graph, 41
 - stats_reachable, 41
 - tgba_dupexp_bfs, 41
 - tgba_dupexp_dfs, 42
 - tgba_powerset, 42
- tgba_parse
 - tgba_io, 35
- tgba_parse_error
 - tgba_io, 34
- tgba_parse_error_list
 - tgba_io, 34
- tgba_powerset
 - tgba_misc, 42
- tgba_product
 - spot::tgba_product, 339
 - spot::tgba_succ_iterator_product, 391
- tgba_product::operator=
 - spot::tgba_product, 341
- tgba_reachable_iterator
 - spot::tgba_reachable_iterator, 344
- tgba_reachable_iterator_breadth_first
 - spot::tgba_reachable_iterator_breadth_first, 348
- tgba_reachable_iterator_depth_first

- spot::tgba_reachable_iterator_depth_first, 352
- tgba_reduc
 - spot::tgba_reduc, 360
- tgba_reduc::nb_set_acc_cond
 - spot::tgba_reduc, 367
- tgba_reduction
 - Reduce_All, 39
 - Reduce_None, 39
 - Reduce_quotient_Del_Sim, 39
 - Reduce_quotient_Dir_Sim, 39
 - Reduce_Scc, 39
 - Reduce_transition_Del_Sim, 39
 - Reduce_transition_Dir_Sim, 39
- tgba_reduction
 - dn_v, 39
 - free_relation_simulation, 39
 - get_delayed_relation_simulation, 40
 - get_direct_relation_simulation, 40
 - reduc_tgba_sim, 40
 - reduce_tgba_options, 39
 - s_v, 39
 - sn_v, 39
- tgba_run
 - print_tgba_run, 52
 - project_tgba_run, 52
 - reduce_run, 52
 - replay_tgba_run, 53
 - spot::tgba_run, 369, 370
 - tgba_run_to_tgba, 53
- tgba_run_dotty_decorator
 - spot::tgba_run_dotty_decorator, 372
- tgba_run_to_tgba
 - tgba_run, 53
- tgba_save_reachable
 - tgba_io, 35
- tgba_sba_proxy
 - spot::tgba_sba_proxy, 377
- tgba_state_
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 253
 - spot::parity_game_graph_direct, 259
- tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, 385
- tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, 390
- tgba_tba_proxy
 - spot::tgba_tba_proxy, 395
- tgba_tba_proxy::operator=
 - spot::tgba_tba_proxy, 397
- tgba_to_evtgba
 - spot, 74
 - tgba2evtgba.hh, 432
- tgbaalgos/ Directory Reference, 60
- tgbaalgos/bfssteps.hh, 480
- tgbaalgos/dotty.hh, 429
- tgbaalgos/dottydec.hh, 481
- tgbaalgos/dupeexp.hh, 481
- tgbaalgos/emptiness.hh, 482
- tgbaalgos/emptiness_stats.hh, 483
- tgbaalgos/gtec/ Directory Reference, 55
- tgbaalgos/gtec/ce.hh, 484
- tgbaalgos/gtec/explscs.hh, 485
- tgbaalgos/gtec/gtec.hh, 485
- tgbaalgos/gtec/nsheap.hh, 486
- tgbaalgos/gtec/sccstack.hh, 487
- tgbaalgos/gtec/status.hh, 488
- tgbaalgos/gv04.hh, 489
- tgbaalgos/lbtt.hh, 489
- tgbaalgos/ltl2tgba_fm.hh, 490
- tgbaalgos/ltl2tgba_lacim.hh, 490
- tgbaalgos/magic.hh, 491
- tgbaalgos/neverclaim.hh, 492
- tgbaalgos/powerset.hh, 492
- tgbaalgos/projrun.hh, 493
- tgbaalgos/randomgraph.hh, 494
- tgbaalgos/reachiter.hh, 430
- tgbaalgos/reducerun.hh, 494
- tgbaalgos/reductgba_sim.hh, 494
- tgbaalgos/replayrun.hh, 496
- tgbaalgos/rundotdec.hh, 496
- tgbaalgos/save.hh, 432
- tgbaalgos/se05.hh, 497
- tgbaalgos/stats.hh, 498
- tgbaalgos/tau03.hh, 498
- tgbaalgos/tau03opt.hh, 499
- tgbaalgos/weight.hh, 499
- tgbaalparse/ Directory Reference, 61
- tgbaalparse/public.hh, 436
- tgbaalreduc.hh
 - simulation_relation, 479
 - state_couple, 479
- the_acceptance_cond_
 - spot::tgba_tba_proxy, 398
- ThirdStep
 - spot::minato_isop::local_vars, 221
- time_info
 - spot::time_info, 398
- timer
 - spot::timer_map, 402
- tm
 - spot::timer_map, 402
- tm_type
 - spot::timer_map, 401
- to_spin_string
 - ltl_io, 22
- to_string
 - ltl_io, 22
 - spot::duplicator_node, 160

- spot::duplicator_node_delayed, 165
- spot::spoiler_node, 279
- spot::spoiler_node_delayed, 282
- todo
 - spot::couvreur99_check_shy, 146
 - spot::evtgba_reachable_iterator_breadth_first, 197
 - spot::evtgba_reachable_iterator_depth_first, 201
 - spot::parity_game_graph, 246
 - spot::parity_game_graph_delayed, 253
 - spot::parity_game_graph_direct, 259
 - spot::tgba_reachable_iterator_breadth_first, 350
 - spot::tgba_reachable_iterator_depth_first, 354
 - spot::tgba_reduc, 369
- todo_
 - spot::minato_isop, 220
- todo_item
 - spot::couvreur99_check_shy::todo_item, 147
- todo_list
 - spot::couvreur99_check_shy, 143
- top
 - spot::scc_stack, 272, 273
- total_
 - spot::timer, 400
- total_1_
 - spot::ltl::random_ltl, 264
- total_2_
 - spot::ltl::random_ltl, 264
- total_2_and_more_
 - spot::ltl::random_ltl, 264
- transition_annotation
 - spot::tgba, 305
 - spot::tgba_bdd_concrete, 312
 - spot::tgba_explicit, 330
 - spot::tgba_product, 341
 - spot::tgba_reduc, 367
 - spot::tgba_sba_proxy, 379
 - spot::tgba_tba_proxy, 397
- transition_list
 - spot::evtgba_explicit, 183
- transitions
 - spot::couvreur99_check, 135
 - spot::couvreur99_check_shy, 145
 - spot::ec_statistics, 168
 - spot::tgba_statistics, 380
- transitions_
 - spot::ec_statistics, 168
- Translating LTL formulae into TGBA, 35
- True
 - spot::ltl::constant, 128
- true_instance
 - spot::ltl::constant, 129
- tunabbrev.hh
 - unabbreviate_ltl, 455
- type
 - spot::ltl::binop, 114
 - spot::ltl::constant, 128
 - spot::ltl::multop, 227
 - spot::ltl::unop, 412
- U
 - spot::ltl::binop, 115
- unabbreviate_logic
 - ltl_rewriting, 24
- unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, 405
- unabbreviate_ltl
 - spot::ltl, 78
 - tunabbrev.hh, 455
- unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, 408
- unop
 - spot::ltl::unop, 412
- unref
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::constant, 129
 - spot::ltl::formula, 206
 - spot::ltl::multop, 229
 - spot::ltl::ref_formula, 269
 - spot::ltl::unop, 414
 - spot::symbol, 299
- unref_
 - spot::ltl::atomic_prop, 88
 - spot::ltl::binop, 116
 - spot::ltl::constant, 129
 - spot::ltl::formula, 207
 - spot::ltl::multop, 229
 - spot::ltl::ref_formula, 269
 - spot::ltl::unop, 414
- unregister_all_my_variables
 - spot::bdd_dict, 103
- unregister_variable
 - spot::bdd_dict, 103
- unsigned_fun
 - spot::acss_statistics, 80
 - spot::ars_statistics, 82
 - spot::couvreur99_check, 133
 - spot::couvreur99_check_result, 138
 - spot::couvreur99_check_shy, 143
 - spot::ec_statistics, 167
 - spot::unsigned_statistics, 416
- unsigned_statistics_copy
 - spot::unsigned_statistics_copy, 417
- update_sums
 - spot::ltl::random_ltl, 264

- utime
 - spot::time_info, 398
 - spot::timer, 400
- v1
 - spot::minato_isop::local_vars, 222
- val
 - spot::ltl::constant, 129
- val_
 - spot::ltl::constant, 130
- val_name
 - spot::ltl::constant, 129
- var_formula_map
 - spot::bdd_dict, 104
- var_map
 - spot::bdd_dict, 104
- var_refs
 - spot::bdd_dict, 104
- var_set
 - spot::tgba_bdd_core_data, 320
- varandnext_set
 - spot::tgba_bdd_core_data, 320
- vars
 - spot::minato_isop::local_vars, 222
- vec
 - spot::ltl::multop, 226
- version
 - misc_tools, 29
- vf_map
 - spot::bdd_dict, 100
- visit
 - spot::ltl::clone_visitor, 120
 - spot::ltl::const_visitor, 125
 - spot::ltl::postfix_visitor, 261
 - spot::ltl::simplify_f_g_visitor, 276
 - spot::ltl::unabbreviate_logic_visitor, 405
 - spot::ltl::unabbreviate_ltl_visitor, 408
 - spot::ltl::visitor, 419
- vr_map
 - spot::bdd_dict, 100
- wang32_hash
 - hash_funcs, 29
- weight
 - spot::weight, 420
- weight_vector
 - spot::weight, 420
- where_
 - spot::gspn_exeption, 212
- X
 - spot::ltl::unop, 412
- Xor
 - spot::ltl::binop, 114