

spot  
0.6

Generated by Doxygen 1.6.3

Fri Apr 16 09:40:33 2010

## Contents

<b>1</b>	<b>Main Page</b>	<b>2</b>
1.1	The Spot Library . . . . .	2
1.2	This Document . . . . .	2
1.3	Handy starting points . . . . .	2
<b>2</b>	<b>Deprecated List</b>	<b>2</b>
<b>3</b>	<b>Bug List</b>	<b>2</b>
<b>4</b>	<b>Module Documentation</b>	<b>3</b>
4.1	LTL formulae . . . . .	3
4.1.1	Detailed Description . . . . .	3
4.2	SABA (State-based Alternating Büchi Automata) . . . . .	3
4.2.1	Detailed Description . . . . .	3
4.3	TGBA (Transition-based Generalized Büchi Automata) . . . . .	3
4.3.1	Detailed Description . . . . .	4
4.4	Emptiness-check algorithms for SSP . . . . .	4
4.4.1	Function Documentation . . . . .	4
4.5	Input/Output of LTL formulae . . . . .	4
4.5.1	Typedef Documentation . . . . .	6
4.5.2	Function Documentation . . . . .	6
4.6	Essential LTL types . . . . .	10
4.6.1	Function Documentation . . . . .	10
4.7	LTL Abstract Syntax Tree . . . . .	11
4.8	LTL environments . . . . .	11
4.8.1	Detailed Description . . . . .	12
4.9	Algorithms for LTL formulae . . . . .	12
4.10	Derivable visitors . . . . .	12
4.11	Rewriting LTL formulae . . . . .	12
4.11.1	Enumeration Type Documentation . . . . .	13
4.11.2	Function Documentation . . . . .	13
4.12	Miscellaneous algorithms for LTL formulae . . . . .	15
4.12.1	Typedef Documentation . . . . .	15
4.12.2	Function Documentation . . . . .	16
4.13	Miscellaneous helper algorithms . . . . .	18
4.13.1	Detailed Description . . . . .	19

4.13.2 Function Documentation . . . . .	19
4.14 Hashing functions . . . . .	20
4.14.1 Function Documentation . . . . .	21
4.15 Random functions . . . . .	21
4.15.1 Function Documentation . . . . .	22
4.16 Essential SABA types . . . . .	23
4.17 Essential TGBA types . . . . .	24
4.18 TGBA representations . . . . .	25
4.19 TGBA algorithms . . . . .	25
4.19.1 Function Documentation . . . . .	26
4.20 TGBA on-the-fly algorithms . . . . .	26
4.21 Input/Output of TGBA . . . . .	26
4.21.1 Typedef Documentation . . . . .	27
4.21.2 Function Documentation . . . . .	28
4.22 Translating LTL formulae into TGBA . . . . .	29
4.22.1 Function Documentation . . . . .	30
4.23 Algorithm patterns . . . . .	33
4.24 TGBA simplifications . . . . .	33
4.24.1 Typedef Documentation . . . . .	34
4.24.2 Enumeration Type Documentation . . . . .	35
4.24.3 Function Documentation . . . . .	35
4.25 Miscellaneous algorithms on TGBA . . . . .	36
4.25.1 Function Documentation . . . . .	37
4.26 Decorating the dot output . . . . .	38
4.27 Emptiness-checks . . . . .	38
4.27.1 Detailed Description . . . . .	39
4.28 Emptiness-check algorithms . . . . .	39
4.28.1 Function Documentation . . . . .	40
4.29 TGBA runs and supporting functions . . . . .	48
4.29.1 Function Documentation . . . . .	49
4.30 Emptiness-check statistics . . . . .	50
<b>5 Directory Documentation</b> . . . . .	<b>51</b>
5.1 eltlparse/ Directory Reference . . . . .	51
5.2 evtgba/ Directory Reference . . . . .	51
5.3 evtgbaalgs/ Directory Reference . . . . .	52
5.4 evtgbaparse/ Directory Reference . . . . .	52

5.5	gspn/ Directory Reference . . . . .	53
5.6	tgbaalgos/gtec/ Directory Reference . . . . .	53
5.7	kripke/ Directory Reference . . . . .	54
5.8	ltlast/ Directory Reference . . . . .	54
5.9	ltlenv/ Directory Reference . . . . .	55
5.10	ltlparse/ Directory Reference . . . . .	56
5.11	ltlvisit/ Directory Reference . . . . .	56
5.12	misc/ Directory Reference . . . . .	57
5.13	nips/ Directory Reference . . . . .	57
5.14	saba/ Directory Reference . . . . .	58
5.15	sabaalgos/ Directory Reference . . . . .	58
5.16	sautparse/ Directory Reference . . . . .	59
5.17	tgba/ Directory Reference . . . . .	59
5.18	tgbaalgos/ Directory Reference . . . . .	60
5.19	tgbaparse/ Directory Reference . . . . .	61
<b>6</b>	<b>Namespace Documentation</b>	<b>61</b>
6.1	eltly Namespace Reference . . . . .	61
6.1.1	Function Documentation . . . . .	62
6.2	ltly Namespace Reference . . . . .	64
6.2.1	Function Documentation . . . . .	66
6.3	sauty Namespace Reference . . . . .	68
6.3.1	Function Documentation . . . . .	68
6.4	spot Namespace Reference . . . . .	70
6.4.1	Typedef Documentation . . . . .	84
6.4.2	Function Documentation . . . . .	85
6.5	spot::eltl Namespace Reference . . . . .	91
6.6	spot::ltl Namespace Reference . . . . .	92
6.6.1	Function Documentation . . . . .	96
6.7	spot::ltl::formula_tree Namespace Reference . . . . .	96
6.7.1	Detailed Description . . . . .	97
6.7.2	Typedef Documentation . . . . .	97
6.7.3	Enumeration Type Documentation . . . . .	97
6.7.4	Function Documentation . . . . .	98
<b>7</b>	<b>Class Documentation</b>	<b>98</b>
7.1	spot::acss_statistics Class Reference . . . . .	98



7.1.1	Detailed Description . . . . .	101
7.1.2	Member Typedef Documentation . . . . .	101
7.1.3	Constructor & Destructor Documentation . . . . .	101
7.1.4	Member Function Documentation . . . . .	101
7.1.5	Member Data Documentation . . . . .	102
7.2	spot::bdd_dict::anon_free_list Class Reference . . . . .	102
7.2.1	Member Typedef Documentation . . . . .	105
7.2.2	Constructor & Destructor Documentation . . . . .	105
7.2.3	Member Function Documentation . . . . .	106
7.2.4	Member Data Documentation . . . . .	107
7.3	spot::ars_statistics Class Reference . . . . .	107
7.3.1	Detailed Description . . . . .	110
7.3.2	Member Typedef Documentation . . . . .	110
7.3.3	Constructor & Destructor Documentation . . . . .	110
7.3.4	Member Function Documentation . . . . .	110
7.3.5	Member Data Documentation . . . . .	111
7.4	spot::ltl::atomic_prop Class Reference . . . . .	111
7.4.1	Detailed Description . . . . .	115
7.4.2	Member Typedef Documentation . . . . .	115
7.4.3	Constructor & Destructor Documentation . . . . .	115
7.4.4	Member Function Documentation . . . . .	115
7.4.5	Member Data Documentation . . . . .	117
7.5	spot::ltl::automatop Class Reference . . . . .	118
7.5.1	Detailed Description . . . . .	121
7.5.2	Member Typedef Documentation . . . . .	121
7.5.3	Constructor & Destructor Documentation . . . . .	122
7.5.4	Member Function Documentation . . . . .	122
7.5.5	Member Data Documentation . . . . .	124
7.6	spot::barand< gen > Class Template Reference . . . . .	125
7.6.1	Detailed Description . . . . .	125
7.6.2	Constructor & Destructor Documentation . . . . .	125
7.6.3	Member Function Documentation . . . . .	126
7.6.4	Member Data Documentation . . . . .	126
7.7	spot::bdd_allocator Class Reference . . . . .	126
7.7.1	Detailed Description . . . . .	130
7.7.2	Member Typedef Documentation . . . . .	130

7.7.3	Constructor & Destructor Documentation	130
7.7.4	Member Function Documentation	130
7.7.5	Member Data Documentation	132
7.8	spot::bdd_dict Class Reference	132
7.8.1	Detailed Description	137
7.8.2	Member Typedef Documentation	137
7.8.3	Constructor & Destructor Documentation	138
7.8.4	Member Function Documentation	138
7.8.5	Member Data Documentation	141
7.9	spot::bdd_less_than Struct Reference	143
7.9.1	Detailed Description	143
7.9.2	Member Function Documentation	143
7.10	spot::bdd_ordered Class Reference	143
7.10.1	Constructor & Destructor Documentation	144
7.10.2	Member Function Documentation	144
7.10.3	Member Data Documentation	144
7.11	spot::bfs_steps Class Reference	145
7.11.1	Detailed Description	146
7.11.2	Constructor & Destructor Documentation	146
7.11.3	Member Function Documentation	146
7.11.4	Member Data Documentation	147
7.12	spot::ltl::binop Class Reference	148
7.12.1	Detailed Description	151
7.12.2	Member Typedef Documentation	151
7.12.3	Member Enumeration Documentation	152
7.12.4	Constructor & Destructor Documentation	152
7.12.5	Member Function Documentation	152
7.12.6	Member Data Documentation	154
7.13	spot::char_ptr_less_than Struct Reference	155
7.13.1	Detailed Description	155
7.13.2	Member Function Documentation	155
7.14	spot::ltl::clone_visitor Class Reference	156
7.14.1	Detailed Description	158
7.14.2	Constructor & Destructor Documentation	158
7.14.3	Member Function Documentation	158
7.14.4	Member Data Documentation	159

7.15	<a href="#">spot::scc_stack::connected_component Struct Reference</a>	159
7.15.1	<a href="#">Constructor &amp; Destructor Documentation</a>	161
7.15.2	<a href="#">Member Data Documentation</a>	161
7.16	<a href="#">spot::connected_component_hash_set Class Reference</a>	161
7.16.1	<a href="#">Detailed Description</a>	164
7.16.2	<a href="#">Member Typedef Documentation</a>	164
7.16.3	<a href="#">Constructor &amp; Destructor Documentation</a>	164
7.16.4	<a href="#">Member Function Documentation</a>	164
7.16.5	<a href="#">Member Data Documentation</a>	165
7.17	<a href="#">spot::connected_component_hash_set_factory Class Reference</a>	165
7.17.1	<a href="#">Detailed Description</a>	167
7.17.2	<a href="#">Constructor &amp; Destructor Documentation</a>	167
7.17.3	<a href="#">Member Function Documentation</a>	167
7.18	<a href="#">spot::ltl::const_visitor Struct Reference</a>	168
7.18.1	<a href="#">Detailed Description</a>	168
7.18.2	<a href="#">Constructor &amp; Destructor Documentation</a>	168
7.18.3	<a href="#">Member Function Documentation</a>	168
7.19	<a href="#">spot::ltl::constant Class Reference</a>	169
7.19.1	<a href="#">Detailed Description</a>	173
7.19.2	<a href="#">Member Enumeration Documentation</a>	173
7.19.3	<a href="#">Constructor &amp; Destructor Documentation</a>	173
7.19.4	<a href="#">Member Function Documentation</a>	173
7.19.5	<a href="#">Member Data Documentation</a>	175
7.20	<a href="#">spot::couvreur99_check Class Reference</a>	175
7.20.1	<a href="#">Detailed Description</a>	179
7.20.2	<a href="#">Member Typedef Documentation</a>	179
7.20.3	<a href="#">Constructor &amp; Destructor Documentation</a>	179
7.20.4	<a href="#">Member Function Documentation</a>	180
7.20.5	<a href="#">Member Data Documentation</a>	183
7.21	<a href="#">spot::couvreur99_check_result Class Reference</a>	183
7.21.1	<a href="#">Detailed Description</a>	187
7.21.2	<a href="#">Member Typedef Documentation</a>	187
7.21.3	<a href="#">Constructor &amp; Destructor Documentation</a>	187
7.21.4	<a href="#">Member Function Documentation</a>	187
7.21.5	<a href="#">Member Data Documentation</a>	189
7.22	<a href="#">spot::couvreur99_check_shy Class Reference</a>	190

7.22.1	Detailed Description	194
7.22.2	Member Typedef Documentation	194
7.22.3	Constructor & Destructor Documentation	194
7.22.4	Member Function Documentation	194
7.22.5	Member Data Documentation	198
7.23	spot::couvreur99_check_status Class Reference	199
7.23.1	Detailed Description	201
7.23.2	Constructor & Destructor Documentation	201
7.23.3	Member Function Documentation	201
7.23.4	Member Data Documentation	201
7.24	spot::ltl::declarative_environment Class Reference	202
7.24.1	Detailed Description	205
7.24.2	Member Typedef Documentation	205
7.24.3	Constructor & Destructor Documentation	205
7.24.4	Member Function Documentation	205
7.24.5	Member Data Documentation	206
7.25	spot::ltl::default_environment Class Reference	206
7.25.1	Detailed Description	209
7.25.2	Constructor & Destructor Documentation	209
7.25.3	Member Function Documentation	209
7.26	spot::delayed_simulation_relation Class Reference	210
7.27	spot::direct_simulation_relation Class Reference	210
7.28	spot::taa_succ_iterator::distance_sort Struct Reference	210
7.28.1	Member Function Documentation	210
7.29	spot::dotty_decorator Class Reference	210
7.29.1	Detailed Description	212
7.29.2	Constructor & Destructor Documentation	212
7.29.3	Member Function Documentation	212
7.30	spot::duplicator_node Class Reference	213
7.30.1	Detailed Description	216
7.30.2	Constructor & Destructor Documentation	216
7.30.3	Member Function Documentation	216
7.30.4	Member Data Documentation	218
7.31	spot::duplicator_node_delayed Class Reference	219
7.31.1	Detailed Description	221
7.31.2	Constructor & Destructor Documentation	221

7.31.3	Member Function Documentation	222
7.31.4	Member Data Documentation	224
7.32	spot::ec_statistics Class Reference	225
7.32.1	Detailed Description	227
7.32.2	Member Typedef Documentation	227
7.32.3	Constructor & Destructor Documentation	227
7.32.4	Member Function Documentation	228
7.32.5	Member Data Documentation	229
7.33	spot::emptiness_check Class Reference	230
7.33.1	Detailed Description	232
7.33.2	Constructor & Destructor Documentation	232
7.33.3	Member Function Documentation	232
7.33.4	Member Data Documentation	234
7.34	spot::emptiness_check_instantiator Class Reference	234
7.34.1	Constructor & Destructor Documentation	236
7.34.2	Member Function Documentation	236
7.34.3	Member Data Documentation	237
7.35	spot::emptiness_check_result Class Reference	237
7.35.1	Detailed Description	240
7.35.2	Constructor & Destructor Documentation	240
7.35.3	Member Function Documentation	240
7.35.4	Member Data Documentation	241
7.36	spot::lrl::environment Class Reference	242
7.36.1	Detailed Description	243
7.36.2	Constructor & Destructor Documentation	243
7.36.3	Member Function Documentation	243
7.37	spot::evtgba Class Reference	243
7.37.1	Constructor & Destructor Documentation	245
7.37.2	Member Function Documentation	245
7.38	spot::evtgba_explicit Class Reference	246
7.38.1	Member Typedef Documentation	249
7.38.2	Constructor & Destructor Documentation	250
7.38.3	Member Function Documentation	250
7.38.4	Member Data Documentation	252
7.39	spot::evtgba_iterator Class Reference	252
7.39.1	Constructor & Destructor Documentation	252

7.39.2	Member Function Documentation	253
7.40	spot::evtgba_product Class Reference	253
7.40.1	Member Typedef Documentation	256
7.40.2	Constructor & Destructor Documentation	256
7.40.3	Member Function Documentation	256
7.40.4	Member Data Documentation	258
7.41	spot::evtgba_reachable_iterator Class Reference	258
7.41.1	Detailed Description	261
7.41.2	Member Typedef Documentation	261
7.41.3	Constructor & Destructor Documentation	261
7.41.4	Member Function Documentation	262
7.41.5	Member Data Documentation	263
7.42	spot::evtgba_reachable_iterator_breadth_first Class Reference	263
7.42.1	Detailed Description	266
7.42.2	Member Typedef Documentation	266
7.42.3	Constructor & Destructor Documentation	266
7.42.4	Member Function Documentation	266
7.42.5	Member Data Documentation	268
7.43	spot::evtgba_reachable_iterator_depth_first Class Reference	268
7.43.1	Detailed Description	271
7.43.2	Member Typedef Documentation	271
7.43.3	Constructor & Destructor Documentation	271
7.43.4	Member Function Documentation	271
7.43.5	Member Data Documentation	273
7.44	spot::explicit_connected_component Class Reference	273
7.44.1	Detailed Description	275
7.44.2	Constructor & Destructor Documentation	276
7.44.3	Member Function Documentation	276
7.44.4	Member Data Documentation	276
7.45	spot::explicit_connected_component_factory Class Reference	277
7.45.1	Detailed Description	277
7.45.2	Constructor & Destructor Documentation	277
7.45.3	Member Function Documentation	278
7.46	spot::explicit_state_conjunction Class Reference	278
7.46.1	Detailed Description	281
7.46.2	Member Typedef Documentation	281

7.46.3	Constructor & Destructor Documentation	281
7.46.4	Member Function Documentation	282
7.46.5	Member Data Documentation	283
7.47	spot::fair_kripke Class Reference	283
7.47.1	Member Function Documentation	286
7.48	spot::fair_kripke_succ_iterator Class Reference	290
7.48.1	Constructor & Destructor Documentation	293
7.48.2	Member Function Documentation	293
7.48.3	Member Data Documentation	295
7.49	spot::ltl::formula Class Reference	295
7.49.1	Detailed Description	297
7.49.2	Constructor & Destructor Documentation	297
7.49.3	Member Function Documentation	298
7.49.4	Member Data Documentation	299
7.50	spot::ltl::formula_ptr_hash Struct Reference	299
7.50.1	Detailed Description	300
7.50.2	Member Function Documentation	300
7.51	spot::ltl::formula_ptr_less_than Struct Reference	300
7.51.1	Detailed Description	300
7.51.2	Member Function Documentation	300
7.52	spot::free_list Class Reference	301
7.52.1	Detailed Description	302
7.52.2	Member Typedef Documentation	302
7.52.3	Constructor & Destructor Documentation	303
7.52.4	Member Function Documentation	303
7.52.5	Member Data Documentation	304
7.53	spot::future_conditions_collector Class Reference	304
7.53.1	Detailed Description	308
7.53.2	Member Typedef Documentation	308
7.53.3	Constructor & Destructor Documentation	308
7.53.4	Member Function Documentation	308
7.53.5	Member Data Documentation	312
7.54	spot::gspn_exception Class Reference	312
7.54.1	Detailed Description	313
7.54.2	Constructor & Destructor Documentation	313
7.54.3	Member Function Documentation	313

7.54.4	Member Data Documentation	313
7.55	spot::gspn_interface Class Reference	314
7.55.1	Constructor & Destructor Documentation	316
7.55.2	Member Function Documentation	316
7.55.3	Member Data Documentation	316
7.56	spot::gspn_ssp_interface Class Reference	317
7.56.1	Constructor & Destructor Documentation	319
7.56.2	Member Function Documentation	319
7.56.3	Member Data Documentation	319
7.57	spot::kripke Class Reference	319
7.57.1	Constructor & Destructor Documentation	322
7.57.2	Member Function Documentation	322
7.58	spot::ltl::language_containment_checker Class Reference	326
7.58.1	Member Typedef Documentation	328
7.58.2	Constructor & Destructor Documentation	328
7.58.3	Member Function Documentation	329
7.58.4	Member Data Documentation	329
7.59	spot::minato_isop::local_vars Struct Reference	330
7.59.1	Detailed Description	330
7.59.2	Member Enumeration Documentation	331
7.59.3	Constructor & Destructor Documentation	331
7.59.4	Member Data Documentation	331
7.60	eltly::location Class Reference	332
7.60.1	Detailed Description	334
7.60.2	Constructor & Destructor Documentation	334
7.60.3	Member Function Documentation	334
7.60.4	Member Data Documentation	335
7.61	sauty::location Class Reference	335
7.61.1	Detailed Description	337
7.61.2	Constructor & Destructor Documentation	337
7.61.3	Member Function Documentation	337
7.61.4	Member Data Documentation	338
7.62	ltly::location Class Reference	338
7.62.1	Detailed Description	340
7.62.2	Constructor & Destructor Documentation	340
7.62.3	Member Function Documentation	340



7.62.4	Member Data Documentation	341
7.63	spot::loopless_modular_mixed_radix_gray_code Class Reference	341
7.63.1	Detailed Description	342
7.63.2	Constructor & Destructor Documentation	343
7.63.3	Member Function Documentation	343
7.63.4	Member Data Documentation	344
7.64	spot::ltl::ltl_file Class Reference	345
7.64.1	Detailed Description	345
7.64.2	Constructor & Destructor Documentation	345
7.64.3	Member Function Documentation	346
7.64.4	Member Data Documentation	346
7.65	spot::minato_isop Class Reference	346
7.65.1	Detailed Description	347
7.65.2	Constructor & Destructor Documentation	347
7.65.3	Member Function Documentation	347
7.65.4	Member Data Documentation	348
7.66	spot::ltl::multop Class Reference	348
7.66.1	Detailed Description	352
7.66.2	Member Typedef Documentation	352
7.66.3	Member Enumeration Documentation	353
7.66.4	Constructor & Destructor Documentation	353
7.66.5	Member Function Documentation	353
7.66.6	Member Data Documentation	356
7.67	spot::ltl::nfa Class Reference	356
7.67.1	Detailed Description	359
7.67.2	Member Typedef Documentation	359
7.67.3	Constructor & Destructor Documentation	359
7.67.4	Member Function Documentation	360
7.67.5	Member Data Documentation	361
7.68	spot::nips_exception Class Reference	362
7.68.1	Detailed Description	362
7.68.2	Constructor & Destructor Documentation	362
7.68.3	Member Function Documentation	362
7.68.4	Member Data Documentation	363
7.69	spot::nips_interface Class Reference	363
7.69.1	Detailed Description	365

7.69.2	Constructor & Destructor Documentation	365
7.69.3	Member Function Documentation	365
7.69.4	Member Data Documentation	365
7.70	spot::ltl::formula_tree::node Struct Reference	366
7.70.1	Constructor & Destructor Documentation	366
7.71	spot::ltl::formula_tree::node_atomic Struct Reference	366
7.71.1	Member Data Documentation	368
7.72	spot::ltl::formula_tree::node_binop Struct Reference	368
7.72.1	Member Data Documentation	369
7.73	spot::ltl::formula_tree::node_multop Struct Reference	370
7.73.1	Member Data Documentation	371
7.74	spot::ltl::formula_tree::node_nfa Struct Reference	371
7.74.1	Member Data Documentation	372
7.75	spot::ltl::formula_tree::node_unop Struct Reference	372
7.75.1	Member Data Documentation	374
7.76	spot::numbered_state_heap Class Reference	374
7.76.1	Detailed Description	376
7.76.2	Member Typedef Documentation	376
7.76.3	Constructor & Destructor Documentation	376
7.76.4	Member Function Documentation	376
7.77	spot::numbered_state_heap_const_iterator Class Reference	378
7.77.1	Detailed Description	378
7.77.2	Constructor & Destructor Documentation	378
7.77.3	Member Function Documentation	378
7.78	spot::numbered_state_heap_factory Class Reference	379
7.78.1	Detailed Description	380
7.78.2	Constructor & Destructor Documentation	380
7.78.3	Member Function Documentation	380
7.79	spot::numbered_state_heap_hash_map Class Reference	380
7.79.1	Detailed Description	383
7.79.2	Member Typedef Documentation	383
7.79.3	Constructor & Destructor Documentation	383
7.79.4	Member Function Documentation	384
7.79.5	Member Data Documentation	385
7.80	spot::numbered_state_heap_hash_map_factory Class Reference	385
7.80.1	Detailed Description	387

7.80.2	Constructor & Destructor Documentation	387
7.80.3	Member Function Documentation	387
7.81	spot::ltl::random_ltl::op_proba Struct Reference	388
7.81.1	Member Typedef Documentation	389
7.81.2	Member Function Documentation	389
7.81.3	Member Data Documentation	389
7.82	spot::option_map Class Reference	389
7.82.1	Detailed Description	390
7.82.2	Member Function Documentation	390
7.82.3	Friends And Related Function Documentation	392
7.82.4	Member Data Documentation	392
7.83	spot::ltl::multop::paircmp Struct Reference	392
7.83.1	Detailed Description	392
7.83.2	Member Function Documentation	392
7.84	spot::parity_game_graph Class Reference	392
7.84.1	Detailed Description	395
7.84.2	Member Typedef Documentation	396
7.84.3	Constructor & Destructor Documentation	396
7.84.4	Member Function Documentation	396
7.84.5	Member Data Documentation	398
7.85	spot::parity_game_graph_delayed Class Reference	399
7.85.1	Detailed Description	403
7.85.2	Member Typedef Documentation	403
7.85.3	Constructor & Destructor Documentation	403
7.85.4	Member Function Documentation	404
7.85.5	Member Data Documentation	406
7.86	spot::parity_game_graph_direct Class Reference	407
7.86.1	Detailed Description	410
7.86.2	Member Typedef Documentation	411
7.86.3	Constructor & Destructor Documentation	411
7.86.4	Member Function Documentation	411
7.86.5	Member Data Documentation	413
7.87	sauty::position Class Reference	414
7.87.1	Detailed Description	415
7.87.2	Constructor & Destructor Documentation	415
7.87.3	Member Function Documentation	415

7.87.4	Member Data Documentation	415
7.88	eltlyy::position Class Reference	416
7.88.1	Detailed Description	417
7.88.2	Constructor & Destructor Documentation	417
7.88.3	Member Function Documentation	417
7.88.4	Member Data Documentation	417
7.89	ltlyy::position Class Reference	418
7.89.1	Detailed Description	419
7.89.2	Constructor & Destructor Documentation	419
7.89.3	Member Function Documentation	419
7.89.4	Member Data Documentation	419
7.90	spot::ltl::postfix_visitor Class Reference	420
7.90.1	Detailed Description	423
7.90.2	Constructor & Destructor Documentation	423
7.90.3	Member Function Documentation	423
7.91	spot::ptr_hash< T > Struct Template Reference	424
7.91.1	Detailed Description	425
7.91.2	Member Function Documentation	425
7.92	spot::ltl::random_ltl Class Reference	425
7.92.1	Detailed Description	427
7.92.2	Constructor & Destructor Documentation	427
7.92.3	Member Function Documentation	427
7.92.4	Member Data Documentation	429
7.93	spot::ltl::language_containment_checker::record_ Struct Reference	430
7.93.1	Member Typedef Documentation	431
7.93.2	Member Data Documentation	431
7.94	spot::ltl::ref_formula Class Reference	431
7.94.1	Detailed Description	434
7.94.2	Constructor & Destructor Documentation	434
7.94.3	Member Function Documentation	435
7.94.4	Member Data Documentation	436
7.95	spot::rsymbol Class Reference	436
7.95.1	Constructor & Destructor Documentation	438
7.95.2	Member Function Documentation	438
7.95.3	Member Data Documentation	439
7.96	spot::saba Class Reference	439

7.96.1 Detailed Description . . . . .	441
7.96.2 Constructor & Destructor Documentation . . . . .	441
7.96.3 Member Function Documentation . . . . .	441
7.96.4 Member Data Documentation . . . . .	443
7.97 spot::saba_complement_tgba Class Reference . . . . .	443
7.97.1 Detailed Description . . . . .	445
7.97.2 Constructor & Destructor Documentation . . . . .	446
7.97.3 Member Function Documentation . . . . .	446
7.97.4 Member Data Documentation . . . . .	447
7.98 spot::saba_reachable_iterator Class Reference . . . . .	447
7.98.1 Detailed Description . . . . .	450
7.98.2 Member Typedef Documentation . . . . .	450
7.98.3 Constructor & Destructor Documentation . . . . .	450
7.98.4 Member Function Documentation . . . . .	451
7.98.5 Member Data Documentation . . . . .	452
7.99 spot::saba_reachable_iterator_breadth_first Class Reference . . . . .	453
7.99.1 Detailed Description . . . . .	455
7.99.2 Member Typedef Documentation . . . . .	455
7.99.3 Constructor & Destructor Documentation . . . . .	455
7.99.4 Member Function Documentation . . . . .	455
7.99.5 Member Data Documentation . . . . .	457
7.100spot::saba_reachable_iterator_depth_first Class Reference . . . . .	458
7.100.1 Detailed Description . . . . .	460
7.100.2 Member Typedef Documentation . . . . .	460
7.100.3 Constructor & Destructor Documentation . . . . .	460
7.100.4 Member Function Documentation . . . . .	460
7.100.5 Member Data Documentation . . . . .	462
7.101spot::saba_state Class Reference . . . . .	463
7.101.1 Detailed Description . . . . .	463
7.101.2 Constructor & Destructor Documentation . . . . .	463
7.101.3 Member Function Documentation . . . . .	463
7.102spot::saba_state_conjunction Class Reference . . . . .	464
7.102.1 Detailed Description . . . . .	466
7.102.2 Constructor & Destructor Documentation . . . . .	466
7.102.3 Member Function Documentation . . . . .	466
7.103spot::saba_state_ptr_equal Struct Reference . . . . .	467

7.103.1 Detailed Description . . . . .	467
7.103.2 Member Function Documentation . . . . .	468
7.104spot::saba_state_ptr_hash Struct Reference . . . . .	468
7.104.1 Detailed Description . . . . .	468
7.104.2 Member Function Documentation . . . . .	468
7.105spot::saba_state_ptr_less_than Struct Reference . . . . .	469
7.105.1 Detailed Description . . . . .	469
7.105.2 Member Function Documentation . . . . .	469
7.106spot::saba_state_shared_ptr_equal Struct Reference . . . . .	469
7.106.1 Detailed Description . . . . .	469
7.106.2 Member Function Documentation . . . . .	470
7.107spot::saba_state_shared_ptr_hash Struct Reference . . . . .	470
7.107.1 Detailed Description . . . . .	470
7.107.2 Member Function Documentation . . . . .	470
7.108spot::saba_state_shared_ptr_less_than Struct Reference . . . . .	471
7.108.1 Detailed Description . . . . .	471
7.108.2 Member Function Documentation . . . . .	471
7.109spot::saba_succ_iterator Class Reference . . . . .	471
7.109.1 Detailed Description . . . . .	472
7.109.2 Constructor & Destructor Documentation . . . . .	472
7.109.3 Member Function Documentation . . . . .	472
7.110spot::scc_map::scc Struct Reference . . . . .	473
7.110.1 Constructor & Destructor Documentation . . . . .	474
7.110.2 Member Data Documentation . . . . .	474
7.111spot::scc_map Class Reference . . . . .	475
7.111.1 Detailed Description . . . . .	478
7.111.2 Member Typedef Documentation . . . . .	478
7.111.3 Constructor & Destructor Documentation . . . . .	479
7.111.4 Member Function Documentation . . . . .	479
7.111.5 Member Data Documentation . . . . .	482
7.112spot::scc_stack Class Reference . . . . .	483
7.112.1 Member Typedef Documentation . . . . .	483
7.112.2 Member Function Documentation . . . . .	484
7.112.3 Member Data Documentation . . . . .	485
7.113spot::scc_stats Struct Reference . . . . .	485
7.113.1 Member Function Documentation . . . . .	485

7.113.2 Member Data Documentation . . . . .	485
7.114spot::scs_set Struct Reference . . . . .	486
7.114.1 Member Data Documentation . . . . .	487
7.115spot::ltl::simplify_f_g_visitor Class Reference . . . . .	487
7.115.1 Detailed Description . . . . .	490
7.115.2 Member Typedef Documentation . . . . .	490
7.115.3 Constructor & Destructor Documentation . . . . .	490
7.115.4 Member Function Documentation . . . . .	490
7.115.5 Member Data Documentation . . . . .	491
7.116eltly::slice< T, S > Class Template Reference . . . . .	491
7.116.1 Detailed Description . . . . .	492
7.116.2 Constructor & Destructor Documentation . . . . .	492
7.116.3 Member Function Documentation . . . . .	492
7.116.4 Member Data Documentation . . . . .	492
7.117ltly::slice< T, S > Class Template Reference . . . . .	493
7.117.1 Detailed Description . . . . .	493
7.117.2 Constructor & Destructor Documentation . . . . .	493
7.117.3 Member Function Documentation . . . . .	493
7.117.4 Member Data Documentation . . . . .	493
7.118sauty::slice< T, S > Class Template Reference . . . . .	494
7.118.1 Detailed Description . . . . .	494
7.118.2 Constructor & Destructor Documentation . . . . .	494
7.118.3 Member Function Documentation . . . . .	494
7.118.4 Member Data Documentation . . . . .	495
7.119spot::spoiler_node Class Reference . . . . .	495
7.119.1 Detailed Description . . . . .	497
7.119.2 Constructor & Destructor Documentation . . . . .	497
7.119.3 Member Function Documentation . . . . .	497
7.119.4 Member Data Documentation . . . . .	499
7.120spot::spoiler_node_delayed Class Reference . . . . .	499
7.120.1 Detailed Description . . . . .	502
7.120.2 Constructor & Destructor Documentation . . . . .	502
7.120.3 Member Function Documentation . . . . .	502
7.120.4 Member Data Documentation . . . . .	504
7.121sauty::stack< T, S > Class Template Reference . . . . .	505
7.121.1 Member Typedef Documentation . . . . .	506

7.121.2 Constructor & Destructor Documentation . . . . .	506
7.121.3 Member Function Documentation . . . . .	506
7.121.4 Member Data Documentation . . . . .	507
7.122ltlyy::stack< T, S > Class Template Reference . . . . .	507
7.122.1 Member Typedef Documentation . . . . .	508
7.122.2 Constructor & Destructor Documentation . . . . .	508
7.122.3 Member Function Documentation . . . . .	508
7.122.4 Member Data Documentation . . . . .	509
7.123eltlyy::stack< T, S > Class Template Reference . . . . .	510
7.123.1 Member Typedef Documentation . . . . .	510
7.123.2 Constructor & Destructor Documentation . . . . .	510
7.123.3 Member Function Documentation . . . . .	511
7.123.4 Member Data Documentation . . . . .	512
7.124spot::evtgba_explicit::state Struct Reference . . . . .	512
7.124.1 Member Data Documentation . . . . .	512
7.125spot::state Class Reference . . . . .	512
7.125.1 Detailed Description . . . . .	513
7.125.2 Constructor & Destructor Documentation . . . . .	513
7.125.3 Member Function Documentation . . . . .	513
7.126spot::state_bdd Class Reference . . . . .	514
7.126.1 Detailed Description . . . . .	517
7.126.2 Constructor & Destructor Documentation . . . . .	517
7.126.3 Member Function Documentation . . . . .	517
7.126.4 Member Data Documentation . . . . .	518
7.127spot::state_evtgba_explicit Class Reference . . . . .	518
7.127.1 Detailed Description . . . . .	521
7.127.2 Constructor & Destructor Documentation . . . . .	521
7.127.3 Member Function Documentation . . . . .	521
7.127.4 Member Data Documentation . . . . .	522
7.128spot::state_explicit Class Reference . . . . .	522
7.128.1 Detailed Description . . . . .	525
7.128.2 Constructor & Destructor Documentation . . . . .	525
7.128.3 Member Function Documentation . . . . .	525
7.128.4 Member Data Documentation . . . . .	526
7.129spot::state_product Class Reference . . . . .	526
7.129.1 Detailed Description . . . . .	529



7.129.2 Constructor & Destructor Documentation . . . . .	529
7.129.3 Member Function Documentation . . . . .	530
7.129.4 Member Data Documentation . . . . .	531
7.130spot::state_ptr_equal Struct Reference . . . . .	531
7.130.1 Detailed Description . . . . .	531
7.130.2 Member Function Documentation . . . . .	531
7.131spot::state_ptr_hash Struct Reference . . . . .	532
7.131.1 Detailed Description . . . . .	532
7.131.2 Member Function Documentation . . . . .	532
7.132spot::state_ptr_less_than Struct Reference . . . . .	532
7.132.1 Detailed Description . . . . .	532
7.132.2 Member Function Documentation . . . . .	533
7.133spot::state_set Class Reference . . . . .	533
7.133.1 Detailed Description . . . . .	536
7.133.2 Constructor & Destructor Documentation . . . . .	536
7.133.3 Member Function Documentation . . . . .	536
7.133.4 Member Data Documentation . . . . .	537
7.134spot::state_shared_ptr_equal Struct Reference . . . . .	537
7.134.1 Detailed Description . . . . .	538
7.134.2 Member Function Documentation . . . . .	538
7.135spot::state_shared_ptr_hash Struct Reference . . . . .	538
7.135.1 Detailed Description . . . . .	538
7.135.2 Member Function Documentation . . . . .	538
7.136spot::state_shared_ptr_less_than Struct Reference . . . . .	539
7.136.1 Detailed Description . . . . .	539
7.136.2 Member Function Documentation . . . . .	539
7.137spot::state_union Class Reference . . . . .	539
7.137.1 Detailed Description . . . . .	542
7.137.2 Constructor & Destructor Documentation . . . . .	542
7.137.3 Member Function Documentation . . . . .	543
7.137.4 Member Data Documentation . . . . .	544
7.138spot::tgba_run::step Struct Reference . . . . .	544
7.138.1 Member Data Documentation . . . . .	545
7.139spot::string_hash Struct Reference . . . . .	545
7.139.1 Detailed Description . . . . .	545
7.139.2 Member Function Documentation . . . . .	545

7.140	<a href="#">spot::ltl::succ_iterator Class Reference</a>	546
7.140.1	<a href="#">Constructor &amp; Destructor Documentation</a>	546
7.140.2	<a href="#">Member Function Documentation</a>	546
7.140.3	<a href="#">Member Data Documentation</a>	546
7.141	<a href="#">spot::couvreur99_check_shy::successor Struct Reference</a>	547
7.141.1	<a href="#">Constructor &amp; Destructor Documentation</a>	547
7.141.2	<a href="#">Member Data Documentation</a>	548
7.142	<a href="#">spot::symbol Class Reference</a>	548
7.142.1	<a href="#">Member Typedef Documentation</a>	549
7.142.2	<a href="#">Constructor &amp; Destructor Documentation</a>	549
7.142.3	<a href="#">Member Function Documentation</a>	549
7.142.4	<a href="#">Member Data Documentation</a>	550
7.143	<a href="#">spot::taa_succ_iterator Class Reference</a>	550
7.143.1	<a href="#">Member Typedef Documentation</a>	553
7.143.2	<a href="#">Constructor &amp; Destructor Documentation</a>	554
7.143.3	<a href="#">Member Function Documentation</a>	554
7.143.4	<a href="#">Member Data Documentation</a>	555
7.144	<a href="#">spot::taa_tgba Class Reference</a>	556
7.144.1	<a href="#">Detailed Description</a>	560
7.144.2	<a href="#">Member Typedef Documentation</a>	560
7.144.3	<a href="#">Constructor &amp; Destructor Documentation</a>	561
7.144.4	<a href="#">Member Function Documentation</a>	561
7.144.5	<a href="#">Member Data Documentation</a>	564
7.145	<a href="#">spot::taa_tgba_formula Class Reference</a>	565
7.145.1	<a href="#">Member Typedef Documentation</a>	569
7.145.2	<a href="#">Constructor &amp; Destructor Documentation</a>	570
7.145.3	<a href="#">Member Function Documentation</a>	570
7.145.4	<a href="#">Member Data Documentation</a>	575
7.146	<a href="#">spot::taa_tgba_labelled&lt; label, label_hash &gt; Class Template Reference</a>	576
7.146.1	<a href="#">Detailed Description</a>	580
7.146.2	<a href="#">Member Typedef Documentation</a>	581
7.146.3	<a href="#">Constructor &amp; Destructor Documentation</a>	581
7.146.4	<a href="#">Member Function Documentation</a>	581
7.146.5	<a href="#">Member Data Documentation</a>	587
7.147	<a href="#">spot::taa_tgba_string Class Reference</a>	588
7.147.1	<a href="#">Member Typedef Documentation</a>	591

7.147.2 Constructor & Destructor Documentation . . . . .	592
7.147.3 Member Function Documentation . . . . .	592
7.147.4 Member Data Documentation . . . . .	597
7.148spot::tgba Class Reference . . . . .	598
7.148.1 Detailed Description . . . . .	601
7.148.2 Constructor & Destructor Documentation . . . . .	602
7.148.3 Member Function Documentation . . . . .	602
7.148.4 Member Data Documentation . . . . .	605
7.149spot::tgba_bdd_concrete Class Reference . . . . .	606
7.149.1 Detailed Description . . . . .	610
7.149.2 Constructor & Destructor Documentation . . . . .	610
7.149.3 Member Function Documentation . . . . .	611
7.149.4 Member Data Documentation . . . . .	615
7.150spot::tgba_bdd_concrete_factory Class Reference . . . . .	615
7.150.1 Detailed Description . . . . .	618
7.150.2 Member Typedef Documentation . . . . .	618
7.150.3 Constructor & Destructor Documentation . . . . .	618
7.150.4 Member Function Documentation . . . . .	618
7.150.5 Member Data Documentation . . . . .	620
7.151spot::tgba_bdd_core_data Struct Reference . . . . .	620
7.151.1 Detailed Description . . . . .	623
7.151.2 Constructor & Destructor Documentation . . . . .	623
7.151.3 Member Function Documentation . . . . .	623
7.151.4 Member Data Documentation . . . . .	624
7.152spot::tgba_bdd_factory Class Reference . . . . .	627
7.152.1 Detailed Description . . . . .	628
7.152.2 Constructor & Destructor Documentation . . . . .	628
7.152.3 Member Function Documentation . . . . .	628
7.153spot::tgba_explicit Class Reference . . . . .	628
7.153.1 Detailed Description . . . . .	632
7.153.2 Member Typedef Documentation . . . . .	632
7.153.3 Constructor & Destructor Documentation . . . . .	633
7.153.4 Member Function Documentation . . . . .	633
7.153.5 Member Data Documentation . . . . .	637
7.154spot::tgba_explicit_formula Class Reference . . . . .	638
7.154.1 Member Typedef Documentation . . . . .	643

7.154.2 Constructor & Destructor Documentation . . . . .	643
7.154.3 Member Function Documentation . . . . .	643
7.154.4 Member Data Documentation . . . . .	649
7.155spot::tgba_explicit_labelled< label, label_hash > Class Template Reference . . . . .	650
7.155.1 Detailed Description . . . . .	655
7.155.2 Member Typedef Documentation . . . . .	655
7.155.3 Constructor & Destructor Documentation . . . . .	655
7.155.4 Member Function Documentation . . . . .	655
7.155.5 Member Data Documentation . . . . .	661
7.156spot::tgba_explicit_string Class Reference . . . . .	662
7.156.1 Member Typedef Documentation . . . . .	667
7.156.2 Constructor & Destructor Documentation . . . . .	667
7.156.3 Member Function Documentation . . . . .	667
7.156.4 Member Data Documentation . . . . .	673
7.157spot::tgba_explicit_succ_iterator Class Reference . . . . .	674
7.157.1 Detailed Description . . . . .	677
7.157.2 Constructor & Destructor Documentation . . . . .	677
7.157.3 Member Function Documentation . . . . .	677
7.157.4 Member Data Documentation . . . . .	678
7.158spot::tgba_kv_complement Class Reference . . . . .	679
7.158.1 Detailed Description . . . . .	683
7.158.2 Constructor & Destructor Documentation . . . . .	683
7.158.3 Member Function Documentation . . . . .	683
7.158.4 Member Data Documentation . . . . .	686
7.159spot::tgba_product Class Reference . . . . .	687
7.159.1 Detailed Description . . . . .	691
7.159.2 Constructor & Destructor Documentation . . . . .	691
7.159.3 Member Function Documentation . . . . .	691
7.159.4 Member Data Documentation . . . . .	695
7.160spot::tgba_reachable_iterator Class Reference . . . . .	695
7.160.1 Detailed Description . . . . .	698
7.160.2 Member Typedef Documentation . . . . .	698
7.160.3 Constructor & Destructor Documentation . . . . .	698
7.160.4 Member Function Documentation . . . . .	699
7.160.5 Member Data Documentation . . . . .	700
7.161spot::tgba_reachable_iterator_breadth_first Class Reference . . . . .	700

7.161.1 Detailed Description . . . . .	703
7.161.2 Member Typedef Documentation . . . . .	703
7.161.3 Constructor & Destructor Documentation . . . . .	703
7.161.4 Member Function Documentation . . . . .	703
7.161.5 Member Data Documentation . . . . .	705
7.162spot::tgba_reachable_iterator_depth_first Class Reference . . . . .	705
7.162.1 Detailed Description . . . . .	708
7.162.2 Member Typedef Documentation . . . . .	708
7.162.3 Constructor & Destructor Documentation . . . . .	708
7.162.4 Member Function Documentation . . . . .	708
7.162.5 Member Data Documentation . . . . .	710
7.163spot::tgba_reduc Class Reference . . . . .	710
7.163.1 Detailed Description . . . . .	716
7.163.2 Member Typedef Documentation . . . . .	716
7.163.3 Constructor & Destructor Documentation . . . . .	717
7.163.4 Member Function Documentation . . . . .	717
7.163.5 Member Data Documentation . . . . .	726
7.164spot::tgba_run Struct Reference . . . . .	728
7.164.1 Detailed Description . . . . .	728
7.164.2 Member Typedef Documentation . . . . .	728
7.164.3 Constructor & Destructor Documentation . . . . .	729
7.164.4 Member Function Documentation . . . . .	729
7.164.5 Member Data Documentation . . . . .	729
7.165spot::tgba_run_dotty_decorator Class Reference . . . . .	729
7.165.1 Detailed Description . . . . .	732
7.165.2 Member Typedef Documentation . . . . .	732
7.165.3 Constructor & Destructor Documentation . . . . .	732
7.165.4 Member Function Documentation . . . . .	732
7.165.5 Member Data Documentation . . . . .	733
7.166spot::tgba_safracomplement Class Reference . . . . .	734
7.166.1 Detailed Description . . . . .	736
7.166.2 Constructor & Destructor Documentation . . . . .	737
7.166.3 Member Function Documentation . . . . .	737
7.166.4 Member Data Documentation . . . . .	740
7.167spot::tgba_sba_proxy Class Reference . . . . .	741
7.167.1 Detailed Description . . . . .	744

7.167.2 Member Typedef Documentation . . . . .	744
7.167.3 Constructor & Destructor Documentation . . . . .	744
7.167.4 Member Function Documentation . . . . .	744
7.167.5 Member Data Documentation . . . . .	747
7.168spot::tgba_scc Class Reference . . . . .	748
7.168.1 Detailed Description . . . . .	750
7.168.2 Constructor & Destructor Documentation . . . . .	751
7.168.3 Member Function Documentation . . . . .	751
7.168.4 Member Data Documentation . . . . .	754
7.169spot::tgba_sgba_proxy Class Reference . . . . .	754
7.169.1 Detailed Description . . . . .	758
7.169.2 Constructor & Destructor Documentation . . . . .	758
7.169.3 Member Function Documentation . . . . .	758
7.169.4 Member Data Documentation . . . . .	761
7.170spot::tgba_statistics Struct Reference . . . . .	762
7.170.1 Member Function Documentation . . . . .	762
7.170.2 Member Data Documentation . . . . .	762
7.171spot::tgba_succ_iterator Class Reference . . . . .	762
7.171.1 Detailed Description . . . . .	763
7.171.2 Constructor & Destructor Documentation . . . . .	764
7.171.3 Member Function Documentation . . . . .	764
7.172spot::tgba_succ_iterator_concrete Class Reference . . . . .	765
7.172.1 Detailed Description . . . . .	768
7.172.2 Constructor & Destructor Documentation . . . . .	768
7.172.3 Member Function Documentation . . . . .	769
7.172.4 Member Data Documentation . . . . .	770
7.173spot::tgba_succ_iterator_product Class Reference . . . . .	771
7.173.1 Detailed Description . . . . .	773
7.173.2 Constructor & Destructor Documentation . . . . .	773
7.173.3 Member Function Documentation . . . . .	774
7.173.4 Friends And Related Function Documentation . . . . .	775
7.173.5 Member Data Documentation . . . . .	775
7.174spot::tgba_succ_iterator_union Class Reference . . . . .	776
7.174.1 Detailed Description . . . . .	779
7.174.2 Constructor & Destructor Documentation . . . . .	779
7.174.3 Member Function Documentation . . . . .	779

7.174.4 Friends And Related Function Documentation . . . . .	781
7.174.5 Member Data Documentation . . . . .	781
7.175spot::tgba_tba_proxy Class Reference . . . . .	782
7.175.1 Detailed Description . . . . .	785
7.175.2 Member Typedef Documentation . . . . .	785
7.175.3 Constructor & Destructor Documentation . . . . .	785
7.175.4 Member Function Documentation . . . . .	785
7.175.5 Member Data Documentation . . . . .	789
7.176spot::tgba_union Class Reference . . . . .	789
7.176.1 Detailed Description . . . . .	793
7.176.2 Constructor & Destructor Documentation . . . . .	793
7.176.3 Member Function Documentation . . . . .	794
7.176.4 Member Data Documentation . . . . .	797
7.177spot::time_info Struct Reference . . . . .	798
7.177.1 Detailed Description . . . . .	798
7.177.2 Constructor & Destructor Documentation . . . . .	798
7.177.3 Member Data Documentation . . . . .	799
7.178spot::timer Class Reference . . . . .	799
7.178.1 Detailed Description . . . . .	801
7.178.2 Constructor & Destructor Documentation . . . . .	801
7.178.3 Member Function Documentation . . . . .	801
7.178.4 Member Data Documentation . . . . .	802
7.179spot::timer_map Class Reference . . . . .	802
7.179.1 Detailed Description . . . . .	803
7.179.2 Member Typedef Documentation . . . . .	803
7.179.3 Member Function Documentation . . . . .	803
7.179.4 Member Data Documentation . . . . .	805
7.180spot::couvreur99_check_shy::todo_item Struct Reference . . . . .	805
7.180.1 Constructor & Destructor Documentation . . . . .	806
7.180.2 Member Data Documentation . . . . .	806
7.181spot::tgba_explicit::transition Struct Reference . . . . .	806
7.181.1 Detailed Description . . . . .	807
7.181.2 Member Data Documentation . . . . .	807
7.182spot::taa_tgba::transition Struct Reference . . . . .	808
7.182.1 Detailed Description . . . . .	809
7.182.2 Member Data Documentation . . . . .	809

7.183spot::ltl::nfa::transition Struct Reference . . . . .	809
7.183.1 Detailed Description . . . . .	810
7.183.2 Member Data Documentation . . . . .	810
7.184spot::evtgba_explicit::transition Struct Reference . . . . .	811
7.184.1 Detailed Description . . . . .	812
7.184.2 Member Data Documentation . . . . .	812
7.185spot::ltl::automatop::tripleicmp Struct Reference . . . . .	812
7.185.1 Detailed Description . . . . .	812
7.185.2 Member Function Documentation . . . . .	812
7.186spot::ltl::unabbreviate_logic_visitor Class Reference . . . . .	813
7.186.1 Detailed Description . . . . .	815
7.186.2 Member Typedef Documentation . . . . .	815
7.186.3 Constructor & Destructor Documentation . . . . .	815
7.186.4 Member Function Documentation . . . . .	815
7.186.5 Member Data Documentation . . . . .	816
7.187spot::ltl::unabbreviate_ltl_visitor Class Reference . . . . .	816
7.187.1 Detailed Description . . . . .	819
7.187.2 Member Typedef Documentation . . . . .	819
7.187.3 Constructor & Destructor Documentation . . . . .	819
7.187.4 Member Function Documentation . . . . .	819
7.187.5 Member Data Documentation . . . . .	820
7.188spot::ltl::unop Class Reference . . . . .	820
7.188.1 Detailed Description . . . . .	824
7.188.2 Member Typedef Documentation . . . . .	824
7.188.3 Member Enumeration Documentation . . . . .	824
7.188.4 Constructor & Destructor Documentation . . . . .	825
7.188.5 Member Function Documentation . . . . .	825
7.188.6 Member Data Documentation . . . . .	827
7.189spot::unsigned_statistics Struct Reference . . . . .	827
7.189.1 Member Typedef Documentation . . . . .	829
7.189.2 Constructor & Destructor Documentation . . . . .	829
7.189.3 Member Function Documentation . . . . .	829
7.189.4 Member Data Documentation . . . . .	829
7.190spot::unsigned_statistics_copy Class Reference . . . . .	829
7.190.1 Detailed Description . . . . .	830
7.190.2 Member Typedef Documentation . . . . .	830



7.190.3 Constructor & Destructor Documentation . . . . .	830
7.190.4 Member Function Documentation . . . . .	830
7.190.5 Member Data Documentation . . . . .	831
7.191 spot::ltl::visitor Struct Reference . . . . .	831
7.191.1 Detailed Description . . . . .	832
7.191.2 Constructor & Destructor Documentation . . . . .	833
7.191.3 Member Function Documentation . . . . .	833
7.192 spot::weight Class Reference . . . . .	833
7.192.1 Detailed Description . . . . .	834
7.192.2 Member Typedef Documentation . . . . .	834
7.192.3 Constructor & Destructor Documentation . . . . .	835
7.192.4 Member Function Documentation . . . . .	835
7.192.5 Friends And Related Function Documentation . . . . .	835
7.192.6 Member Data Documentation . . . . .	836
<b>8 File Documentation</b>	<b>836</b>
8.1 mainpage.dox File Reference . . . . .	836
8.2 gspn/common.hh File Reference . . . . .	836
8.3 nips/common.hh File Reference . . . . .	837
8.4 gspn/gspn.hh File Reference . . . . .	837
8.5 gspn/ssp.hh File Reference . . . . .	838
8.6 nips/nips.hh File Reference . . . . .	839
8.6.1 Typedef Documentation . . . . .	840
8.7 eltlparse/location.hh File Reference . . . . .	841
8.8 ltlparse/location.hh File Reference . . . . .	842
8.9 sautparse/location.hh File Reference . . . . .	843
8.10 eltlparse/position.hh File Reference . . . . .	844
8.11 ltlparse/position.hh File Reference . . . . .	845
8.12 sautparse/position.hh File Reference . . . . .	846
8.13 eltlparse/public.hh File Reference . . . . .	847
8.14 evtgbaparse/public.hh File Reference . . . . .	848
8.15 ltlparse/public.hh File Reference . . . . .	849
8.16 tgba/public.hh File Reference . . . . .	851
8.17 tgbaparse/public.hh File Reference . . . . .	852
8.18 eltlparse/stack.hh File Reference . . . . .	853
8.19 ltlparse/stack.hh File Reference . . . . .	853
8.20 sautparse/stack.hh File Reference . . . . .	854

8.21	<a href="#">evtgba/evtgba.hh File Reference</a>	854
8.22	<a href="#">evtgba/evtgbaiter.hh File Reference</a>	855
8.23	<a href="#">evtgba/explicit.hh File Reference</a>	856
8.24	<a href="#">evtgba/product.hh File Reference</a>	857
8.25	<a href="#">evtgba/symbol.hh File Reference</a>	857
8.26	<a href="#">evtgbalgos/dotty.hh File Reference</a>	858
8.27	<a href="#">ltlvisit/dotty.hh File Reference</a>	859
8.28	<a href="#">tgbaalgos/dotty.hh File Reference</a>	860
8.29	<a href="#">evtgbalgos/reachiter.hh File Reference</a>	860
8.30	<a href="#">tgbaalgos/reachiter.hh File Reference</a>	861
8.31	<a href="#">evtgbalgos/save.hh File Reference</a>	862
8.32	<a href="#">tgbaalgos/save.hh File Reference</a>	863
8.33	<a href="#">evtgbalgos/tgba2evtgba.hh File Reference</a>	864
8.34	<a href="#">kripke/fairkripke.hh File Reference</a>	864
8.35	<a href="#">kripke/kripke.hh File Reference</a>	865
8.36	<a href="#">ltlast/allnodes.hh File Reference</a>	865
8.36.1	<a href="#">Detailed Description</a>	866
8.37	<a href="#">ltlast/atomic_prop.hh File Reference</a>	866
8.37.1	<a href="#">Detailed Description</a>	867
8.38	<a href="#">ltlast/automatop.hh File Reference</a>	867
8.38.1	<a href="#">Detailed Description</a>	868
8.39	<a href="#">ltlast/binop.hh File Reference</a>	868
8.39.1	<a href="#">Detailed Description</a>	869
8.40	<a href="#">ltlast/constant.hh File Reference</a>	869
8.40.1	<a href="#">Detailed Description</a>	870
8.41	<a href="#">ltlast/formula.hh File Reference</a>	870
8.41.1	<a href="#">Detailed Description</a>	871
8.42	<a href="#">ltlast/formula_tree.hh File Reference</a>	871
8.42.1	<a href="#">Detailed Description</a>	873
8.43	<a href="#">ltlast/multop.hh File Reference</a>	873
8.43.1	<a href="#">Detailed Description</a>	874
8.44	<a href="#">ltlast/nfa.hh File Reference</a>	874
8.44.1	<a href="#">Detailed Description</a>	875
8.45	<a href="#">ltlast/predecl.hh File Reference</a>	875
8.45.1	<a href="#">Detailed Description</a>	875
8.46	<a href="#">ltlast/refformula.hh File Reference</a>	875

8.46.1 Detailed Description . . . . .	876
8.47 Itlast/unop.hh File Reference . . . . .	876
8.47.1 Detailed Description . . . . .	877
8.48 Itlast/visitor.hh File Reference . . . . .	877
8.48.1 Detailed Description . . . . .	878
8.49 Itlenv/declenv.hh File Reference . . . . .	878
8.50 Itlenv/defaultenv.hh File Reference . . . . .	879
8.51 Itlenv/environment.hh File Reference . . . . .	880
8.52 Itlparse/Itlfile.hh File Reference . . . . .	880
8.53 Itlvisit/apcollect.hh File Reference . . . . .	881
8.54 Itlvisit/basicreduce.hh File Reference . . . . .	882
8.55 Itlvisit/clone.hh File Reference . . . . .	883
8.56 Itlvisit/contain.hh File Reference . . . . .	883
8.57 Itlvisit/destroy.hh File Reference . . . . .	884
8.58 Itlvisit/dump.hh File Reference . . . . .	885
8.59 Itlvisit/length.hh File Reference . . . . .	886
8.60 Itlvisit/lunabbrev.hh File Reference . . . . .	886
8.61 Itlvisit/venoform.hh File Reference . . . . .	887
8.62 Itlvisit/postfix.hh File Reference . . . . .	888
8.63 Itlvisit/randomltl.hh File Reference . . . . .	889
8.64 Itlvisit/reduce.hh File Reference . . . . .	889
8.65 Itlvisit/simpfg.hh File Reference . . . . .	891
8.66 Itlvisit/syntimpl.hh File Reference . . . . .	891
8.67 Itlvisit/tostring.hh File Reference . . . . .	892
8.68 Itlvisit/tunabbrev.hh File Reference . . . . .	893
8.69 misc/bareword.hh File Reference . . . . .	894
8.70 misc/bddalloc.hh File Reference . . . . .	894
8.71 misc/bddlt.hh File Reference . . . . .	895
8.72 misc/bddop.hh File Reference . . . . .	895
8.73 misc/escape.hh File Reference . . . . .	896
8.74 misc/freelist.hh File Reference . . . . .	897
8.75 misc/hash.hh File Reference . . . . .	897
8.76 misc/hashfunc.hh File Reference . . . . .	898
8.77 misc/ltstr.hh File Reference . . . . .	899
8.78 misc/memusage.hh File Reference . . . . .	900
8.79 misc/minato.hh File Reference . . . . .	900

8.80 misc/modgray.hh File Reference . . . . .	900
8.81 misc/optionmap.hh File Reference . . . . .	901
8.82 misc/random.hh File Reference . . . . .	901
8.83 misc/timer.hh File Reference . . . . .	902
8.84 misc/version.hh File Reference . . . . .	903
8.85 saba/explicitstateconjunction.hh File Reference . . . . .	903
8.86 saba/saba.hh File Reference . . . . .	904
8.87 saba/sabacomplementtgba.hh File Reference . . . . .	905
8.88 saba/sabastate.hh File Reference . . . . .	906
8.89 saba/sabasucciter.hh File Reference . . . . .	908
8.90 sabaalgos/sabadotty.hh File Reference . . . . .	908
8.91 sabaalgos/sabareachiter.hh File Reference . . . . .	909
8.92 tgba/bdddict.hh File Reference . . . . .	910
8.93 tgba/bddprint.hh File Reference . . . . .	911
8.94 tgba/formula2bdd.hh File Reference . . . . .	912
8.95 tgba/futurecondcol.hh File Reference . . . . .	913
8.96 tgba/state.hh File Reference . . . . .	913
8.97 tgba/statebdd.hh File Reference . . . . .	915
8.98 tgba/succiter.hh File Reference . . . . .	915
8.99 tgba/succiterconcrete.hh File Reference . . . . .	916
8.100tgba/taatgba.hh File Reference . . . . .	917
8.101tgba/tgba.hh File Reference . . . . .	918
8.102tgba/tgababddconcrete.hh File Reference . . . . .	919
8.103tgba/tgababddconcretefactory.hh File Reference . . . . .	920
8.104tgba/tgababddconcreteproduct.hh File Reference . . . . .	920
8.105tgba/tgababddcoredata.hh File Reference . . . . .	921
8.106tgba/tgababddfactory.hh File Reference . . . . .	922
8.107tgba/tgbaexplicit.hh File Reference . . . . .	923
8.108tgba/tgbakvcomplement.hh File Reference . . . . .	924
8.109tgba/tgbaproduct.hh File Reference . . . . .	925
8.110tgba/tgbareduc.hh File Reference . . . . .	926
8.111tgba/tgbsafracomplement.hh File Reference . . . . .	927
8.111.1 Define Documentation . . . . .	928
8.112tgba/tgbascc.hh File Reference . . . . .	928
8.113tgba/tgbasgba.hh File Reference . . . . .	929
8.114tgba/tgbatba.hh File Reference . . . . .	930

8.115tgba/tgbaunion.hh File Reference . . . . .	931
8.116tgbaalgos/bfssteps.hh File Reference . . . . .	932
8.117tgbaalgos/cutsc.c File Reference . . . . .	932
8.118tgbaalgos/dottydec.hh File Reference . . . . .	933
8.119tgbaalgos/duexp.hh File Reference . . . . .	934
8.120tgbaalgos/eltl2tgba_lacim.hh File Reference . . . . .	935
8.121tgbaalgos/emptiness.hh File Reference . . . . .	936
8.122tgbaalgos/emptiness_stats.hh File Reference . . . . .	937
8.123tgbaalgos/gtec/ce.hh File Reference . . . . .	938
8.124tgbaalgos/gtec/explsc.c File Reference . . . . .	939
8.125tgbaalgos/gtec/gtec.hh File Reference . . . . .	939
8.126tgbaalgos/gtec/nsheap.hh File Reference . . . . .	940
8.127tgbaalgos/gtec/scstack.hh File Reference . . . . .	941
8.128tgbaalgos/gtec/status.hh File Reference . . . . .	942
8.129tgbaalgos/gv04.hh File Reference . . . . .	943
8.130tgbaalgos/lbtt.hh File Reference . . . . .	943
8.131tgbaalgos/ltl2taa.hh File Reference . . . . .	944
8.132tgbaalgos/ltl2tgba_fm.hh File Reference . . . . .	945
8.133tgbaalgos/ltl2tgba_lacim.hh File Reference . . . . .	946
8.134tgbaalgos/magic.hh File Reference . . . . .	947
8.135tgbaalgos/neverclaim.hh File Reference . . . . .	948
8.136tgbaalgos/powerset.hh File Reference . . . . .	949
8.137tgbaalgos/projrun.hh File Reference . . . . .	949
8.138tgbaalgos/randomgraph.hh File Reference . . . . .	950
8.139tgbaalgos/reducerun.hh File Reference . . . . .	950
8.140tgbaalgos/reductgba_sim.hh File Reference . . . . .	951
8.141tgbaalgos/replayrun.hh File Reference . . . . .	953
8.142tgbaalgos/rundotdec.hh File Reference . . . . .	953
8.143tgbaalgos/sc.c File Reference . . . . .	954
8.144tgbaalgos/scfilter.hh File Reference . . . . .	955
8.145tgbaalgos/se05.hh File Reference . . . . .	955
8.146tgbaalgos/stats.hh File Reference . . . . .	956
8.147tgbaalgos/tau03.hh File Reference . . . . .	957
8.148tgbaalgos/tau03opt.hh File Reference . . . . .	957
8.149tgbaalgos/weight.hh File Reference . . . . .	958

# 1 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 Deprecated List

Member [spot::ltl::clone](#)(const formula \*f) Use `f->clone()` instead.

Member [spot::ltl::destroy](#)(const formula \*f) Use `f->destroy()` instead.

# 3 Bug List

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 `courcoubetis.92.fmsd`.

Member [spot::get\\_delayed\\_relation\\_simulation](#)(const tgba \*a, std::ostream &os, int opt=-1) Does not work for generalized automata.

## 4 Module Documentation

### 4.1 LTL formulae

#### Modules

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

#### 4.1.1 Detailed Description

This module gathers types and definitions related to LTL formulae.

### 4.2 SABA (State-based Alternating Büchi Automata)

#### Classes

- class [spot::saba\\_complement\\_tgba](#)  
*Complement a TGBA and produce a SABA.  
The original TGBA is transformed into a States-based Büchi Automaton.*

#### Modules

- [Essential SABA types](#)

#### 4.2.1 Detailed Description

Spot was centered around non-deterministic [TGBA \(Transition-based Generalized Büchi Automata\)](#). Alternating automata are an extension to non-deterministic automata, and are presented with [spot::saba](#). This type and its cousins are listed [here](#). This is an abstract interface.

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

#### Classes

- class [spot::future\\_conditions\\_collector](#)  
*Wrap a tgba to offer information about upcoming conditions.  
This class is a [spot::tgba](#) wrapper that simply add a new method, [future\\_conditions\(\)](#), to any [spot::tgba](#).*
- class [spot::tgba\\_scc](#)  
*Wrap a tgba to offer information about strongly connected components.  
This class is a [spot::tgba](#) wrapper that simply add a new method [scc\\_of\\_state\(\)](#) to retrieve the number of a SCC a state belongs to.*

## Modules

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

### 4.3.1 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](#).

## 4.4 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, bool stack_inclusion=true, bool double_inclusion=false, bool reversed_double_inclusion=false, bool no_decomp=false)`

### 4.4.1 Function Documentation

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

**4.4.1.2** `couvreur99_check * spot::couvreur99\_check\_ssp\_shy (const tgba * ssp_automata, bool stack_inclusion = true, bool double_inclusion = false, bool reversed_double_inclusion = false, bool no_decomp = false)`

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

## 4.5 Input/Output of LTL formulae

### Classes

- class [spot::ltl::ltl\\_file](#)  
*Read LTL formulae from a file, one by one.*
- class [spot::ltl::random\\_ltl](#)



*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.*

### Typedefs

- typedef std::pair< std::string, std::string > [spot::eltl::spair](#)
- typedef std::pair< [eltly::location](#), spair > [spot::eltl::parse\\_error](#)  
A parse diagnostic <location, <file, message>>.
- typedef std::list< parse\_error > [spot::eltl::parse\\_error\\_list](#)  
A list of parser diagnostics, as filled by parse.
- typedef std::pair< [ltly::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::eltl::parse\\_file](#) (const std::string &filename, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
Build a formula from a text file.
- formula \* [spot::eltl::parse\\_string](#) (const std::string &eltl\_string, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
Build a formula from an ECTL string.
- bool [spot::eltl::format\\_parse\\_errors](#) (std::ostream &os, parse\_error\_list &error\_list)  
Format diagnostics produced by [spot::eltl::parse](#).
- formula \* [spot::ltl::parse](#) (const std::string &ltl\_string, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
Build a formula from an LTL string.
- bool [spot::ltl::format\\_parse\\_errors](#) (std::ostream &os, const std::string &ltl\_string, parse\_error\_list &error\_list)  
Format diagnostics produced by [spot::ltl::parse](#).
- std::ostream & [spot::ltl::dotty](#) (std::ostream &os, const formula \*f)  
Write a formula tree using dot's syntax.
- std::ostream & [spot::ltl::dump](#) (std::ostream &os, const formula \*f)  
Dump a formula tree.
- std::ostream & [spot::ltl::to\\_string](#) (const formula \*f, std::ostream &os, bool full\_parent=false)

*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ECTL formulae).*

- `std::string spot::ltl::to_string` (const formula \*f, bool full\_parent=false)

*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ECTL formulae).*

- `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.*

#### 4.5.1 Typedef Documentation

##### 4.5.1.1 `typedef std::pair<ltl::location, std::string> spot::ltl::parse_error`

A parse diagnostic with its location.

##### 4.5.1.2 `typedef std::pair<eltl::location, spair> spot::eltl::parse_error`

A parse diagnostic <location, <file, message>>.

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

A list of parser diagnostics, as filled by parse.

##### 4.5.1.4 `typedef std::list<parse_error> spot::eltl::parse_error_list`

A list of parser diagnostics, as filled by parse.

##### 4.5.1.5 `typedef std::pair<std::string, std::string> spot::eltl::spair`

#### 4.5.2 Function Documentation

##### 4.5.2.1 `std::ostream& spot::ltl::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/>

**4.5.2.2 `std::ostream& spot::ltl::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.

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

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

**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.

**4.5.2.4 `bool spot::eltl::format_parse_errors (std::ostream & os, parse_error_list & error_list)`**

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

**Parameters**

*os* Where diagnostics should be output.

*error\_list* The error list filled by `spot::eltl::parse` while parsing *eltl\_string*.

**Returns**

`true` iff any diagnostic was output.

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

Build a formula from an LTL string.

#### Parameters

*ltl\_string* The string to parse.

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

*env* The environment into which parsing should take place.

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

#### Returns

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

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

#### Warning

This function is not reentrant.

**4.5.2.6** `formula* spot::eltl::parse_file (const std::string & filename, parse_error_list & error_list, environment & env = default_environment::instance(), bool debug = false)`

Build a formula 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.

*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.

#### Warning

This function is not reentrant.

**4.5.2.7** `formula* spot::ltl::parse_string (const std::string & eltl_string, parse_error_list & error_list, environment & env = default_environment::instance(), bool debug = false)`

Build a formula from an ELTL string.

#### Parameters

*eltl\_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 *eltl\_string*, or 0 if the input was unparsable.

#### Warning

This function is not reentrant.

**4.5.2.8** `std::string spot::ltl::to_spin_string (const formula * f)`

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

#### Parameters

*f* The formula to translate.

**4.5.2.9** `std::ostream& spot::ltl::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.

**4.5.2.10** `std::string spot::ltl::to_string (const formula * f, bool full_parent = false)`

Output a formula as a string which is parsable unless the formula contains automaton operators (used in ELTL formulae).

#### Parameters

*f* The formula to translate.

*full\_parent* Whether or not the string should be fully parenthesized.

#### 4.5.2.11 `std::ostream& spot::ltl::to_string (const formula *f, std::ostream & os, bool full_parent = false)`

Output a formula as a string which is parsable unless the formula contains automaton operators (used in ELTL formulae).

##### Parameters

- f* The formula to translate.
- os* The stream where it should be output.
- full\_parent* Whether or not the string should be fully parenthesized.

## 4.6 Essential LTL types

### Classes

- class `spot::ltl::formula`  
*An LTL formula.*  
*The only way you can work with a formula is to build a `spot::ltl::visitor` or `spot::ltl::const_visitor`.*
- struct `spot::ltl::visitor`  
*Formula visitor that can modify the formula.*  
*Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.*
- 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.*

#### 4.6.1 Function Documentation

##### 4.6.1.1 `formula* spot::ltl::clone (const formula *f)`

Clone a formula.

##### Deprecated

Use `f->clone()` instead.

#### 4.6.1.2 void spot::ltl::destroy (const formula \*f)

Destroys a formula.

#### Deprecated

Use `f->destroy()` instead.

## 4.7 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.*  
*The only way you can work with a formula is to build a `spot::ltl::visitor` or `spot::ltl::const_visitor`.*
- class `spot::ltl::multop`  
*Multi-operand operators.*  
*These operators are considered commutative and associative.*
- class `spot::ltl::ref_formula`  
*A reference-counted LTL formula.*
- class `spot::ltl::unop`  
*Unary operators.*

## 4.8 LTL environments

### Classes

- class `spot::ltl::declarative_environment`  
*A declarative environment.*  
*This environment recognizes all atomic propositions that have been previously declared. It will reject other.*
- class `spot::ltl::default_environment`  
*A laxist environment.*  
*This environment recognizes all atomic propositions.*

### 4.8.1 Detailed Description

LTL environment implementations.

## 4.9 Algorithms for LTL formulae

### Modules

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

### 4.10 Derivable visitors

#### Classes

- class [spot::ltl::clone\\_visitor](#)  
*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::formula::clone](#) instead, it is way faster.*
- class [spot::ltl::unabbreviate\\_logic\\_visitor](#)  
*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](#).*
- class [spot::ltl::postfix\\_visitor](#)  
*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.*
- 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.*  
*The rewriting performed on logical operator is the same as the one done by [spot::ltl::unabbreviate\\_logic\\_visitor](#).*

### 4.11 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\\_Containment\\_Checks](#) = 8, [spot::ltl::Reduce\\_Containment\\_Checks\\_Stronger](#) = 16,  
[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. This will rewrite binary operators such as `binop::Implies`, `binop::Equals`, and `binop::Xor`, using only `unop::Not`, `multop::Or`, and `multop::And`.*
- formula \* `spot::ltl::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.*
- formula \* `spot::ltl::reduce` (const formula \*f, int opt=Reduce\_All)  
*Reduce a formula f.*
- formula \* `spot::ltl::simplify_f_g` (const formula \*f)  
*Replace `true`  $\cup$  `f` and `false`  $\cap$  `g` by `F` `f` and `G` `g`.*

### 4.11.1 Enumeration Type Documentation

#### 4.11.1.1 enum `spot::ltl::reduce_options`

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

#### Enumerator:

- 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\_Containment\_Checks*** Tauriainen containment checks.
- Reduce\_Containment\_Checks\_Stronger*** Tauriainen containment checks (stronger version).
- Reduce\_All*** All reductions.

### 4.11.2 Function Documentation

#### 4.11.2.1 formula\* `spot::ltl::basic_reduce` (const formula \*f)

Basic rewritings.

#### 4.11.2.2 formula\* spot::ltl::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.)

#### 4.11.2.3 formula\* spot::ltl::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

#### 4.11.2.4 formula\* spot::ltl::simplify\_f\_g (const formula \*f)

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

Perform the following rewriting (from left to right):

- `true U a = F a`
- `a M true = F a`
- `false R a = G a`
- `a W false = G a`

#### 4.11.2.5 formula\* spot::ltl::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`.

## 4.12 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.*  
*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).*
- bool [spot::ltl::is\\_eventual](#) (const formula \*f)  
*Check whether a formula is a pure eventuality.*  
*Pure eventuality formulae are defined in.*
- bool [spot::ltl::is\\_universal](#) (const formula \*f)  
*Check whether a formula is purely universal.*  
*Purely universal formulae are defined in.*
- bool [spot::ltl::syntactic\\_implication](#) (const formula \*f1, const formula \*f2)  
*Syntactic implication.*  
*This comes from.*
- bool [spot::ltl::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.*

### 4.12.1 Typedef Documentation

#### 4.12.1.1 typedef std::set<atomic\_prop\*, formula\_ptr\_less\_than> spot::ltl::atomic\_prop\_set

Set of atomic propositions.

### 4.12.2 Function Documentation

#### 4.12.2.1 `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.

##### 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.

#### 4.12.2.2 `bool spot::ltl::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.

#### 4.12.2.3 `bool spot::ltl::is_FG (const formula *f)`

Whether a formula starts with FG.

#### 4.12.2.4 `bool spot::ltl::is_GF (const formula *f)`

Whether a formula starts with GF.

**4.12.2.5 bool spot::ltl::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.

**4.12.2.6 int spot::ltl::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.  $a \mid b \mid c$  has length 4, because  $\mid$  is represented once internally).

**4.12.2.7 bool spot::ltl::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}
/// }
///

```

**4.12.2.8 bool spot::ltl::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](#)

## 4.13 Miscellaneous helper algorithms

Whether a word is bare.

### 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*`.  
This is meant to be used as a comparison functor for STL map whose key are of type `const char*`.*
- class [spot::minato\\_isop](#)  
*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.*
- class [spot::loopless\\_modular\\_mixed\\_radix\\_gray\\_code](#)  
*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.*
- class [spot::option\\_map](#)  
*Manage a map of options.  
Each option is defined by a string and is associated to an integer value.*
- 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.*

## Modules

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

## 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.*

### 4.13.1 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.

### 4.13.2 Function Documentation

#### 4.13.2.1 `std::string spot::escape_str (const std::string & str)`

Escape " and \ characters in *str*.

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

Escape " and \ characters in *str*.

#### 4.13.2.3 `bool spot::is_bare_word (const char * str)`

#### 4.13.2.4 `std::string spot::quote_unless_bare_word (const std::string & str)`

Double-quote words that are not bare.

See also

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

#### 4.13.2.5 `const char* spot::version ()`

Return Spot's version.

## 4.14 Hashing functions

### Classes

- struct [spot::ltl::formula\\_ptr\\_hash](#)  
*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*`.*
- struct [spot::ptr\\_hash< T >](#)  
*A hash function for pointers.*
- struct [spot::string\\_hash](#)  
*A hash function for strings.*
- struct [spot::saba\\_state\\_ptr\\_hash](#)  
*Hash Function for `saba_state*`.  
This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `saba_state*`.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_hash](#)  
*Hash Function for `shared_saba_state (shared_ptr<const saba_state*>)`.  
This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `shared_saba_state`.*
- struct [spot::state\\_ptr\\_hash](#)  
*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*`.*
- struct [spot::state\\_shared\\_ptr\\_hash](#)  
*Hash Function for `shared_state (shared_ptr<const state*>)`.  
This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `shared_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.*



#### 4.14.1 Function Documentation

##### 4.14.1.1 `size_t spot::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>

Referenced by `spot::string_hash::operator()`.

##### 4.14.1.2 `size_t spot::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>

## 4.15 Random functions

### Classes

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

### Functions

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

#### 4.15.1 Function Documentation

##### 4.15.1.1 `double spot::bmrnd ()`

Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).  
This uses the polar form of the Box-Muller transform to generate random values.

##### 4.15.1.2 `double spot::drand ()`

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

See also

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

##### 4.15.1.3 `int spot::mrand (int max)`

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

See also

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

##### 4.15.1.4 `double spot::nrnd ()`

Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).  
This uses a polynomial approximation of the inverse cumulated density function from Odeh & Evans, Journal of Applied Statistics, 1974, vol 23, pp 96-97.

##### 4.15.1.5 `int spot::prand (double p)`

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

**Precondition**

$p > 0$

#### 4.15.1.6 `int spot::rrand (int min, int max)`

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

See also

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

#### 4.15.1.7 `void spot::srand (unsigned int seed)`

Reset the seed of the pseudo-random number generator.

See also

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

## 4.16 Essential SABA types

### Classes

- class [spot::explicit\\_state\\_conjunction](#)  
*Basic implementation of [saba\\_state\\_conjunction](#).  
This class provides a basic implementation to iterate over a conjunction of states of a saba.*
- class [spot::saba](#)  
*A State-based Alternating (Generalized) Büchi Automaton.  
Browsing such automaton can be achieved using two functions: `get_init_state`, and `succ_iter`.  
The former returns the initial state while the latter lists the successor states of any state.*
- class [spot::saba\\_state](#)  
*Abstract class for saba states.*
- struct [spot::saba\\_state\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for `saba_state*`.  
This is meant to be used as a comparison functor for STL map whose key are of type `saba_state*`.*
- struct [spot::saba\\_state\\_ptr\\_equal](#)  
*An Equivalence Relation for `saba_state*`.  
This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type `saba_state*`.*
- struct [spot::saba\\_state\\_ptr\\_hash](#)  
*Hash Function for `saba_state*`.  
This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type `saba_state*`.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for `shared_saba_state (shared_ptr<const saba_state*>)`.  
This is meant to be used as a comparison functor for STL map whose key are of type `shared_saba_state`.*

- struct [spot::saba\\_state\\_shared\\_ptr\\_equal](#)  
*An Equivalence Relation for `shared_saba_state` (`shared_ptr<const saba_state*>`).  
This is meant to be used as a comparison functor for Sgi `hash_map` whose key are of type `shared_saba_state`.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_hash](#)  
*Hash Function for `shared_saba_state` (`shared_ptr<const saba_state*>`).  
This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `shared_saba_state`.*
- class [spot::saba\\_state\\_conjunction](#)  
*Iterate over a conjunction of `saba_state`.  
This class provides the basic functionalities required to iterate over a conjunction of states of a saba.*
- class [spot::saba\\_succ\\_iterator](#)  
*Iterate over the successors of a `saba_state`.  
This class provides the basic functionalities required to iterate over the successors of a state of a saba. Since transitions of an alternating automaton are defined as a boolean function with conjunctions (universal) and disjunctions (non-deterministic),.*

## 4.17 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*`.  
This is meant to be used as a comparison functor for STL `map` whose key are of type `state*`.*
- struct [spot::state\\_ptr\\_equal](#)  
*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*`.*
- struct [spot::state\\_ptr\\_hash](#)  
*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*`.*
- struct [spot::state\\_shared\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for `shared_state` (`shared_ptr<const state*>`).  
This is meant to be used as a comparison functor for STL `map` whose key are of type `shared_state`.*
- struct [spot::state\\_shared\\_ptr\\_equal](#)  
*An Equivalence Relation for `shared_state` (`shared_ptr<const state*>`).  
This is meant to be used as a comparison functor for Sgi `hash_map` whose key are of type `shared_state`.*
- struct [spot::state\\_shared\\_ptr\\_hash](#)

Hash Function for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a hash functor for Sgi's `hash_map` whose key are of type `shared_state`.

- class `spot::tgba_succ_iterator`

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.

- class `spot::tgba`

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).

## 4.18 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`

## 4.19 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.

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

### 4.19.1 Function Documentation

#### 4.19.1.1 `tgba_bdd_concrete* spot::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.

## 4.20 TGBA on-the-fly algorithms

### Classes

- class `spot::tgba_kv_complement`  
*Build a complemented automaton.  
 The construction comes from:*
- class `spot::state_product`  
*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.*
- class `spot::tgba_safra_complement`  
*Build a complemented automaton.  
 It creates an automaton that recognizes the negated language of aut.*
- class `spot::tgba_sgba_proxy`  
*Change the labeling-mode of `spot::tgba` on the fly, producing a state-based generalized Büchi automaton.  
 This class acts as a proxy in front of a `spot::tgba`, that should label on states on-the-fly. The result is still a `spot::tgba`, but acceptances conditions are also on states.*
- class `spot::tgba_tba_proxy`  
*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).*
- class `spot::tgba_sba_proxy`  
*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.*
- class `spot::state_union`  
*A state for `spot::tgba_union`.  
 This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.*

## 4.21 Input/Output of TGBA

### Modules

- `Decorating the dot output`

## Typedefs

- `typedef std::pair< tgbayy::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.*  
*The dd argument allows to customize the output in various ways. See [this page](#) for a list of available decorators.*
- `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, bool comments=false)`  
*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_string * spot::tgba_parse (const std::string &filename, tgba_parse_error_list &error_list, bdd_dict *dict, ltl::environment &env=ltl::default_environment::instance(), ltl::environment &envacc=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](#).*

### 4.21.1 Typedef Documentation

#### 4.21.1.1 `typedef std::pair<tgbayy::location, std::string> spot::tgba_parse_error`

A parse diagnostic with its location.

#### 4.21.1.2 `typedef std::list<tgba_parse_error> spot::tgba_parse_error_list`

A list of parser diagnostics, as filled by parse.

### 4.21.2 Function Documentation

#### 4.21.2.1 `std::ostream& spot::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.

#### 4.21.2.2 `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`.

##### 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.

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

Print reachable states in LBTT format.

##### Parameters

*g* The automata to print.

*os* Where to print.

#### 4.21.2.4 `std::ostream& spot::never_claim_reachable (std::ostream & os, const tgba_sba_proxy * g, const ltl::formula * f = 0, bool comments = false)`

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.

*comments* Whether to comment each state of the never clause with the label of the *g* automaton.



**4.21.2.5** `tgba_explicit_string*` `spot::tgba_parse` (`const std::string & filename`,  
`tgba_parse_error_list & error_list`, `bdd_dict * dict`, `ltl::environment & env =`  
`ltl::default_environment::instance()`, `ltl::environment & envacc =`  
`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 of atomic proposition into which parsing should take place.

*envacc* The environment of acceptance conditions 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.

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

Save reachable states in text format.

## 4.22 Translating LTL formulae into TGBA

### Functions

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

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

*This is based on the following paper.*

- `taa_tgba *` `spot::ltl_to_taa` (`const ltl::formula *f`, `bdd_dict *dict`, `bool refined_rules=false`)

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

*This is based on the following.*

- `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.  
This is based on the following paper:

- `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.  
This is based on the following paper:

### 4.22.1 Function Documentation

#### 4.22.1.1 `tgba_bdd_concrete * spot::eltl_to_tgba_lacim` (const ltl::formula \*f, bdd\_dict \*dict)

Build a `spot::tgba_bdd_concrete` from an ETLTL 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*.

#### 4.22.1.2 `taa_tgba * spot::ltl_to_taa` (const ltl::formula \*f, bdd\_dict \*dict, bool refined\_rules = false)

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

This is based on the following.

```
/// @techreport{HUT-TCS-A104,
///   address = {Espoo, Finland},
///   author  = {Heikki Tauriainen},
///   month   = {September},
///   note    = {Doctoral dissertation},
///   number  = {A104},
///   pages   = {xii+229},
```

```

/// title   = {Automata and Linear Temporal Logic: Translations
///           with Transition-Based Acceptance},
/// type    = {Research Report},
/// year    = {2006}
/// }
///

```

### Parameters

*f* The formula to translate into an automaton.

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

*refined\_rules* If this parameter is set, refined rules are used.

### Returns

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

**4.22.1.3** [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.

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*.

### 4.22.1.4 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.

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{\'}ebec {\'}a Montr{\'}eal},
/// 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*.

## 4.23 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.*

## 4.24 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

- const 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.*

#### 4.24.1 Typedef Documentation

##### 4.24.1.1 typedef std::vector<duplicator\_node\*> spot::dn\_v

##### 4.24.1.2 typedef std::vector<const state\*> spot::s\_v

##### 4.24.1.3 typedef std::vector<spoiler\_node\*> spot::sn\_v

### 4.24.2 Enumeration Type Documentation

#### 4.24.2.1 enum spot::reduce\_tgba\_options

Options for reduce.

##### Enumerator:

*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.

### 4.24.3 Function Documentation

#### 4.24.3.1 void spot::free\_relation\_simulation (delayed\_simulation\_relation \* rel)

To free a simulation relation.

#### 4.24.3.2 void spot::free\_relation\_simulation (direct\_simulation\_relation \* rel)

To free a simulation relation.

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

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

##### Bug

Does not work for generalized automata.

#### 4.24.3.4 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  $f$ .

#### 4.24.3.5 `const tgba* spot::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.

## 4.25 Miscellaneous algorithms on TGBA

### Classes

- class `spot::bfs_steps`

*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.*

- 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.*

*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.*

- `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.*



### 4.25.1 Function Documentation

**4.25.1.1** `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 = &lt;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.)

### 4.25.1.2 `tgba_statistics spot::stats_reachable (const tgba * g)`

Compute statistics for an automaton.

#### 4.25.1.3 `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.

#### 4.25.1.4 `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.

#### 4.25.1.5 `tgba_explicit* spot::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.

## 4.26 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`.  
An instance of this class can be passed to `spot::dotty_reachable`.*

## 4.27 Emptiness-checks

### 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`

### Modules

- Emptiness-check algorithms for SSP
- Emptiness-check algorithms
- TGBA runs and supporting functions
- Emptiness-check statistics

### 4.27.1 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 function to 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 find 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.

## 4.28 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_sc05_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*.

### 4.28.1 Function Documentation

#### 4.28.1.1 `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*.

#### 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`

#### 4.28.1.2 `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*.

##### 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](#)

#### 4.28.1.3 `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.

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. ( $\langle \text{Sigma}, Q, \text{delta}, q, F \rangle$  is the automaton to check,  $H$  is an associative array mapping each state to its positive DFS order or 0 if it is dead,  $\text{SCC}$  is and  $\text{ACC}$  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::couvereur99\\_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.

#### 4.28.1.4 `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.

##### 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}
/// }
///

```

#### 4.28.1.5 `emptiness_check* spot::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;
///

```

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
///   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`.

#### 4.28.1.6 `emptiness_check* spot::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](#)

#### 4.28.1.7 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*.

**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;
///     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.

#### 4.28.1.8 emptiness\_check\* spot::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}
/// }
///

```

#### 4.28.1.9 emptiness\_check\* spot::magic\_search (const tgba \* a, option\_map o = option\_map())

Wrapper for the two magic\_search implementations.

This wrapper calls `explicit_magic_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.

#### 4.28.1.10 emptiness\_check\* spot::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.

## 4.29 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.  
If a tgba\_run has been generated on a product, or any other on-the-fly algorithm with tgba operands.,*
- `tgba_run * spot::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.*
- `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`.

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.

#### 4.29.1 Function Documentation

##### 4.29.1.1 `std::ostream& spot::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).

##### 4.29.1.2 `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`.

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

##### 4.29.1.3 `tgba_run* spot::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*.

##### 4.29.1.4 `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`.

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

##### 4.29.1.5 `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).

#### Precondition

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

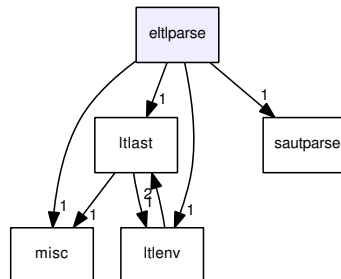
## 4.30 Emptiness-check statistics

#### 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.*

## 5 Directory Documentation

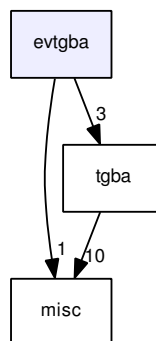
### 5.1 eltlparse/ Directory Reference



#### Files

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

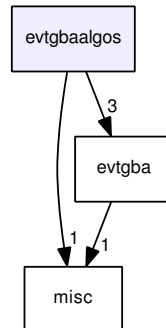
### 5.2 evtgba/ Directory Reference



#### Files

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

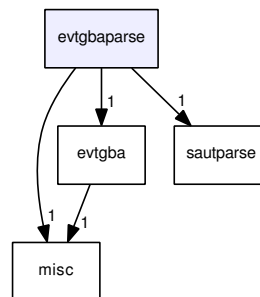
### 5.3 evtgbaalgos/ Directory Reference



#### Files

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

### 5.4 evtgbaparse/ Directory Reference

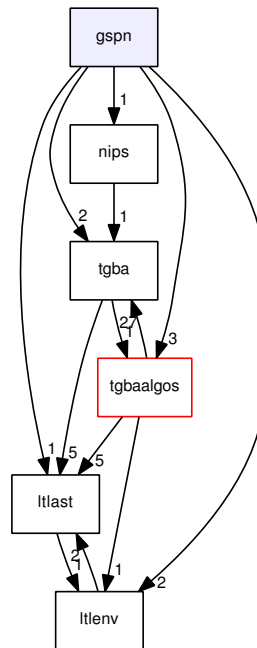


#### Files

- file [public.hh](#)



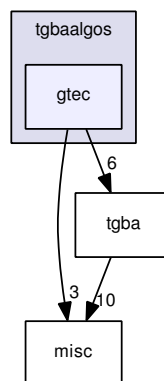
## 5.5 gspn/ Directory Reference



### Files

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

## 5.6 tgbaalgos/gtec/ Directory Reference

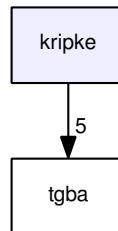


### Files

- file [ce.hh](#)
- file [explscc.hh](#)

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

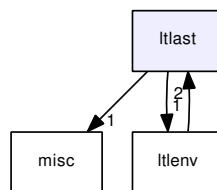
## 5.7 kripke/ Directory Reference



### Files

- file [fairkripke.hh](#)
- file [kripke.hh](#)

## 5.8 Itlast/ Directory Reference

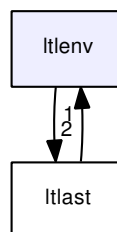


### Files

- file [allnodes.hh](#)  
*Define all LTL node types.*
- file [atomic\\_prop.hh](#)  
*LTL atomic propositions.*
- file [automatop.hh](#)  
*ELTL automaton operators.*
- file [binop.hh](#)  
*LTL binary operators.*
- file [constant.hh](#)  
*LTL constants.*

- file [formula.hh](#)  
*LTL formula interface.*
- file [formula\\_tree.hh](#)  
*Trees representing formulae where atomic propositions are unknown.*
- file [multop.hh](#)  
*LTL multi-operand operators.*
- file [nfa.hh](#)  
*NFA interface used by automatop.*
- file [predecl.hh](#)  
*Predeclare all LTL node types.*
- file [reformula.hh](#)  
*Reference-counted LTL formulae.*
- file [unop.hh](#)  
*LTL unary operators.*
- file [visitor.hh](#)  
*LTL visitor interface.*

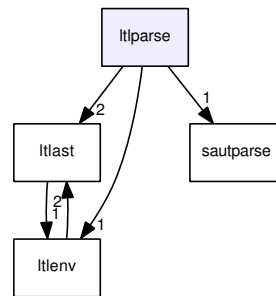
## 5.9 Itlenv/ Directory Reference



### Files

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

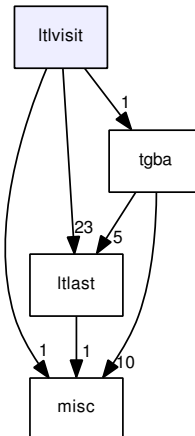
## 5.10 Itlparse/ Directory Reference



### Files

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

## 5.11 Itlvisit/ Directory Reference

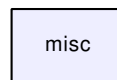


### Files

- file [apcollect.hh](#)
- file [basicreduce.hh](#)
- file [clone.hh](#)
- file [contain.hh](#)
- file [destroy.hh](#)
- file [dotty.hh](#)
- file [dump.hh](#)
- file [length.hh](#)
- file [lunabbrev.hh](#)

- file [neniform.hh](#)
- file [postfix.hh](#)
- file [randomltl.hh](#)
- file [reduce.hh](#)
- file [simpfg.hh](#)
- file [syntimpl.hh](#)
- file [tostring.hh](#)
- file [tunabbrev.hh](#)

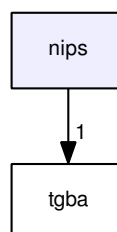
## 5.12 misc/ Directory Reference



### Files

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

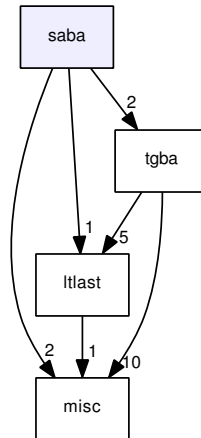
## 5.13 nips/ Directory Reference



### Files

- file [common.hh](#)
- file [nips.hh](#)

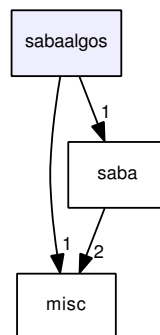
## 5.14 saba/ Directory Reference



### Files

- file [explicitstateconjunction.hh](#)
- file [saba.hh](#)
- file [sabacomplementtgba.hh](#)
- file [sabastate.hh](#)
- file [sabasucciter.hh](#)

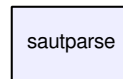
## 5.15 sabaalgos/ Directory Reference



### Files

- file [sabadotty.hh](#)
- file [sabareachiter.hh](#)

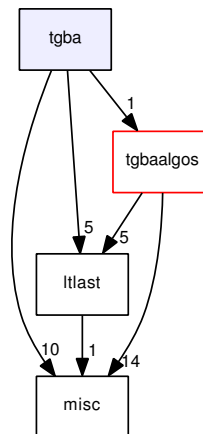
## 5.16 sautparse/ Directory Reference



### Files

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

## 5.17 tgba/ Directory Reference

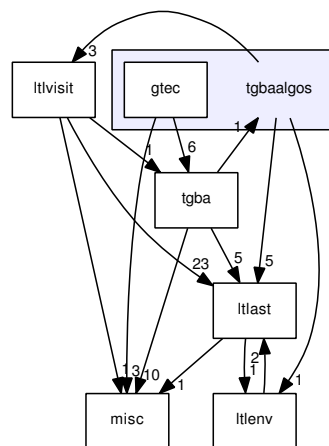


### Files

- file [bdddict.hh](#)
- file [bddprint.hh](#)
- file [formula2bdd.hh](#)
- file [futurecondcol.hh](#)
- file [public.hh](#)
- file [state.hh](#)
- file [statebdd.hh](#)
- file [succiter.hh](#)
- file [succiterconcrete.hh](#)
- file [taatgba.hh](#)
- file [tgba.hh](#)
- file [tgbabddconcrete.hh](#)
- file [tgbabddconcretefactory.hh](#)
- file [tgbabddconcreteproduct.hh](#)
- file [tgbabddcoredata.hh](#)
- file [tgbabddfactory.hh](#)
- file [tgbaexplicit.hh](#)

- file [tgbakvcomplement.hh](#)
- file [tgbaproduct.hh](#)
- file [tgbareduc.hh](#)
- file [tgbafracomplement.hh](#)
- file [tgbascc.hh](#)
- file [tgbaagba.hh](#)
- file [tgbatba.hh](#)
- file [tgbaunion.hh](#)

## 5.18 tgbaalgos/ Directory Reference



### Directories

- directory [gtec](#)

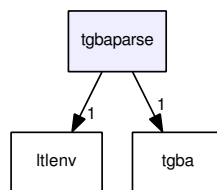
### Files

- file [bfssteps.hh](#)
- file [cutscc.hh](#)
- file [dotty.hh](#)
- file [dottydec.hh](#)
- file [dupexp.hh](#)
- file [eltl2tgba\\_lacim.hh](#)
- file [emptiness.hh](#)
- file [emptiness\\_stats.hh](#)
- file [gv04.hh](#)
- file [lbtt.hh](#)
- file [ltl2taa.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 [scc.hh](#)
- file [sccfilter.hh](#)
- file [se05.hh](#)
- file [stats.hh](#)
- file [tau03.hh](#)
- file [tau03opt.hh](#)
- file [weight.hh](#)

## 5.19 tgbaparse/ Directory Reference



### Files

- file [public.hh](#)

## 6 Namespace Documentation

### 6.1 eltly Namespace Reference

#### Classes

- class [location](#)  
*Abstract a location.*
- class [position](#)  
*Abstract a position.*
- class [stack](#)
- class [slice](#)  
*Present a slice of the top of a stack.*

## Functions

- const [location](#) [operator+](#) (const [location](#) &begin, const [location](#) &end)  
*Join two location objects to create a location.*
- const [location](#) [operator+](#) (const [location](#) &begin, unsigned int width)  
*Add two location objects.*
- [location](#) & [operator+=](#) ([location](#) &res, unsigned int width)  
*Add and assign a location.*
- bool [operator==](#) (const [location](#) &loc1, const [location](#) &loc2)  
*Compare two location objects.*
- bool [operator!=](#) (const [location](#) &loc1, const [location](#) &loc2)  
*Compare two location objects.*
- std::ostream & [operator<<](#) (std::ostream &ostr, const [location](#) &loc)  
*Intercept output stream redirection.*
- const [position](#) & [operator+=](#) ([position](#) &res, const int width)  
*Add and assign a position.*
- const [position](#) [operator+](#) (const [position](#) &begin, const int width)  
*Add two position objects.*
- const [position](#) & [operator-=](#) ([position](#) &res, const int width)  
*Add and assign a position.*
- const [position](#) [operator-](#) (const [position](#) &begin, const int width)  
*Add two position objects.*
- bool [operator==](#) (const [position](#) &pos1, const [position](#) &pos2)  
*Compare two position objects.*
- bool [operator!=](#) (const [position](#) &pos1, const [position](#) &pos2)  
*Compare two position objects.*
- std::ostream & [operator<<](#) (std::ostream &ostr, const [position](#) &pos)  
*Intercept output stream redirection.*

### 6.1.1 Function Documentation

#### 6.1.1.1 bool eltlyy::operator!= (const position & pos1, const position & pos2) [inline]

Compare two position objects.

**6.1.1.2 bool eltly::operator!= (const location & *loc1*, const location & *loc2*) [inline]**

Compare two location objects.

**6.1.1.3 const position eltly::operator+ (const position & *begin*, const int *width*) [inline]**

Add two position objects.

**6.1.1.4 const location eltly::operator+ (const location & *begin*, unsigned int *width*) [inline]**

Add two location objects.

References eltly::location::columns().

**6.1.1.5 const location eltly::operator+ (const location & *begin*, const location & *end*) [inline]**

Join two location objects to create a location.

References eltly::location::end.

**6.1.1.6 const position& eltly::operator+= (position & *res*, const int *width*) [inline]**

Add and assign a position.

References eltly::position::columns().

**6.1.1.7 location& eltly::operator+= (location & *res*, unsigned int *width*) [inline]**

Add and assign a location.

References eltly::location::columns().

**6.1.1.8 const position eltly::operator- (const position & *begin*, const int *width*) [inline]**

Add two position objects.

**6.1.1.9 const position& eltly::operator-= (position & *res*, const int *width*) [inline]**

Add and assign a position.

#### 6.1.1.10 `std::ostream& eltly::operator<< (std::ostream & ostr, const position & pos) [inline]`

Intercept output stream redirection.

##### Parameters

- ostr* the destination output stream
- pos* a reference to the position to redirect

References `eltly::position::column`, `eltly::position::filename`, and `eltly::position::line`.

#### 6.1.1.11 `std::ostream& eltly::operator<< (std::ostream & ostr, const location & loc) [inline]`

Intercept output stream redirection.

##### Parameters

- ostr* the destination output stream
- loc* a reference to the location to redirect

Avoid duplicate information.

#### 6.1.1.12 `bool eltly::operator== (const position & pos1, const position & pos2) [inline]`

Compare two position objects.

References `eltly::position::column`, `eltly::position::filename`, and `eltly::position::line`.

#### 6.1.1.13 `bool eltly::operator== (const location & loc1, const location & loc2) [inline]`

Compare two location objects.

## 6.2 Itlly Namespace Reference

### Classes

- class [location](#)  
*Abstract a location.*
- class [position](#)  
*Abstract a position.*

- class `stack`
- class `slice`

*Present a slice of the top of a stack.*

## Functions

- const `location operator+` (const `location` &begin, const `location` &end)  
*Join two location objects to create a location.*
- const `location operator+` (const `location` &begin, unsigned int width)  
*Add two location objects.*
- `location & operator+=` (`location` &res, unsigned int width)  
*Add and assign a location.*
- bool `operator==` (const `location` &loc1, const `location` &loc2)  
*Compare two location objects.*
- bool `operator!=` (const `location` &loc1, const `location` &loc2)  
*Compare two location objects.*
- std::ostream & `operator<<` (std::ostream &ostr, const `location` &loc)  
*Intercept output stream redirection.*
- const `position & operator+=` (`position` &res, const int width)  
*Add and assign a position.*
- const `position operator+` (const `position` &begin, const int width)  
*Add two position objects.*
- const `position & operator-=` (`position` &res, const int width)  
*Add and assign a position.*
- const `position operator-` (const `position` &begin, const int width)  
*Add two position objects.*
- bool `operator==` (const `position` &pos1, const `position` &pos2)  
*Compare two position objects.*
- bool `operator!=` (const `position` &pos1, const `position` &pos2)  
*Compare two position objects.*
- std::ostream & `operator<<` (std::ostream &ostr, const `position` &pos)  
*Intercept output stream redirection.*

### 6.2.1 Function Documentation

#### 6.2.1.1 `bool Itlly::operator!=(const position & pos1, const position & pos2) [inline]`

Compare two position objects.

#### 6.2.1.2 `bool Itlly::operator!=(const location & loc1, const location & loc2) [inline]`

Compare two location objects.

#### 6.2.1.3 `const position Itlly::operator+(const position & begin, const int width) [inline]`

Add two position objects.

#### 6.2.1.4 `const location Itlly::operator+(const location & begin, unsigned int width) [inline]`

Add two location objects.

References `Itlly::location::columns()`.

#### 6.2.1.5 `const location Itlly::operator+(const location & begin, const location & end) [inline]`

Join two location objects to create a location.

References `Itlly::location::end`.

#### 6.2.1.6 `const position& Itlly::operator+=(position & res, const int width) [inline]`

Add and assign a position.

References `Itlly::position::columns()`.

#### 6.2.1.7 `location& Itlly::operator+=(location & res, unsigned int width) [inline]`

Add and assign a location.

References `Itlly::location::columns()`.

#### 6.2.1.8 `const position Itlly::operator-(const position & begin, const int width) [inline]`

Add two position objects.

**6.2.1.9** `const position& Itlly::operator+= (position & res, const int width) [inline]`

Add and assign a position.

**6.2.1.10** `std::ostream& Itlly::operator<< (std::ostream & ostr, const position & pos) [inline]`

Intercept output stream redirection.

#### Parameters

*ostr* the destination output stream

*pos* a reference to the position to redirect

References Itlly::position::column, Itlly::position::filename, and Itlly::position::line.

**6.2.1.11** `std::ostream& Itlly::operator<< (std::ostream & ostr, const location & loc) [inline]`

Intercept output stream redirection.

#### Parameters

*ostr* the destination output stream

*loc* a reference to the location to redirect

Avoid duplicate information.

**6.2.1.12** `bool Itlly::operator== (const position & pos1, const position & pos2) [inline]`

Compare two position objects.

References Itlly::position::column, Itlly::position::filename, and Itlly::position::line.

**6.2.1.13** `bool Itlly::operator== (const location & loc1, const location & loc2) [inline]`

Compare two location objects.

## 6.3 sautyy Namespace Reference

### Classes

- class [location](#)  
*Abstract a location.*
- class [position](#)  
*Abstract a position.*
- class [stack](#)
- class [slice](#)  
*Present a slice of the top of a stack.*

### Functions

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

### 6.3.1 Function Documentation

#### 6.3.1.1 const position sautyy::operator+ (const position & *begin*, const int *width*) [[inline](#)]

Add two position objects.



**6.3.1.2 const location sautty::operator+ (const location & *begin*, unsigned int *width*) [inline]**

Add two location objects.

References sautty::location::columns().

**6.3.1.3 const location sautty::operator+ (const location & *begin*, const location & *end*) [inline]**

Join two location objects to create a location.

References sautty::location::end.

**6.3.1.4 const position& sautty::operator+= (position & *res*, const int *width*) [inline]**

Add and assign a position.

References sautty::position::columns().

**6.3.1.5 location& sautty::operator+= (location & *res*, unsigned int *width*) [inline]**

Add and assign a location.

References sautty::location::columns().

**6.3.1.6 const position sautty::operator- (const position & *begin*, const int *width*) [inline]**

Add two position objects.

**6.3.1.7 const position& sautty::operator-= (position & *res*, const int *width*) [inline]**

Add and assign a position.

**6.3.1.8 std::ostream& sautty::operator<< (std::ostream & *ostr*, const position & *pos*) [inline]**

Intercept output stream redirection.

**Parameters**

*ostr* the destination output stream

*pos* a reference to the position to redirect

References `sauty::position::column`, `sauty::position::filename`, and `sauty::position::line`.

### 6.3.1.9 `std::ostream& sauty::operator<< (std::ostream & ostr, const location & loc) [inline]`

Intercept output stream redirection.

#### Parameters

*ostr* the destination output stream

*loc* a reference to the location to redirect

Avoid duplicate information.

## 6.4 spot Namespace Reference

### Namespaces

- namespace [eltl](#)
- namespace [ltl](#)

### 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 [fair\\_kripke\\_succ\\_iterator](#)
- class [fair\\_kripke](#)
- class [kripke](#)
- 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*`.  
This is meant to be used as a comparison functor for STL map whose key are of type `const char*`.*
- class [minato\\_isop](#)  
*Generate an irredundant sum-of-products (ISOP) form of a BDD function.  
This algorithm implements a derecursived version the Minato-Morreale algorithm presented in the following paper.*
- class [loopless\\_modular\\_mixed\\_radix\\_gray\\_code](#)  
*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.*
- class [option\\_map](#)  
*Manage a map of options.  
Each option is defined by a string and is associated to an integer value.*
- 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 [explicit\\_state\\_conjunction](#)  
*Basic implementation of [saba\\_state\\_conjunction](#).  
This class provides a basic implementation to iterate over a conjunction of states of a saba.*
- class [saba](#)

*A State-based Alternating (Generalized) Büchi Automaton.*

*Browsing such automaton can be achieved using two functions: `get_init_state`, and `succ_iter`.*

*The former returns the initial state while the latter lists the successor states of any state.*

- class [saba\\_complement\\_tgba](#)

*Complement a TGBA and produce a SABA.*

*The original TGBA is transformed into a States-based Büchi Automaton.*

- class [saba\\_state](#)

*Abstract class for saba states.*

- struct [saba\\_state\\_ptr\\_less\\_than](#)

*Strict Weak Ordering for `saba_state*`.*

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

- struct [saba\\_state\\_ptr\\_equal](#)

*An Equivalence Relation for `saba_state*`.*

*This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type `saba_state*`.*

- struct [saba\\_state\\_ptr\\_hash](#)

*Hash Function for `saba_state*`.*

*This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type `saba_state*`.*

- struct [saba\\_state\\_shared\\_ptr\\_less\\_than](#)

*Strict Weak Ordering for `shared_saba_state (shared_ptr<const saba_state*>)`.*

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

- struct [saba\\_state\\_shared\\_ptr\\_equal](#)

*An Equivalence Relation for `shared_saba_state (shared_ptr<const saba_state*>)`.*

*This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type `shared_saba_state`.*

- struct [saba\\_state\\_shared\\_ptr\\_hash](#)

*Hash Function for `shared_saba_state (shared_ptr<const saba_state*>)`.*

*This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type `shared_saba_state`.*

- class [saba\\_state\\_conjunction](#)

*Iterate over a conjunction of `saba_state`.*

*This class provides the basic functionalities required to iterate over a conjunction of states of a saba.*

- class [saba\\_succ\\_iterator](#)

*Iterate over the successors of a `saba_state`.*

*This class provides the basic functionalities required to iterate over the successors of a state of a saba. Since transitions of an alternating automaton are defined as a boolean function with conjunctions (universal) and disjunctions (non-deterministic),.*

- class [saba\\_reachable\\_iterator](#)

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

- class [saba\\_reachable\\_iterator\\_depth\\_first](#)

*An implementation of `spot::saba_reachable_iterator` that browses states depth first.*

- class `saba_reachable_iterator_breadth_first`

*An implementation of `spot::saba_reachable_iterator` that browses states breadth first.*

- class `bdd_dict`
- class `future_conditions_collector`

*Wrap a `tgba` to offer information about upcoming conditions.*

*This class is a `spot::tgba` wrapper that simply add a new method, `future_conditions()`, to any `spot::tgba`.*

- class `state`

*Abstract class for states.*

- struct `state_ptr_less_than`

*Strict Weak Ordering for `state*`.*

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

- struct `state_ptr_equal`

*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*`.*

- struct `state_ptr_hash`

*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*`.*

- struct `state_shared_ptr_less_than`

*Strict Weak Ordering for `shared_state (shared_ptr<const state*>)`.*

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

- struct `state_shared_ptr_equal`

*An Equivalence Relation for `shared_state (shared_ptr<const state*>)`.*

*This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type `shared_state`.*

- struct `state_shared_ptr_hash`

*Hash Function for `shared_state (shared_ptr<const state*>)`.*

*This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type `shared_state`.*

- class `state_bdd`

- class `tgba_succ_iterator`

*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.*

- class `tgba_succ_iterator_concrete`

- class `taa_tgba`

*A self-loop Transition-based Alternating Automaton (TAA) which is seen as a TGBA (abstract class, see below).*

- class `state_set`

*Set of states deriving from `spot::state`.*

- class [taa\\_succ\\_iterator](#)
- class [taa\\_tgba\\_labelled](#)
- class [taa\\_tgba\\_string](#)
- class [taa\\_tgba\\_formula](#)
- class [tgba](#)

*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).*

- 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 [tgba\\_explicit\\_labelled](#)

*A `tgba_explicit` instance with states labeled by a given type.*

- class [tgba\\_explicit\\_string](#)
- class [tgba\\_explicit\\_formula](#)
- class [bdd\\_ordered](#)
- class [tgba\\_kv\\_complement](#)

*Build a complemented automaton.*

*The construction comes from:*

- class [state\\_product](#)

*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.*

- 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\\_safracomplement](#)

*Build a complemented automaton.*

*It creates an automaton that recognizes the negated language of aut.*

- class [tgba\\_scc](#)

*Wrap a tgba to offer information about strongly connected components.*

*This class is a [spot::tgba](#) wrapper that simply add a new method [scc\\_of\\_state\(\)](#) to retrieve the number of a SCC a state belongs to.*

- class [tgba\\_sgba\\_proxy](#)

*Change the labeling-mode of [spot::tgba](#) on the fly, producing a state-based generalized Büchi automaton.*

*This class acts as a proxy in front of a [spot::tgba](#), that should label on states on-the-fly. The result is still a [spot::tgba](#), but acceptances conditions are also on states.*

- class [tgba\\_tba\\_proxy](#)

*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).*

- class [tgba\\_sba\\_proxy](#)

*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.*

- class [state\\_union](#)

*A state for [spot::tgba\\_union](#).*

*This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.*

- class [tgba\\_succ\\_iterator\\_union](#)

*Iterate over the successors of an union computed on the fly.*

- class [tgba\\_union](#)

*A lazy union. (States are computed on the fly.).*

- class [bfs\\_steps](#)

*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.*

- struct [sccs\\_set](#)

- 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`.  
An instance of this class can be passed to `spot::dotty_reachable`.*
- struct [scc\\_stats](#)
- class [scc\\_map](#)  
*Build a map of Strongly Connected components in in a 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](#)
- class [nips\\_exception](#)  
*An exception used to forward NIPS errors.*

- class [nips\\_interface](#)

*An interface to provide a PROMELA front-end.*

### Typedefs

- typedef std::set< const [symbol](#) \* > [symbol\\_set](#)
- typedef std::set< [rsymbol](#) > [rsymbol\\_set](#)
- typedef std::pair< evtgbayy::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 [fair\\_kripke\\_succ\\_iterator](#) [kripke\\_succ\\_iterator](#)
- typedef boost::shared\_ptr< const [saba\\_state](#) > [shared\\_saba\\_state](#)
- typedef boost::shared\_ptr< const [state](#) > [shared\\_state](#)
- typedef std::vector< [bdd\\_ordered](#) > [acc\\_list\\_t](#)
- 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< [tgbayy::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.*

- bdd `compute_all_acceptance_conditions` (bdd neg\_acceptance\_conditions)

*Compute all acceptance conditions from all neg acceptance conditions.*

- bdd `compute_neg_acceptance_conditions` (bdd all\_acceptance\_conditions)

*Compute neg acceptance conditions from all acceptance conditions.*

- 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.*

- int `memusage` ()

*Total number of pages in use by the program.*

- 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 `mrnd` (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 `nrnd` ()

*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 & [saba\\_dotty\\_reachable](#) (std::ostream &os, const [saba](#) \*g)  
*Print reachable states in dot format.*
- 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.  
This function builds the resulting product as another [spot::tgba\\_bdd\\_concrete](#) automaton.*

- void `display_safra` (const `tgba_safra_complement` \*a)  
*Produce a dot output of the Safra automaton associated to a.*
- `std::vector< std::vector< sccs_set * > > * find_paths (tgba *a, const scc_map &m)`
- unsigned `max_spanning_paths` (`std::vector< sccs_set * > *paths`, `scc_map` &m)
- `std::list< tgba * > split_tgba (tgba *a, const scc_map &m, unsigned split_number)`
- `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.*
- `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.*
- `tgba_bdd_concrete * eltl_to_tgba_lacim (const ltl::formula *f, bdd_dict *dict)`  
*Build a `spot::tgba_bdd_concrete` from an ETLT formula.*  
*This is based on the following paper.*
- `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.*
- `taa_tgba * ltl_to_taa (const ltl::formula *f, bdd_dict *dict, bool refined_rules=false)`  
*Build a `spot::taa*` from an LTL formula.*  
*This is based on the following.*
- `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.*
- `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:

- `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, bool comments=false)  
Print reachable states in Spin never claim format.
- `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.
- `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,.
- `tgba * random_graph` (int n, float d, const `ltl::atomic_prop_set` \*ap, `bdd_dict` \*dict, int n\_acc=0, float a=0.1, float t=0.5, `ltl::environment` \*env=&`ltl::default_environment::instance`())  
Construct a `tgba` randomly.
- `tgba_run * reduce_run` (const `tgba` \*a, const `tgba_run` \*org)  
Reduce an accepting run.  
Return a run which is accepting for and that is no longer that org.
- const `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*.

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.

- *std::ostream* & *tgba\_save\_reachable* (*std::ostream* &os, const *tgba* \*g)  
Save reachable states in text format.
- *scc\_stats build\_scc\_stats* (const *tgba* \*a)
- *scc\_stats build\_scc\_stats* (const *scc\_map* &m)
- *std::ostream* & *dump\_scc\_dot* (const *tgba* \*a, *std::ostream* &out, bool verbose=false)
- *std::ostream* & *dump\_scc\_dot* (const *scc\_map* &m, *std::ostream* &out, bool verbose=false)
- *tgba* \* *scc\_filter* (const *tgba* \*aut, bool remove\_all\_useless=false)  
Prune unaccepting SCCs and remove superfluous acceptance conditions.
- *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\_string* \* *tgba\_parse* (const *std::string* &filename, *tgba\_parse\_error\_list* &error\_list, *bdd\_dict* \*dict, *ltl::environment* &env=*ltl::default\_environment::instance()*, *ltl::environment* &envacc=*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)
- *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, bool stack\_inclusion=true, bool double\_inclusion=false, bool reversed\_double\_inclusion=false, bool no\_decomp=false)
- *std::ostream* & *operator<<* (*std::ostream* &os, const *nips\_exception* &e)

### 6.4.1 Typedef Documentation

**6.4.1.1** `typedef std::vector<bdd_ordered> spot::acc_list_t`

**6.4.1.2** `typedef std::pair<evtgba::location, std::string> spot::evtgb_parse_error`

A parse diagnostic with its location.

**6.4.1.3** `typedef std::list<evtgb_parse_error> spot::evtgb_parse_error_list`

A list of parser diagnostics, as filled by parse.

**6.4.1.4** `typedef fair_kripke_succ_iterator spot::kripke_succ_iterator`

**6.4.1.5** `typedef std::set<rsymbol> spot::rsymbol_set`

**6.4.1.6** `typedef boost::shared_ptr<const saba_state> spot::shared_saba_state`

**6.4.1.7** `typedef boost::shared_ptr<const state> spot::shared_state`

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

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

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



## 6.4.2 Function Documentation

### 6.4.2.1 `std::string spot::bdd_format_accset (const bdd_dict * dict, bdd b)`

Format a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

#### Parameters

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

*b* The BDD to print.

#### Returns

The BDD formatted as a string.

### 6.4.2.2 `std::string spot::bdd_format_formula (const bdd_dict * dict, bdd b)`

Format a BDD as a formula.

#### Parameters

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

*b* The BDD to print.

#### Returns

The BDD formatted as a string.

### 6.4.2.3 `std::string spot::bdd_format_sat (const bdd_dict * dict, bdd b)`

Format a BDD as a list of literals.

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

#### Parameters

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

*b* The BDD to print.

#### Returns

The BDD formatted as a string.

#### 6.4.2.4 `std::string spot::bdd_format_set (const bdd_dict * dict, bdd b)`

Format a BDD as a set.

##### Parameters

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

*b* The BDD to print.

##### Returns

The BDD formatted as a string.

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

Print a BDD as a list of acceptance conditions.

This is used when saving a TGBA.

##### Parameters

*os* The output stream.

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

*b* The BDD to print.

##### Returns

The BDD formatted as a string.

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

Print a BDD as a set of acceptance conditions.

This is used when saving a TGBA.

##### Parameters

*os* The output stream.

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

*b* The BDD to print.

##### Returns

The BDD formatted as a string.

**6.4.2.7 std::ostream& spot::bdd\_print\_dot (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a diagram in dotty format.

**Parameters**

*os* The output stream.

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

*b* The BDD to print.

**6.4.2.8 std::ostream& spot::bdd\_print\_formula (std::ostream & *os*, const bdd\_dict \* *dict*, bdd *b*)**

Print a BDD as a formula.

**Parameters**

*os* The output stream.

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

*b* The BDD to print.

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

Print a BDD as a list of literals.

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

**Parameters**

*os* The output stream.

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

*b* The BDD to print.

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

Print a BDD as a set.

**Parameters**

*os* The output stream.

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

*b* The BDD to print.

**6.4.2.11** `std::ostream& spot::bdd_print_table (std::ostream & os, const bdd_dict * dict, bdd b)`

Print a BDD as a table.

**Parameters**

*os* The output stream.

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

*b* The BDD to print.

**6.4.2.12** `const ltl::formula* spot::bdd_to_formula (bdd f, const bdd_dict * d)`**6.4.2.13** `scc_stats spot::build_scc_stats (const scc_map & m)`**6.4.2.14** `scc_stats spot::build_scc_stats (const tgba * a)`**6.4.2.15** `bdd spot::compute_all_acceptance_conditions (bdd neg_acceptance_conditions)`

Compute all acceptance conditions from all neg acceptance conditions.

**6.4.2.16** `bdd spot::compute_neg_acceptance_conditions (bdd all_acceptance_conditions)`

Compute neg acceptance conditions from all acceptance conditions.

**6.4.2.17** `void spot::display_safra (const tgba_safra_complement * a)`

Produce a dot output of the Safra automaton associated to *a*.

**Parameters**

*a* The `tgba_safra_complement` with an intermediate Safra automaton to display

**6.4.2.18** `std::ostream& spot::dotty_reachable (std::ostream & os, const evtgba * g)`

Print reachable states in dot format.

**6.4.2.19** `std::ostream& spot::dump_scc_dot (const scc_map & m, std::ostream & out, bool verbose = false)`

**6.4.2.20** `std::ostream& spot::dump_scc_dot (const tgba * a, std::ostream & out, bool verbose = false)`

**6.4.2.21** `evtgba_explicit* spot::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.

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

Save reachable states in text format.

**6.4.2.23** `std::vector<std::vector<sccs_set* > >* spot::find_paths (tgba * a, const scc_map & m)`

**6.4.2.24** `bool spot::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.

**6.4.2.25** `bdd spot::formula_to_bdd (const ltl::formula *f, bdd_dict *d, void *for_me)`

**6.4.2.26** `unsigned spot::max_spanning_paths (std::vector< sccs_set * > *paths, scc_map &m)`

**6.4.2.27** `int spot::memusage ()`

Total number of pages in use by the program.

**Returns**

The total number of pages in use by the program if known. -1 otherwise.

**6.4.2.28** `std::ostream& spot::operator<< (std::ostream &os, const nips_exception &e)`

**6.4.2.29** `std::ostream& spot::operator<< (std::ostream &os, const gspn_exception &e)`

**6.4.2.30** `std::ostream& spot::saba_dotty_reachable (std::ostream &os, const saba *g)`

Print reachable states in dot format.

**6.4.2.31** `tgba* spot::scc_filter (const tgba *aut, bool remove_all_useless = false)`

Prune unaccepting SCCs and remove superfluous acceptance conditions.

This functions will explore the SCCs of the automaton and remove dead SCCs (unaccepting, and with no exit path leading to an accepting SCC).

Additionally, this will try to remove useless acceptance conditions. This operation may diminish the number of acceptance condition of the automaton, for instance when two acceptance conditions are always used together we only keep one (but it will never remove all acceptance conditions, even if it would be OK to have zero).

Acceptance conditions on transitions going to non-accepting SCC are all removed. Acceptance conditions going to an accepting SCC and coming from another SCC are only removed if *remove\_all\_useless* is set. The default value of *remove\_all\_useless* is `false` because some algorithms (like the degeneralization) will work better if transition going to an accepting SCC are accepting.

**6.4.2.32** `std::list<tgba*> spot::split_tgba (tgba * a, const scc_map & m, unsigned split_number)`

**6.4.2.33** `evtgba_explicit* spot::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.)

## 6.5 spot::eltl Namespace Reference

### Typedefs

- typedef std::pair< std::string, std::string > [spair](#)
- typedef std::pair< [eltly::location](#), [spair](#) > [parse\\_error](#)  
A parse diagnostic <location, <file, message>>.
- typedef std::list< [parse\\_error](#) > [parse\\_error\\_list](#)  
A list of parser diagnostics, as filled by *parse*.

### Functions

- [formula](#) \* [parse\\_file](#) (const std::string &filename, [parse\\_error\\_list](#) &error\_list, [environment](#) &env=default\_environment::instance(), bool debug=false)  
Build a formula from a text file.
- [formula](#) \* [parse\\_string](#) (const std::string &eltl\_string, [parse\\_error\\_list](#) &error\_list, [environment](#) &env=default\_environment::instance(), bool debug=false)  
Build a formula from an ECTL string.
- bool [format\\_parse\\_errors](#) (std::ostream &os, [parse\\_error\\_list](#) &error\_list)  
Format diagnostics produced by *spot::eltl::parse*.

## 6.6 spot::ltl Namespace Reference

### Namespaces

- namespace [formula\\_tree](#)  
*Trees representing formulae where atomic propositions are unknown.*

### Classes

- class [atomic\\_prop](#)  
*Atomic propositions.*
- class [automatop](#)  
*Automaton operators.*
- class [binop](#)  
*Binary operator.*
- class [constant](#)  
*A constant (True or False).*
- class [formula](#)  
*An LTL formula.*  
*The only way you can work with a formula is to build a [spot::ltl::visitor](#) or [spot::ltl::const\\_visitor](#).*
- struct [formula\\_ptr\\_less\\_than](#)  
*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*`.*
- struct [formula\\_ptr\\_hash](#)  
*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*`.*
- class [multop](#)  
*Multi-operand operators.*  
*These operators are considered commutative and associative.*
- class [nfa](#)  
*Nondeterministic Finite Automata used by automata operators.*
- class [succ\\_iterator](#)
- class [ref\\_formula](#)  
*A reference-counted LTL formula.*
- class [unop](#)  
*Unary operators.*
- struct [visitor](#)  
*Formula visitor that can modify the formula.*  
*Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.*



- struct [const\\_visitor](#)  
*Formula visitor that cannot modify the formula.*
- class [declarative\\_environment](#)  
*A declarative environment.  
This environment recognizes all atomic propositions that have been previously declared. It will reject other.*
- class [default\\_environment](#)  
*A laxist environment.  
This environment recognizes all atomic propositions.*
- class [environment](#)  
*An environment that describes atomic propositions.*
- class [ltl\\_file](#)  
*Read LTL formulae from a file, one by one.*
- class [clone\\_visitor](#)  
*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::formula::clone](#) instead, it is way faster.*
- class [language\\_containment\\_checker](#)
- class [unabbreviate\\_logic\\_visitor](#)  
*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](#).*
- class [postfix\\_visitor](#)  
*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.*
- class [random\\_ltl](#)  
*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.*
- class [simplify\\_f\\_g\\_visitor](#)  
*Replace  $true \cup f$  and  $false \cap g$  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.  
The rewriting performed on logical operator is the same as the one done by [spot::ltl::unabbreviate\\_logic\\_visitor](#).*

## Typedefs

- typedef std::pair< [ltl::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\\_Containment\\_Checks](#) = 8, [Reduce\\_Containment\\_Checks\\_Stronger](#) = 16, [Reduce\\_All](#) = -1U  
}
- Options for [spot::ltl::reduce](#).*

## Functions

- [formula](#) \* [parse](#) (const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list, [environment](#) &env=default\_environment::instance(), bool debug=false)  
*Build a formula from an LTL string.*
- bool [format\\_parse\\_errors](#) (std::ostream &os, const std::string &ltl\_string, [parse\\_error\\_list](#) &error\_list)  
*Format diagnostics produced by [spot::ltl::parse](#).*
- [atomic\\_prop\\_set](#) \* [atomic\\_prop\\_collect](#) (const [formula](#) \*f, [atomic\\_prop\\_set](#) \*s=0)  
*Return the set of atomic propositions occurring in a formula.*
- [formula](#) \* [basic\\_reduce](#) (const [formula](#) \*f)  
*Basic rewritings.*
- bool [is\\_GF](#) (const [formula](#) \*f)  
*Whether a formula starts with GF.*
- bool [is\\_FG](#) (const [formula](#) \*f)  
*Whether a formula starts with FG.*
- [formula](#) \* [clone](#) (const [formula](#) \*f)  
*Clone a formula.*
- [formula](#) \* [reduce\\_tau03](#) (const [formula](#) \*f, bool stronger=true)  
*Reduce a formula using language containment relationships.*

- 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.*  
*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.  $a \mid b \mid c$  has length 4, because  $\mid$  is represented once internally).*
- **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**.*
- **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.*
- **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.*  
*Pure eventuality formulae are defined in.*
- bool **is\_universal** (const **formula** \*f)  
*Check whether a formula is purely universal.*  
*Purely universal formulae are defined in.*
- **formula** \* **simplify\_f\_g** (const **formula** \*f)  
*Replace  $true \cup f$  and  $false \cap g$  by  $F \cup f$  and  $G \cap g$ .*
- bool **syntactic\_implication** (const **formula** \*f1, const **formula** \*f2)  
*Syntactic implication.*  
*This comes from.*
- 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.*
- std::ostream & **to\_string** (const **formula** \*f, std::ostream &os, bool full\_parent=false)  
*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ELTL formulae).*

- std::string [to\\_string](#) (const [formula](#) \*f, bool full\_parent=false)  
*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ECTL formulae).*
- std::ostream & [to\\_spin\\_string](#) (const [formula](#) \*f, std::ostream &os)  
*Output a formula as a (parsable by Spin) string.*
- std::string [to\\_spin\\_string](#) (const [formula](#) \*f)  
*Convert a formula into a (parsable by Spin) string.*
- [formula](#) \* [unabbreviate\\_ltl](#) (const [formula](#) \*f)  
*Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.*

### 6.6.1 Function Documentation

#### 6.6.1.1 formula\* spot::ltl::reduce\_tau03 (const formula \*f, bool stronger = true)

Reduce a formula using language containment relationships.

The method is taken from table 4.1 in

```

///@TechReport{   tauriainen.03.a83,
///  author = {Heikki Tauriainen},
///  title = {On Translating Linear Temporal Logic into Alternating and
///    Nondeterministic Automata},
///  institution = {Helsinki University of Technology, Laboratory for
///    Theoretical Computer Science},
///  address = {Espoo, Finland},
///  month = dec,
///  number      = {A83},
///  pages = {132},
///  type = {Research Report},
///  year = {2003},
///  note = {Reprint of Licentiate's thesis}
///}
///
/// (The "dagged" cells in the tables are not handled here.)
///
/// If \a stronger is set, additional rules are used to further
/// reduce some U, R, and X usages.
///

```

#### 6.6.1.2 formula\* spot::ltl::unabbreviate\_ltl (const formula \*f)

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

The rewriting performed on logical operator is the same as the one done by [spot::ltl::unabbreviate\\_logic](#).

This will also rewrite unary operators such as [unop::F](#), and [unop::G](#), using only [binop::U](#), and [binop::R](#).

## 6.7 spot::ltl::formula\_tree Namespace Reference

Trees representing formulae where atomic propositions are unknown.

## Classes

- struct [node](#)
- struct [node\\_unop](#)
- struct [node\\_binop](#)
- struct [node\\_multop](#)
- struct [node\\_nfa](#)
- struct [node\\_atomic](#)

## Typedefs

- typedef boost::shared\_ptr< [node](#) > [node\\_ptr](#)  
*We use boost::shared\_ptr to easily handle deletion.*

## Enumerations

- enum { [True](#) = -1, [False](#) = -2 }  
*Integer values for True and False used in node\_atomic.*

## Functions

- [formula \\* instantiate](#) (const [node\\_ptr](#) np, const std::vector< [formula](#) \* > &v)
- [size\\_t arity](#) (const [node\\_ptr](#) np)  
*Get the arity.*

### 6.7.1 Detailed Description

Trees representing formulae where atomic propositions are unknown. Forward declaration. NFA's labels are represented by nodes which are defined in [formula\\_tree.hh](#), included in nfa.cc.

### 6.7.2 Typedef Documentation

#### 6.7.2.1 typedef boost::shared\_ptr<node> spot::ltl::formula\_tree::node\_ptr

We use boost::shared\_ptr to easily handle deletion.

### 6.7.3 Enumeration Type Documentation

#### 6.7.3.1 anonymous enum

Integer values for True and False used in [node\\_atomic](#).

**Enumerator:**

*True*

*False*

#### 6.7.4 Function Documentation

##### 6.7.4.1 `size_t spot::ltl::formula_tree::arity (const node_ptr np)`

Get the arity.

##### 6.7.4.2 `formula* spot::ltl::formula_tree::instanciate (const node_ptr np, const std::vector< formula * > & v)`

Instantiate the formula corresponding to the node with atomic propositions taken from *v*.

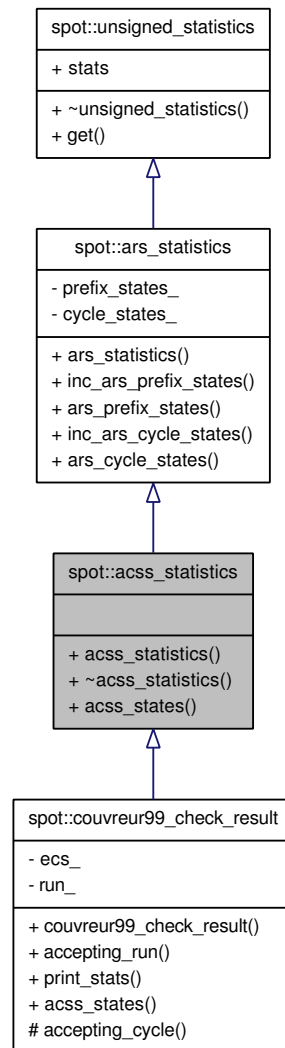
## 7 Class Documentation

### 7.1 `spot::acss_statistics` Class Reference

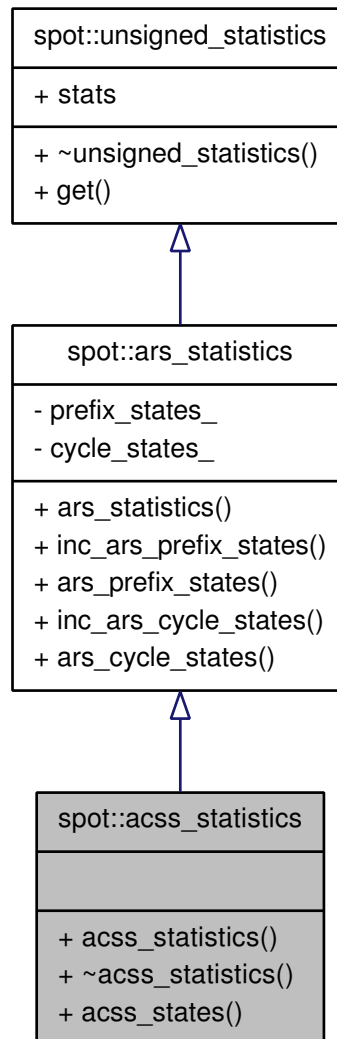
Accepting Cycle Search Space statistics.

```
#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

### 7.1.1 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.

### 7.1.2 Member Typedef Documentation

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

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

### 7.1.3 Constructor & Destructor Documentation

**7.1.3.1** `spot::acss_statistics::acss_statistics () [inline]`

References `acss_states()`, and `spot::unsigned_statistics::stats`.

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

### 7.1.4 Member Function Documentation

**7.1.4.1** `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](#).

Referenced by `acss_statistics()`.

#### 7.1.4.2 unsigned spot::ars\_statistics::ars\_cycle\_states () const [inline, inherited]

References spot::ars\_statistics::cycle\_states\_.

Referenced by spot::ars\_statistics::ars\_statistics().

#### 7.1.4.3 unsigned spot::ars\_statistics::ars\_prefix\_states () const [inline, inherited]

References spot::ars\_statistics::prefix\_states\_.

Referenced by spot::ars\_statistics::ars\_statistics().

#### 7.1.4.4 unsigned spot::unsigned\_statistics::get (const char \* str) const [inline, inherited]

References spot::unsigned\_statistics::stats.

#### 7.1.4.5 void spot::ars\_statistics::inc\_ars\_cycle\_states () [inline, inherited]

References spot::ars\_statistics::cycle\_states\_.

#### 7.1.4.6 void spot::ars\_statistics::inc\_ars\_prefix\_states () [inline, inherited]

References spot::ars\_statistics::prefix\_states\_.

### 7.1.5 Member Data Documentation

#### 7.1.5.1 stats\_map spot::unsigned\_statistics::stats [inherited]

Referenced by acss\_statistics(), spot::ars\_statistics::ars\_statistics(), spot::ec\_statistics::ec\_statistics(), spot::unsigned\_statistics::get(), and spot::unsigned\_statistics\_copy::seteq().

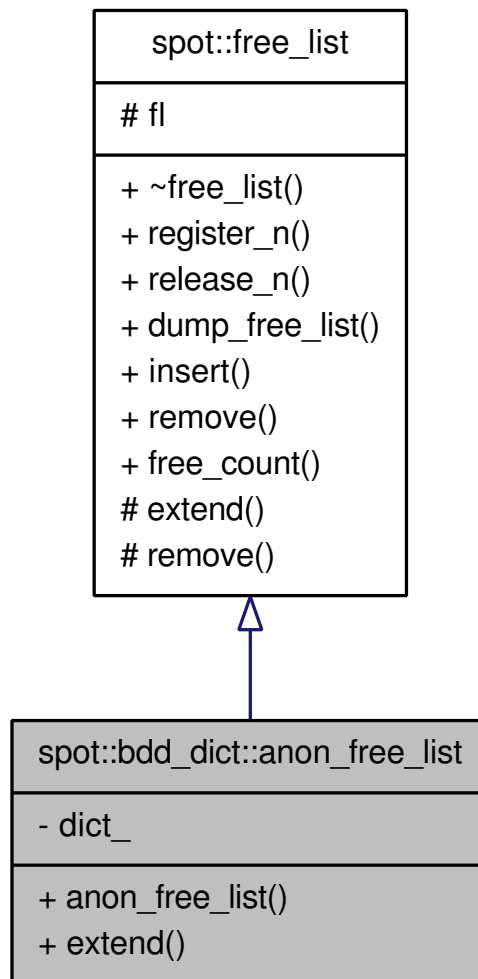
The documentation for this class was generated from the following file:

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

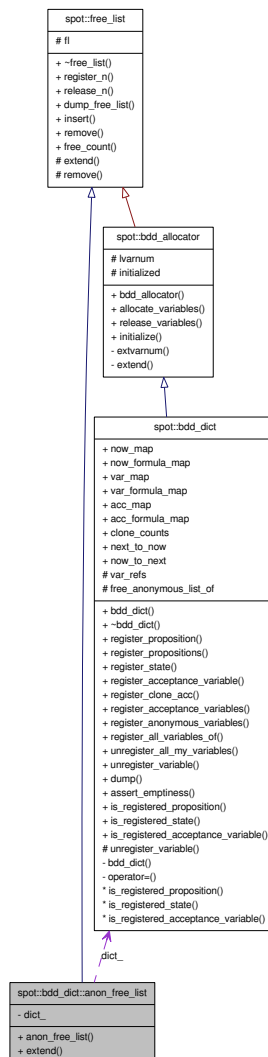
## 7.2 spot::bdd\_dict::anon\_free\_list Class Reference

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

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



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



## Public Member Functions

- [anon\\_free\\_list](#) ([bdd\\_dict](#) \*d=0)
- virtual int [extend](#) (int n)
- int [register\\_n](#) (int n)  
*Find n consecutive integers.*
- void [release\\_n](#) (int base, int n)  
*Release n consecutive integers starting at base.*
- std::ostream & [dump\\_free\\_list](#) (std::ostream &os) const  
*Dump the list to os for debugging.*
- void [insert](#) (int base, int n)  
*Extend the list by inserting a new pos-length pair.*

- void [remove](#) (int base, int n=0)  
*Remove  $n$  consecutive entries from the list, starting at  $base$ .*
- int [free\\_count](#) () const  
*Return the number of free integers on the list.*

### Protected Types

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

### Protected Member Functions

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

### Protected Attributes

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

### Private Attributes

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

#### 7.2.1 Member Typedef Documentation

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

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

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

#### 7.2.2 Constructor & Destructor Documentation

**7.2.2.1** spot::bdd\_dict::anon\_free\_list::anon\_free\_list (bdd\_dict \*  $d = 0$ )

### 7.2.3 Member Function Documentation

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

Dump the list to *os* for debugging.

#### 7.2.3.2 `virtual int spot::bdd_dict::anon_free_list::extend (int n)` `[virtual]`

Allocate *n* integer.

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

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

#### 7.2.3.3 `int spot::free_list::free_count () const` `[inherited]`

Return the number of free integers on the list.

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

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

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

Find *n* consecutive integers.

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

#### Returns

the first integer of the range

#### 7.2.3.6 `void spot::free_list::release_n (int base, int n)` `[inherited]`

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

#### 7.2.3.7 `void spot::free_list::remove (free_list_type::iterator i, int base, int n)` `[protected, inherited]`

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

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

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

#### 7.2.4 Member Data Documentation

##### 7.2.4.1 bdd\_dict\* spot::bdd\_dict::anon\_free\_list::dict\_ [private]

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

Tracks unused BDD variables.

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

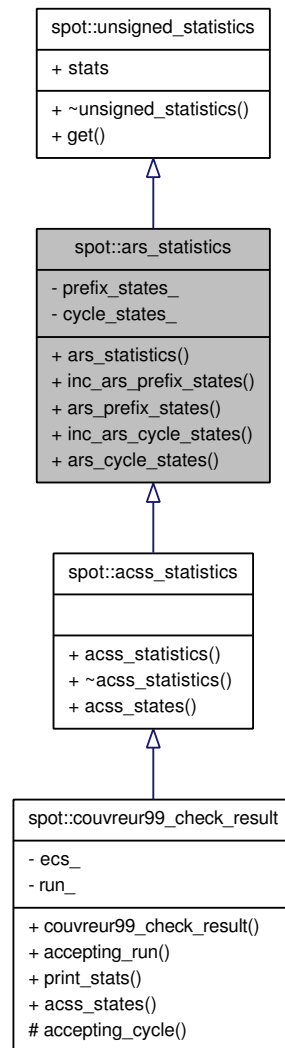
- [tgba/bdddict.hh](#)

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

Accepting Run Search statistics.

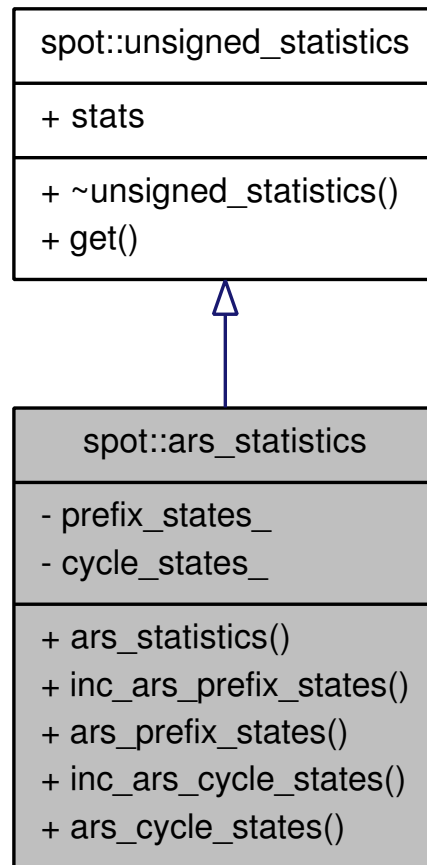
```
#include <tgbaalgos/emptiness_stats.hh>
```

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*

### 7.3.1 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.

### 7.3.2 Member Typedef Documentation

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

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

### 7.3.3 Constructor & Destructor Documentation

**7.3.3.1** `spot::ars_statistics::ars_statistics () [inline]`

References [ars\\_cycle\\_states\(\)](#), [ars\\_prefix\\_states\(\)](#), and [spot::unsigned\\_statistics::stats](#).

### 7.3.4 Member Function Documentation

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

References [cycle\\_states\\_](#).

Referenced by [ars\\_statistics\(\)](#).

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

References [prefix\\_states\\_](#).

Referenced by [ars\\_statistics\(\)](#).

#### 7.3.4.3 unsigned spot::unsigned\_statistics::get (const char \* *str*) const [inline, inherited]

References spot::unsigned\_statistics::stats.

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

References cycle\_states\_.

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

References prefix\_states\_.

### 7.3.5 Member Data Documentation

#### 7.3.5.1 unsigned spot::ars\_statistics::cycle\_states\_ [private]

states visited to construct the prefix

Referenced by ars\_cycle\_states(), and inc\_ars\_cycle\_states().

#### 7.3.5.2 unsigned spot::ars\_statistics::prefix\_states\_ [private]

Referenced by ars\_prefix\_states(), and inc\_ars\_prefix\_states().

#### 7.3.5.3 stats\_map spot::unsigned\_statistics::stats [inherited]

Referenced by spot::acss\_statistics::acss\_statistics(), ars\_statistics(), spot::ec\_statistics::ec\_statistics(), spot::unsigned\_statistics::get(), and spot::unsigned\_statistics\_copy::seteq().

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

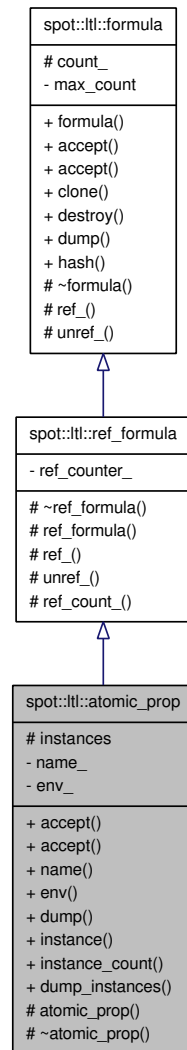
- [tgbaalgorithms/emptiness\\_stats.hh](#)

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

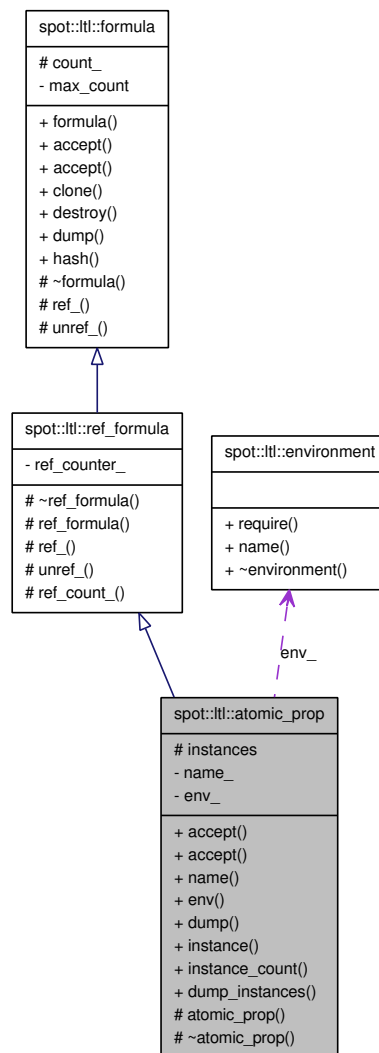
Atomic propositions.

```
#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.
- virtual std::string `dump ()` const

*Return a canonic representation of the atomic proposition.*

- `formula * clone () const`  
*clone this node*
- `void destroy () const`  
*release this node*
- `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.*

### 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.*

### Protected Attributes

- `size_t count_`  
*The hash key of this formula.*

**Static Protected Attributes**

- static [map instances](#)

**Private Attributes**

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

**7.4.1 Detailed Description**

Atomic propositions.

**7.4.2 Member Typedef Documentation**

**7.4.2.1** `typedef std::map<pair, atomic_prop*> spot::ltl::atomic_prop::map` `[protected]`

**7.4.2.2** `typedef std::pair<std::string, environment*> spot::ltl::atomic_prop::pair`  
`[protected]`

**7.4.3 Constructor & Destructor Documentation**

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

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

**7.4.4 Member Function Documentation**

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

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

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

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

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

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

#### 7.4.4.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

#### 7.4.4.4 void spot::ltl::formula::destroy () const [inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.4.4.5 virtual std::string spot::ltl::atomic\_prop::dump () const [virtual]

Return a canonic representation of the atomic proposition.

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

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

List all instances of atomic propositions. For debugging.

#### 7.4.4.7 environment& spot::ltl::atomic\_prop::env () const

Get the environment of the atomic proposition.

#### 7.4.4.8 size\_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

#### 7.4.4.9 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*.



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

Number of instantiated atomic propositions. For debugging.

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

Get the name of the atomic proposition.

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

increment reference counter if any

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

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

Number of references to this formula.

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

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

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

**7.4.5 Member Data Documentation****7.4.5.1** `size_t spot::ltl::formula::count_ [protected, inherited]`

The hash key of this formula.

Referenced by `spot::ltl::formula::hash()`.

**7.4.5.2** `environment* spot::ltl::atomic_prop::env_ [private]`**7.4.5.3** `map spot::ltl::atomic_prop::instances [static, protected]`

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

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

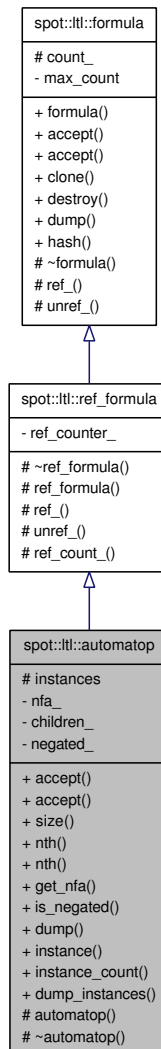
- [ltlast/atomic\\_prop.hh](#)

## 7.5 spot::ltl::automatop Class Reference

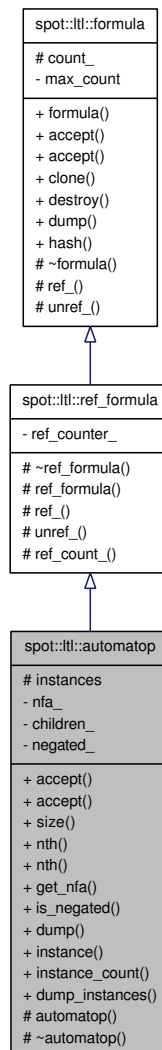
Automaton operators.

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

Inheritance diagram for spot::ltl::automatop:



Collaboration diagram for spot::ltl::automatop:



## Classes

- struct `triplecmp`  
Comparison functor used internally by `ltl::automatop`.

## Public Types

- typedef `std::vector< formula * >` `vec`  
List of formulae.

## 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 argument.*
- const `formula * nth (unsigned n)` const  
*Get the nth argument.*
- `formula * nth (unsigned n)`  
*Get the nth argument.*
- const `spot::ltl::nfa::ptr get_nfa ()` const  
*Get the NFA of this operator.*
- bool `is_negated ()` const  
*Whether the automaton is negated.*
- std::string `dump ()` const  
*Return a canonic representation of the atomic proposition.*
- `formula * clone ()` const  
*clone this node*
- void `destroy ()` const  
*release this node*
- size\_t `hash ()` const  
*Return a hash key for the formula.*

### Static Public Member Functions

- static `automatop * instance (const nfa::ptr nfa, vec *v, bool negated)`  
*Build a spot::ltl::automatop with many children.*
- static unsigned `instance_count ()`  
*Number of instantiated multop operators. For debugging.*
- static std::ostream & `dump_instances (std::ostream &os)`  
*Dump all instances. For debugging.*

### Protected Types

- typedef std::pair< std::pair< `nfa::ptr`, bool >, `vec * >` `triplet`
- typedef std::map< `triplet`, `automatop *`, `tripletcmp` > `map`

### Protected Member Functions

- `automatop` (const `nfa::ptr`, `vec` \*v, bool negated)
- virtual `~automatop` ()
- 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.*

### Protected Attributes

- size\_t `count_`  
*The hash key of this formula.*

### Static Protected Attributes

- static `map instances`

### Private Attributes

- const `nfa::ptr nfa_`
- `vec` \* `children_`
- bool `negated_`

#### 7.5.1 Detailed Description

Automaton operators.

#### 7.5.2 Member Typedef Documentation

**7.5.2.1** `typedef std::map<triplet, automatop*, tripletcmp> spot::ltl::automatop::map`  
[protected]

**7.5.2.2** `typedef std::pair<std::pair<nfa::ptr, bool>, vec*> spot::ltl::automatop::triplet`  
[protected]

### 7.5.2.3 typedef std::vector<formula\*> spot::ltl::automatop::vec

List of formulae.

## 7.5.3 Constructor & Destructor Documentation

### 7.5.3.1 spot::ltl::automatop::automatop (const nfa::ptr, vec \* v, bool *negated*) [protected]

### 7.5.3.2 virtual spot::ltl::automatop::~~automatop () [protected, virtual]

## 7.5.4 Member Function Documentation

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

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

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

### 7.5.4.2 virtual void spot::ltl::automatop::accept (visitor & v) [virtual]

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

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

### 7.5.4.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

### 7.5.4.4 void spot::ltl::formula::destroy () const [inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.5.4.5** `std::string spot::ltl::automatop::dump () const` `[virtual]`

Return a canonic representation of the atomic proposition.

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

**7.5.4.6** `static std::ostream& spot::ltl::automatop::dump_instances (std::ostream & os)`  
`[static]`

Dump all instances. For debugging.

**7.5.4.7** `const spot::ltl::nfa::ptr spot::ltl::automatop::get_nfa () const`

Get the NFA of this operator.

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

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

**7.5.4.9** `static automatop* spot::ltl::automatop::instance (const nfa::ptr nfa, vec * v, bool negated)`  
`[static]`

Build a [spot::ltl::automatop](#) with many children.

This vector is acquired by the [spot::ltl::automatop](#) class, the caller should allocate it with `new`, but not use it (especially not destroy it) after it has been passed to [spot::ltl::automatop](#).

**7.5.4.10** `static unsigned spot::ltl::automatop::instance_count ()` `[static]`

Number of instantiated multop operators. For debugging.

**7.5.4.11** `bool spot::ltl::automatop::is_negated () const`

Whether the automaton is negated.

**7.5.4.12** `formula* spot::ltl::automatop::nth (unsigned n)`

Get the  $n$ th argument.

Starting with  $n = 0$ .

#### 7.5.4.13 `const formula* spot::ltl::automatop::nth (unsigned $n$ ) const`

Get the  $n$ th argument.

Starting with  $n = 0$ .

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

increment reference counter if any

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

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

Number of references to this formula.

#### 7.5.4.16 `unsigned spot::ltl::automatop::size () const`

Get the number of argument.

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

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

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

### 7.5.5 Member Data Documentation

#### 7.5.5.1 `vec* spot::ltl::automatop::children_ [private]`

#### 7.5.5.2 `size_t spot::ltl::formula::count_ [protected, inherited]`

The hash key of this formula.

Referenced by `spot::ltl::formula::hash()`.



7.5.5.3 map spot::ltl::automatop::instances [static, protected]

7.5.5.4 bool spot::ltl::automatop::negated\_ [private]

7.5.5.5 const nfa::ptr spot::ltl::automatop::nfa\_ [private]

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

- [ltlast/automatop.hh](#)

## 7.6 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\\_](#)

### 7.6.1 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 [nrand\(\)](#) or [bmrand\(\)](#).

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.

### 7.6.2 Constructor & Destructor Documentation

7.6.2.1 template<double(\*)() gen> spot::barand< gen >::barand (int  $n$ , double  $p$ ) [inline]

### 7.6.3 Member Function Documentation

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

References `spot::barand< gen >::m_`, `spot::barand< gen >::n_`, and `spot::barand< gen >::s_`.

### 7.6.4 Member Data Documentation

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

Referenced by `spot::barand< gen >::rand()`.

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

Referenced by `spot::barand< gen >::rand()`.

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

Referenced by `spot::barand< gen >::rand()`.

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

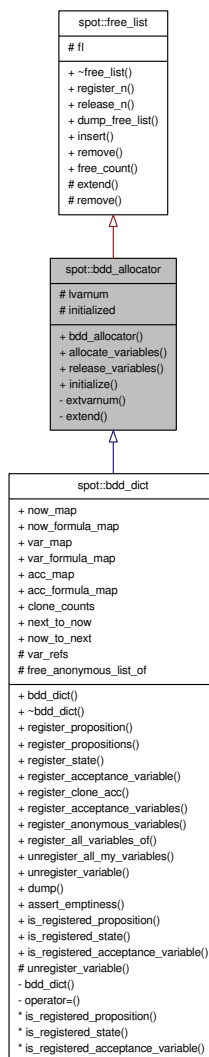
- [misc/random.hh](#)

## 7.7 spot::bdd\_allocator Class Reference

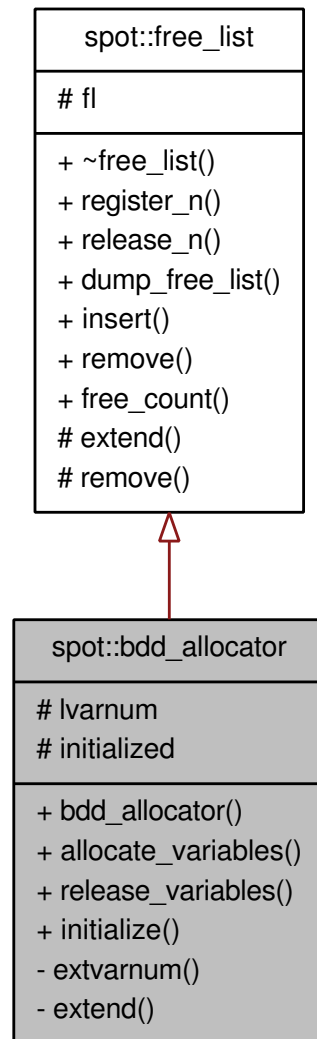
Manage ranges of variables.

```
#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`.*
- int `free_count` () const  
*Return the number of free integers on the list.*

**Private Attributes**

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

**7.7.1 Detailed Description**

Manage ranges of variables.

**7.7.2 Member Typedef Documentation**

**7.7.2.1** `typedef std::list<pos_lenght_pair> spot::free_list::free_list_type [protected, inherited]`

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

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

**7.7.3 Constructor & Destructor Documentation**

**7.7.3.1** `spot::bdd_allocator::bdd_allocator ()`

Default constructor.

**7.7.4 Member Function Documentation**

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

Allocate *n* BDD variables.

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

Dump the list to *os* for debugging.

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

Allocate *n* integer.

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

Implements `spot::free_list`.

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

Require more variables.

#### 7.7.4.5 `int spot::free_list::free_count () const [inherited]`

Return the number of free integers on the list.

#### 7.7.4.6 `static void spot::bdd_allocator::initialize () [static]`

Initialize the BDD library.

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

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

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

Find  $n$  consecutive integers.

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

#### Returns

the first integer of the range

#### 7.7.4.9 `void spot::free_list::release_n (int base, int n) [inherited]`

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

#### 7.7.4.10 `void spot::bdd_allocator::release_variables (int base, int n)`

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

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

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

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

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

### 7.7.5 Member Data Documentation

**7.7.5.1** free\_list\_type spot::free\_list::fl [protected, inherited]

Tracks unused BDD variables.

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

Whether the BDD library has been initialized.

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

number of variables in use in this allocator.

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

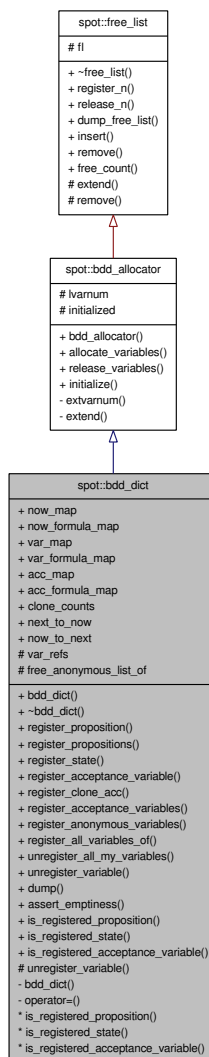
- misc/[bddalloc.hh](#)

## 7.8 spot::bdd\_dict Class Reference

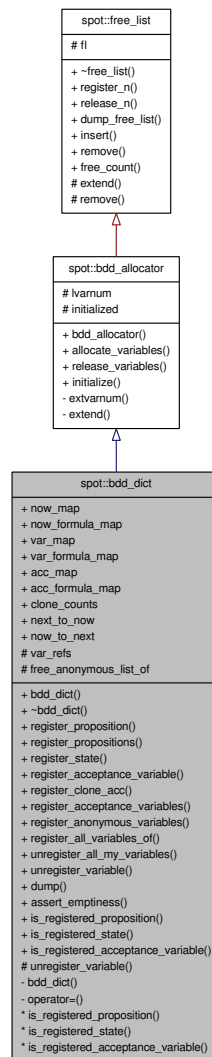
```
#include <tgba/bdddict.hh>
```



Inheritance diagram for spot::bdd\_dict:



Collaboration diagram for spot::bdd\_dict:



## Classes

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

## 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.*
- typedef `std::map< int, int >` [cc\\_map](#)  
*Clone counts.*

**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.*
- [int register\\_clone\\_acc](#) (int var, const void \*for\_me)  
*Clone an acceptance variable VAR for FOR\_ME.*
- [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\\_emptyiness](#) () 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.*
- [cc\\_map](#) [clone\\_counts](#)
- [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 \*, [anon\\_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)

#### 7.8.1 Detailed Description

Map BDD variables to formulae.

#### 7.8.2 Member Typedef Documentation

##### 7.8.2.1 `typedef std::map<int, int> spot::bdd_dict::cc_map`

Clone counts.

##### 7.8.2.2 `typedef std::map<const void*, anon_free_list> spot::bdd_dict::free_anonymous_list_of_type [protected]`

List of unused anonymous variable number for each automaton.

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

Formula-to-BDD-variable maps.

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

BDD-variable reference counts.

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

BDD-variable-to-formula maps.

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

## 7.8.3 Constructor & Destructor Documentation

### 7.8.3.1 `spot::bdd_dict::bdd_dict ()`

### 7.8.3.2 `spot::bdd_dict::~~bdd_dict ()`

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

## 7.8.4 Member Function Documentation

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

Allocate *n* BDD variables.

### 7.8.4.2 `void spot::bdd_dict::assert_emptyness () const`

Make sure the dictionary is empty.

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

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

Dump all variables for debugging.

#### Parameters

*os* The output stream.

**7.8.4.4** `static void spot::bdd_allocator::initialize ()` [`static`, `inherited`]

Initialize the BDD library.

**7.8.4.5** `bool spot::bdd_dict::is_registered_acceptance_variable (const ltl::formula *f, const void *by_me)`**7.8.4.6** `bool spot::bdd_dict::is_registered_proposition (const ltl::formula *f, const void *by_me)`

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

**7.8.4.7** `bool spot::bdd_dict::is_registered_state (const ltl::formula *f, const void *by_me)`**7.8.4.8** `bdd_dict& spot::bdd_dict::operator= (const bdd_dict &other)` [`private`]**7.8.4.9** `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.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

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

Register BDD variables as acceptance variables.

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

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

Duplicate the variable usage of another object.

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

#### 7.8.4.12 `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.

#### 7.8.4.13 `int spot::bdd_dict::register_clone_acc (int var, const void *for_me)`

Clone an acceptance variable VAR for FOR\_ME.

This is used in products TGBAs when both operands share the same acceptance variables but they need to be distinguished in the result.

#### 7.8.4.14 `int spot::bdd_dict::register_proposition (const ltl::formula *f, const void *for_me)`

Register an atomic proposition.

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

##### Returns

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

#### 7.8.4.15 `void spot::bdd_dict::register_propositions (bdd f, const void *for_me)`

Register BDD variables as atomic propositions.

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



**7.8.4.16** `int spot::bdd_dict::register_state (const ltl::formula *f, const void *for_me)`

Register a couple of Now/Next variables.

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

**Returns**

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

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

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

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

Release all variables used by an object.

Usually called in the destructor if *me*.

**7.8.4.19** `void spot::bdd_dict::unregister_variable (vr_map::iterator &cur, const void *me)`  
**[protected]****7.8.4.20** `void spot::bdd_dict::unregister_variable (int var, const void *me)`

Release a variable used by *me*.

**7.8.5 Member Data Documentation****7.8.5.1** `vf_map spot::bdd_dict::acc_formula_map`

Maps BDD variables to acceptance conditions.

**7.8.5.2** `fv_map spot::bdd_dict::acc_map`

Maps acceptance conditions to BDD variables.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

### 7.8.5.3 `cc_map spot::bdd_dict::clone_counts`

### 7.8.5.4 `free_anonymous_list_of_type spot::bdd_dict::free_anonymous_list_of` `[protected]`

### 7.8.5.5 `bool spot::bdd_allocator::initialized` `[static, protected, inherited]`

Whether the BDD library has been initialized.

### 7.8.5.6 `int spot::bdd_allocator::lvarnum` `[protected, inherited]`

number of variables in use in this allocator.

### 7.8.5.7 `bddPair* spot::bdd_dict::next_to_now`

Map Next variables to Now variables.

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

### 7.8.5.8 `vf_map spot::bdd_dict::now_formula_map`

Maps "Now" BDD variables to formulae.

### 7.8.5.9 `fv_map spot::bdd_dict::now_map`

Maps formulae to "Now" BDD variables.

### 7.8.5.10 `bddPair* spot::bdd_dict::now_to_next`

Map Now variables to Next variables.

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

### 7.8.5.11 `vf_map spot::bdd_dict::var_formula_map`

Maps BDD variables to atomic propositions.

#### 7.8.5.12 fv\_map spot::bdd\_dict::var\_map

Maps atomic propositions to BDD variables.

#### 7.8.5.13 vr\_map spot::bdd\_dict::var\_refs [protected]

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

- [tgba/bdddict.hh](#)

## 7.9 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

#### 7.9.1 Detailed Description

Comparison functor for BDDs.

#### 7.9.2 Member Function Documentation

##### 7.9.2.1 bool spot::bdd\_less\_than::operator() (const bdd & left, const bdd & right) const [inline]

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

- [misc/bddlt.hh](#)

## 7.10 spot::bdd\_ordered Class Reference

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

### Public Member Functions

- [bdd\\_ordered](#) ()
- [bdd\\_ordered](#) (int [bdd\\_](#), unsigned [order\\_](#))
- unsigned [order](#) () const
- unsigned & [order](#) ()
- bdd [get\\_bdd](#) () const

### Private Attributes

- int [bdd\\_](#)
- unsigned [order\\_](#)

### 7.10.1 Constructor & Destructor Documentation

**7.10.1.1** spot::bdd\_ordered::bdd\_ordered () [[inline](#)]

**7.10.1.2** spot::bdd\_ordered::bdd\_ordered (int *bdd\_*, unsigned *order\_*) [[inline](#)]

### 7.10.2 Member Function Documentation

**7.10.2.1** bdd spot::bdd\_ordered::get\_bdd () const [[inline](#)]

References [order\\_](#).

**7.10.2.2** unsigned& spot::bdd\_ordered::order () [[inline](#)]

References [order\\_](#).

**7.10.2.3** unsigned spot::bdd\_ordered::order () const [[inline](#)]

### 7.10.3 Member Data Documentation

**7.10.3.1** int spot::bdd\_ordered::bdd\_ [[private](#)]

**7.10.3.2** unsigned spot::bdd\_ordered::order\_ [[private](#)]

Referenced by [get\\_bdd\(\)](#), and [order\(\)](#).

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

- [tgba/tgbakvcomplement.hh](#)

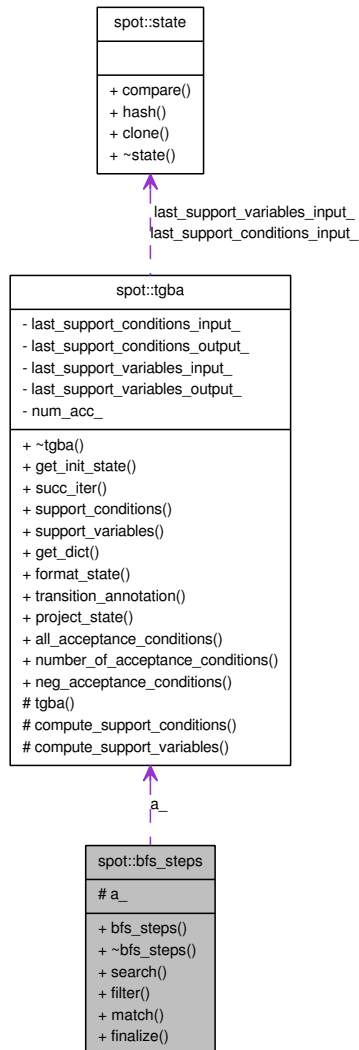
## 7.11 spot::bfs\_steps Class Reference

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.

```
#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.*

### 7.11.1 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()`.

### 7.11.2 Constructor & Destructor Documentation

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

#### 7.11.2.2 virtual `spot::bfs_steps::~bfs_steps` () **[virtual]**

### 7.11.3 Member Function Documentation

#### 7.11.3.1 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.

**7.11.3.2** `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).

**7.11.3.3** `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.

**7.11.3.4** `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.

### 7.11.4 Member Data Documentation

**7.11.4.1** `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:

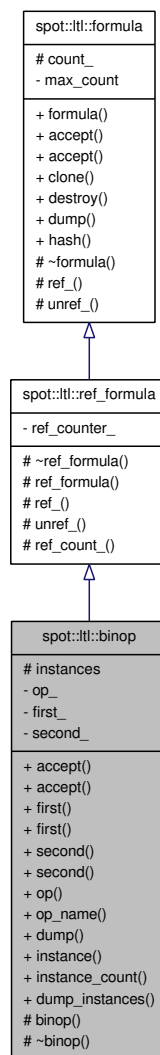
- [tgbaalgos/bfssteps.hh](#)

## 7.12 spot::ltl::binop Class Reference

Binary operator.

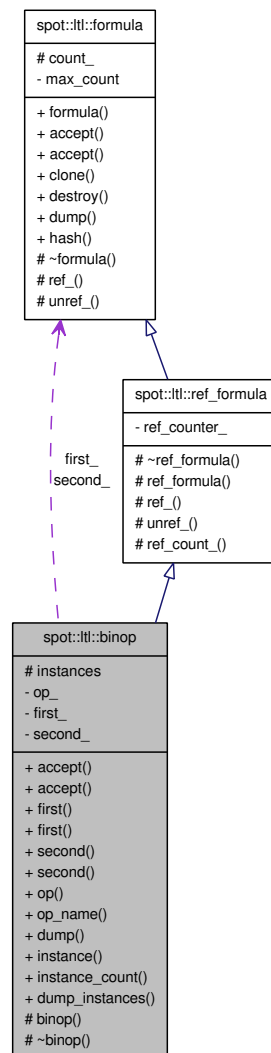
```
#include <ltlast/binop.hh>
```

Inheritance diagram for spot::ltl::binop:





Collaboration diagram for spot::ltl::binop:



## Public Types

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

## 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.*
- `virtual std::string dump () const`  
*Return a canonic representation of the atomic proposition.*
- `formula * clone () const`  
*clone this node*
- `void destroy () const`  
*release this node*
- `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 std::ostream & dump_instances (std::ostream &os)`  
*Dump all instances. For debugging.*

### Protected Types

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

### Protected Attributes

- `size_t` `count_`  
*The hash key of this formula.*

### Static Protected Attributes

- static `map` `instances`

### Private Attributes

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

#### 7.12.1 Detailed Description

Binary operator.

#### 7.12.2 Member Typedef Documentation

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

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

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

### 7.12.3 Member Enumeration Documentation

#### 7.12.3.1 enum spot::ltl::binop::type

Different kinds of binary operators

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

**Enumerator:**

*Xor*

*Implies*

*Equiv*

*U*

*R*

*W*

*M*

### 7.12.4 Constructor & Destructor Documentation

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

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

### 7.12.5 Member Function Documentation

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

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

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

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

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

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

#### 7.12.5.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

**7.12.5.4 void spot::ltl::formula::destroy () const [inherited]**

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.12.5.5 virtual std::string spot::ltl::binop::dump () const [virtual]**

Return a canonic representation of the atomic proposition.

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

**7.12.5.6 static std::ostream& spot::ltl::binop::dump\_instances (std::ostream & os) [static]**

Dump all instances. For debugging.

**7.12.5.7 formula\* spot::ltl::binop::first ()**

Get the first operand.

**7.12.5.8 const formula\* spot::ltl::binop::first () const**

Get the first operand.

**7.12.5.9 size\_t spot::ltl::formula::hash () const [inline, inherited]**

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

**7.12.5.10 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*.

**7.12.5.11 static unsigned spot::ltl::binop::instance\_count () [static]**

Number of instantiated binary operators. For debugging.

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

Get the type of this operator.

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

Get the type of this operator, as a string.

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

increment reference counter if any

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

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

Number of references to this formula.

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

Get the second operand.

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

Get the second operand.

**7.12.5.18** `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](#).

**7.12.6** Member Data Documentation**7.12.6.1** `size_t spot::ltl::formula::count_ [protected, inherited]`

The hash key of this formula.

Referenced by `spot::ltl::formula::hash()`.

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

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

**7.12.6.4** `type spot::ltl::binop::op_ [private]`

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

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

- [ltlast/binop.hh](#)

## 7.13 spot::char\_ptr\_less\_than Struct Reference

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*`.

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

### Public Member Functions

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

#### 7.13.1 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;
```

#### 7.13.2 Member Function Documentation

**7.13.2.1** `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](#)

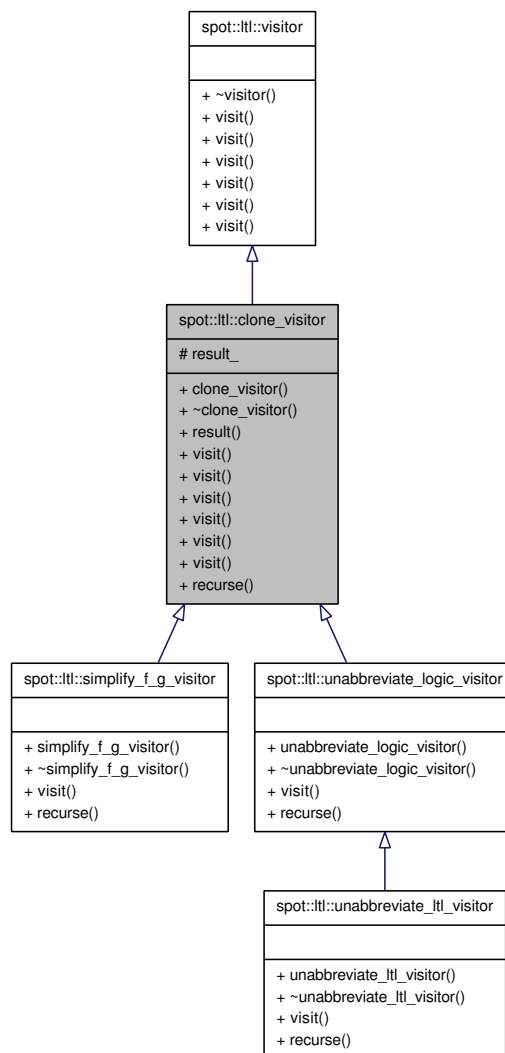
## 7.14 spot::ltl::clone\_visitor Class Reference

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::formula::clone](#) instead, it is way faster.

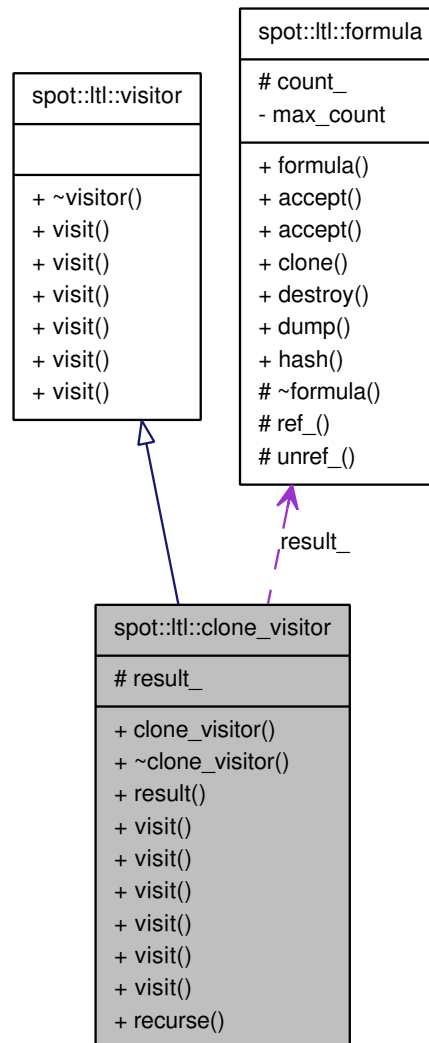
```
#include <ltlvisit/clone.hh>
```

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 (automatop *mo)`
- `void visit (multop *mo)`
- `void visit (constant *c)`
- `virtual formula * recurse (formula *f)`

**Protected Attributes**

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

**7.14.1 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::formula::clone](#) instead, it is way faster.

**7.14.2 Constructor & Destructor Documentation****7.14.2.1 spot::ltl::clone\_visitor::clone\_visitor ()****7.14.2.2 virtual spot::ltl::clone\_visitor::~~clone\_visitor () [virtual]****7.14.3 Member Function Documentation****7.14.3.1 virtual formula\* spot::ltl::clone\_visitor::recurse (formula \*f) [virtual]**

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

**7.14.3.2 formula\* spot::ltl::clone\_visitor::result () const****7.14.3.3 void spot::ltl::clone\_visitor::visit (constant \*c) [virtual]**

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

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

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

**7.14.3.5 void spot::ltl::clone\_visitor::visit (automatop \*mo) [virtual]**

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

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

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

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

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

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

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

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

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

### 7.14.4 Member Data Documentation

#### 7.14.4.1 `formula* spot::ltl::clone_visitor::result_ [protected]`

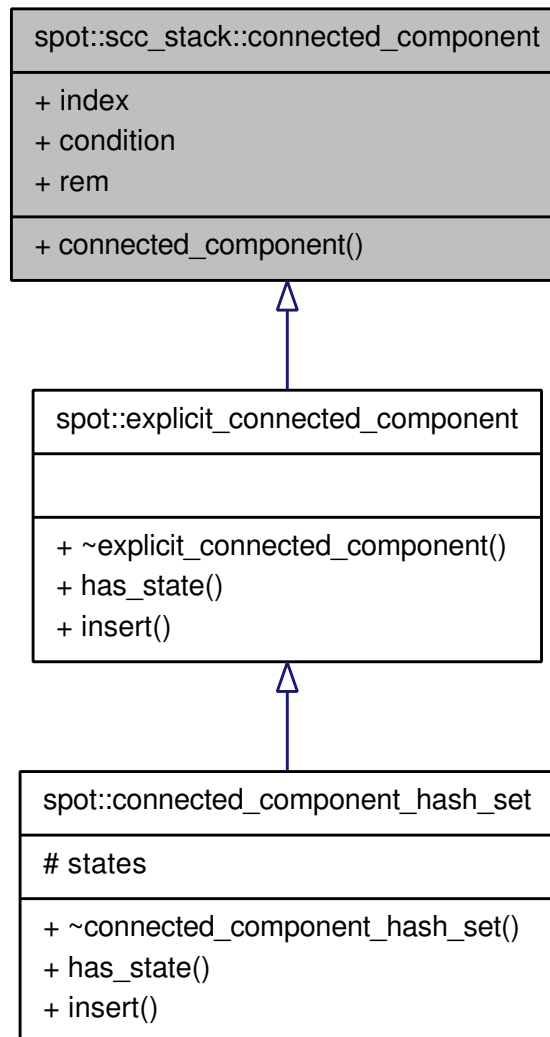
The documentation for this class was generated from the following file:

- [ltlvisit/clone.hh](#)

## 7.15 `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`

### 7.15.1 Constructor & Destructor Documentation

#### 7.15.1.1 spot::scc\_stack::connected\_component::connected\_component (int *index* = -1)

### 7.15.2 Member Data Documentation

#### 7.15.2.1 bdd spot::scc\_stack::connected\_component::condition

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

#### 7.15.2.2 int spot::scc\_stack::connected\_component::index

Index of the SCC.

#### 7.15.2.3 std::list<const state\*> spot::scc\_stack::connected\_component::rem

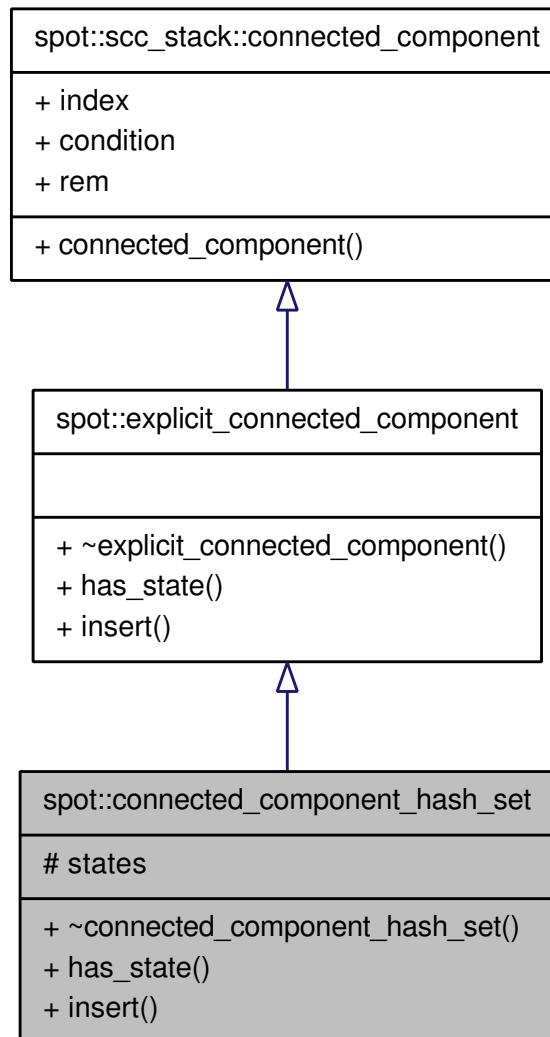
The documentation for this struct was generated from the following file:

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

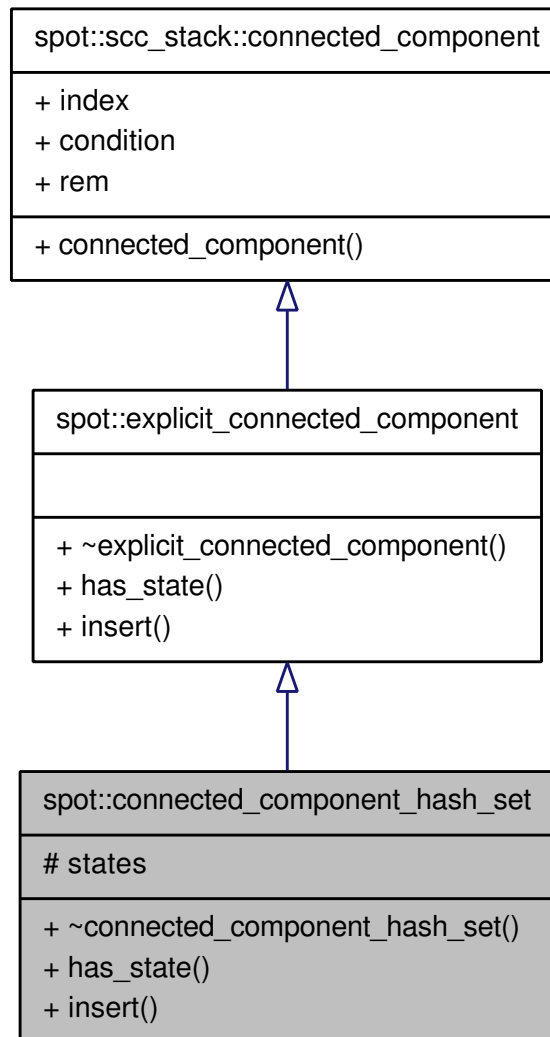
## 7.16 spot::connected\_component\_hash\_set Class Reference

```
#include <tgbaalgos/gtec/explscch.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](#)

#### 7.16.1 Detailed Description

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

#### 7.16.2 Member Typedef Documentation

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

#### 7.16.3 Constructor & Destructor Documentation

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

#### 7.16.4 Member Function Documentation

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

Check if the SCC contains states *s*.

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

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

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

Insert a new state in the SCC.

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



### 7.16.5 Member Data Documentation

#### 7.16.5.1 bdd spot::scc\_stack::connected\_component::condition [inherited]

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

#### 7.16.5.2 int spot::scc\_stack::connected\_component::index [inherited]

Index of the SCC.

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

#### 7.16.5.4 set\_type spot::connected\_component\_hash\_set::states [protected]

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

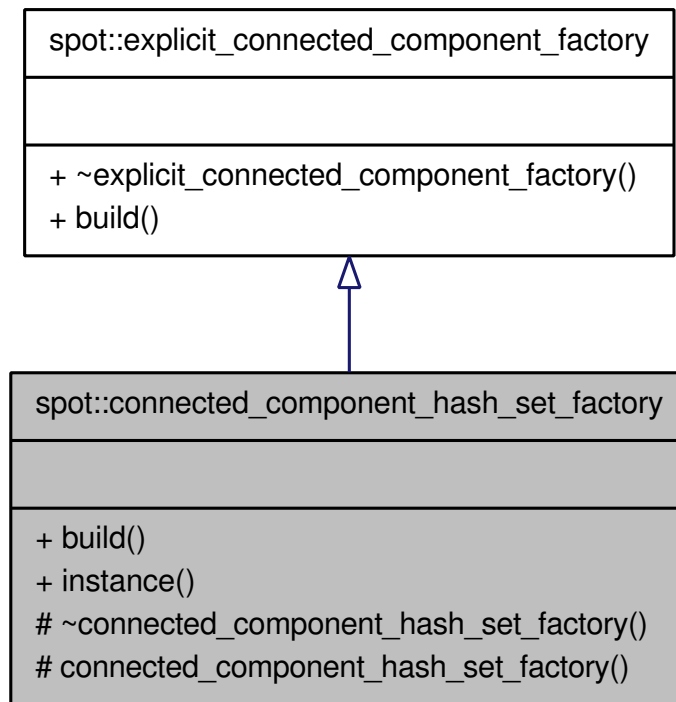
- [tgbaalgos/gtec/explsc.hh](#)

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

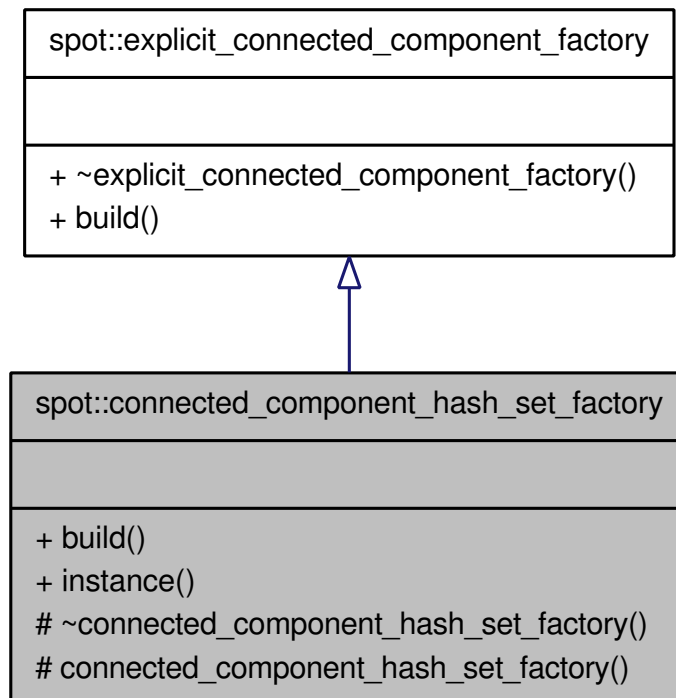
Factory for [connected\\_component\\_hash\\_set](#).

```
#include <tgbaalgos/gtec/explsc.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.

#### 7.17.1 Detailed Description

Factory for [connected\\_component\\_hash\\_set](#). This class is a singleton. Retrieve the instance using [instance\(\)](#).

#### 7.17.2 Constructor & Destructor Documentation

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

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

Construction is forbidden.

#### 7.17.3 Member Function Documentation

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

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

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

### 7.17.3.2 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/explscs.hh](#)

## 7.18 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
- virtual void [visit](#) (const [automatop](#) \*node)=0

### 7.18.1 Detailed Description

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

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

### 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 virtual spot::ltl::const\_visitor::~const\_visitor() [inline, virtual]

### 7.18.3 Member Function Documentation

#### 7.18.3.1 virtual void spot::ltl::const\_visitor::visit (const automatop \* node) [pure virtual]

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

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

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

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

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

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

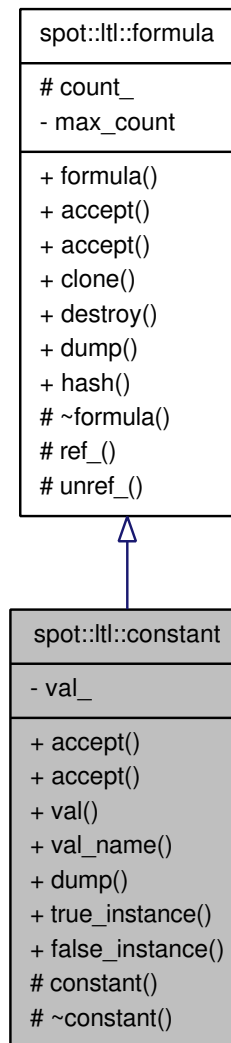
- ltlast/[visitor.hh](#)

## 7.19 spot::ltl::constant Class Reference

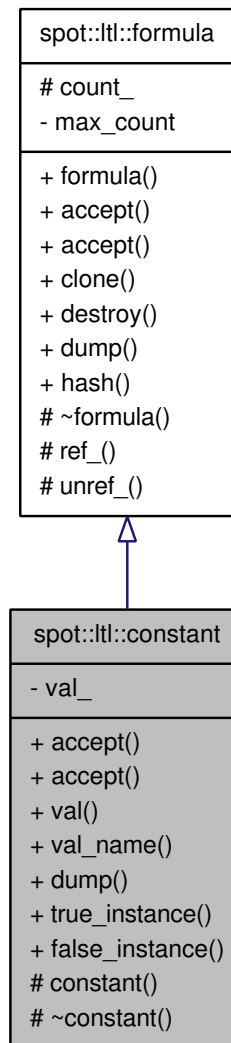
A constant (True or False).

```
#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.*
- `virtual std::string dump () const`  
*Return a canonic representation of the formula.*
- `formula * clone () const`  
*clone this node*
- `void destroy () const`  
*release this node*
- `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)`.*

### 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).*

### Protected Attributes

- `size_t count_`  
*The hash key of this formula.*

### Private Attributes

- `type val_`



### 7.19.1 Detailed Description

A constant (True or False).

### 7.19.2 Member Enumeration Documentation

#### 7.19.2.1 enum spot::ltl::constant::type

Enumerator:

*False*

*True*

### 7.19.3 Constructor & Destructor Documentation

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

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

### 7.19.4 Member Function Documentation

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

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

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

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

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

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

#### 7.19.4.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

**7.19.4.4 void spot::ltl::formula::destroy () const [inherited]**

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.19.4.5 virtual std::string spot::ltl::constant::dump () const [virtual]**

Return a canonic representation of the formula.

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

**7.19.4.6 static constant\* spot::ltl::constant::false\_instance () [static]**

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

**7.19.4.7 size\_t spot::ltl::formula::hash () const [inline, inherited]**

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

**7.19.4.8 virtual void spot::ltl::formula::ref\_ () [protected, virtual, inherited]**

increment reference counter if any

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

**7.19.4.9 static constant\* spot::ltl::constant::true\_instance () [static]**

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

**7.19.4.10 virtual bool spot::ltl::formula::unref\_ () [protected, virtual, inherited]**

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

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

#### 7.19.4.11 type spot::ltl::constant::val () const

Return the value of the constant.

#### 7.19.4.12 const char\* spot::ltl::constant::val\_name () const

Return the value of the constant as a string.

### 7.19.5 Member Data Documentation

#### 7.19.5.1 size\_t spot::ltl::formula::count\_ [protected, inherited]

The hash key of this formula.

Referenced by spot::ltl::formula::hash().

#### 7.19.5.2 type spot::ltl::constant::val\_ [private]

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

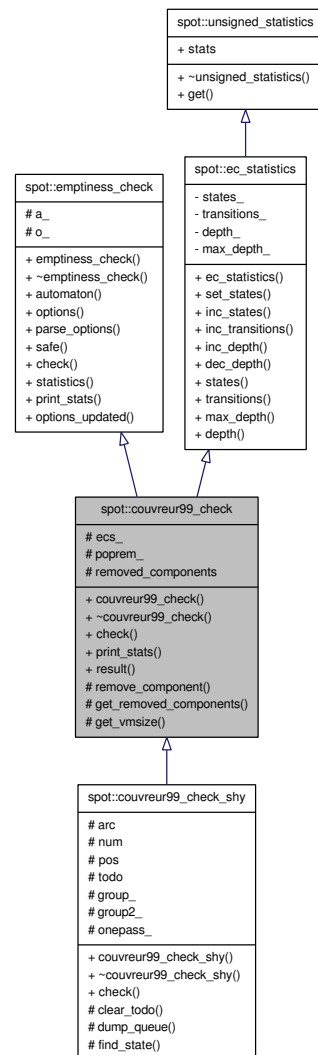
- ltlast/[constant.hh](#)

## 7.20 spot::couvreur99\_check Class Reference

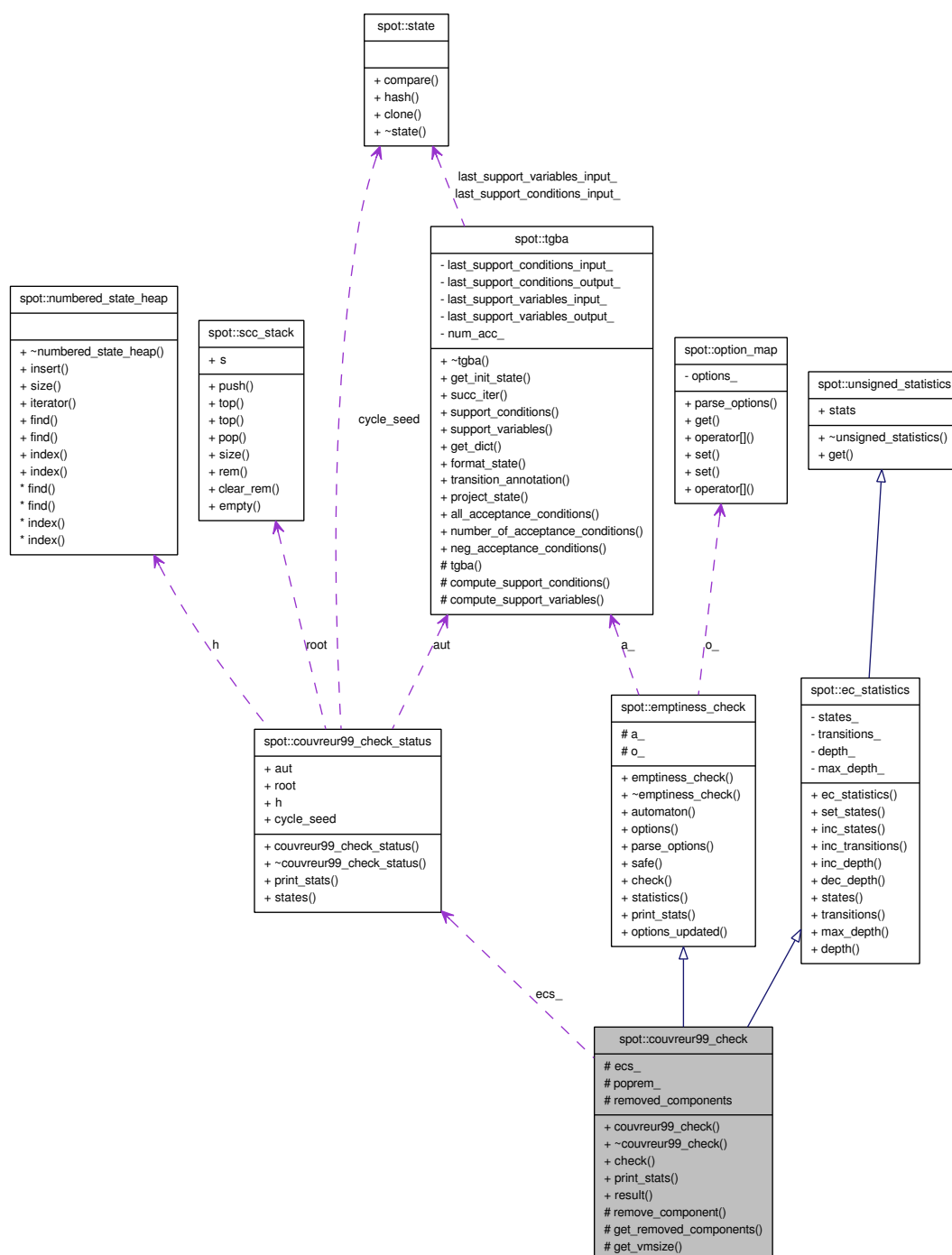
An implementation of the Couvreur99 emptiness-check algorithm.

```
#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.*
- unsigned [get\\_removed\\_components](#) () const
- unsigned [get\\_vmsize](#) () const

### Protected Attributes

- [couvreur99\\_check\\_status](#) \* [ecs\\_](#)
- bool [poprem\\_](#)  
*Whether to store the state to be removed.*
- unsigned [removed\\_components](#)  
*Number of dead SCC removed by the algorithm.*
- const [tgba](#) \* [a\\_](#)  
*The automaton.*
- [option\\_map](#) [o\\_](#)  
*The options.*

#### 7.20.1 Detailed Description

An implementation of the Couvreur99 emptiness-check algorithm. See the documentation for [spot::couvreur99](#).

#### 7.20.2 Member Typedef Documentation

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

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

#### 7.20.3 Constructor & Destructor Documentation

**7.20.3.1** `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 ())`

### 7.20.3.2 virtual spot::couvreur99\_check::~~couvreur99\_check () [virtual]

## 7.20.4 Member Function Documentation

### 7.20.4.1 const tgba\* spot::emptiness\_check::automaton () const [inline, inherited]

The automaton that this emptiness-check inspects.

References [spot::emptiness\\_check::a\\_](#).

### 7.20.4.2 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](#).

### 7.20.4.3 void spot::ec\_statistics::dec\_depth (unsigned *n* = 1) [inline, inherited]

References [spot::ec\\_statistics::depth\\_](#).

### 7.20.4.4 unsigned spot::ec\_statistics::depth () const [inline, inherited]

References [spot::ec\\_statistics::depth\\_](#).

### 7.20.4.5 unsigned spot::unsigned\_statistics::get (const char \* *str*) const [inline, inherited]

References [spot::unsigned\\_statistics::stats](#).

### 7.20.4.6 unsigned spot::couvreur99\_check::get\_removed\_components () const [protected]

### 7.20.4.7 unsigned spot::couvreur99\_check::get\_vmsize () const [protected]



**7.20.4.8 void spot::ec\_statistics::inc\_depth (unsigned  $n = 1$ ) [inline, inherited]**

References spot::ec\_statistics::depth\_, and spot::ec\_statistics::max\_depth\_.

**7.20.4.9 void spot::ec\_statistics::inc\_states () [inline, inherited]**

References spot::ec\_statistics::states\_.

**7.20.4.10 void spot::ec\_statistics::inc\_transitions () [inline, inherited]**

References spot::ec\_statistics::transitions\_.

**7.20.4.11 unsigned spot::ec\_statistics::max\_depth () const [inline, inherited]**

References spot::ec\_statistics::max\_depth\_.

Referenced by spot::ec\_statistics::ec\_statistics().

**7.20.4.12 const option\_map& spot::emptiness\_check::options () const [inline, inherited]**

Return the options parametrizing how the emptiness check is realized.

References spot::emptiness\_check::o\_.

**7.20.4.13 virtual void spot::emptiness\_check::options\_updated (const option\_map & old) [virtual, inherited]**

Notify option updates.

**7.20.4.14 const char\* spot::emptiness\_check::parse\_options (char \* options) [inherited]**

Modify the algorithm options.

**7.20.4.15 virtual std::ostream& spot::couvreur99\_check::print\_stats (std::ostream & os) const [virtual]**

Print statistics, if any.

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

**7.20.4.16 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.

**7.20.4.17 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.

**7.20.4.18 virtual bool spot::emptiness\_check::safe () const [virtual, inherited]**

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

**7.20.4.19 void spot::ec\_statistics::set\_states (unsigned *n*) [inline, inherited]**

References `spot::ec_statistics::states_`.

**7.20.4.20 unsigned spot::ec\_statistics::states () const [inline, inherited]**

References `spot::ec_statistics::states_`.

Referenced by `spot::ec_statistics::ec_statistics()`.

**7.20.4.21 virtual const unsigned\_statistics\* spot::emptiness\_check::statistics () const  
[virtual, inherited]**

Return statistics, if available.

**7.20.4.22 unsigned spot::ec\_statistics::transitions () const [inline, inherited]**

References `spot::ec_statistics::transitions_`.

Referenced by `spot::ec_statistics::ec_statistics()`.

### 7.20.5 Member Data Documentation

#### 7.20.5.1 `const tgba* spot::emptiness_check::a_` `[protected, inherited]`

The automaton.

Referenced by `spot::emptiness_check::automaton()`.

#### 7.20.5.2 `couvreur99_check_status* spot::couvreur99_check::ecs_` `[protected]`

#### 7.20.5.3 `option_map spot::emptiness_check::o_` `[protected, inherited]`

The options.

Referenced by `spot::emptiness_check::options()`.

#### 7.20.5.4 `bool spot::couvreur99_check::poprem_` `[protected]`

Whether to store the state to be removed.

#### 7.20.5.5 `unsigned spot::couvreur99_check::removed_components` `[protected]`

Number of dead SCC removed by the algorithm.

#### 7.20.5.6 `stats_map spot::unsigned_statistics::stats` `[inherited]`

Referenced by `spot::acss_statistics::acss_statistics()`, `spot::ars_statistics::ars_statistics()`, `spot::ec_statistics::ec_statistics()`, `spot::unsigned_statistics::get()`, and `spot::unsigned_statistics_copy::seteq()`.

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

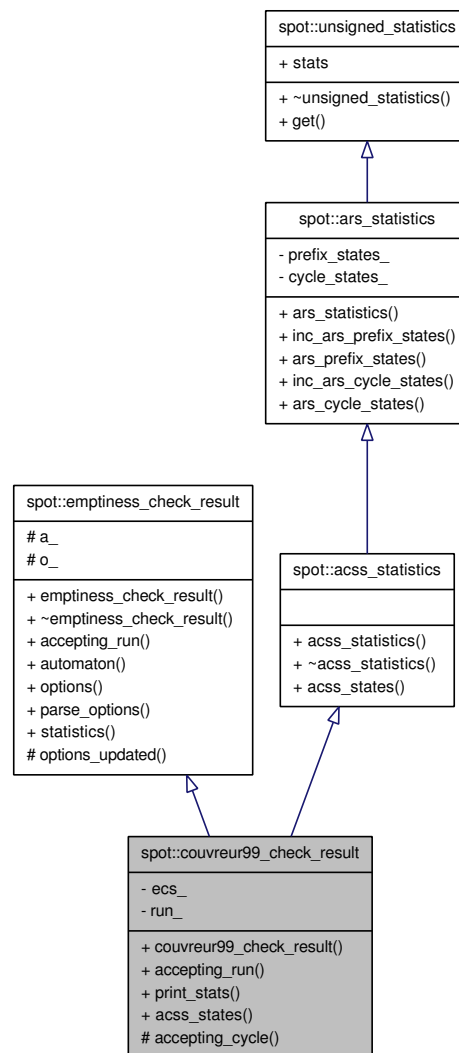
- `tgbaalgos/gtec/gtec.hh`

## 7.21 `spot::couvreur99_check_result` Class Reference

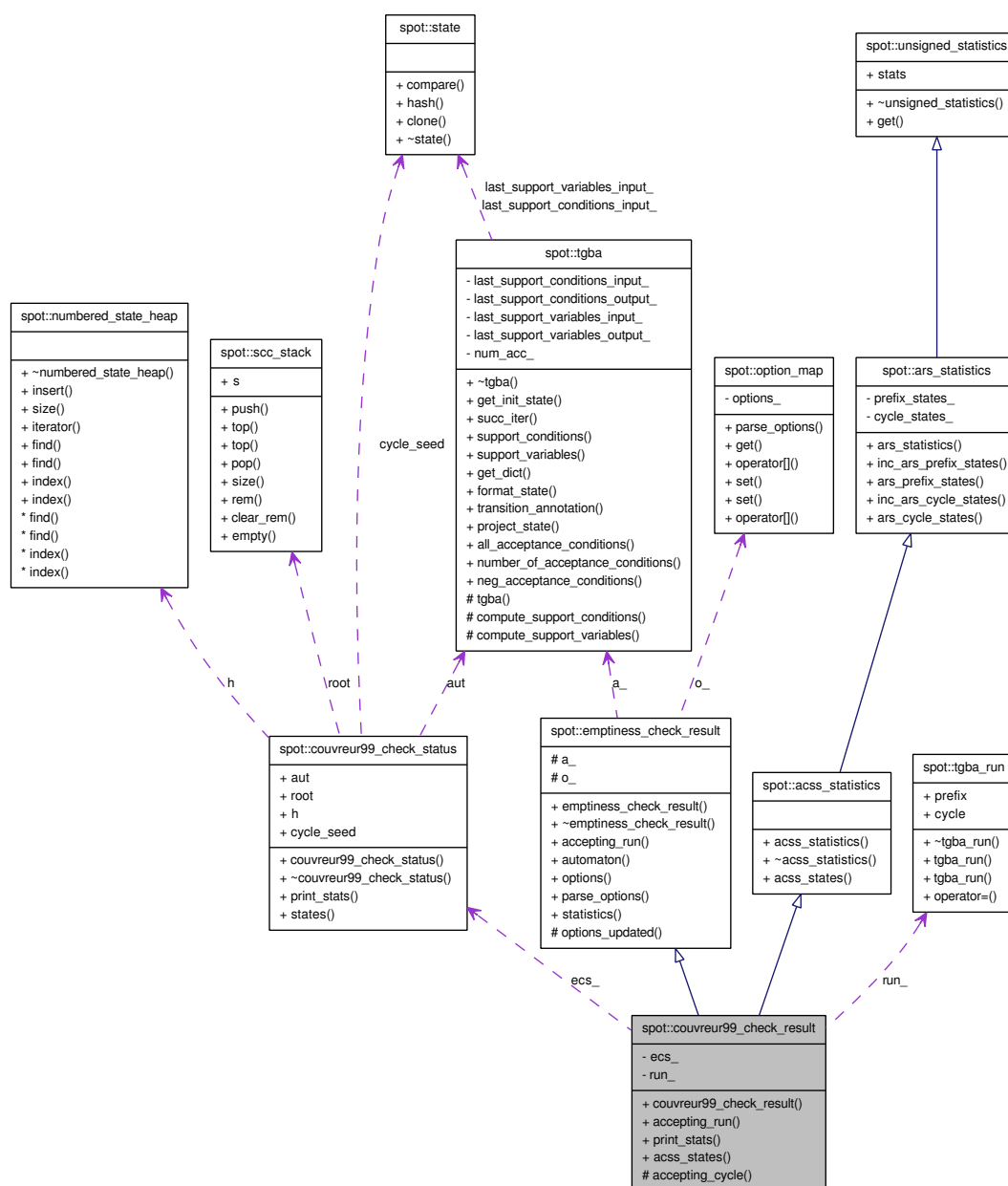
Compute a counter example from a `spot::couvreur99_check_status`.

```
#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 [couvreur99\\_check\\_status](#) \* [ecs\\_](#)
- [tgba\\_run](#) \* [run\\_](#)

### 7.21.1 Detailed Description

Compute a counter example from a [spot::couvreur99\\_check\\_status](#).

### 7.21.2 Member Typedef Documentation

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

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

### 7.21.3 Constructor & Destructor Documentation

**7.21.3.1** `spot::couvreur99_check_result::couvreur99_check_result (const  
couvreur99_check_status * ecs, option_map o = option_map ())`

### 7.21.4 Member Function Documentation

**7.21.4.1** `void spot::couvreur99_check_result::accepting_cycle () [protected]`

Called by [accepting\\_run\(\)](#) to find a cycle which traverses all acceptance conditions in the accepted SCC.

**7.21.4.2** `virtual tgba_run* spot::couvreur99_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](#).

**7.21.4.3** `virtual unsigned spot::couvreur99_check_result::acss_states () const [virtual]`

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

Implements [spot::acss\\_statistics](#).

**7.21.4.4 unsigned spot::ars\_statistics::ars\_cycle\_states () const [inline, inherited]**

References spot::ars\_statistics::cycle\_states\_.

Referenced by spot::ars\_statistics::ars\_statistics().

**7.21.4.5 unsigned spot::ars\_statistics::ars\_prefix\_states () const [inline, inherited]**

References spot::ars\_statistics::prefix\_states\_.

Referenced by spot::ars\_statistics::ars\_statistics().

**7.21.4.6 const tgba\* spot::emptiness\_check\_result::automaton () const [inline, inherited]**

The automaton on which an [accepting\\_run\(\)](#) was found.

References spot::emptiness\_check\_result::a\_.

**7.21.4.7 unsigned spot::unsigned\_statistics::get (const char \* str) const [inline, inherited]**

References spot::unsigned\_statistics::stats.

**7.21.4.8 void spot::ars\_statistics::inc\_ars\_cycle\_states () [inline, inherited]**

References spot::ars\_statistics::cycle\_states\_.

**7.21.4.9 void spot::ars\_statistics::inc\_ars\_prefix\_states () [inline, inherited]**

References spot::ars\_statistics::prefix\_states\_.

**7.21.4.10 const option\_map& spot::emptiness\_check\_result::options () const [inline, inherited]**

Return the options parametrizing how the accepting run is computed.

References spot::emptiness\_check\_result::o\_.



**7.21.4.11** virtual void spot::emptiness\_check\_result::options\_updated (const option\_map & *old*)  
[protected, virtual, inherited]

Notify option updates.

**7.21.4.12** const char\* spot::emptiness\_check\_result::parse\_options (char \* *options*)  
[inherited]

Modify the algorithm options.

**7.21.4.13** void spot::couvreur99\_check\_result::print\_stats (std::ostream & *os*) const

**7.21.4.14** virtual const unsigned\_statistics\* spot::emptiness\_check\_result::statistics () const  
[virtual, inherited]

Return statistics, if available.

## 7.21.5 Member Data Documentation

**7.21.5.1** const tgba\* spot::emptiness\_check\_result::a\_ [protected, inherited]

The automaton.

Referenced by spot::emptiness\_check\_result::automaton().

**7.21.5.2** const couvreur99\_check\_status\* spot::couvreur99\_check\_result::ecs\_ [private]

**7.21.5.3** option\_map spot::emptiness\_check\_result::o\_ [protected, inherited]

The options.

Referenced by spot::emptiness\_check\_result::options().

**7.21.5.4** tgba\_run\* spot::couvreur99\_check\_result::run\_ [private]

## 7.21.5.5 stats\_map spot::unsigned\_statistics::stats [inherited]

Referenced by `spot::acss_statistics::acss_statistics()`, `spot::ars_statistics::ars_statistics()`, `spot::ec_statistics::ec_statistics()`, `spot::unsigned_statistics::get()`, and `spot::unsigned_statistics_copy::seteq()`.

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

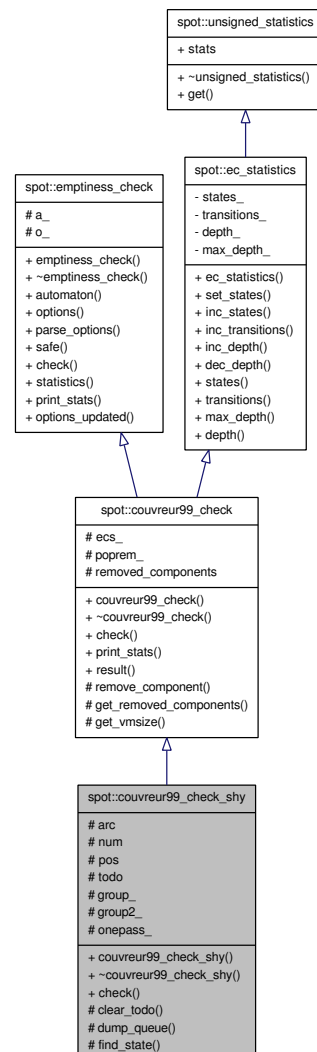
- `tgbaalgos/gtec/ce.hh`

## 7.22 spot::couvreur99\_check\_shy Class Reference

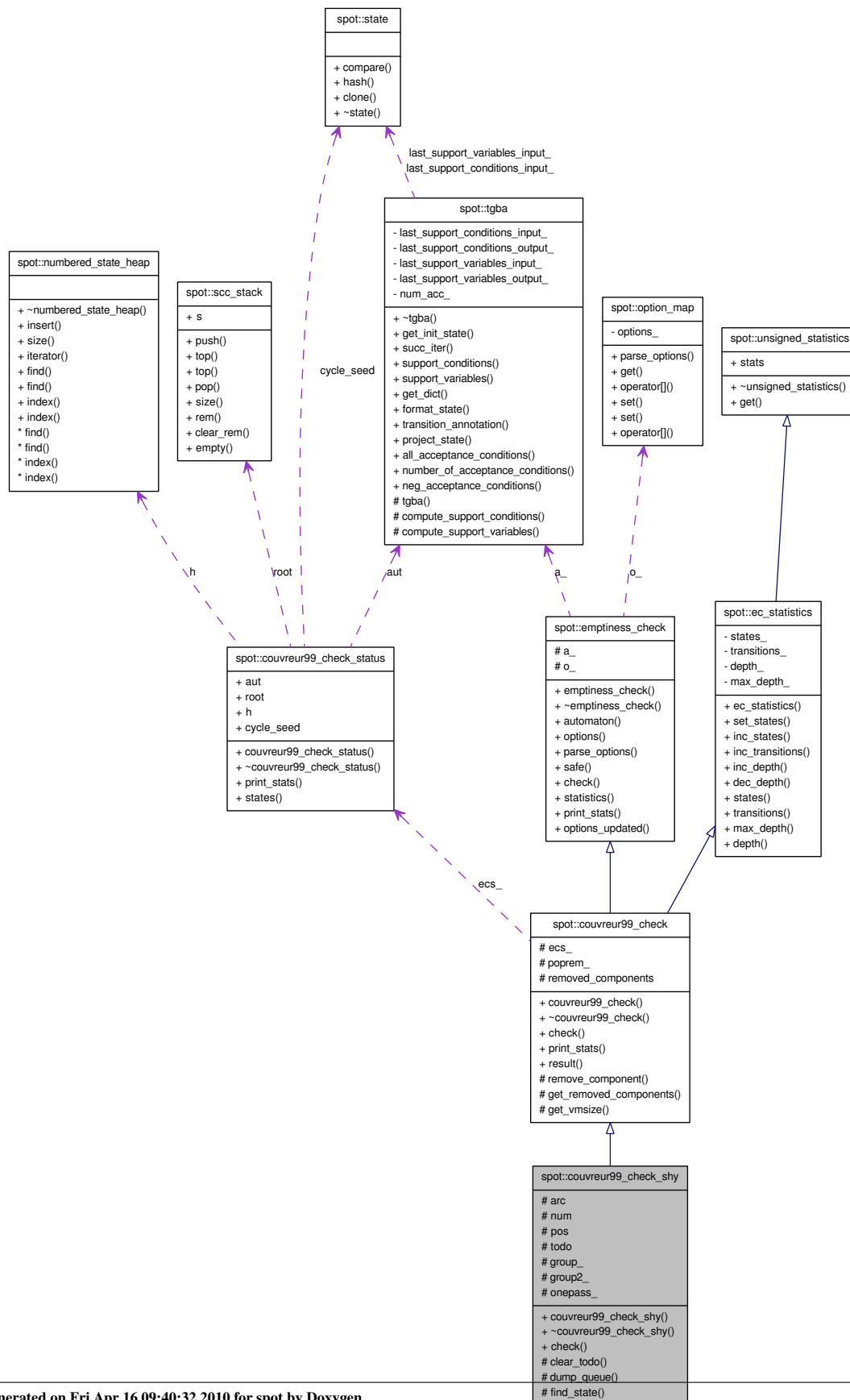
A version of `spot::couvreur99_check` that tries to visit known states first.

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

Inheritance diagram for `spot::couvreur99_check_shy`:



Collaboration diagram for spot::couvreur99\_check\_shy:



## Classes

- struct [successor](#)
- struct [todo\\_item](#)

## 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](#) ()
- void [dump\\_queue](#) (std::ostream &os=std::cerr)  
*Dump the queue for debugging.*
- virtual [numbered\\_state\\_heap::state\\_index\\_p](#) [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.*
- unsigned [get\\_removed\\_components](#) () const
- unsigned [get\\_vmsize](#) () const

### Protected Attributes

- std::stack< bdd > [arc](#)
- int [num](#)
- succ\_queue::iterator [pos](#)
- [todo\\_list](#) [todo](#)
- bool [group\\_](#)  
*Whether successors should be grouped for states in the same SCC.*
- bool [group2\\_](#)
- bool [onepass\\_](#)
- [couvreur99\\_check\\_status](#) \* [ecs\\_](#)
- bool [poprem\\_](#)  
*Whether to store the state to be removed.*
- unsigned [removed\\_components](#)  
*Number of dead SCC removed by the algorithm.*
- const [tgba](#) \* [a\\_](#)  
*The automaton.*
- [option\\_map](#) [o\\_](#)  
*The options.*

### 7.22.1 Detailed Description

A version of [spot::couvreur99\\_check](#) that tries to visit known states first. See the documentation for [spot::couvreur99](#).

### 7.22.2 Member Typedef Documentation

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

**7.22.2.2** `typedef std::list<successor> spot::couvreur99_check_shy::succ_queue [protected]`

**7.22.2.3** `typedef std::list<todo_item> spot::couvreur99_check_shy::todo_list [protected]`

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

### 7.22.3 Constructor & Destructor Documentation

**7.22.3.1** `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 ())`

**7.22.3.2** `virtual spot::couvreur99_check_shy::~~couvreur99_check_shy () [virtual]`

### 7.22.4 Member Function Documentation

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

The automaton that this emptiness-check inspects.

References [spot::emptiness\\_check::a\\_](#).

**7.22.4.2** `virtual emptiness_check_result* spot::couvreur99_check_shy::check () [virtual]`

Check whether the automaton's language is empty.

Reimplemented from [spot::couvreur99\\_check](#).

**7.22.4.3** `void spot::couvreur99_check_shy::clear_todo () [protected]`**7.22.4.4** `void spot::ec_statistics::dec_depth (unsigned n = 1) [inline, inherited]`

References `spot::ec_statistics::depth_`.

**7.22.4.5** `unsigned spot::ec_statistics::depth () const [inline, inherited]`

References `spot::ec_statistics::depth_`.

**7.22.4.6** `void spot::couvreur99_check_shy::dump_queue (std::ostream & os = std::cerr) [protected]`

Dump the queue for debugging.

**7.22.4.7** `virtual numbered_state_heap::state_index_p 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.

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

References `spot::unsigned_statistics::stats`.

**7.22.4.9** unsigned spot::couvreur99\_check::get\_removed\_components () const [protected, inherited]

**7.22.4.10** unsigned spot::couvreur99\_check::get\_vmsize () const [protected, inherited]

**7.22.4.11** void spot::ec\_statistics::inc\_depth (unsigned  $n = 1$ ) [inline, inherited]

References spot::ec\_statistics::depth\_, and spot::ec\_statistics::max\_depth\_.

**7.22.4.12** void spot::ec\_statistics::inc\_states () [inline, inherited]

References spot::ec\_statistics::states\_.

**7.22.4.13** void spot::ec\_statistics::inc\_transitions () [inline, inherited]

References spot::ec\_statistics::transitions\_.

**7.22.4.14** unsigned spot::ec\_statistics::max\_depth () const [inline, inherited]

References spot::ec\_statistics::max\_depth\_.

Referenced by spot::ec\_statistics::ec\_statistics().

**7.22.4.15** const option\_map& spot::emptiness\_check::options () const [inline, inherited]

Return the options parametrizing how the emptiness check is realized.

References spot::emptiness\_check::o\_.

**7.22.4.16** virtual void spot::emptiness\_check::options\_updated (const option\_map & *old*) [virtual, inherited]

Notify option updates.



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

Modify the algorithm options.

**7.22.4.18** `virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const`  
`[virtual, inherited]`

Print statistics, if any.

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

**7.22.4.19** `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.

**7.22.4.20** `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.

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

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

**7.22.4.22** `void spot::ec_statistics::set_states (unsigned n)` `[inline, inherited]`

References `spot::ec_statistics::states_`.

**7.22.4.23** `unsigned spot::ec_statistics::states () const` `[inline, inherited]`

References `spot::ec_statistics::states_`.

Referenced by `spot::ec_statistics::ec_statistics()`.

**7.22.4.24** `virtual const unsigned_statistics* spot::emptiness_check::statistics () const [virtual, inherited]`

Return statistics, if available.

**7.22.4.25** `unsigned spot::ec_statistics::transitions () const [inline, inherited]`

References spot::ec\_statistics::transitions\_.

Referenced by spot::ec\_statistics::ec\_statistics().

## 7.22.5 Member Data Documentation

**7.22.5.1** `const tgba* spot::emptiness_check::a_ [protected, inherited]`

The automaton.

Referenced by spot::emptiness\_check::automaton().

**7.22.5.2** `std::stack<bdd> spot::couvreur99_check_shy::arc [protected]`

**7.22.5.3** `couvreur99_check_status* spot::couvreur99_check::ecs_ [protected, inherited]`

**7.22.5.4** `bool spot::couvreur99_check_shy::group2_ [protected]`

**7.22.5.5** `bool spot::couvreur99_check_shy::group_ [protected]`

Whether successors should be grouped for states in the same SCC.

**7.22.5.6** `int spot::couvreur99_check_shy::num [protected]`

**7.22.5.7** `option_map spot::emptiness_check::o_ [protected, inherited]`

The options.

Referenced by spot::emptiness\_check::options().

**7.22.5.8** `bool spot::couvreur99_check_shy::onepass_ [protected]`

**7.22.5.9** `bool spot::couvreur99_check::poprem_ [protected, inherited]`

Whether to store the state to be removed.

**7.22.5.10** `succ_queue::iterator spot::couvreur99_check_shy::pos [protected]`

**7.22.5.11** `unsigned spot::couvreur99_check::removed_components [protected, inherited]`

Number of dead SCC removed by the algorithm.

**7.22.5.12** `stats_map spot::unsigned_statistics::stats [inherited]`

Referenced by spot::acss\_statistics::acss\_statistics(), spot::ars\_statistics::ars\_statistics(), spot::ec\_statistics::ec\_statistics(), spot::unsigned\_statistics::get(), and spot::unsigned\_statistics\_copy::seteq().

**7.22.5.13** `todo_list spot::couvreur99_check_shy::todo [protected]`

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

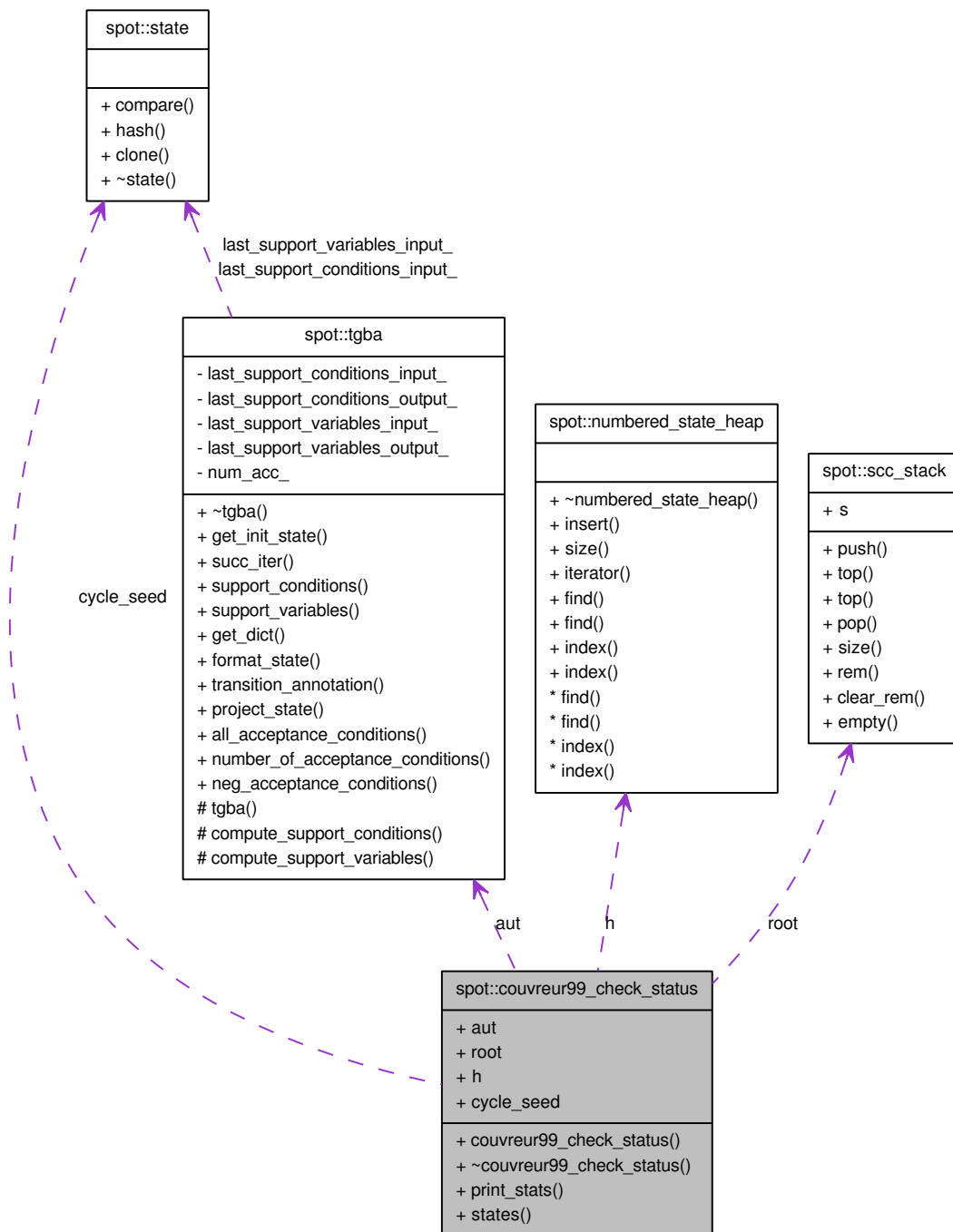
- `tgbaalgos/gtec/gtec.hh`

## 7.23 spot::couvreur99\_check\_status Class Reference

The status of the emptiness-check on success.

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

Collaboration diagram for spot::couvreur99\_check\_status:



## Public Member Functions

- `couvreur99_check_status` (const `tgba` \*`aut`, const `numbered_state_heap_factory` \*`nshf`)
- `~couvreur99_check_status` ()
- void `print_stats` (std::ostream &`os`) const  
Output statistics about this object.

- `int states () const`  
*Return the number of states visited by the search.*

### Public Attributes

- `const tgba * aut`
- `scc_stack root`
- `numbered_state_heap * h`  
*Heap of visited states.*
- `const state * cycle_seed`

### 7.23.1 Detailed Description

The status of the emptiness-check on success. This contains everything needed to construct a counter-example: the automata, the stack of SCCs traversed by the counter-example, and the heap of visited states with their indexes.

### 7.23.2 Constructor & Destructor Documentation

**7.23.2.1** `spot::couvreur99_check_status::couvreur99_check_status (const tgba * aut, const numbered_state_heap_factory * nshf)`

**7.23.2.2** `spot::couvreur99_check_status::~~couvreur99_check_status ()`

### 7.23.3 Member Function Documentation

**7.23.3.1** `void spot::couvreur99_check_status::print_stats (std::ostream & os) const`

Output statistics about this object.

**7.23.3.2** `int spot::couvreur99_check_status::states () const`

Return the number of states visited by the search.

### 7.23.4 Member Data Documentation

**7.23.4.1** `const tgba* spot::couvreur99_check_status::aut`

**7.23.4.2** `const state* spot::couvreur99_check_status::cycle_seed`

**7.23.4.3** `numbered_state_heap* spot::couvreur99_check_status::h`

Heap of visited states.

**7.23.4.4** `scc_stack spot::couvreur99_check_status::root`

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

- `tgbaalgos/gtec/status.hh`

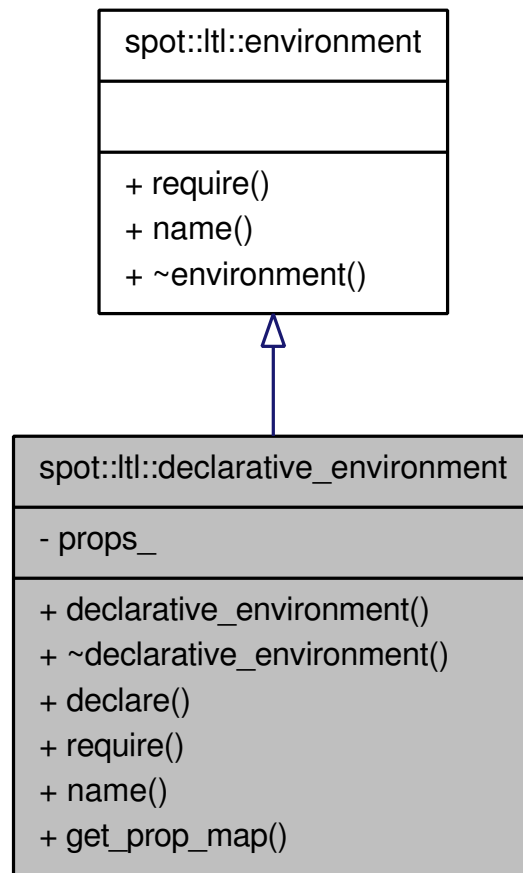
## 7.24 spot::ltl::declarative\_environment Class Reference

A declarative environment.

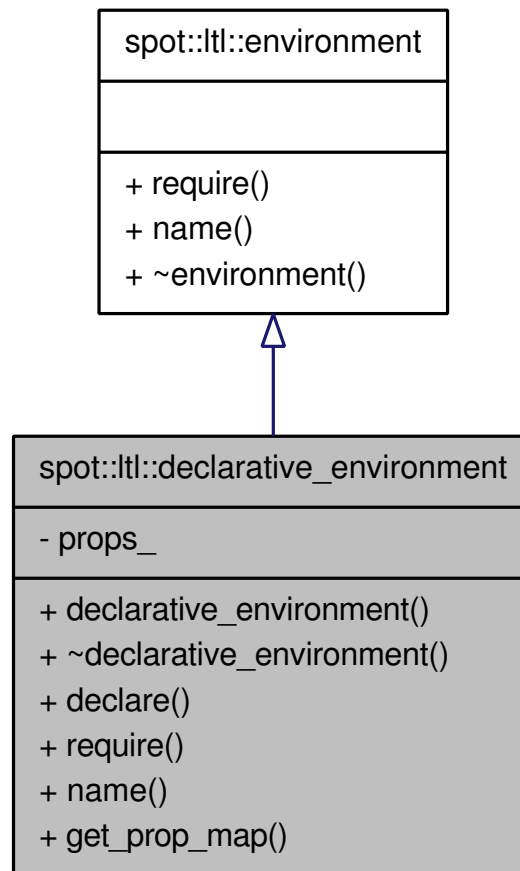
This environment recognizes all atomic propositions that have been previously declared. It will reject other.

```
#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_`

#### 7.24.1 Detailed Description

A declarative environment.

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

#### 7.24.2 Member Typedef Documentation

- 7.24.2.1 `typedef std::map<const std::string, ltl::atomic_prop*> spot::ltl::declarative_environment::prop_map`

#### 7.24.3 Constructor & Destructor Documentation

- 7.24.3.1 `spot::ltl::declarative_environment::declarative_environment ()`

- 7.24.3.2 `spot::ltl::declarative_environment::~~declarative_environment ()`

#### 7.24.4 Member Function Documentation

- 7.24.4.1 `bool spot::ltl::declarative_environment::declare (const std::string & prop_str)`

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

- 7.24.4.2 `const prop_map& spot::ltl::declarative_environment::get_prop_map () const`

Get the map of atomic proposition known to this environment.

- 7.24.4.3 `virtual const std::string& spot::ltl::declarative_environment::name () [virtual]`

Get the name of the environment.

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

#### 7.24.4.4 `virtual ltl::formula* spot::ltl::declarative_environment::require (const std::string & prop_str) [virtual]`

Obtain the formula associated to *prop\_str*.

Usually *prop\_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated `spot::ltl::atomic_prop`.

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a `spot::ltl::formula` instead of an `spot::ltl::atomic_prop`, because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

#### Returns

0 iff *prop\_str* is not part of the environment, or the associated `spot::ltl::formula` otherwise.

Implements `spot::ltl::environment`.

#### 7.24.5 Member Data Documentation

##### 7.24.5.1 `prop_map spot::ltl::declarative_environment::props_ [private]`

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

- `ltlenv/declenv.hh`

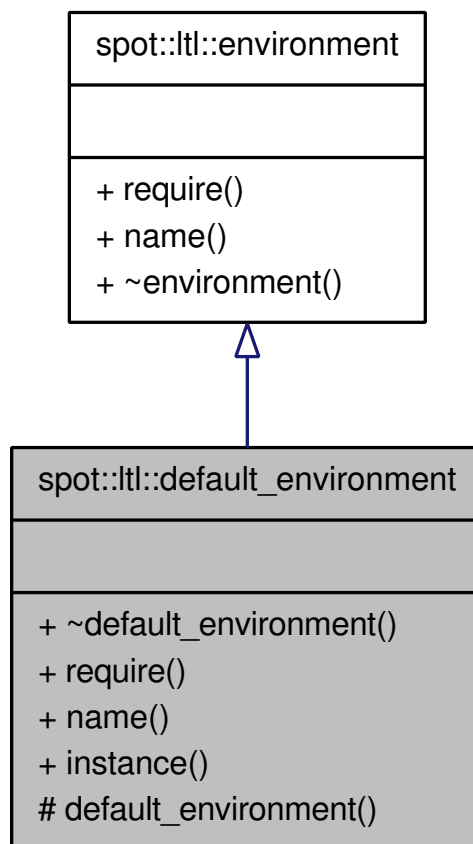
## 7.25 `spot::ltl::default_environment` Class Reference

A laxist environment.

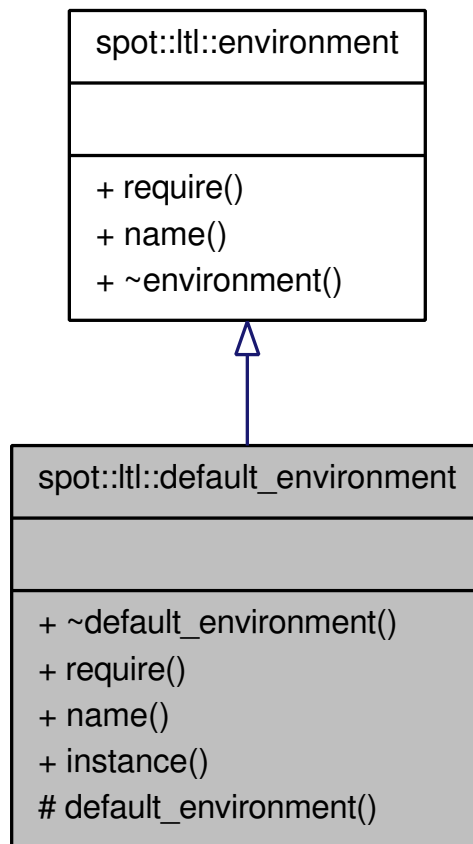
This environment recognizes all atomic propositions.

```
#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 ()`

### 7.25.1 Detailed Description

A laxist environment.

This environment recognizes all atomic propositions. This is a singleton. Use [default\\_environment::instance\(\)](#) to obtain the instance.

### 7.25.2 Constructor & Destructor Documentation

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

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

### 7.25.3 Member Function Documentation

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

Get the sole instance of [spot::ltl::default\\_environment](#).

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

Get the name of the environment.

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

**7.25.3.3** `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](#)

## 7.26 `spot::delayed_simulation_relation` Class Reference

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

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

- [tgba/tgbareduc.hh](#)

## 7.27 `spot::direct_simulation_relation` Class Reference

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

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

- [tgba/tgbareduc.hh](#)

## 7.28 `spot::taa_succ_iterator::distance_sort` Struct Reference

### Public Member Functions

- `bool operator() (const iterator\_pair &lhs, const iterator\_pair &rhs) const`

### 7.28.1 Member Function Documentation

**7.28.1.1** `bool spot::taa_succ_iterator::distance_sort::operator() (const iterator\_pair &lhs, const iterator\_pair &rhs) const` [`inline`]

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

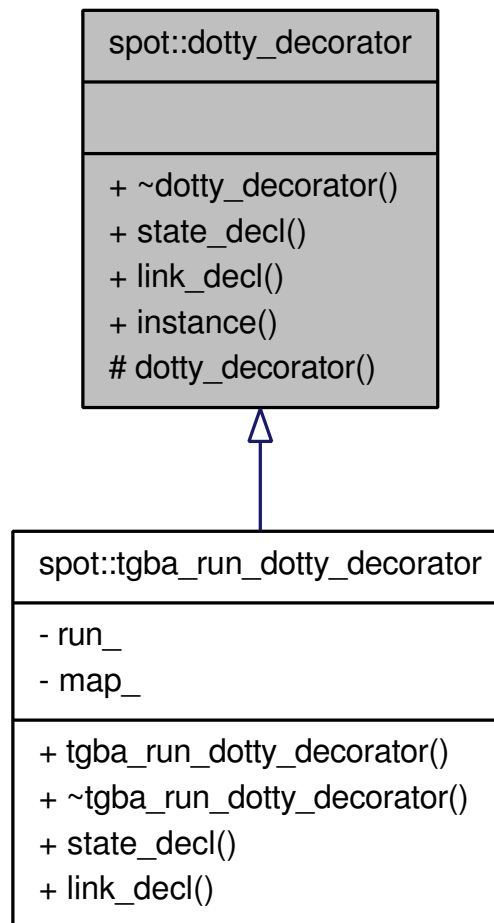
- [tgba/taatgba.hh](#)

## 7.29 `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\(\)](#)

**7.29.1 Detailed Description**

Choose state and link styles for [spot::dotty\\_reachable](#).

**7.29.2 Constructor & Destructor Documentation****7.29.2.1 virtual spot::dotty\_decorator::~~dotty\_decorator() [virtual]****7.29.2.2 spot::dotty\_decorator::dotty\_decorator() [protected]****7.29.3 Member Function Documentation****7.29.3.1 static dotty\_decorator\* spot::dotty\_decorator::instance() [static]**

Get the unique instance of the default [dotty\\_decorator](#).

**7.29.3.2 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](#).



### 7.29.3.3 virtual std::string spot::dotty\_decorator::state\_decl (const tgba \* *a*, const state \* *s*, int *n*, tgba\_succ\_iterator \* *si*, const std::string & *label*) [virtual]

Compute the style of a state.

This function should output a string of the form [label="foo", style=bar, ...]. The default implementation will simply output [label="LABEL"] with LABEL replaced by the value of *label*.

#### Parameters

- a* the automaton being drawn
- s* the state being drawn (owned by the caller)
- n* a unique number for this state
- si* an iterator over the successors of this state (owned by the caller, but can be freely iterated)
- label* the computed name of this state

Reimplemented in [spot::tgba\\_run\\_dotty\\_decorator](#).

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

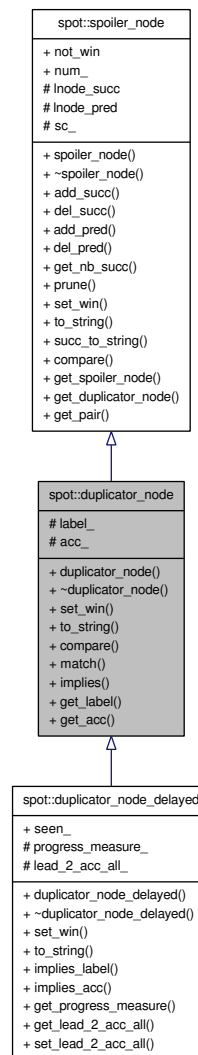
- [tgbaalgos/dottydec.hh](#)

## 7.30 spot::duplicator\_node Class Reference

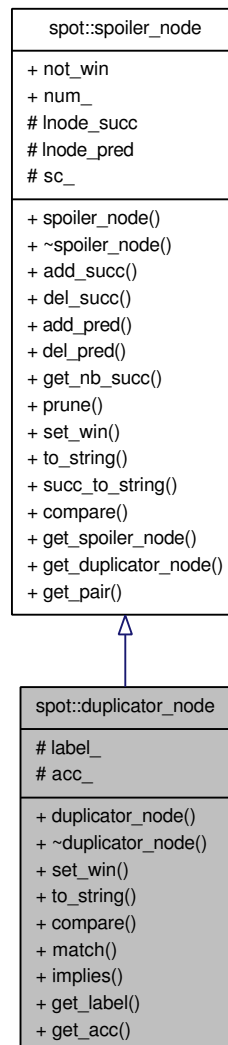
Duplicator node of parity game graph.

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

Inheritance diagram for spot::duplicator\_node:



Collaboration diagram for spot::duplicator\_node:



## Public Member Functions

- `duplicator_node` (const `state` \*d\_node, const `state` \*s\_node, bdd l, bdd a, int num)
- virtual `~duplicator_node` ()
- virtual bool `set_win` ()
- virtual std::string `to_string` (const `tgba` \*a)
- virtual bool `compare` (`spoiler_node` \*n)
- bool `match` (bdd l, bdd a)
- bool `implies` (bdd l, bdd a)
- bdd `get_label` () const
- bdd `get_acc` () const
- bool `add_succ` (`spoiler_node` \*n)  
*Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.*
- void `del_succ` (`spoiler_node` \*n)

- virtual void `add_pred` (`spoiler_node *n`)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const `state` \* `get_spoiler_node` ()
- const `state` \* `get_duplicator_node` ()
- `state_couple` \* `get_pair` ()

#### Public Attributes

- bool `not_win`
- int `num_`

#### Protected Attributes

- bdd `label_`
- bdd `acc_`
- `sn_v` \* `lnode_succ`
- `sn_v` \* `lnode_pred`
- `state_couple` \* `sc_`

### 7.30.1 Detailed Description

Duplicator node of parity game graph.

### 7.30.2 Constructor & Destructor Documentation

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

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

### 7.30.3 Member Function Documentation

**7.30.3.1** `virtual void spot::spoiler_node::add_pred (spoiler_node * n)` **[virtual, inherited]**

**7.30.3.2** `bool spot::spoiler_node::add_succ (spoiler_node * n)` **[inherited]**

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

**7.30.3.3** `virtual bool spot::duplicator_node::compare (spoiler_node * n)` `[virtual]`

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

#### **7.30.4 Member Data Documentation**

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

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

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

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

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

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

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

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

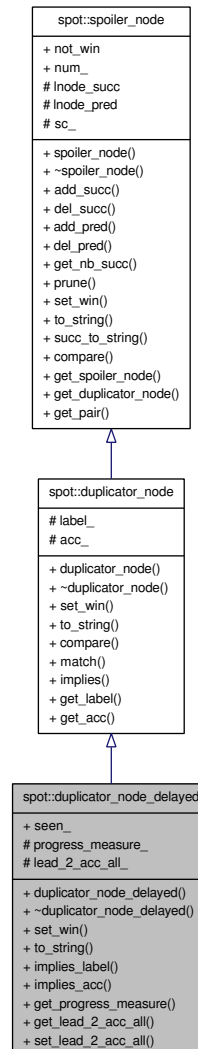
- [tgbaalgos/reductgba\\_sim.hh](#)

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

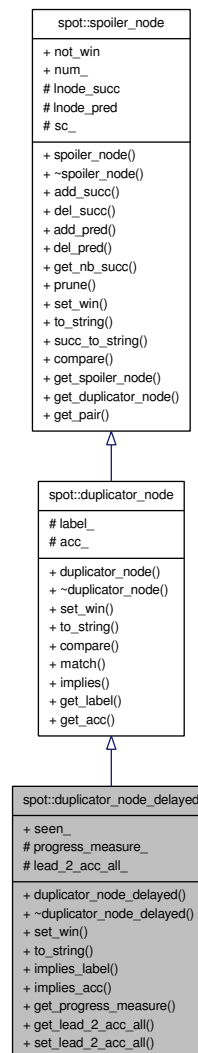
Duplicator node of parity game graph for delayed simulation.

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

Inheritance diagram for spot::duplicator\_node\_delayed:



Collaboration diagram for spot::duplicator\_node\_delayed:



## Public Member Functions

- [duplicator\\_node\\_delayed](#) (const [state](#) \*d\_node, const [state](#) \*s\_node, bdd l, bdd a, int num)
- [~duplicator\\_node\\_delayed](#) ()
- bool [set\\_win](#) ()

*Return true if the progress\_measure has changed.*

- virtual std::string [to\\_string](#) (const [tgba](#) \*a)
- bool [implies\\_label](#) (bdd l)
- bool [implies\\_acc](#) (bdd a)
- int [get\\_progress\\_measure](#) ()
- bool [get\\_lead\\_2\\_acc\\_all](#) ()
- bool [set\\_lead\\_2\\_acc\\_all](#) (bdd acc=bddfalse)
- virtual bool [compare](#) ([spoiler\\_node](#) \*n)
- bool [match](#) (bdd l, bdd a)



- bool `implies` (bdd `l`, bdd `a`)
- bdd `get_label` () const
- bdd `get_acc` () const
- bool `add_succ` (spoiler\_node \*`n`)

*Add a successor. Return true if `n` wasn't yet in the list of successor, false otherwise.*

- void `del_succ` (spoiler\_node \*`n`)
- virtual void `add_pred` (spoiler\_node \*`n`)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual std::string `succ_to_string` ()
- const state \* `get_spoiler_node` ()
- const state \* `get_duplicator_node` ()
- state\_couple \* `get_pair` ()

### Public Attributes

- bool `seen_`
- bool `not_win`
- int `num_`

### Protected Attributes

- int `progress_measure_`
- bool `lead_2_acc_all_`
- bdd `label_`
- bdd `acc_`
- sn\_v \* `lnode_succ`
- sn\_v \* `lnode_pred`
- state\_couple \* `sc_`

#### 7.31.1 Detailed Description

Duplicator node of parity game graph for delayed simulation.

#### 7.31.2 Constructor & Destructor Documentation

- 7.31.2.1** spot::duplicator\_node\_delayed::duplicator\_node\_delayed (const state \* `d_node`, const state \* `s_node`, bdd `l`, bdd `a`, int `num`)

- 7.31.2.2** spot::duplicator\_node\_delayed::~~duplicator\_node\_delayed ()

### 7.31.3 Member Function Documentation

**7.31.3.1** `virtual void spot::spoiler_node::add_pred (spoiler_node * n)` [`virtual`, `inherited`]

**7.31.3.2** `bool spot::spoiler_node::add_succ (spoiler_node * n)` [`inherited`]

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**7.31.3.19** `bool spot::duplicator_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

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

Return true if the `progress_measure` has changed.

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

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

**7.31.3.22** `virtual std::string spot::duplicator_node_delayed::to_string (const tgba * a)`  
[`virtual`]

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

#### 7.31.4 Member Data Documentation

**7.31.4.1** `bdd spot::duplicator_node::acc_` [`protected`, `inherited`]

**7.31.4.2** `bdd spot::duplicator_node::label_` [`protected`, `inherited`]

**7.31.4.3** `bool spot::duplicator_node_delayed::lead_2_acc_all_` [`protected`]

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

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

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

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

**7.31.4.8** `int spot::duplicator_node_delayed::progress_measure_` [`protected`]

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

## 7.31.4.10 bool spot::duplicator\_node\_delayed::seen\_

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

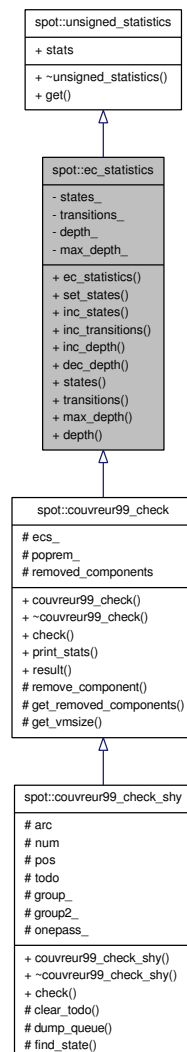
- [tgbaalgos/reductgba\\_sim.hh](#)

## 7.32 spot::ec\_statistics Class Reference

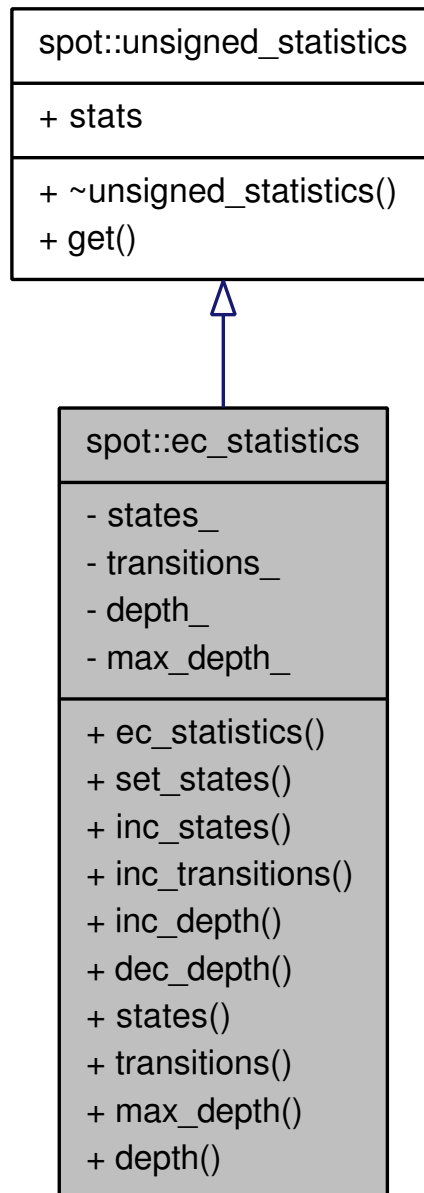
Emptiness-check statistics.

```
#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 distinct visited states*
- unsigned [depth\\_](#)  
*number of visited transitions*
- unsigned [max\\_depth\\_](#)  
*maximal depth of the stack(s)*

#### 7.32.1 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.

#### 7.32.2 Member Typedef Documentation

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

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

#### 7.32.3 Constructor & Destructor Documentation

**7.32.3.1** `spot::ec_statistics::ec_statistics () [inline]`

References [max\\_depth\(\)](#), [states\(\)](#), [spot::unsigned\\_statistics::stats](#), and [transitions\(\)](#).

### 7.32.4 Member Function Documentation

#### 7.32.4.1 void spot::ec\_statistics::dec\_depth (unsigned *n* = 1) [inline]

References `depth_`.

#### 7.32.4.2 unsigned spot::ec\_statistics::depth () const [inline]

References `depth_`.

#### 7.32.4.3 unsigned spot::unsigned\_statistics::get (const char \* *str*) const [inline, inherited]

References `spot::unsigned_statistics::stats`.

#### 7.32.4.4 void spot::ec\_statistics::inc\_depth (unsigned *n* = 1) [inline]

References `depth_`, and `max_depth_`.

#### 7.32.4.5 void spot::ec\_statistics::inc\_states () [inline]

References `states_`.

#### 7.32.4.6 void spot::ec\_statistics::inc\_transitions () [inline]

References `transitions_`.

#### 7.32.4.7 unsigned spot::ec\_statistics::max\_depth () const [inline]

References `max_depth_`.

Referenced by `ec_statistics()`.

#### 7.32.4.8 void spot::ec\_statistics::set\_states (unsigned *n*) [inline]

References `states_`.



#### 7.32.4.9 unsigned spot::ec\_statistics::states () const [inline]

References states\_.

Referenced by ec\_statistics().

#### 7.32.4.10 unsigned spot::ec\_statistics::transitions () const [inline]

References transitions\_.

Referenced by ec\_statistics().

### 7.32.5 Member Data Documentation

#### 7.32.5.1 unsigned spot::ec\_statistics::depth\_ [private]

number of visited transitions

Referenced by dec\_depth(), depth(), and inc\_depth().

#### 7.32.5.2 unsigned spot::ec\_statistics::max\_depth\_ [private]

maximal depth of the stack(s)

Referenced by inc\_depth(), and max\_depth().

#### 7.32.5.3 unsigned spot::ec\_statistics::states\_ [private]

Referenced by inc\_states(), set\_states(), and states().

#### 7.32.5.4 stats\_map spot::unsigned\_statistics::stats [inherited]

Referenced by spot::acss\_statistics::acss\_statistics(), spot::ars\_statistics::ars\_statistics(), ec\_statistics(), spot::unsigned\_statistics::get(), and spot::unsigned\_statistics\_copy::seteq().

#### 7.32.5.5 unsigned spot::ec\_statistics::transitions\_ [private]

number of distinct visited states

Referenced by inc\_transitions(), and transitions().

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

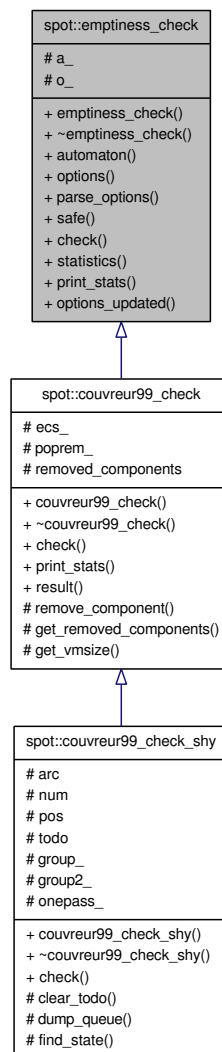
- [tgbaalgos/emptiness\\_stats.hh](#)

### 7.33 spot::emptiness\_check Class Reference

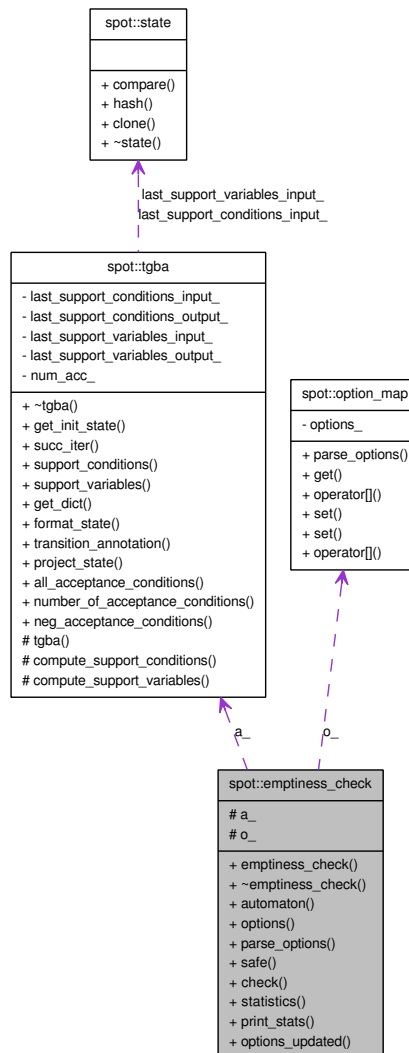
Common interface to emptiness check algorithms.

```
#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.*

#### 7.33.1 Detailed Description

Common interface to emptiness check algorithms.

#### 7.33.2 Constructor & Destructor Documentation

**7.33.2.1** `spot::emptiness_check::emptiness_check (const tgba * a, option_map o = option_map ())`  
[inline]

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

#### 7.33.3 Member Function Documentation

**7.33.3.1** `const tgba* spot::emptiness_check::automaton () const` [inline]

The automaton that this emptiness-check inspects.

References `a_`.

### 7.33.3.2 `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::couvereur99_check`, and `spot::couvereur99_check_shy`.

### 7.33.3.3 `const option_map& spot::emptiness_check::options () const [inline]`

Return the options parametrizing how the emptiness check is realized.

References `o_`.

### 7.33.3.4 `virtual void spot::emptiness_check::options_updated (const option_map & old) [virtual]`

Notify option updates.

### 7.33.3.5 `const char* spot::emptiness_check::parse_options (char * options)`

Modify the algorithm options.

### 7.33.3.6 `virtual std::ostream& spot::emptiness_check::print_stats (std::ostream & os) const [virtual]`

Print statistics, if any.

Reimplemented in `spot::couvereur99_check`.

### 7.33.3.7 `virtual bool spot::emptiness_check::safe () const [virtual]`

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

### 7.33.3.8 `virtual const unsigned_statistics* spot::emptiness_check::statistics () const` `[virtual]`

Return statistics, if available.

## 7.33.4 Member Data Documentation

### 7.33.4.1 `const tgba* spot::emptiness_check::a_` `[protected]`

The automaton.

Referenced by `automaton()`.

### 7.33.4.2 `option_map spot::emptiness_check::o_` `[protected]`

The options.

Referenced by `options()`.

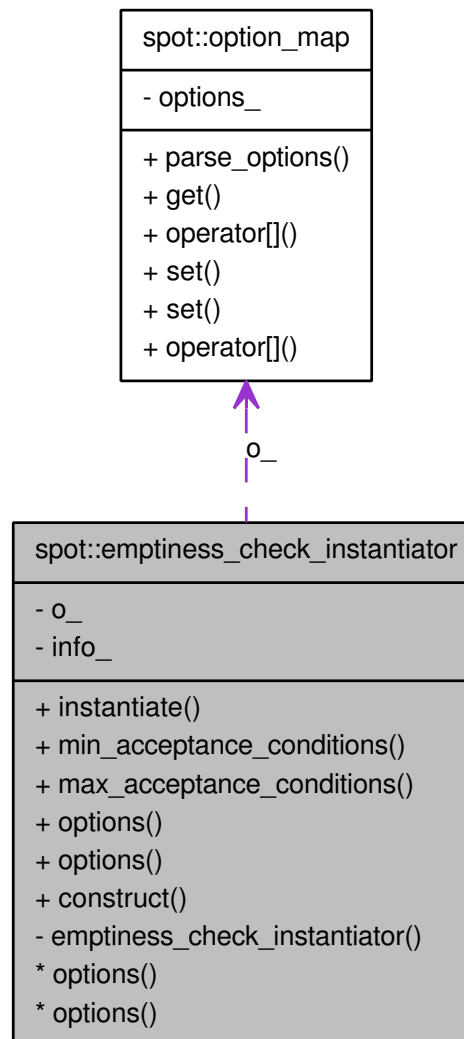
The documentation for this class was generated from the following file:

- `tgbaalgos/emptiness.hh`

## 7.34 `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.*
- `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.*
- `const option_map & options () const`
- `option_map & options ()`

### 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\_

#### 7.34.1 Constructor & Destructor Documentation

**7.34.1.1** `spot::emptiness_check_instantiator::emptiness_check_instantiator (option_map o, void * i) [private]`

#### 7.34.2 Member Function Documentation

**7.34.2.1** `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.

**7.34.2.2** `emptiness_check* spot::emptiness_check_instantiator::instantiate (const tgba * a) const`

Actually instantiate the emptiness check, for `a`.

**7.34.2.3** `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.



#### 7.34.2.4 unsigned int spot::emptiness\_check\_instantiator::min\_acceptance\_conditions () const

Minimum number of acceptance conditions supported by the emptiness check.

#### 7.34.2.5 option\_map& spot::emptiness\_check\_instantiator::options () [inline]

References o\_.

#### 7.34.2.6 const option\_map& spot::emptiness\_check\_instantiator::options () const [inline]

Accessor to the options.

References o\_.

### 7.34.3 Member Data Documentation

#### 7.34.3.1 void\* spot::emptiness\_check\_instantiator::info\_ [private]

#### 7.34.3.2 option\_map spot::emptiness\_check\_instantiator::o\_ [private]

Referenced by options().

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

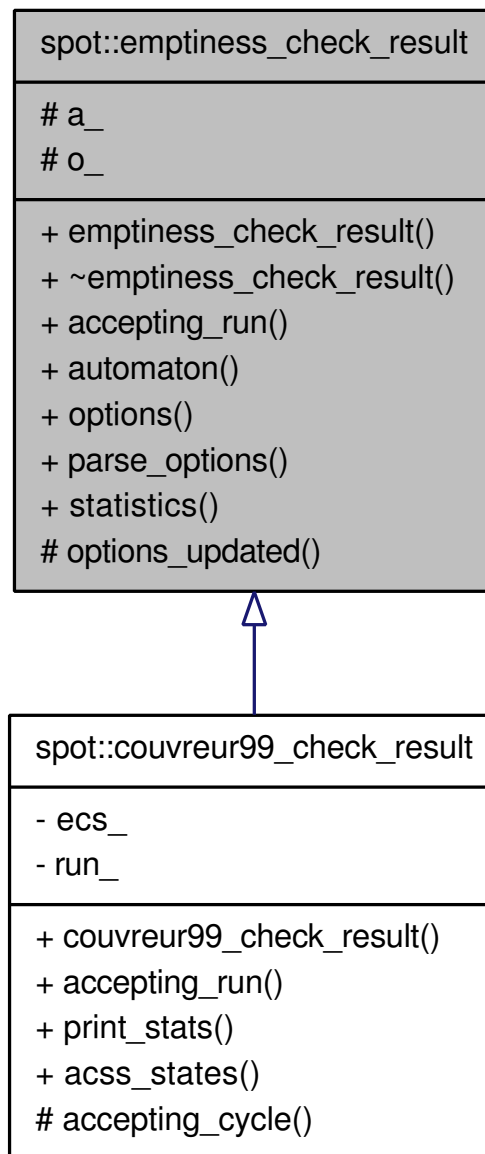
- [tgbaalgos/emptiness.hh](#)

## 7.35 spot::emptiness\_check\_result Class Reference

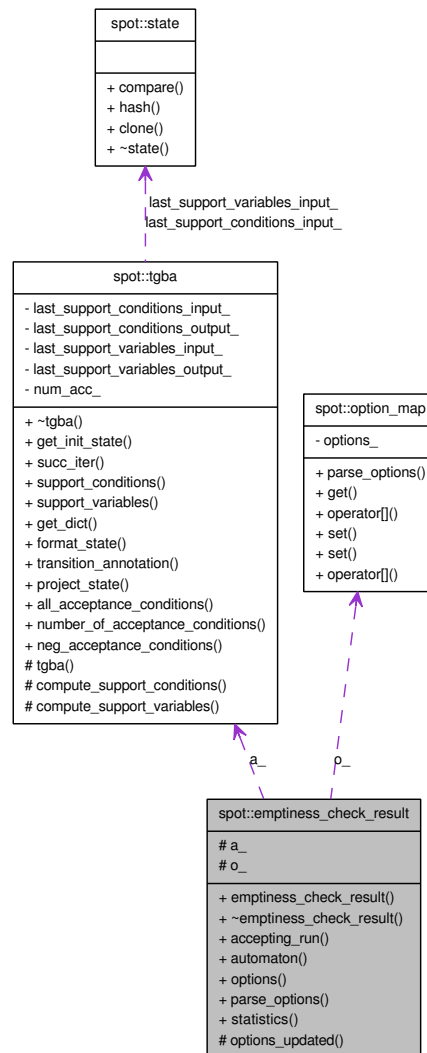
The result of an emptiness check.

```
#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.*

#### 7.35.1 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.

#### 7.35.2 Constructor & Destructor Documentation

**7.35.2.1** `spot::emptiness_check_result::emptiness_check_result (const tgba * a, option_map o = option_map ()) [inline]`

**7.35.2.2** `virtual spot::emptiness_check_result::~~emptiness_check_result () [inline, virtual]`

#### 7.35.3 Member Function Documentation

**7.35.3.1** `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](#).

#### 7.35.3.2 `const tgba* spot::emptiness_check_result::automaton () const` `[inline]`

The automaton on which an [accepting\\_run\(\)](#) was found.

References `a_`.

#### 7.35.3.3 `const option_map& spot::emptiness_check_result::options () const` `[inline]`

Return the options parametrizing how the accepting run is computed.

References `o_`.

#### 7.35.3.4 `virtual void spot::emptiness_check_result::options_updated (const option_map & old)` `[protected, virtual]`

Notify option updates.

#### 7.35.3.5 `const char* spot::emptiness_check_result::parse_options (char * options)`

Modify the algorithm options.

#### 7.35.3.6 `virtual const unsigned_statistics* spot::emptiness_check_result::statistics () const` `[virtual]`

Return statistics, if available.

### 7.35.4 Member Data Documentation

#### 7.35.4.1 `const tgba* spot::emptiness_check_result::a_` `[protected]`

The automaton.

Referenced by `automaton()`.

#### 7.35.4.2 `option_map spot::emptiness_check_result::o_` `[protected]`

The options.

Referenced by options().

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

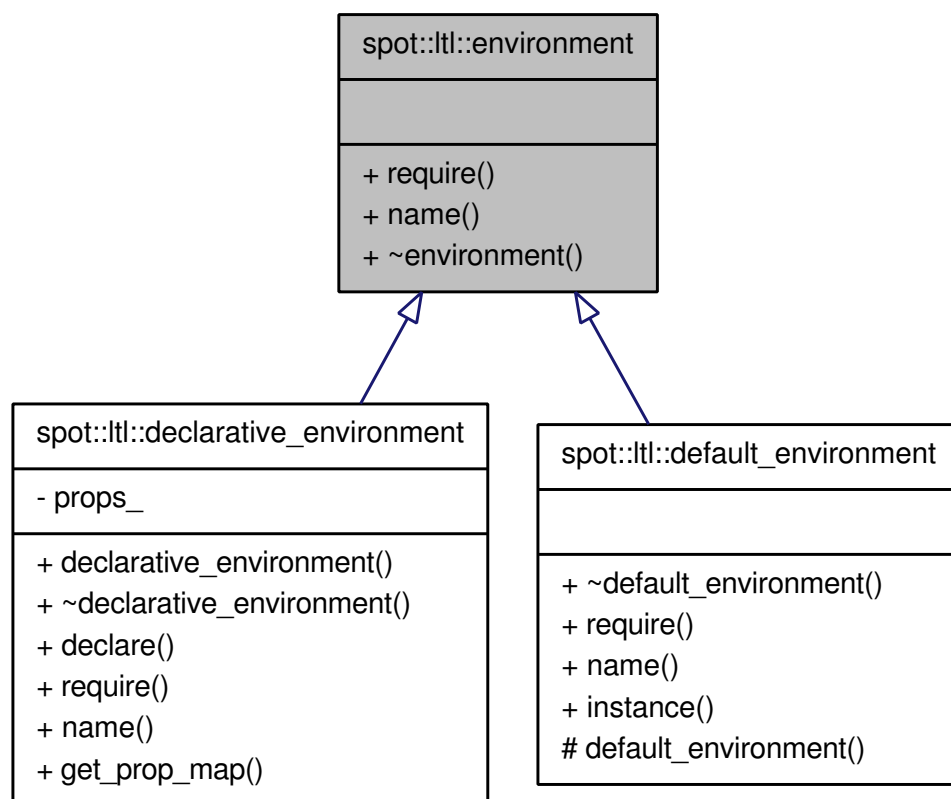
- [tgbaalgorithms/emptiness.hh](#)

## 7.36 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](#) ()

### 7.36.1 Detailed Description

An environment that describes atomic propositions.

### 7.36.2 Constructor & Destructor Documentation

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

### 7.36.3 Member Function Documentation

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

Get the name of the environment.

Implemented in [spot::ltl::declarative\\_environment](#), and [spot::ltl::default\\_environment](#).

#### 7.36.3.2 virtual formula\* spot::ltl::environment::require(const std::string & prop\_str) [pure virtual]

Obtain the formula associated to *prop\_str*.

Usually *prop\_str*, is the name of an atomic proposition, and `spot::ltl::require` simply returns the associated [spot::ltl::atomic\\_prop](#).

Note this is not a `const` method. Some environments will "create" the atomic proposition when requested.

We return a [spot::ltl::formula](#) instead of an [spot::ltl::atomic\\_prop](#), because this will allow nifty tricks (e.g., we could name formulae in an environment, and let the parser build a larger tree from these).

#### Returns

0 iff *prop\_str* is not part of the environment, or the associated [spot::ltl::formula](#) otherwise.

Implemented in [spot::ltl::declarative\\_environment](#), and [spot::ltl::default\\_environment](#).

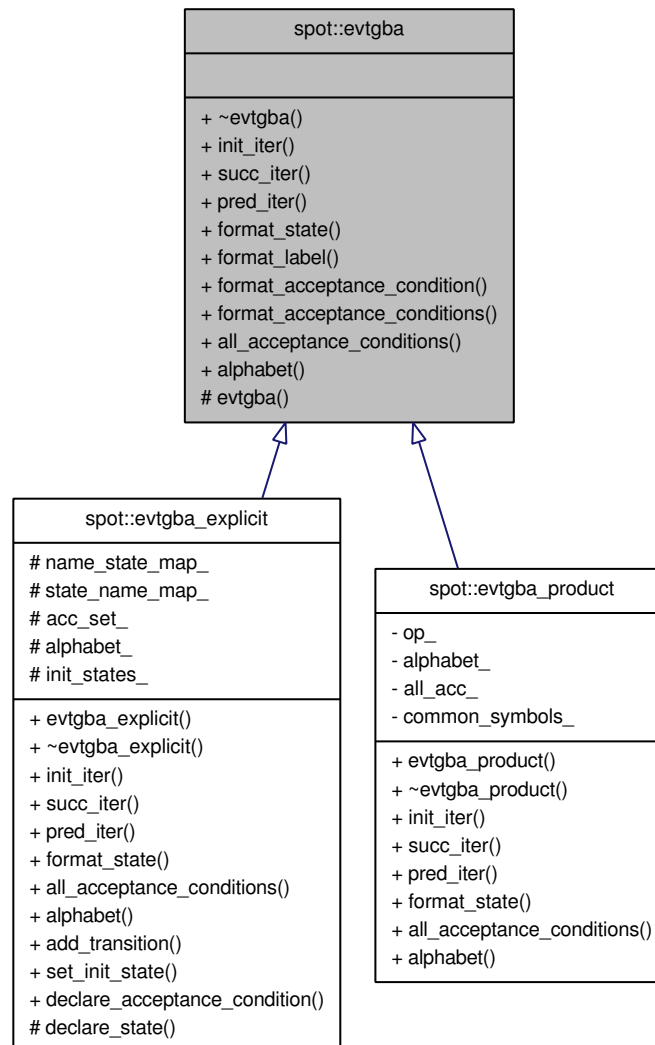
The documentation for this class was generated from the following file:

- [ltlenv/environment.hh](#)

## 7.37 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](#) ()

## 7.37.1 Constructor & Destructor Documentation

### 7.37.1.1 spot::evtgba::evtgba () [protected]

### 7.37.1.2 virtual spot::evtgba::~~evtgba () [virtual]

## 7.37.2 Member Function Documentation

### 7.37.2.1 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](#).

### 7.37.2.2 virtual const [symbol\\_set](#)& spot::evtgba::alphabet () const [pure virtual]

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

### 7.37.2.3 virtual std::string spot::evtgba::format\_acceptance\_condition (const [symbol](#) \* *symbol*) const [virtual]

### 7.37.2.4 virtual std::string spot::evtgba::format\_acceptance\_conditions (const [symbol\\_set](#) & *symset*) const [virtual]

### 7.37.2.5 virtual std::string spot::evtgba::format\_label (const [symbol](#) \* *symbol*) const [virtual]

**7.37.2.6** `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 that owns the state.

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**7.37.2.7** `virtual evtgba_iterator* spot::evtgba::init_iter () const` `[pure virtual]`

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**7.37.2.8** `virtual evtgba_iterator* spot::evtgba::pred_iter (const state * s) const` `[pure virtual]`

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

**7.37.2.9** `virtual evtgba_iterator* spot::evtgba::succ_iter (const state * s) const` `[pure virtual]`

Implemented in [spot::evtgba\\_explicit](#), and [spot::evtgba\\_product](#).

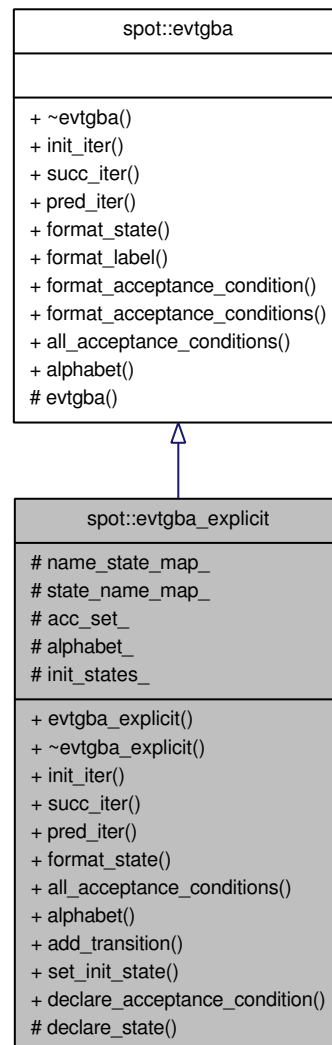
The documentation for this class was generated from the following file:

- [evtgba/evtgba.hh](#)

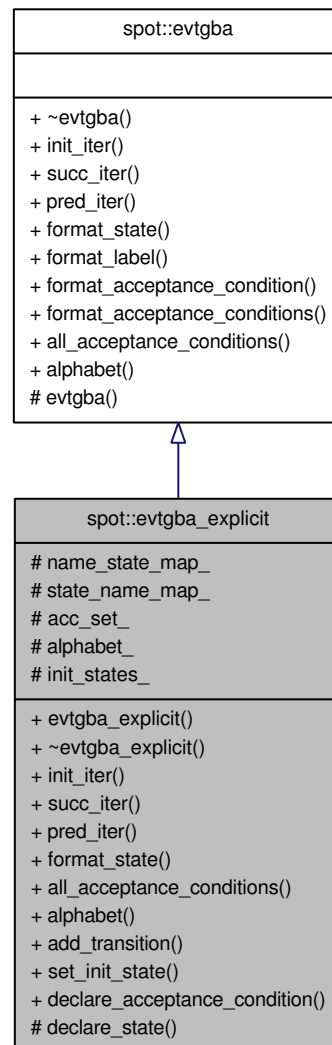
## 7.38 `spot::evtgba_explicit` Class Reference

```
#include <evtgba/explicit.hh>
```

Inheritance diagram for spot::evtgba\_explicit:



Collaboration diagram for spot::evtgba\_explicit:



## Classes

- struct [state](#)
- struct [transition](#)

*Explicit transitions (used by [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  
*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
- [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 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\\_](#)

#### 7.38.1 Member Typedef Documentation

**7.38.1.1** typedef Sgi::hash\_map<const std::string, [evtgba\\_explicit::state](#)\*, [string\\_hash](#)>  
[spot::evtgba\\_explicit::ns\\_map](#) [**protected**]

**7.38.1.2** typedef Sgi::hash\_map<const [evtgba\\_explicit::state](#)\*, std::string,  
[ptr\\_hash](#)<[evtgba\\_explicit::state](#)> > [spot::evtgba\\_explicit::sn\\_map](#) [**protected**]

**7.38.1.3** typedef std::list<transition\*> spot::evtgba\_explicit::transition\_list

## 7.38.2 Constructor & Destructor Documentation

**7.38.2.1** spot::evtgba\_explicit::evtgba\_explicit ()

**7.38.2.2** virtual spot::evtgba\_explicit::~~evtgba\_explicit () [virtual]

## 7.38.3 Member Function Documentation

**7.38.3.1** transition\* spot::evtgba\_explicit::add\_transition (const std::string & *source*, const rsymbol & *label*, rsymbol\_set *acc*, const std::string & *dest*)

**7.38.3.2** 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 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](#).

**7.38.3.3** virtual const symbol\_set& spot::evtgba\_explicit::alphabet () const [virtual]

Implements [spot::evtgba](#).

**7.38.3.4** void spot::evtgba\_explicit::declare\_acceptance\_condition (const rsymbol & *acc*)

**7.38.3.5** state\* spot::evtgba\_explicit::declare\_state (const std::string & *name*) [protected]

**7.38.3.6** `virtual std::string spot::evtgba::format_acceptance_condition (const symbol * symbol) const` `[virtual, inherited]`

**7.38.3.7** `virtual std::string spot::evtgba::format_acceptance_conditions (const symbol_set & symset) const` `[virtual, inherited]`

**7.38.3.8** `virtual std::string spot::evtgba::format_label (const symbol * symbol) const` `[virtual, inherited]`

**7.38.3.9** `virtual std::string spot::evtgba_explicit::format_state (const spot::state * state) const` `[virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::evtgba](#).

**7.38.3.10** `virtual evtgba_iterator* spot::evtgba_explicit::init_iter () const` `[virtual]`

Implements [spot::evtgba](#).

**7.38.3.11** `virtual evtgba_iterator* spot::evtgba_explicit::pred_iter (const spot::state * s) const` `[virtual]`

Implements [spot::evtgba](#).

**7.38.3.12** `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.

**7.38.3.13** `virtual evtgba_iterator* spot::evtgba_explicit::succ_iter (const spot::state * s) const` `[virtual]`

Implements [spot::evtgba](#).

### 7.38.4 Member Data Documentation

7.38.4.1 symbol\_set spot::evtgba\_explicit::acc\_set\_ [protected]

7.38.4.2 symbol\_set spot::evtgba\_explicit::alphabet\_ [protected]

7.38.4.3 transition\_list spot::evtgba\_explicit::init\_states\_ [protected]

7.38.4.4 ns\_map spot::evtgba\_explicit::name\_state\_map\_ [protected]

7.38.4.5 sn\_map spot::evtgba\_explicit::state\_name\_map\_ [protected]

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

- [evtgba/explicit.hh](#)

## 7.39 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

### 7.39.1 Constructor & Destructor Documentation

7.39.1.1 virtual spot::evtgba\_iterator::~~evtgba\_iterator () [inline, virtual]



### 7.39.2 Member Function Documentation

**7.39.2.1** virtual symbol\_set spot::evtgba\_iterator::current\_acceptance\_conditions () const  
[pure virtual]

**7.39.2.2** virtual const symbol\* spot::evtgba\_iterator::current\_label () const [pure virtual]

**7.39.2.3** virtual const state\* spot::evtgba\_iterator::current\_state () const [pure virtual]

**7.39.2.4** virtual bool spot::evtgba\_iterator::done () const [pure virtual]

**7.39.2.5** virtual void spot::evtgba\_iterator::first () [pure virtual]

**7.39.2.6** virtual void spot::evtgba\_iterator::next () [pure virtual]

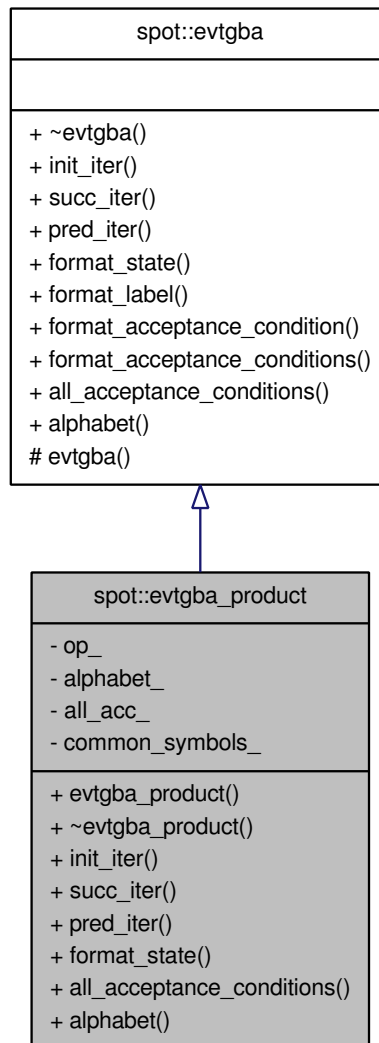
The documentation for this class was generated from the following file:

- evtgba/[evtgbaiter.hh](#)

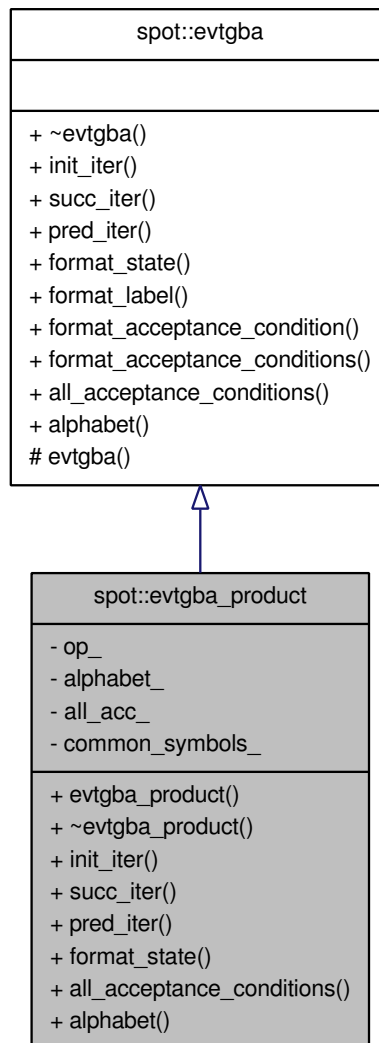
## 7.40 spot::evtgba\_product Class Reference

```
#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\_

### 7.40.1 Member Typedef Documentation

**7.40.1.1** `typedef std::map<const symbol*, std::set<int> > spot::evtgba_product::common_symbol_table`

**7.40.1.2** `typedef std::vector<const evtgba*> spot::evtgba_product::evtgba_product_operands`

### 7.40.2 Constructor & Destructor Documentation

**7.40.2.1** `spot::evtgba_product::evtgba_product (const evtgba_product_operands & op)`

**7.40.2.2** `virtual spot::evtgba_product::~~evtgba_product () [virtual]`

### 7.40.3 Member Function Documentation

**7.40.3.1** `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](#).

**7.40.3.2** virtual const symbol\_set& spot::evtgba\_product::alphabet () const [virtual]

Implements [spot::evtgba](#).

**7.40.3.3** virtual std::string spot::evtgba::format\_acceptance\_condition (const symbol \* *symbol*) const [virtual, inherited]

**7.40.3.4** virtual std::string spot::evtgba::format\_acceptance\_conditions (const symbol\_set & *symset*) const [virtual, inherited]

**7.40.3.5** virtual std::string spot::evtgba::format\_label (const symbol \* *symbol*) const [virtual, inherited]

**7.40.3.6** 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 that owns the state.

Implements [spot::evtgba](#).

**7.40.3.7** virtual evtgba\_iterator\* spot::evtgba\_product::init\_iter () const [virtual]

Implements [spot::evtgba](#).

**7.40.3.8** virtual evtgba\_iterator\* spot::evtgba\_product::pred\_iter (const state \* *s*) const [virtual]

Implements [spot::evtgba](#).

**7.40.3.9** virtual evtgba\_iterator\* spot::evtgba\_product::succ\_iter (const state \* *s*) const [virtual]

Implements [spot::evtgba](#).

#### 7.40.4 Member Data Documentation

7.40.4.1 symbol\_set spot::evtgba\_product::all\_acc\_ [private]

7.40.4.2 symbol\_set spot::evtgba\_product::alphabet\_ [private]

7.40.4.3 common\_symbol\_table spot::evtgba\_product::common\_symbols\_ [private]

7.40.4.4 const evtgba\_product\_operands spot::evtgba\_product::op\_ [private]

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

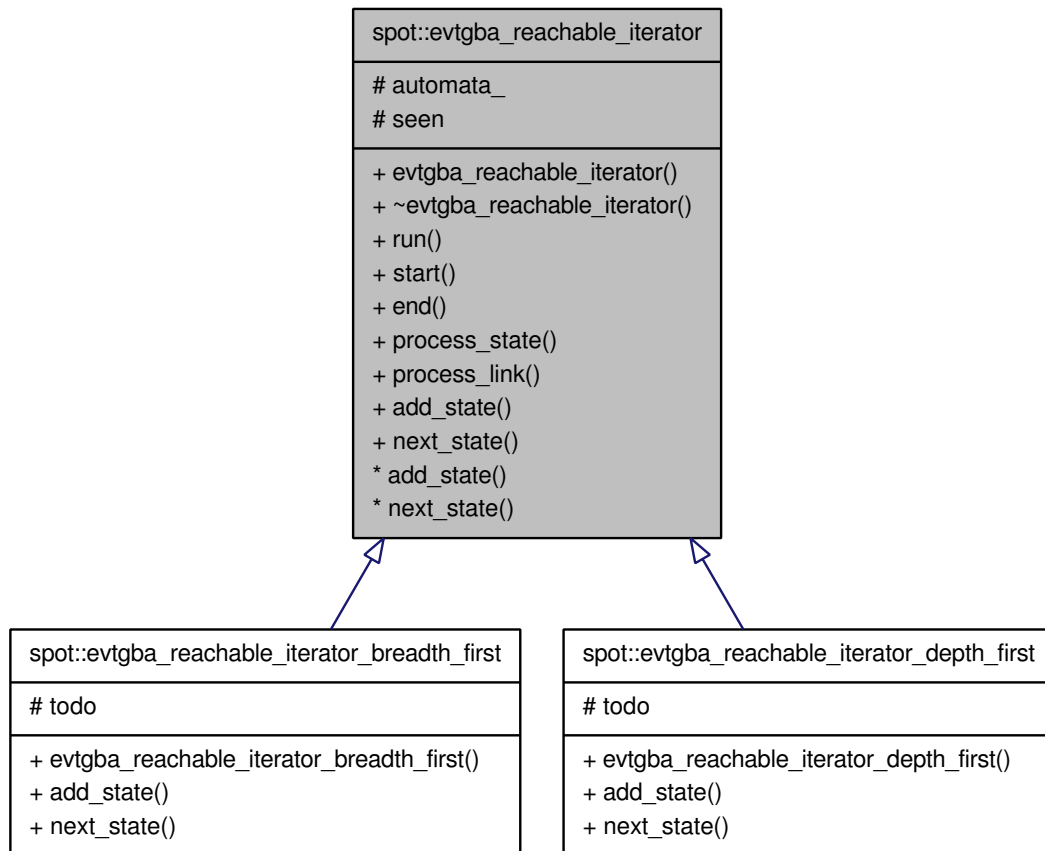
- [evtgba/product.hh](#)

### 7.41 spot::evtgba\_reachable\_iterator Class Reference

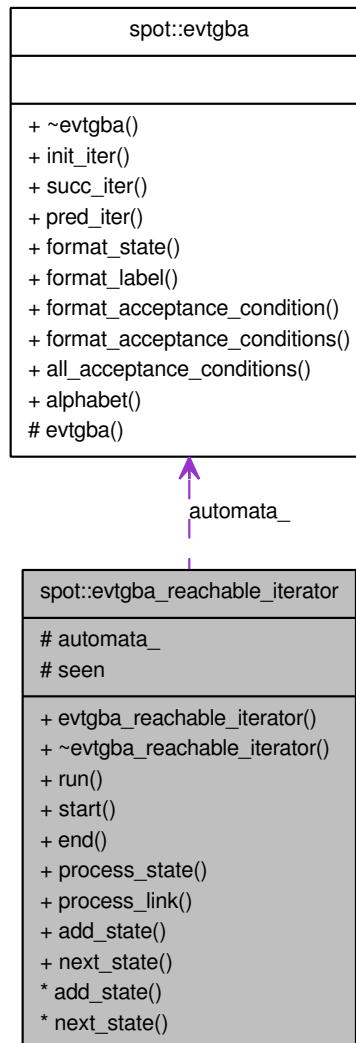
Iterate over all reachable states of a [spot::evtgba](#).

```
#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.

**7.41.1 Detailed Description**

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

**7.41.2 Member Typedef Documentation**

- 7.41.2.1** typedef Sgi::hash\_map<const state\*, int, state\_ptr\_hash, state\_ptr\_equal>  
spot::evtgba\_reachable\_iterator::seen\_map [protected]

**7.41.3 Constructor & Destructor Documentation**

- 7.41.3.1** spot::evtgba\_reachable\_iterator::evtgba\_reachable\_iterator (const evtgba \* a)

- 7.41.3.2** virtual spot::evtgba\_reachable\_iterator::~~evtgba\_reachable\_iterator () [virtual]

### 7.41.4 Member Function Documentation

#### 7.41.4.1 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](#).

#### 7.41.4.2 virtual void spot::evtgba\_reachable\_iterator::end () [virtual]

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

#### 7.41.4.3 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](#).

#### 7.41.4.4 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.

#### 7.41.4.5 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*.

#### 7.41.4.6 `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.

#### 7.41.4.7 `virtual void spot::evtgba_reachable_iterator::start (int n)` `[virtual]`

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

##### Parameters

*n* The number of initial states.

#### 7.41.5 Member Data Documentation

##### 7.41.5.1 `const evtgba* spot::evtgba_reachable_iterator::automata_` `[protected]`

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

##### 7.41.5.2 `seen_map spot::evtgba_reachable_iterator::seen` `[protected]`

States already seen.

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

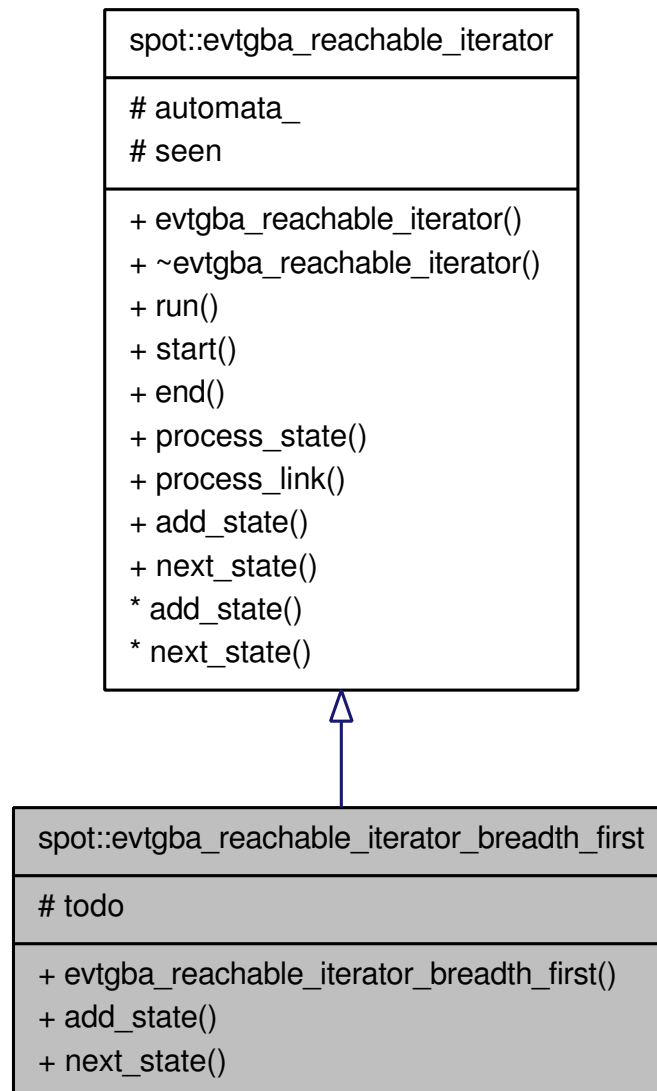
- [evtgbaalgos/reachiter.hh](#)

## 7.42 `spot::evtgba_reachable_iterator_breadth_first` Class Reference

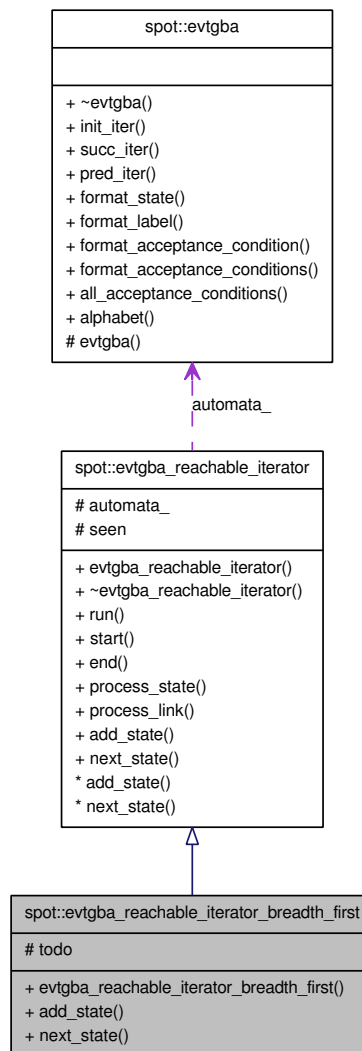
An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states breadth first.

```
#include <evtgbaalgos/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.*

#### 7.42.1 Detailed Description

An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states breadth first.

#### 7.42.2 Member Typedef Documentation

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

#### 7.42.3 Constructor & Destructor Documentation

- 7.42.3.1 `spot::evtgba_reachable_iterator_breadth_first::evtgba_reachable_iterator_breadth_first  
(const evtgba * a)`

#### 7.42.4 Member Function Documentation

- 7.42.4.1 `virtual void spot::evtgba_reachable_iterator_breadth_first::add_state (const state * s)  
[virtual]`

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

**7.42.4.2 virtual void spot::evtgba\_reachable\_iterator::end () [virtual, inherited]**

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

**7.42.4.3 virtual const state\* spot::evtgba\_reachable\_iterator\_breadth\_first::next\_state () [virtual]**

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

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

**7.42.4.4 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.

**7.42.4.5 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*.

**7.42.4.6 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.

**7.42.4.7 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.

#### 7.42.5 Member Data Documentation

##### 7.42.5.1 `const evtgba* spot::evtgba_reachable_iterator::automata_` `[protected, inherited]`

The `spot::evtgba` to explore.

##### 7.42.5.2 `seen_map spot::evtgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

##### 7.42.5.3 `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:

- `evtgbaalgos/reachiter.hh`

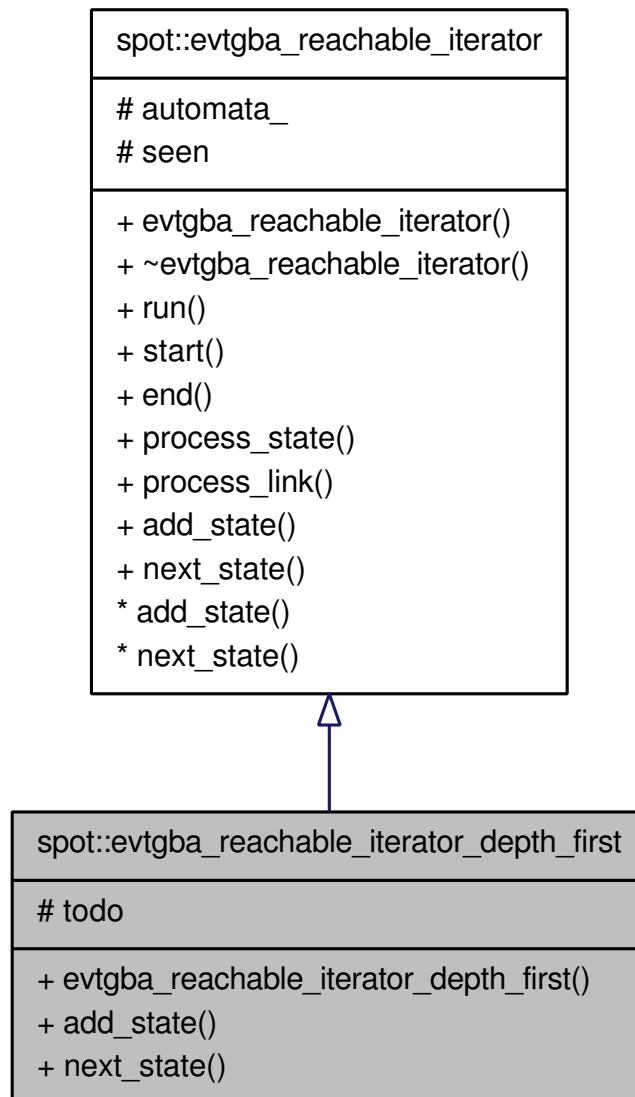
### 7.43 `spot::evtgba_reachable_iterator_depth_first` Class Reference

An implementation of `spot::evtgba_reachable_iterator` that browses states depth first.

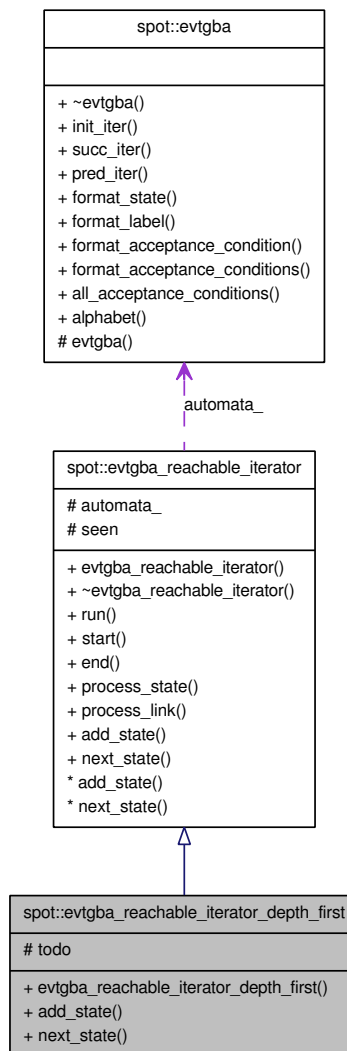
```
#include <evtgbaalgos/reachiter.hh>
```



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.*

#### 7.43.1 Detailed Description

An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states depth first.

#### 7.43.2 Member Typedef Documentation

- 7.43.2.1** typedef Sgi::hash\_map<const state\*, int, state\_ptr\_hash, state\_ptr\_equal>  
spot::evtgba\_reachable\_iterator::seen\_map [protected, inherited]

#### 7.43.3 Constructor & Destructor Documentation

- 7.43.3.1** spot::evtgba\_reachable\_iterator\_depth\_first::evtgba\_reachable\_iterator\_depth\_first  
(const evtgba \* a)

#### 7.43.4 Member Function Documentation

- 7.43.4.1** virtual void spot::evtgba\_reachable\_iterator\_depth\_first::add\_state (const state \* s)  
[virtual]

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

**7.43.4.2 virtual void spot::evtgba\_reachable\_iterator::end () [virtual, inherited]**

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

**7.43.4.3 virtual const state\* spot::evtgba\_reachable\_iterator\_depth\_first::next\_state () [virtual]**

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

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

**7.43.4.4 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.

**7.43.4.5 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*.

**7.43.4.6 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.

**7.43.4.7 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.

#### 7.43.5 Member Data Documentation

##### 7.43.5.1 `const evtgba* spot::evtgba_reachable_iterator::automata_` `[protected, inherited]`

The `spot::evtgba` to explore.

##### 7.43.5.2 `seen_map spot::evtgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

##### 7.43.5.3 `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:

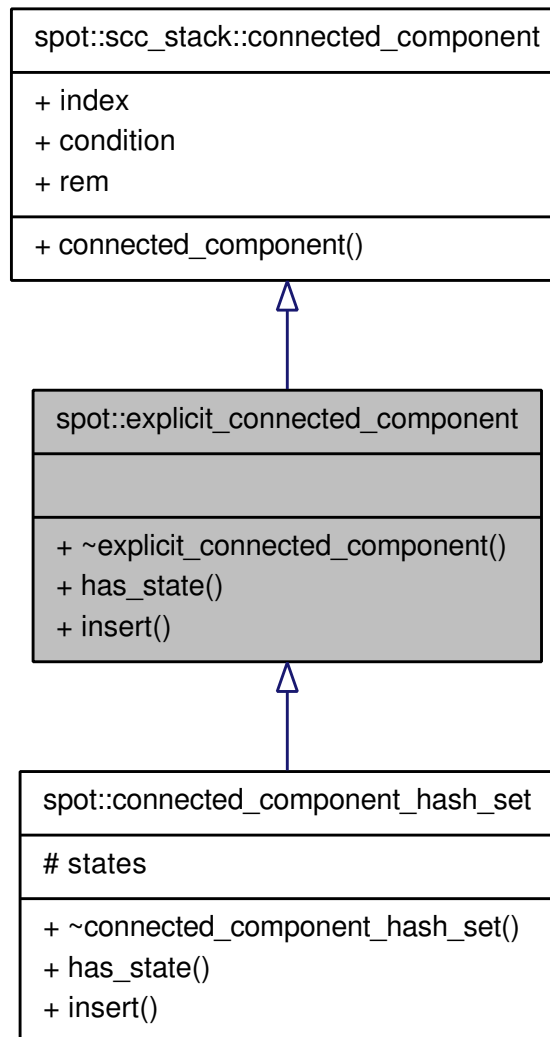
- `evtgbalgorithms/reachiter.hh`

## 7.44 `spot::explicit_connected_component` Class Reference

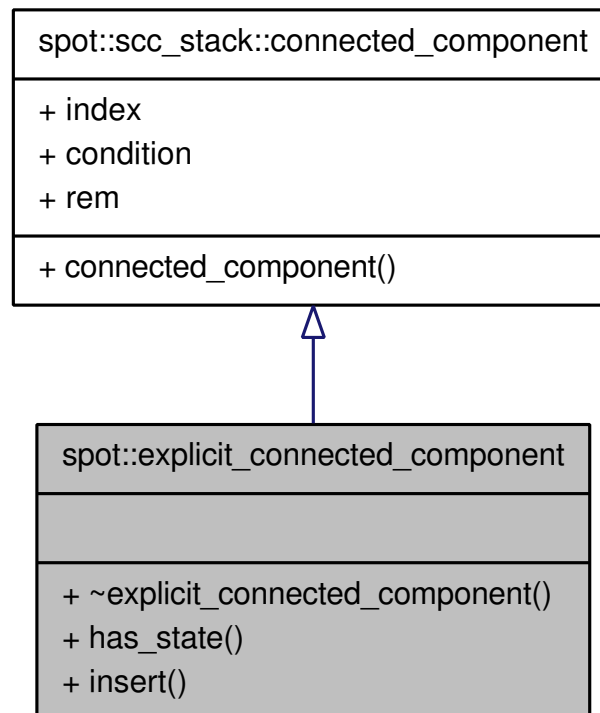
An SCC storing all its states explicitly.

```
#include <tgbalgorithms/gtec/explscch>
```

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`

#### 7.44.1 Detailed Description

An SCC storing all its states explicitly.

### 7.44.2 Constructor & Destructor Documentation

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

### 7.44.3 Member Function Documentation

**7.44.3.1** `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](#).

**7.44.3.2** `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](#).

### 7.44.4 Member Data Documentation

**7.44.4.1** `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.

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

Index of the SCC.

**7.44.4.3** `std::list<const state*> spot::scc_stack::connected_component::rem [inherited]`

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

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

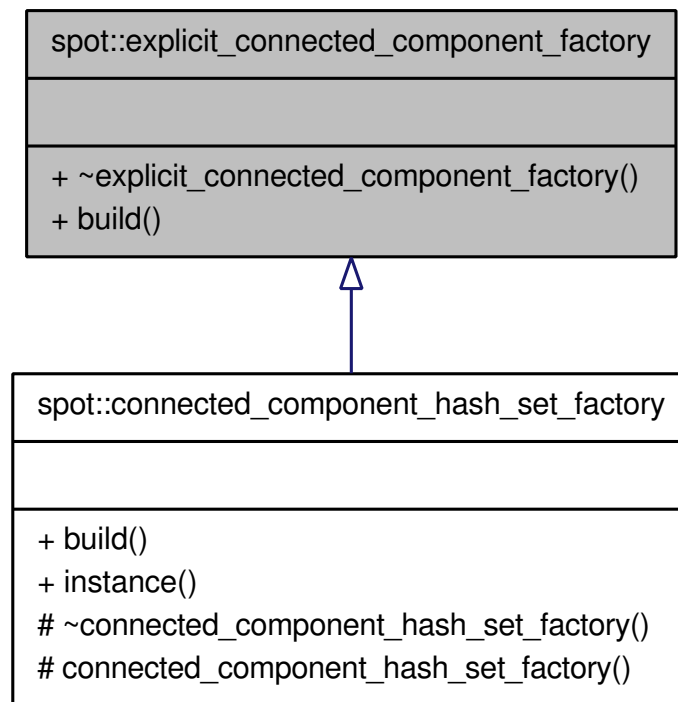


## 7.45 spot::explicit\_connected\_component\_factory Class Reference

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

```
#include <tgbaaalgos/gtec/explscs.hh>
```

Inheritance diagram for spot::explicit\_connected\_component\_factory:



### Public Member Functions

- virtual [~explicit\\_connected\\_component\\_factory](#) ()
- virtual [explicit\\_connected\\_component](#) \* [build](#) () const =0

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

### 7.45.1 Detailed Description

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

### 7.45.2 Constructor & Destructor Documentation

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

### 7.45.3 Member Function Documentation

#### 7.45.3.1 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/explscs.hh](#)

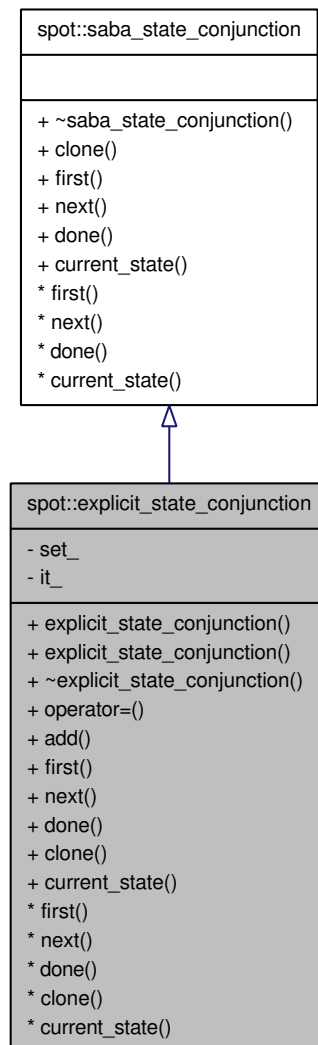
## 7.46 spot::explicit\_state\_conjunction Class Reference

Basic implementation of [saba\\_state\\_conjunction](#).

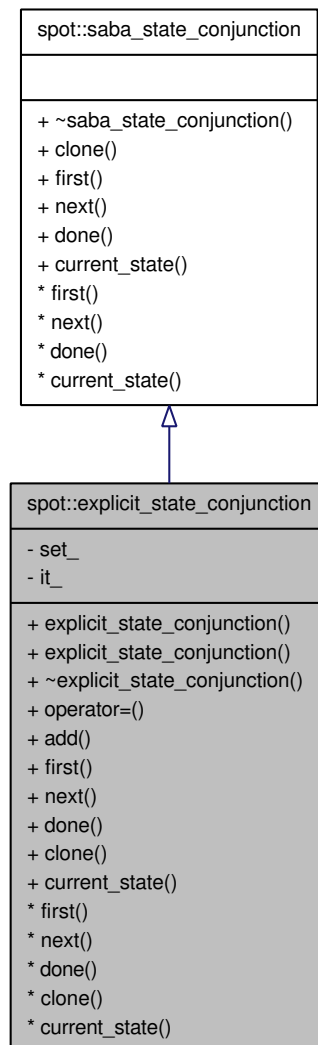
This class provides a basic implementation to iterate over a conjunction of states of a saba.

```
#include <saba/explicitstateconjunction.hh>
```

Inheritance diagram for spot::explicit\_state\_conjunction:



Collaboration diagram for spot::explicit\_state\_conjunction:



## Public Member Functions

- `explicit_state_conjunction ()`
- `explicit_state_conjunction (const explicit_state_conjunction *other)`
- `virtual ~explicit_state_conjunction ()`
- `explicit_state_conjunction * operator= (const explicit_state_conjunction &o)`
- `void add (saba_state *state)`

## Iteration

- `virtual void first ()`  
*Position the iterator on the first successor of the conjunction (if any).*
- `virtual void next ()`  
*Jump to the next successor (if any).*

- virtual bool [done](#) () const  
*Check whether the iteration over a conjunction of states is finished.*

### Inspection

- [explicit\\_state\\_conjunction](#) \* [clone](#) () const  
*Duplicate a this conjunction.*
- virtual [saba\\_state](#) \* [current\\_state](#) () const

### Private Types

- typedef Sgi::hash\_set< [shared\\_saba\\_state](#), [spot::saba\\_state\\_shared\\_ptr\\_hash](#), [spot::saba\\_state\\_shared\\_ptr\\_equal](#) > [saba\\_state\\_set\\_t](#)

### Private Attributes

- [saba\\_state\\_set\\_t](#) [set\\_](#)
- [saba\\_state\\_set\\_t::iterator](#) [it\\_](#)

## 7.46.1 Detailed Description

Basic implementation of [saba\\_state\\_conjunction](#).

This class provides a basic implementation to iterate over a conjunction of states of a saba.

## 7.46.2 Member Typedef Documentation

- 7.46.2.1** typedef Sgi::hash\_set<[shared\\_saba\\_state](#), [spot::saba\\_state\\_shared\\_ptr\\_hash](#), [spot::saba\\_state\\_shared\\_ptr\\_equal](#)> [spot::explicit\\_state\\_conjunction::saba\\_state\\_set\\_t](#) [**private**]

## 7.46.3 Constructor & Destructor Documentation

- 7.46.3.1** [spot::explicit\\_state\\_conjunction::explicit\\_state\\_conjunction](#) ()

- 7.46.3.2** [spot::explicit\\_state\\_conjunction::explicit\\_state\\_conjunction](#) (const [explicit\\_state\\_conjunction](#) \* *other*)

- 7.46.3.3** virtual [spot::explicit\\_state\\_conjunction::~~explicit\\_state\\_conjunction](#) () [**virtual**]

### 7.46.4 Member Function Documentation

#### 7.46.4.1 `void spot::explicit_state_conjunction::add (saba_state * state)`

Add a new state in the conjunction. The class becomes owner of *state*.

#### 7.46.4.2 `explicit_state_conjunction* spot::explicit_state_conjunction::clone () const` **[virtual]**

Duplicate a this conjunction.

Implements [spot::saba\\_state\\_conjunction](#).

#### 7.46.4.3 `virtual saba_state* spot::explicit_state_conjunction::current_state () const` **[virtual]**

Return the a new instance on the current state. This is the caller responsibility to delete the returned state.

Implements [spot::saba\\_state\\_conjunction](#).

#### 7.46.4.4 `virtual bool spot::explicit_state_conjunction::done () const` **[virtual]**

Check whether the iteration over a conjunction of states is finished.

This function should be called after any call to [first \(\)](#) or [next \(\)](#) and before any enquiry about the current state.

Implements [spot::saba\\_state\\_conjunction](#).

#### 7.46.4.5 `virtual void spot::explicit_state_conjunction::first ()` **[virtual]**

Position the iterator on the first successor of the conjunction (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::saba\\_state\\_conjunction](#).

#### 7.46.4.6 `virtual void spot::explicit_state_conjunction::next ()` **[virtual]**

Jump to the next successor (if any).

#### Warning

Again, one should always call [done \(\)](#) to ensure there is a successor.

Implements [spot::saba\\_state\\_conjunction](#).

**7.46.4.7** `explicit_state_conjunction* spot::explicit_state_conjunction::operator= (const explicit_state_conjunction & o)`

## 7.46.5 Member Data Documentation

**7.46.5.1** `saba_state_set_t::iterator spot::explicit_state_conjunction::it_ [private]`

**7.46.5.2** `saba_state_set_t spot::explicit_state_conjunction::set_ [private]`

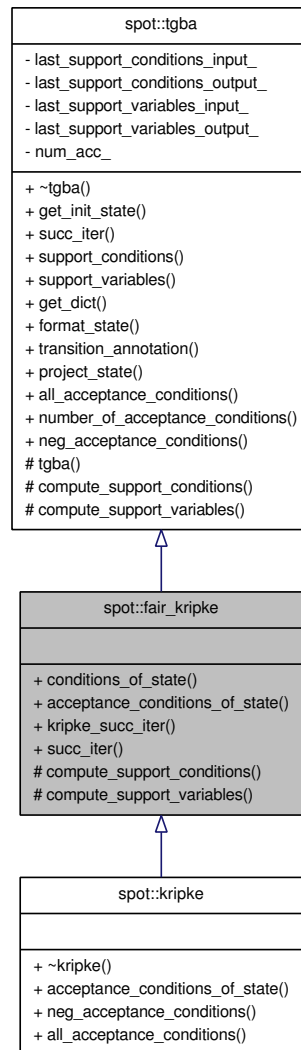
The documentation for this class was generated from the following file:

- [saba/explicitstateconjunction.hh](#)

## 7.47 spot::fair\_kripke Class Reference

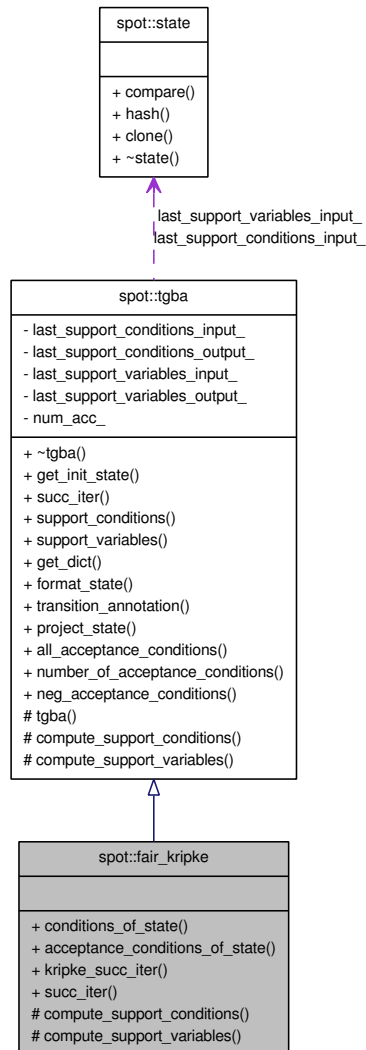
```
#include <kripke/fairkripke.hh>
```

Inheritance diagram for spot::fair\_kripke:





Collaboration diagram for spot::fair\_kripke:



## Public Member Functions

- virtual bdd [conditions\\_of\\_state](#) (const [state](#) \*s) const =0
- virtual bdd [acceptance\\_conditions\\_of\\_state](#) (const [state](#) \*s) const =0
- virtual [fair\\_kripke\\_succ\\_iterator](#) \* [kripke\\_succ\\_iter](#) (const [state](#) \*s)=0
- virtual [fair\\_kripke\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [state](#) \*local\_state, const [state](#) \*global\_state=0, const [tgba](#) \*global\_automaton=0)
- 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

- virtual bdd [compute\\_support\\_conditions](#) (const [state](#) \*s) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [state](#) \*s) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

#### 7.47.1 Member Function Documentation

**7.47.1.1** virtual bdd [spot::fair\\_kripke::acceptance\\_conditions\\_of\\_state](#) (const [state](#) \* s) const  
[pure virtual]

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

**7.47.1.2** virtual bdd [spot::tgba::all\\_acceptance\\_conditions](#) () const [pure 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.

Implemented in `spot::kripke`, `spot::taa_tgba`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

**7.47.1.3** `virtual bdd spot::fair_kripke::compute_support_conditions (const state * state) const [protected, virtual]`

Do the actual computation of `tgba::support_conditions()`.

Implements `spot::tgba`.

**7.47.1.4** `virtual bdd spot::fair_kripke::compute_support_variables (const state * state) const [protected, virtual]`

Do the actual computation of `tgba::support_variables()`.

Implements `spot::tgba`.

**7.47.1.5** `virtual bdd spot::fair_kripke::conditions_of_state (const state * s) const [pure virtual]`

**7.47.1.6** `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 that owns the state.

Implemented in `spot::future_conditions_collector`, `spot::taa_tgba`, `spot::taa_tgba_labelled< label, label_hash >`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_explicit_string`, `spot::tgba_explicit_formula`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_reduc`, `spot::tgba_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, `spot::tgba_union`, `spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >`, and `spot::taa_tgba_labelled< std::string, string_hash >`.

**7.47.1.7** `virtual bdd_dict* spot::tgba::get_dict () const [pure 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.

Implemented in [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

#### 7.47.1.8 virtual state\* spot::tgba::get\_init\_state () const [pure 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.

Implemented in [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), [spot::tgba\\_sba\\_proxy](#), and [spot::tgba\\_union](#).

#### 7.47.1.9 virtual fair\_kripke\_succ\_iterator\* spot::fair\_kripke::kripke\_succ\_iter (const state \* s) [pure virtual]

#### 7.47.1.10 virtual bdd spot::tgba::neg\_acceptance\_conditions () const [pure 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.

Implemented in [spot::kripke](#), [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

#### 7.47.1.11 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

#### 7.47.1.12 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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.47.1.13** `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::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.47.1.14** `virtual fair_kripke_succ_iterator* spot::fair_kripke::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0)` `[virtual]`

**7.47.1.15** `bdd spot::tgba::support_conditions (const state * state) const` `[inherited]`

Get a formula that must hold whatever successor is taken.

### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.47.1.16 `bdd spot::tgba::support_variables (const state * state) const` **[inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.47.1.17 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

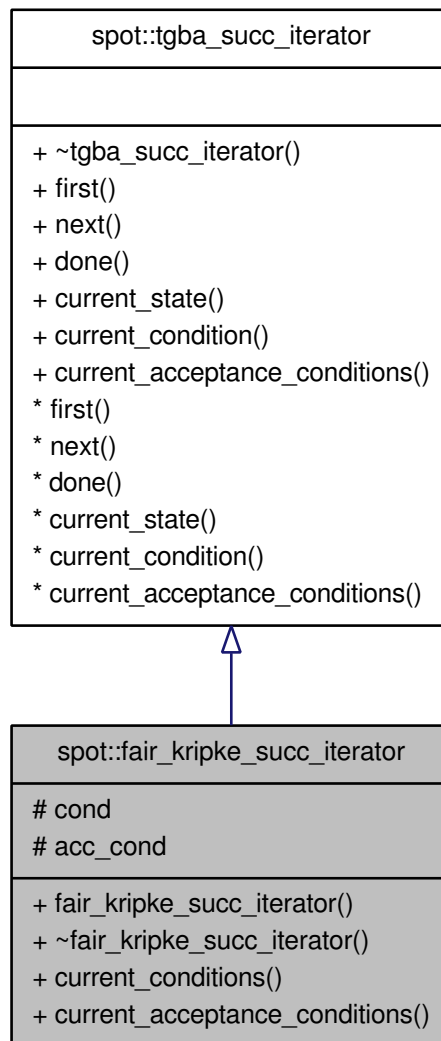
The documentation for this class was generated from the following file:

- `kripke/fairkripke.hh`

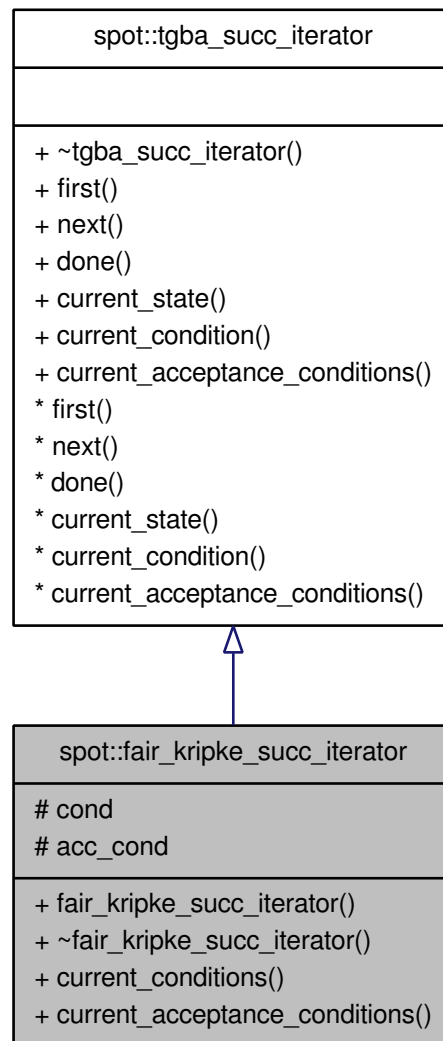
## 7.48 `spot::fair_kripke_succ_iterator` Class Reference

```
#include <kripke/fairkripke.hh>
```

Inheritance diagram for spot::fair\_kripke\_succ\_iterator:



Collaboration diagram for spot::fair\_kripke\_succ\_iterator:



### Public Member Functions

- `fair_kripke_succ_iterator` (const `fair_kripke` \*aut, const `state` \*st)
- virtual `~fair_kripke_succ_iterator` ()
- virtual bdd `current_conditions` () const
- virtual bdd `current_acceptance_conditions` () const

*Get the acceptance conditions on the transition leading to this successor.*

### 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.*

### Protected Attributes

- bdd [cond](#)
- bdd [acc\\_cond](#)

## 7.48.1 Constructor & Destructor Documentation

**7.48.1.1** `spot::fair_kripke_succ_iterator::fair_kripke_succ_iterator (const fair_kripke * aut, const state * st)`

**7.48.1.2** `virtual spot::fair_kripke_succ_iterator::~~fair_kripke_succ_iterator ()` **[virtual]**

## 7.48.2 Member Function Documentation

**7.48.2.1** `virtual bdd spot::fair_kripke_succ_iterator::current_acceptance_conditions () const` **[virtual]**

Get the acceptance conditions on the transition leading to this successor.

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

**7.48.2.2** `virtual bdd spot::tgba_succ_iterator::current_condition () const` **[pure virtual, inherited]**

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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.48.2.3** `virtual bdd spot::fair_kripke_succ_iterator::current_conditions () const [virtual]`

**7.48.2.4** `virtual state* spot::tgba_succ_iterator::current_state () const [pure virtual, inherited]`

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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.48.2.5** `virtual bool spot::tgba_succ_iterator::done () const [pure virtual, inherited]`

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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.48.2.6** `virtual void spot::tgba_succ_iterator::first () [pure virtual, inherited]`

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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.48.2.7** `virtual void spot::tgba_succ_iterator::next () [pure virtual, inherited]`

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::taa_succ_iterator`, `spot::tgba_explicit_succ_iterator`, `spot::tgba_succ_iterator_product`, and `spot::tgba_succ_iterator_union`.

### 7.48.3 Member Data Documentation

**7.48.3.1** `bdd spot::fair_kripke_succ_iterator::acc_cond` `[protected]`

**7.48.3.2** `bdd spot::fair_kripke_succ_iterator::cond` `[protected]`

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

- `kripke/fairkripke.hh`

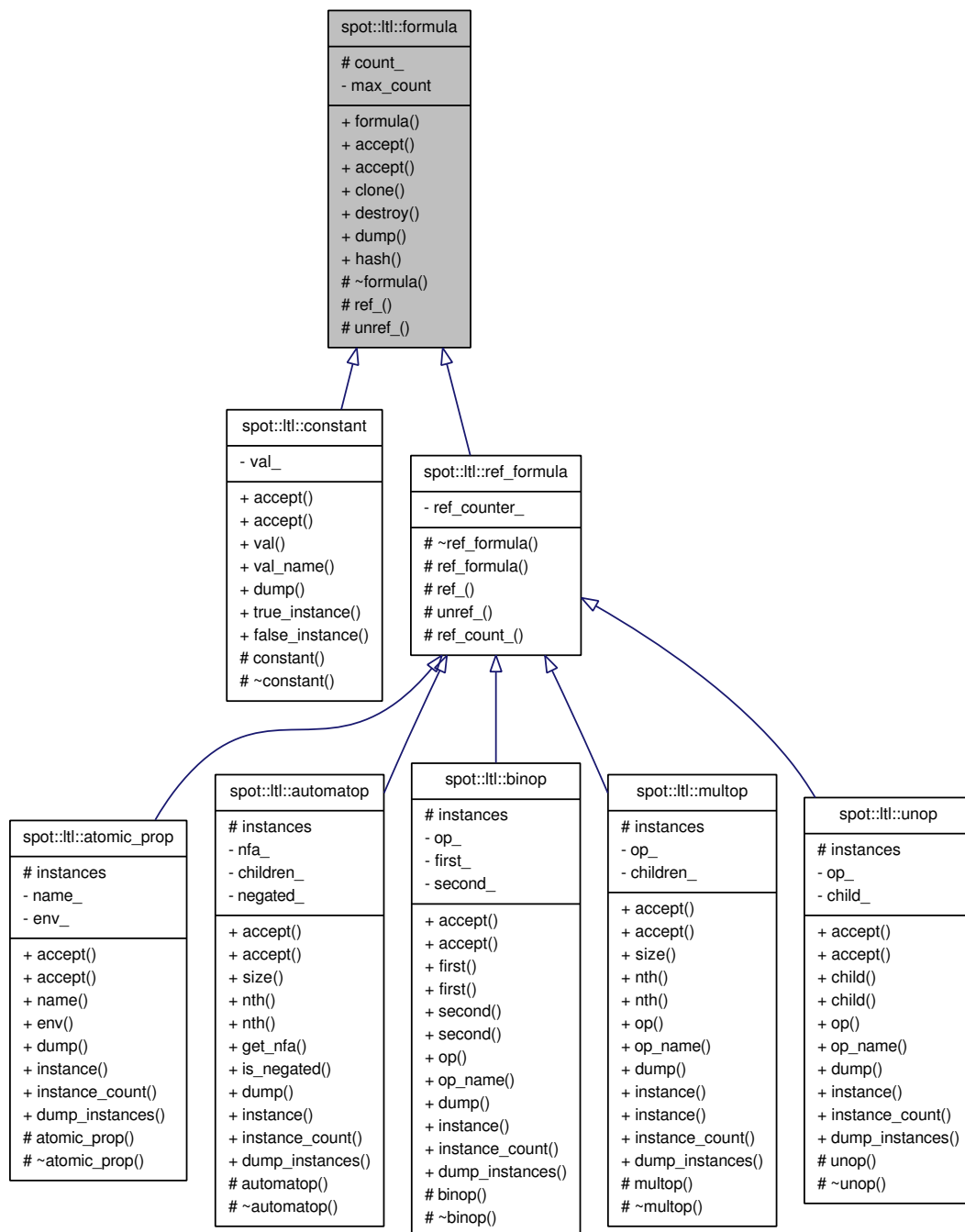
## 7.49 `spot::ltl::formula` Class Reference

An LTL formula.

The only way you can work with a formula is to build a `spot::ltl::visitor` or `spot::ltl::const_visitor`.

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

Inheritance diagram for spot::ltl::formula:



## Public Member Functions

- [formula\(\)](#)
- virtual void [accept](#) (visitor &v)=0

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

- virtual void [accept](#) ([const\\_visitor](#) &v) const =0  
*Entry point for vspot::ltl::const\_visitor instances.*
- [formula](#) \* [clone](#) () const  
*clone this node*
- void [destroy](#) () const  
*release this node*
- virtual std::string [dump](#) () const =0  
*Return a canonic representation of the formula.*
- size\_t [hash](#) () const  
*Return a hash key for the formula.*

### 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).*

### Protected Attributes

- size\_t [count\\_](#)  
*The hash key of this formula.*

### Static Private Attributes

- static size\_t [max\\_count](#)  
*Number of formulae created so far.*

#### 7.49.1 Detailed Description

An LTL formula.

The only way you can work with a formula is to build a [spot::ltl::visitor](#) or [spot::ltl::const\\_visitor](#).

#### 7.49.2 Constructor & Destructor Documentation

##### 7.49.2.1 spot::ltl::formula::formula () [inline]

**7.49.2.2** `virtual spot::ltl::formula::~~formula () [protected, virtual]`**7.49.3** Member Function Documentation**7.49.3.1** `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::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

**7.49.3.2** `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::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

**7.49.3.3** `formula* spot::ltl::formula::clone () const`

clone this node

This increments the reference counter of this node (if one is used).

**7.49.3.4** `void spot::ltl::formula::destroy () const`

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.49.3.5** `virtual std::string spot::ltl::formula::dump () const [pure virtual]`

Return a canonic representation of the formula.

Implemented in [spot::ltl::atomic\\_prop](#), [spot::ltl::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

**7.49.3.6** `size_t spot::ltl::formula::hash () const [inline]`

Return a hash key for the formula.

References count\_.

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

increment reference counter if any

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

#### 7.49.3.8 `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](#).

### 7.49.4 Member Data Documentation

#### 7.49.4.1 `size_t spot::ltl::formula::count_` `[protected]`

The hash key of this formula.

Referenced by `hash()`.

#### 7.49.4.2 `size_t spot::ltl::formula::max_count` `[static, private]`

Number of formulae created so far.

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

- `ltlast/formula.hh`

## 7.50 `spot::ltl::formula_ptr_hash` Struct Reference

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*`.

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

### Public Member Functions

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

### 7.50.1 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;
```

### 7.50.2 Member Function Documentation

#### 7.50.2.1 `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](#)

## 7.51 spot::ltl::formula\_ptr\_less\_than Struct Reference

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*`.

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

### Public Member Functions

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

### 7.51.1 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;
```

### 7.51.2 Member Function Documentation

#### 7.51.2.1 `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](#)

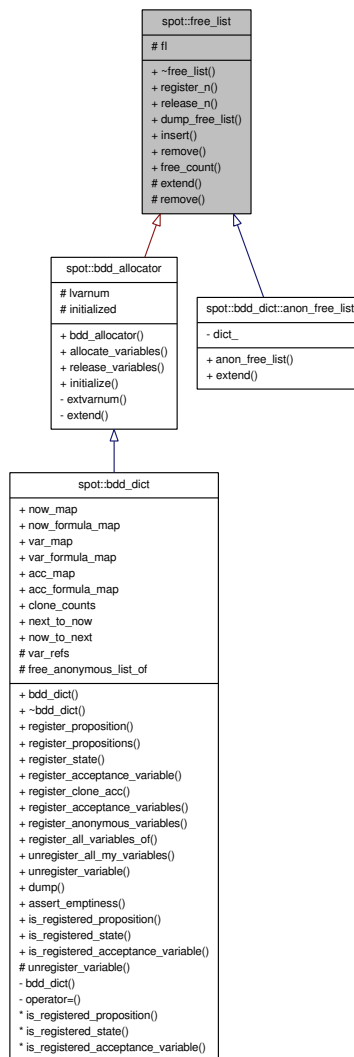


## 7.52 spot::free\_list Class Reference

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-lenght pair.*
- void [remove](#) (int base, int n=0)  
*Remove n consecutive entries from the list, starting at base.*
- int [free\\_count](#) () const  
*Return the number of free integers on the list.*

### Protected Types

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

### Protected Member Functions

- virtual int [extend](#) (int n)=0
- void [remove](#) (free\_list\_type::iterator i, int base, int n)  
*Remove n consecutive entries from the list, starting at base.*

### Protected Attributes

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

#### 7.52.1 Detailed Description

Manage list of free integers.

#### 7.52.2 Member Typedef Documentation

**7.52.2.1** typedef std::list<pos\_lenght\_pair> spot::free\_list::free\_list\_type [protected]

**7.52.2.2** typedef std::pair<int, int> spot::free\_list::pos\_lenght\_pair [protected]

Such pairs describe second free integer starting at first.

### 7.52.3 Constructor & Destructor Documentation

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

### 7.52.4 Member Function Documentation

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

Dump the list to *os* for debugging.

#### 7.52.4.2 `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::anon\\_free\\_list](#).

#### 7.52.4.3 `int spot::free_list::free_count () const`

Return the number of free integers on the list.

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

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

#### 7.52.4.5 `int spot::free_list::register_n (int n)`

Find *n* consecutive integers.

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

#### Returns

the first integer of the range

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

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

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

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

**7.52.4.8** `void spot::free_list::remove (int base, int n = 0)`

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

**7.52.5 Member Data Documentation****7.52.5.1** `free_list_type spot::free_list::fl` `[protected]`

Tracks unused BDD variables.

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

- [misc/freelist.hh](#)

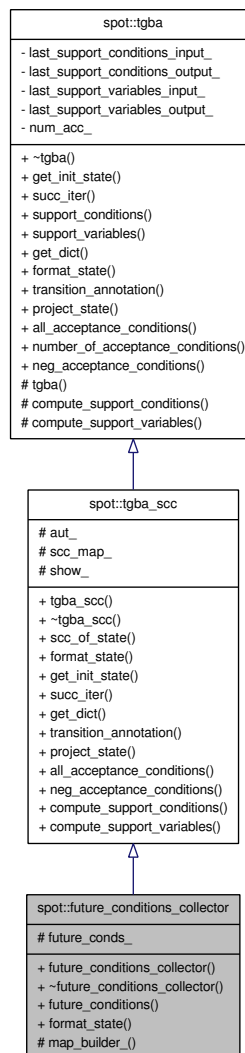
**7.53 `spot::future_conditions_collector` Class Reference**

Wrap a `tgba` to offer information about upcoming conditions.

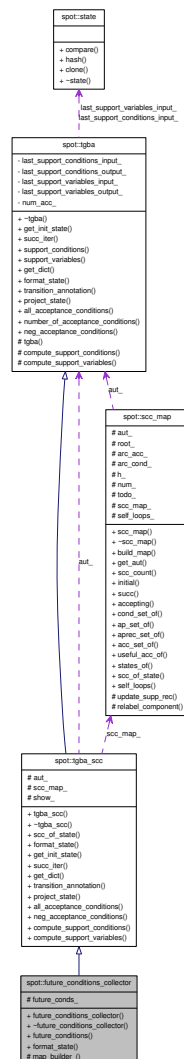
This class is a `spot::tgba` wrapper that simply add a new method, `future_conditions()`, to any `spot::tgba`.

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

Inheritance diagram for spot::future\_conditions\_collector:



Collaboration diagram for spot::future\_conditions\_collector:



## Public Types

- typedef scc\_map::cond\_set cond\_set
- typedef std::vector< cond\_set > fc\_map

## Public Member Functions

- `future_conditions_collector` (const `tgba` \*aut, bool show=false)  
*Create a `future_conditions_collector` wrapper for aut.*
- virtual `~future_conditions_collector` ()
- const `cond_set` & `future_conditions` (const `spot::state` \*s) const  
*Returns the set of future conditions visible after s.*
- virtual std::string `format_state` (const `state` \*state) const

*Format a state for output.*

- unsigned `scc_of_state` (const `spot::state` \*s) const  
*Returns the number of the SCC s belongs to.*
- 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 `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.*
- 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().*
- 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

- void `map_builder_` (unsigned s)

### Protected Attributes

- `fc_map` `future_conds_`
- `const tgba * aut_`
- `scc_map` `scc_map_`
- `bool show_`

#### 7.53.1 Detailed Description

Wrap a `tgba` to offer information about upcoming conditions.

This class is a `spot::tgba` wrapper that simply add a new method, `future_conditions()`, to any `spot::tgba`. This new method returns a set of conditions that can be seen on a transitions accessible (maybe indirectly) from the given state.

#### 7.53.2 Member Typedef Documentation

**7.53.2.1** `typedef scc_map::cond_set spot::future_conditions_collector::cond_set`

**7.53.2.2** `typedef std::vector<cond_set> spot::future_conditions_collector::fc_map`

#### 7.53.3 Constructor & Destructor Documentation

**7.53.3.1** `spot::future_conditions_collector::future_conditions_collector (const tgba * aut, bool show = false)`

Create a `future_conditions_collector` wrapper for `aut`.

If `show` is set to true, then the `format_state()` method will include the set of conditions computed for the given state in its output string.

**7.53.3.2** `virtual spot::future_conditions_collector::~~future_conditions_collector () [virtual]`

#### 7.53.4 Member Function Documentation

**7.53.4.1** `virtual bdd spot::tgba_scc::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 accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.53.4.2** `virtual bdd spot::tgba_scc::compute_support_conditions (const state * state) const`  
`[virtual, inherited]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.53.4.3** `virtual bdd spot::tgba_scc::compute_support_variables (const state * state) const`  
`[virtual, inherited]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.53.4.4** `virtual std::string spot::future_conditions_collector::format_state (const state * state)`  
`const [virtual]`

Format a state for output.

If the constructor was called with *show* set to true, then this method will include the set of conditions computed for *state* by [future\\_conditions\(\)](#) in the output string.

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

**7.53.4.5** `const cond_set& spot::future_conditions_collector::future_conditions (const spot::state * s) const`

Returns the set of future conditions visible after *s*.

**7.53.4.6** `virtual bdd_dict* spot::tgba_scc::get_dict () const` `[virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.53.4.7 virtual state\* spot::tgba\_scc::get\_init\_state () const [virtual, inherited]**

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.53.4.8 void spot::future\_conditions\_collector::map\_builder\_ (unsigned s) [protected]****7.53.4.9 virtual bdd spot::tgba\_scc::neg\_acceptance\_conditions () const [virtual, inherited]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.53.4.10 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**7.53.4.11 virtual state\* spot::tgba\_scc::project\_state (const state \* s, const tgba \* t) const [virtual, inherited]**

Project a state on an automaton.

This converts `s`, into that corresponding [spot::state](#) for `t`. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that `s` and `t` should be compatible (i.e., `s` is a state of `t`).

**Returns**

0 if the projection fails (`s` is unrelated to `t`), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented from [spot::tgba](#).

**7.53.4.12** `unsigned spot::tgba_scc::scc_of_state (const spot::state * s) const` **[inherited]**

Returns the number of the SCC *s* belongs to.

**7.53.4.13** `virtual tgba_succ_iterator* spot::tgba_scc::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`.

**7.53.4.14** `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.

**7.53.4.15** `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.

**7.53.4.16** `virtual std::string spot::tgba_scc::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 from `spot::tgba`.

### 7.53.5 Member Data Documentation

**7.53.5.1** `const tgba* spot::tgba_scc::aut_` `[protected, inherited]`

**7.53.5.2** `fc_map spot::future_conditions_collector::future_conds_` `[protected]`

**7.53.5.3** `scc_map spot::tgba_scc::scc_map_` `[protected, inherited]`

**7.53.5.4** `bool spot::tgba_scc::show_` `[protected, inherited]`

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

- `tgba/futurecondcol.hh`

## 7.54 spot::gspn\_exception Class Reference

An exception used to forward GSPN errors.

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

#### Public Member Functions

- `gspn_exception` (const std::string &where, int err)

- int [get\\_err](#) () const
- std::string [get\\_where](#) () const

#### Private Attributes

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

#### 7.54.1 Detailed Description

An exception used to forward GSPN errors.

#### 7.54.2 Constructor & Destructor Documentation

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

#### 7.54.3 Member Function Documentation

**7.54.3.1** int spot::gspn\_exception::get\_err () const [inline]

References [err\\_](#).

**7.54.3.2** std::string spot::gspn\_exception::get\_where () const [inline]

References [where\\_](#).

#### 7.54.4 Member Data Documentation

**7.54.4.1** int spot::gspn\_exception::err\_ [private]

Referenced by [get\\_err\(\)](#).

**7.54.4.2** std::string spot::gspn\_exception::where\_ [private]

Referenced by [get\\_where\(\)](#).

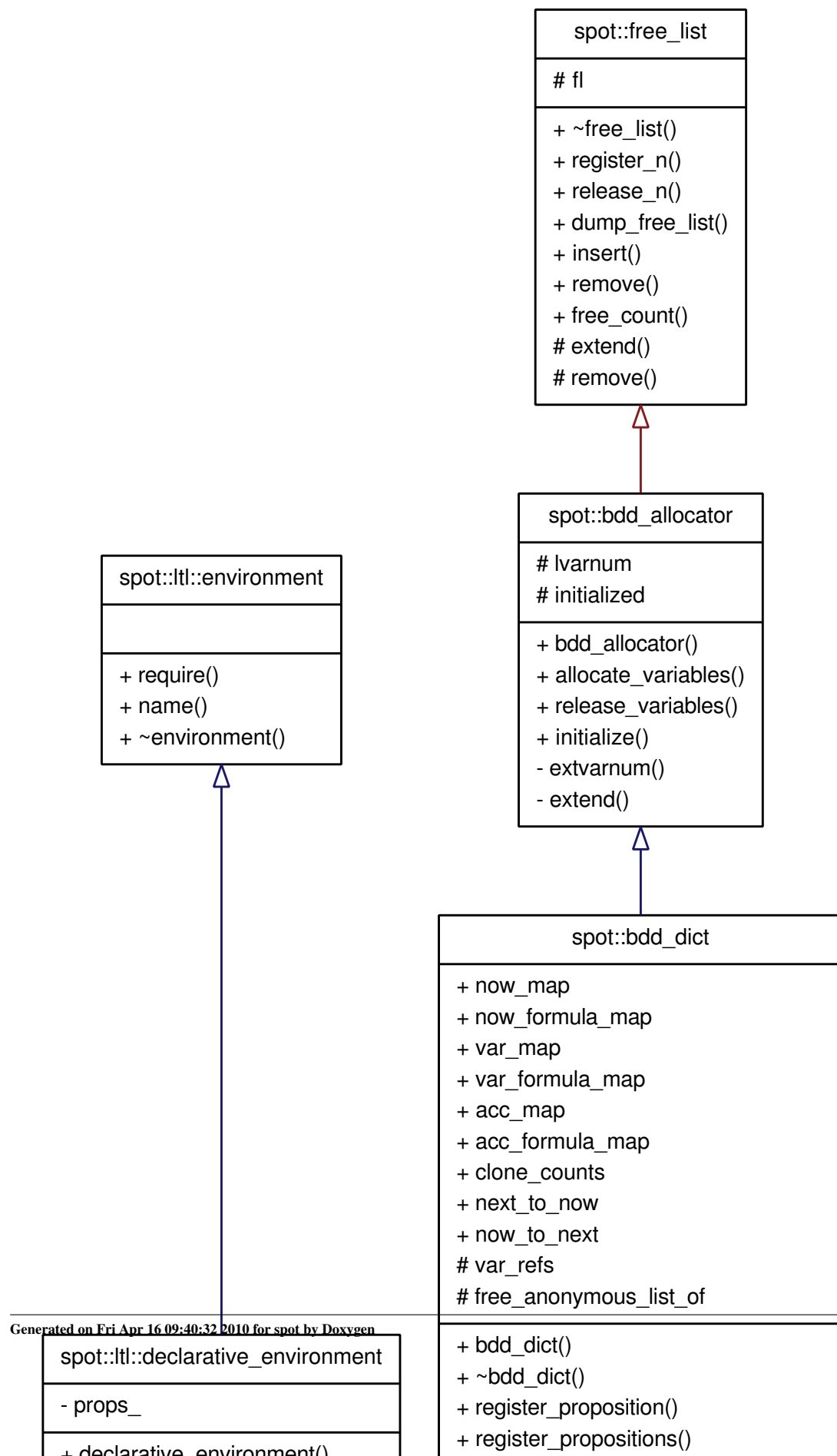
The documentation for this class was generated from the following file:

- [gspn/common.hh](#)

## 7.55 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\\_](#)

### 7.55.1 Constructor & Destructor Documentation

**7.55.1.1** [spot::gspn\\_interface::gspn\\_interface](#) (int *argc*, char \*\* *argv*, [bdd\\_dict](#) \* *dict*, [ltl::declarative\\_environment](#) & *env*, const std::string & *dead* = "true")

**7.55.1.2** [spot::gspn\\_interface::~~gspn\\_interface](#) ()

### 7.55.2 Member Function Documentation

**7.55.2.1** [tgba](#)\* [spot::gspn\\_interface::automaton](#) () const

### 7.55.3 Member Data Documentation

**7.55.3.1** const std::string [spot::gspn\\_interface::dead\\_](#) [[private](#)]

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

**7.55.3.3** [ltl::declarative\\_environment](#)& [spot::gspn\\_interface::env\\_](#) [[private](#)]

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

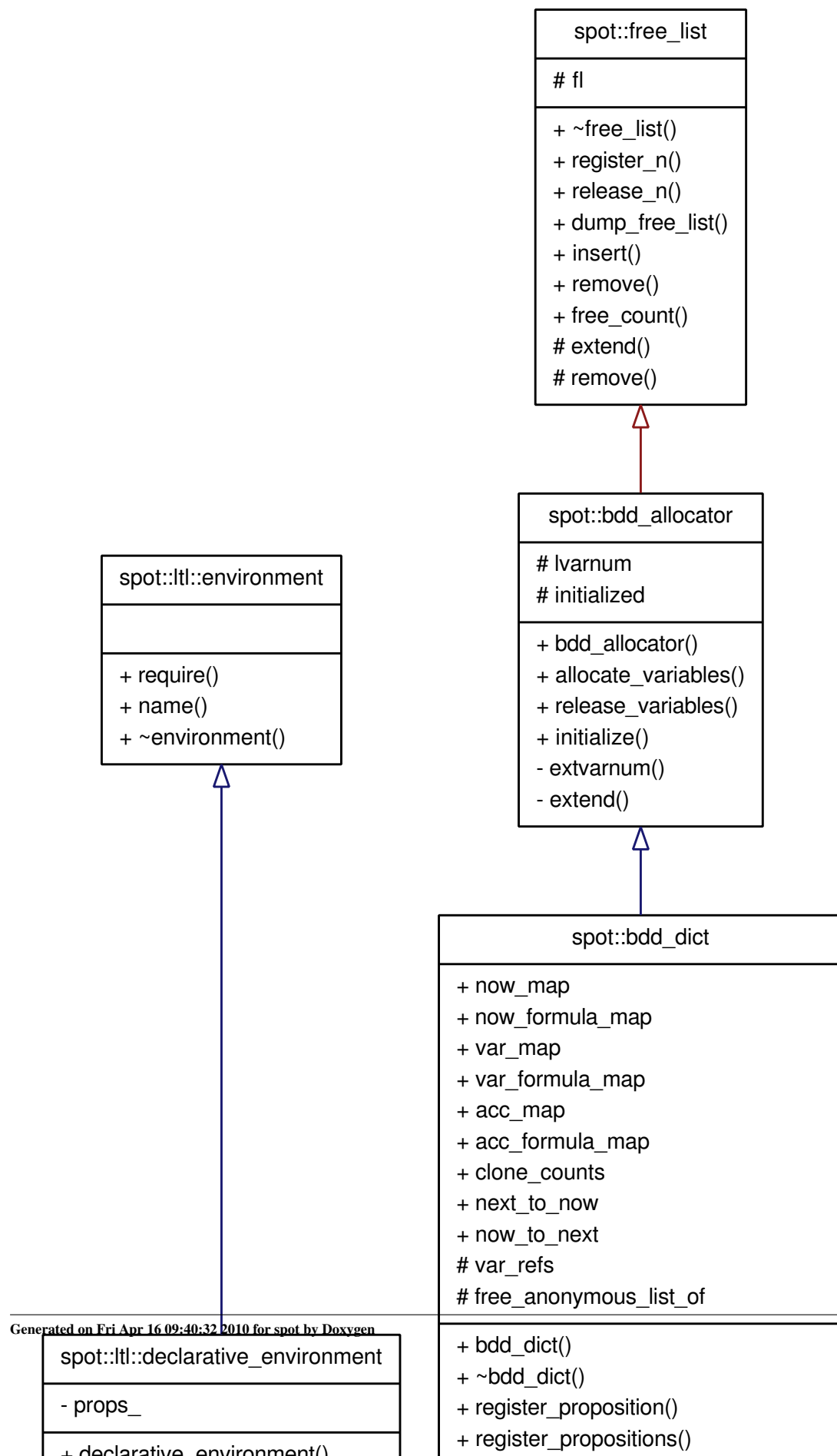
- [gspn/gspn.hh](#)



## 7.56 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, bool doublehash=true, bool pushfront=false)
- [~gspn\\_ssp\\_interface](#) ()
- [tgba](#) \* [automaton](#) (const [tgba](#) \*operand) const

**Private Attributes**

- [bdd\\_dict](#) \* [dict\\_](#)
- const [ltl::declarative\\_environment](#) & [env\\_](#)

**7.56.1 Constructor & Destructor Documentation**

**7.56.1.1** `spot::gspn_ssp_interface::gspn_ssp_interface (int argc, char ** argv, bdd_dict * dict, const ltl::declarative_environment & env, bool inclusion = false, bool doublehash = true, bool pushfront = false)`

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

**7.56.2 Member Function Documentation**

**7.56.2.1** `tgba* spot::gspn_ssp_interface::automaton (const tgba * operand) const`

**7.56.3 Member Data Documentation**

**7.56.3.1** `bdd_dict* spot::gspn_ssp_interface::dict_ [private]`

**7.56.3.2** `const ltl::declarative_environment& spot::gspn_ssp_interface::env_ [private]`

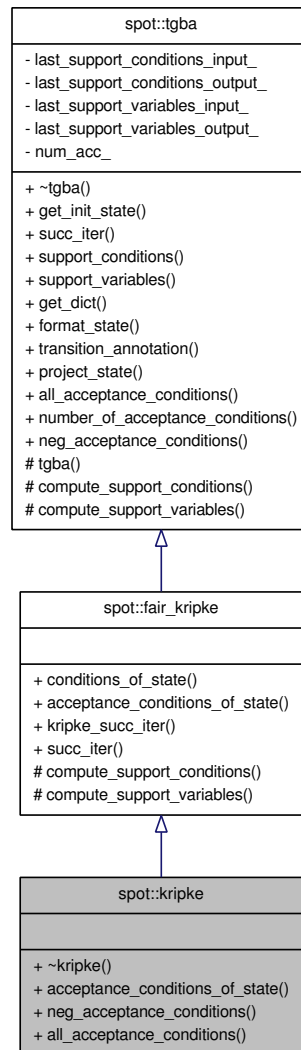
The documentation for this class was generated from the following file:

- [gspn/ssp.hh](#)

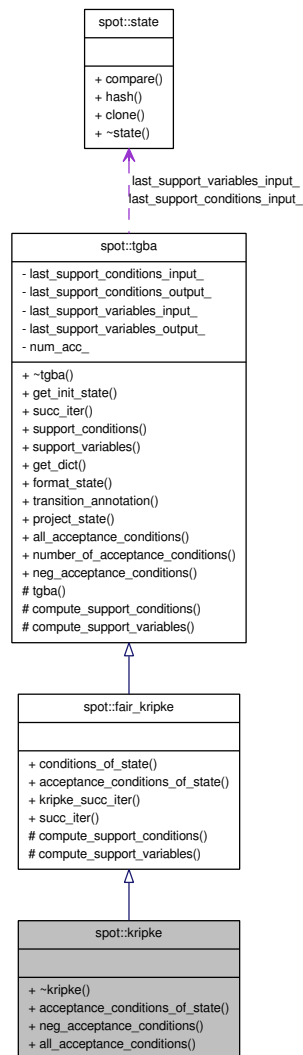
**7.57 spot::kripke Class Reference**

```
#include <kripke/kripke.hh>
```

Inheritance diagram for spot::kripke:



Collaboration diagram for spot::kripke:



## Public Member Functions

- virtual `~kripke()`
- virtual bdd `acceptance_conditions_of_state` (const `state` \*s) const
- virtual bdd `neg_acceptance_conditions` () const  
*Return the conjunction of all negated acceptance variables.*
- virtual bdd `all_acceptance_conditions` () const  
*Return the set of all acceptance conditions used by this automaton.*
- virtual bdd `conditions_of_state` (const `state` \*s) const =0
- virtual `fair_kripke_succ_iterator` \* `kripke_succ_iter` (const `state` \*s)=0
- virtual `fair_kripke_succ_iterator` \* `succ_iter` (const `state` \*local\_state, const `state` \*global\_state=0, const `tgba` \*global\_automaton=0)

- 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 `state * get_init_state` () const =0  
*Get the initial state of the automaton.*
- 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 unsigned int `number_of_acceptance_conditions` () const  
*The number of acceptance conditions.*

### Protected Member Functions

- virtual bdd `compute_support_conditions` (const `state` \*s) const  
*Do the actual computation of `tgba::support_conditions()`.*
- virtual bdd `compute_support_variables` (const `state` \*s) const  
*Do the actual computation of `tgba::support_variables()`.*

## 7.57.1 Constructor & Destructor Documentation

### 7.57.1.1 virtual spot::kripke::~~kripke () [virtual]

## 7.57.2 Member Function Documentation

### 7.57.2.1 virtual bdd spot::kripke::acceptance\_conditions\_of\_state (const `state` \* s) const [virtual]

Implements [spot::fair\\_kripke](#).

#### 7.57.2.2 virtual bdd spot::kripke::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 7.57.2.3 virtual bdd spot::fair\_kripke::compute\_support\_conditions (const state \* state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 7.57.2.4 virtual bdd spot::fair\_kripke::compute\_support\_variables (const state \* state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.57.2.5 virtual bdd spot::fair\_kripke::conditions\_of\_state (const state \* s) const [pure virtual, inherited]

#### 7.57.2.6 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 that owns the state.

Implemented in [spot::future\\_conditions\\_collector](#), [spot::taa\\_tgba](#), [spot::taa\\_tgba\\_labelled< label, label\\_hash >](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_explicit\\_string](#), [spot::tgba\\_explicit\\_formula](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_reduc](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), [spot::tgba\\_union](#), [spot::taa\\_tgba\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#), and [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >](#).

#### 7.57.2.7 virtual bdd\_dict\* spot::tgba::get\_dict () const [pure 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.

Implemented in [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

#### 7.57.2.8 virtual state\* spot::tgba::get\_init\_state () const [pure 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.

Implemented in [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), [spot::tgba\\_sba\\_proxy](#), and [spot::tgba\\_union](#).

#### 7.57.2.9 virtual fair\_kripke\_succ\_iterator\* spot::fair\_kripke::kripke\_succ\_iter (const state \* s) [pure virtual, inherited]

#### 7.57.2.10 virtual bdd spot::kripke::neg\_acceptance\_conditions () const [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

#### 7.57.2.11 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

#### 7.57.2.12 virtual state\* spot::tgba::project\_state (const state \* s, const tgba \* t) const [virtual, inherited]

Project a state on an automaton.



This converts  $s$ , into that corresponding `spot::state` for  $t$ . This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that  $s$  and  $t$  should be compatible (i.e.,  $s$  is a state of  $t$ ).

### Returns

0 if the projection fails ( $s$  is unrelated to  $t$ ), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in `spot::tgba_product`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

**7.57.2.13** `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::taa_tgba`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safracomplement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

**7.57.2.14** `virtual fair_kripke_succ_iterator* spot::fair_kripke::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0)` `[virtual, inherited]`

**7.57.2.15** `bdd spot::tgba::support_conditions (const state * state) const` `[inherited]`

Get a formula that must hold whatever successor is taken.

### Returns

A formula which must be verified for all successors of `state`.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.57.2.16 `bdd spot::tgba::support_variables (const state * state) const` [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.57.2.17 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

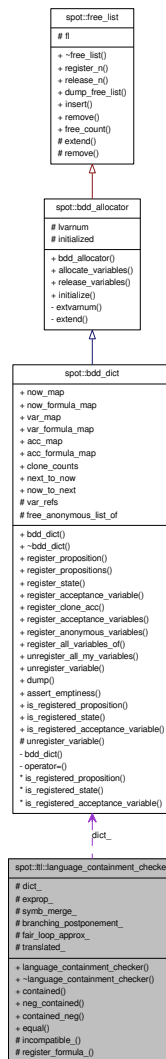
The documentation for this class was generated from the following file:

- [kripke/kripke.hh](#)

## 7.58 spot::ltl::language\_containment\_checker Class Reference

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

Collaboration diagram for spot::ltl::language\_containment\_checker:



## Classes

- struct [record\\_](#)

## Public Member Functions

- [language\\_containment\\_checker](#) ([bdd\\_dict](#) \*dict, bool exprop, bool symb\_merge, bool branching\_postponement, bool fair\_loop\_approx)
- [~language\\_containment\\_checker](#) ()
- bool [contained](#) (const [formula](#) \*l, const [formula](#) \*g)  
*Check whether  $L(l)$  is a subset of  $L(g)$ .*
- bool [neg\\_contained](#) (const [formula](#) \*l, const [formula](#) \*g)  
*Check whether  $L(!l)$  is a subset of  $L(g)$ .*

- bool `contained_neg` (const `formula` \*l, const `formula` \*g)  
*Check whether  $L(l)$  is a subset of  $L(!g)$ .*
- bool `equal` (const `formula` \*l, const `formula` \*g)  
*Check whether  $L(l) = L(g)$ .*

### Protected Member Functions

- bool `incompatible_` (`record_` \*l, `record_` \*g)
- `record_` \* `register_formula_` (const `formula` \*f)

### Protected Attributes

- `bdd_dict` \* `dict_`
- bool `exprop_`
- bool `symb_merge_`
- bool `branching_postponement_`
- bool `fair_loop_approx_`
- `trans_map` `translated_`

### Private Types

- typedef Sgi::hash\_map< const `formula` \*, `record_`, `formula_ptr_hash` > `trans_map`

## 7.58.1 Member Typedef Documentation

- 7.58.1.1** typedef Sgi::hash\_map<const `formula`\*, `record_`, `formula_ptr_hash`>  
spot::ltl::language\_containment\_checker::trans\_map [private]

## 7.58.2 Constructor & Destructor Documentation

- 7.58.2.1** spot::ltl::language\_containment\_checker::language\_containment\_checker (`bdd_dict` \* `dict`, bool `exprop`, bool `symb_merge`, bool `branching_postponement`, bool `fair_loop_approx`)

This class uses `spot::ltl_to_tgba_fm` to translate LTL formulae. See that class for the meaning of these options.

- 7.58.2.2** spot::ltl::language\_containment\_checker::~language\_containment\_checker ()

### 7.58.3 Member Function Documentation

**7.58.3.1** `bool spot::ltl::language_containment_checker::contained (const formula * l, const formula * g)`

Check whether  $L(l)$  is a subset of  $L(g)$ .

**7.58.3.2** `bool spot::ltl::language_containment_checker::contained_neg (const formula * l, const formula * g)`

Check whether  $L(l)$  is a subset of  $L(!g)$ .

**7.58.3.3** `bool spot::ltl::language_containment_checker::equal (const formula * l, const formula * g)`

Check whether  $L(l) = L(g)$ .

**7.58.3.4** `bool spot::ltl::language_containment_checker::incompatible_ (record_ * l, record_ * g)`  
[protected]

**7.58.3.5** `bool spot::ltl::language_containment_checker::neg_contained (const formula * l, const formula * g)`

Check whether  $L(!l)$  is a subset of  $L(g)$ .

**7.58.3.6** `record_* spot::ltl::language_containment_checker::register_formula_ (const formula * f)`  
[protected]

### 7.58.4 Member Data Documentation

**7.58.4.1** `bool spot::ltl::language_containment_checker::branching_postponement_`  
[protected]

**7.58.4.2** `bdd_dict* spot::ltl::language_containment_checker::dict_` [protected]

7.58.4.3 bool spot::ltl::language\_containment\_checker::exprop\_ [protected]

7.58.4.4 bool spot::ltl::language\_containment\_checker::fair\_loop\_approx\_ [protected]

7.58.4.5 bool spot::ltl::language\_containment\_checker::symb\_merge\_ [protected]

7.58.4.6 trans\_map spot::ltl::language\_containment\_checker::translated\_ [protected]

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

- ltlvisit/[contain.hh](#)

## 7.59 spot::minato\_isop::local\_vars Struct Reference

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

### Public Types

- enum { [FirstStep](#), [SecondStep](#), [ThirdStep](#), [FourthStep](#) }

### Public Member Functions

- [local\\_vars](#) (bdd [f\\_min](#), bdd [f\\_max](#), bdd [vars](#))

### Public Attributes

- bdd [f\\_min](#)
- bdd [f\\_max](#)
- enum spot::minato\_isop::local\_vars:: { ... } [step](#)
- bdd [vars](#)
- bdd [v1](#)
- bdd [f0\\_min](#)
- bdd [f0\\_max](#)
- bdd [f1\\_min](#)
- bdd [f1\\_max](#)
- bdd [g0](#)
- bdd [g1](#)

### 7.59.1 Detailed Description

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

## 7.59.2 Member Enumeration Documentation

### 7.59.2.1 anonymous enum

Enumerator:

*FirstStep*  
*SecondStep*  
*ThirdStep*  
*FourthStep*

## 7.59.3 Constructor & Destructor Documentation

### 7.59.3.1 spot::minato\_isop::local\_vars::local\_vars (bdd *f\_min*, bdd *f\_max*, bdd *vars*) [inline]

## 7.59.4 Member Data Documentation

### 7.59.4.1 bdd spot::minato\_isop::local\_vars::f0\_max

### 7.59.4.2 bdd spot::minato\_isop::local\_vars::f0\_min

### 7.59.4.3 bdd spot::minato\_isop::local\_vars::f1\_max

### 7.59.4.4 bdd spot::minato\_isop::local\_vars::f1\_min

### 7.59.4.5 bdd spot::minato\_isop::local\_vars::f\_max

### 7.59.4.6 bdd spot::minato\_isop::local\_vars::f\_min

### 7.59.4.7 bdd spot::minato\_isop::local\_vars::g0

**7.59.4.8** bdd spot::minato\_isop::local\_vars::g1

**7.59.4.9** enum { ... } spot::minato\_isop::local\_vars::step

**7.59.4.10** bdd spot::minato\_isop::local\_vars::v1

**7.59.4.11** bdd spot::minato\_isop::local\_vars::vars

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

- misc/[minato.hh](#)

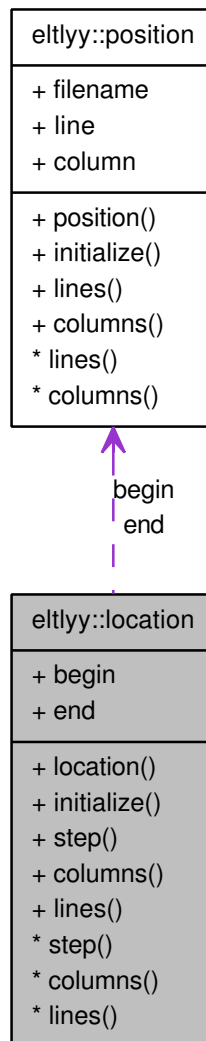
## 7.60 eltlyy::location Class Reference

Abstract a location.

```
#include <eltlparse/location.hh>
```



Collaboration diagram for eltlyy::location:



### Public Member Functions

- `location ()`  
*Construct a location.*
- `void initialize (std::string *fn)`  
*Initialization.*

### Line and Column related manipulators

- `void step ()`  
*Reset initial location to final location.*
- `void columns (unsigned int count=1)`

*Extend the current location to the COUNT next columns.*

- void [lines](#) (unsigned int count=1)  
*Extend the current location to the COUNT next lines.*

### Public Attributes

- [position begin](#)  
*Beginning of the located region.*
- [position end](#)  
*End of the located region.*

### 7.60.1 Detailed Description

Abstract a location.

### 7.60.2 Constructor & Destructor Documentation

#### 7.60.2.1 eltlyy::location::location () [inline]

Construct a location.

### 7.60.3 Member Function Documentation

#### 7.60.3.1 void eltlyy::location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

Referenced by eltlyy::operator+(), and eltlyy::operator+=().

#### 7.60.3.2 void eltlyy::location::initialize (std::string \*fn) [inline]

Initialization.

#### 7.60.3.3 void eltlyy::location::lines (unsigned int count = 1) [inline]

Extend the current location to the COUNT next lines.

#### 7.60.3.4 void eltlyy::location::step () [inline]

Reset initial location to final location.

References begin, end, and eltlyy::position::initialize().

### 7.60.4 Member Data Documentation

#### 7.60.4.1 position eltlyy::location::begin

Beginning of the located region.

Referenced by step().

#### 7.60.4.2 position eltlyy::location::end

End of the located region.

Referenced by eltlyy::operator+(), and step().

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

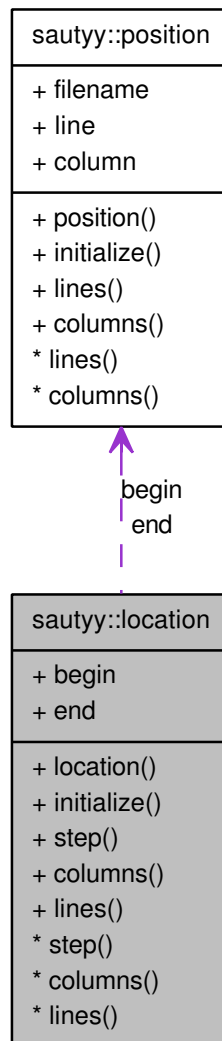
- eltlparse/[location.hh](#)

## 7.61 sautyy::location Class Reference

Abstract a location.

```
#include <sautparse/location.hh>
```

Collaboration diagram for sautyy::location:



### Public Member Functions

- `location ()`  
*Construct a location.*
- `void initialize (std::string *fn)`  
*Initialization.*

### Line and Column related manipulators

- `void step ()`  
*Reset initial location to final location.*
- `void columns (unsigned int count=1)`

*Extend the current location to the COUNT next columns.*

- void [lines](#) (unsigned int count=1)  
*Extend the current location to the COUNT next lines.*

## Public Attributes

- [position begin](#)  
*Beginning of the located region.*
- [position end](#)  
*End of the located region.*

### 7.61.1 Detailed Description

Abstract a location.

### 7.61.2 Constructor & Destructor Documentation

#### 7.61.2.1 sautty::location::location () [inline]

Construct a location.

### 7.61.3 Member Function Documentation

#### 7.61.3.1 void sautty::location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

References begin, and end.

Referenced by sautty::operator+(), and sautty::operator+=().

#### 7.61.3.2 void sautty::location::initialize (std::string \*fn) [inline]

Initialization.

#### 7.61.3.3 void sautty::location::lines (unsigned int count = 1) [inline]

Extend the current location to the COUNT next lines.

References end.

#### 7.61.3.4 void sautyy::location::step () [inline]

Reset initial location to final location.

### 7.61.4 Member Data Documentation

#### 7.61.4.1 position sautyy::location::begin

Beginning of the located region.

Referenced by columns().

#### 7.61.4.2 position sautyy::location::end

End of the located region.

Referenced by columns(), lines(), and sautyy::operator+().

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

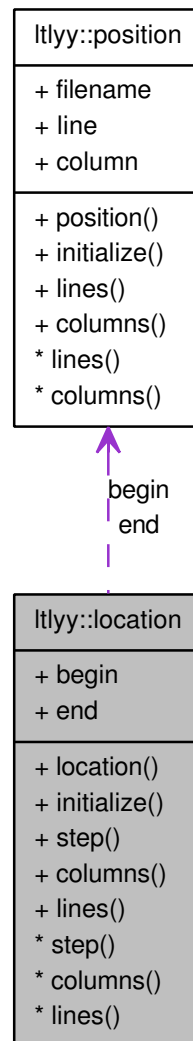
- sautparse/[location.hh](#)

## 7.62 Itlyy::location Class Reference

Abstract a location.

```
#include <ltlparse/location.hh>
```

Collaboration diagram for Itlly::location:



### Public Member Functions

- `location ()`  
*Construct a location.*
- `void initialize (std::string *fn)`  
*Initialization.*

### Line and Column related manipulators

- `void step ()`  
*Reset initial location to final location.*
- `void columns (unsigned int count=1)`

*Extend the current location to the COUNT next columns.*

- void [lines](#) (unsigned int count=1)  
*Extend the current location to the COUNT next lines.*

### Public Attributes

- [position begin](#)  
*Beginning of the located region.*
- [position end](#)  
*End of the located region.*

## 7.62.1 Detailed Description

Abstract a location.

## 7.62.2 Constructor & Destructor Documentation

### 7.62.2.1 Itlyy::location::location () [inline]

Construct a location.

## 7.62.3 Member Function Documentation

### 7.62.3.1 void Itlyy::location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

Referenced by Itlyy::operator+(), and Itlyy::operator+=().

### 7.62.3.2 void Itlyy::location::initialize (std::string \*fn) [inline]

Initialization.

### 7.62.3.3 void Itlyy::location::lines (unsigned int count = 1) [inline]

Extend the current location to the COUNT next lines.



**7.62.3.4 void `ltlyy::location::step()` [inline]**

Reset initial location to final location.

References `begin`, `end`, and `ltlyy::position::initialize()`.

**7.62.4 Member Data Documentation****7.62.4.1 position `ltlyy::location::begin`**

Beginning of the located region.

Referenced by `step()`.

**7.62.4.2 position `ltlyy::location::end`**

End of the located region.

Referenced by `ltlyy::operator+()`, and `step()`.

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

- [ltlparse/location.hh](#)

**7.63 `spot::loopless_modular_mixed_radix_gray_code` Class Reference**

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.

```
#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_`

#### 7.63.1 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.)

### 7.63.2 Constructor & Destructor Documentation

#### 7.63.2.1 `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.

#### 7.63.2.2 `virtual spot::loopless_modular_mixed_radix_gray_code::~~loopless_modular_mixed_radix_gray_code() [virtual]`

### 7.63.3 Member Function Documentation

#### 7.63.3.1 `virtual void spot::loopless_modular_mixed_radix_gray_code::a_first(int j) [pure virtual]`

Reset  $a_j$  to its initial value.

#### 7.63.3.2 `virtual bool spot::loopless_modular_mixed_radix_gray_code::a_last(int j) const [pure virtual]`

Whether  $a_j$  is on its last value.

#### 7.63.3.3 `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.

#### 7.63.3.4 `bool spot::loopless_modular_mixed_radix_gray_code::done () const [inline]`

Whether all tuple have been explored.

References `done_`.

#### 7.63.3.5 `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()`.

#### 7.63.3.6 `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.

References `f_`, and `n_`.

#### 7.63.3.7 `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  $j$ .

### 7.63.4 Member Data Documentation

#### 7.63.4.1 `int* spot::loopless_modular_mixed_radix_gray_code::a_ [protected]`

#### 7.63.4.2 `bool spot::loopless_modular_mixed_radix_gray_code::done_ [protected]`

Referenced by `done()`.

#### 7.63.4.3 `int* spot::loopless_modular_mixed_radix_gray_code::f_ [protected]`

Referenced by `last()`.

7.63.4.4 int\* spot::loopless\_modular\_mixed\_radix\_gray\_code::m\_ [protected]

7.63.4.5 int spot::loopless\_modular\_mixed\_radix\_gray\_code::n\_ [protected]

Referenced by last().

7.63.4.6 int\* spot::loopless\_modular\_mixed\_radix\_gray\_code::non\_one\_radixes\_ [protected]

7.63.4.7 int\* spot::loopless\_modular\_mixed\_radix\_gray\_code::s\_ [protected]

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

- misc/[modgray.hh](#)

## 7.64 spot::ltl::ltl\_file Class Reference

Read LTL formulae from a file, one by one.

```
#include <ltlparse/ltlfile.hh>
```

### Public Member Functions

- [ltl\\_file](#) (const std::string &filename)
- [ltl\\_file](#) (const char \*filename)
- [formula](#) \* [next](#) ()  
*Return the next parsed LTL formula, and 0 at end of file.*

### Private Attributes

- std::ifstream [in](#)

### 7.64.1 Detailed Description

Read LTL formulae from a file, one by one.

### 7.64.2 Constructor & Destructor Documentation

7.64.2.1 spot::ltl::ltl\_file::ltl\_file (const std::string &filename)

### 7.64.2.2 spot::ltl::ltl\_file (const char \*filename)

### 7.64.3 Member Function Documentation

#### 7.64.3.1 formula\* spot::ltl::ltl\_file::next ()

Return the next parsed LTL formula, and 0 at end of file.

### 7.64.4 Member Data Documentation

#### 7.64.4.1 std::ifstream spot::ltl::ltl\_file::in [private]

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

- [ltlparse/ltlfile.hh](#)

## 7.65 spot::minato\_isop Class Reference

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.

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

### Classes

- struct [local\\_vars](#)  
*Internal variables for [minato\\_isop](#).*

### Public Member Functions

- [minato\\_isop](#) (bdd input)  
*Constructor.*
- [minato\\_isop](#) (bdd input, bdd vars)  
*Constructor.*
- bdd [next](#) ()  
*Compute the next sum term of the ISOP form. Return `bddfalse` when all terms have been output.*

### Private Attributes

- std::stack< local\_vars > todo\_
- std::stack< bdd > cube\_
- bdd ret\_

#### 7.65.1 Detailed Description

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

This algorithm implements a derecursed version the Minato-Morreale algorithm presented in the following paper.

```

/// @InProceedings{ minato.92.sasimi,
///   author      = {Shin-ichi Minato},
///   title       = {Fast Generation of Irredundant Sum-of-Products Forms
///                 from Binary Decision Diagrams},
///   booktitle    = {Proceedings of the third Synthesis and Simulation
///                 and Meeting International Interchange workshop
///                 (SASIMI'92)},
///   pages       = {64--73},
///   year        = {1992},
///   address     = {Kobe, Japan},
///   month       = {April}
/// }
///

```

#### 7.65.2 Constructor & Destructor Documentation

##### 7.65.2.1 spot::minato\_isop::minato\_isop (bdd input)

Constructor.

- input The BDD function to translate in ISOP.

##### 7.65.2.2 spot::minato\_isop::minato\_isop (bdd input, bdd vars)

Constructor.

- input The BDD function to translate in ISOP.
- vars The set of BDD variables to factorize in *input*.

#### 7.65.3 Member Function Documentation

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

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

#### 7.65.4 Member Data Documentation

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

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

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

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

- [misc/minato.hh](#)

## 7.66 spot::ltl::multop Class Reference

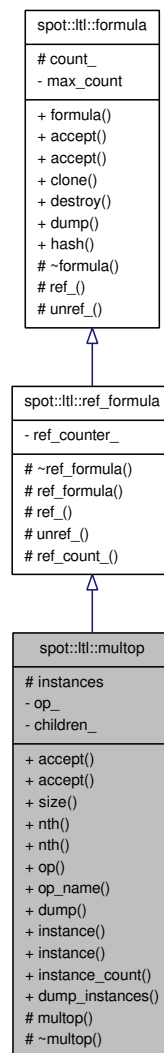
Multi-operand operators.

These operators are considered commutative and associative.

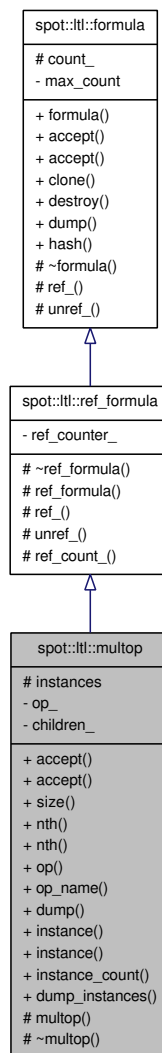
```
#include <ltlast/multop.hh>
```



Inheritance diagram for spot::ltl::multop:



Collaboration diagram for spot::ltl::multop:



## Classes

- struct [paircmp](#)  
*Comparison functor used internally by [ltl::multop](#).*

## Public Types

- enum [type](#) { [Or](#), [And](#) }
- typedef std::vector< [formula](#) \* > [vec](#)  
*List of formulae.*

### Public Member Functions

- virtual void **accept** (**visitor** &v)  
*Entry point for vspot::ltl::visitor instances.*
- virtual void **accept** (**const\_visitor** &v) const  
*Entry point for vspot::ltl::const\_visitor instances.*
- unsigned **size** () const  
*Get the number of children.*
- const **formula** \* **nth** (unsigned n) const  
*Get the nth children.*
- **formula** \* **nth** (unsigned n)  
*Get the nth children.*
- **type op** () const  
*Get the type of this operator.*
- const char \* **op\_name** () const  
*Get the type of this operator, as a string.*
- virtual std::string **dump** () const  
*Return a canonic representation of the atomic proposition.*
- **formula** \* **clone** () const  
*clone this node*
- void **destroy** () const  
*release this node*
- 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 std::ostream & **dump\_instances** (std::ostream &os)  
*Dump all instances. For debugging.*

### Protected Types

- typedef std::pair< type, vec \* > pair
- typedef std::map< pair, multop \*, 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.*

### Protected Attributes

- size\_t count\_  
*The hash key of this formula.*

### Static Protected Attributes

- static map instances

### Private Attributes

- type op\_
- vec \* children\_

#### 7.66.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

#### 7.66.2 Member Typedef Documentation

**7.66.2.1** typedef std::map<pair, multop\*, paircmp> spot::ltl::multop::map [protected]

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

### 7.66.2.3 typedef std::vector<formula\*> spot::ltl::multop::vec

List of formulae.

## 7.66.3 Member Enumeration Documentation

### 7.66.3.1 enum spot::ltl::multop::type

Enumerator:

*Or*

*And*

## 7.66.4 Constructor & Destructor Documentation

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

### 7.66.4.2 virtual spot::ltl::multop::~~multop () [protected, virtual]

## 7.66.5 Member Function Documentation

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

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

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

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

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

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

### 7.66.5.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

**7.66.5.4** `void spot::ltl::formula::destroy () const` **[inherited]**

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.66.5.5** `virtual std::string spot::ltl::multop::dump () const` **[virtual]**

Return a canonic representation of the atomic proposition.

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

**7.66.5.6** `static std::ostream& spot::ltl::multop::dump_instances (std::ostream & os)` **[static]**

Dump all instances. For debugging.

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

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

**7.66.5.8** `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.

**7.66.5.9** `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.

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

Number of instantiated multi-operand operators. For debugging.

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

Get the  $n$ th children.

Starting with  $n = 0$ .

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

Get the  $n$ th children.

Starting with  $n = 0$ .

#### 7.66.5.13 `type spot::ltl::multop::op () const`

Get the type of this operator.

#### 7.66.5.14 `const char* spot::ltl::multop::op_name () const`

Get the type of this operator, as a string.

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

increment reference counter if any

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

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

Number of references to this formula.

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

Get the number of children.

#### 7.66.5.18 `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](#).

### 7.66.6 Member Data Documentation

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

#### 7.66.6.2 `size_t spot::ltl::formula::count_ [protected, inherited]`

The hash key of this formula.

Referenced by `spot::ltl::formula::hash()`.

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

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

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

- [ltlast/multop.hh](#)

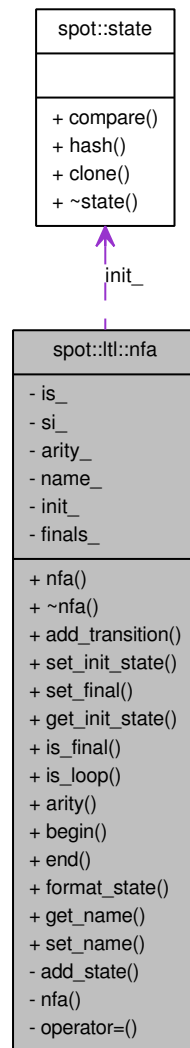
## 7.67 `spot::ltl::nfa` Class Reference

Nondeterministic Finite Automata used by automata operators.

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



Collaboration diagram for spot::ltl::nfa:



## Classes

- struct [transition](#)  
*Explicit transitions.*

## Public Types

- typedef std::list< [transition](#) \* > [state](#)
- typedef boost::shared\_ptr< [formula\\_tree::node](#) > [label](#)
- typedef [succ\\_iterator](#) iterator  
*Iterator over the successors of a state.*
- typedef boost::shared\_ptr< [nfa](#) > [ptr](#)

## Public Member Functions

- [nfa](#) ()
- [~nfa](#) ()
- void [add\\_transition](#) (int src, int dst, const [label](#) lbl)
- void [set\\_init\\_state](#) (int name)
- void [set\\_final](#) (int name)
- const [state](#) \* [get\\_init\\_state](#) ()  
*Get the initial state of the NFA.*
- bool [is\\_final](#) (const [state](#) \*s)  
*Tell whether the given state is final or not.*
- bool [is\\_loop](#) ()  
*Tell whether the NFA is 'loop', i.e. without any final state.*
- unsigned [arity](#) ()  
*Get the 'arity' i.e. max t.cost, for each transition t.*
- [iterator begin](#) (const [state](#) \*s) const  
*Return an iterator on the first succesor (if any) of state.*
- [iterator end](#) (const [state](#) \*s) const  
*Return an iterator just past the last succesor of state.*
- int [format\\_state](#) (const [state](#) \*s) const
- const std::string & [get\\_name](#) () const
- void [set\\_name](#) (const std::string &)

## Private Types

- typedef Sgi::hash\_map< int, [state](#) \*, Sgi::hash< int > > [is\\_map](#)
- typedef Sgi::hash\_map< const [state](#) \*, int, [ptr\\_hash](#)< [state](#) > > [si\\_map](#)

## Private Member Functions

- [state](#) \* [add\\_state](#) (int name)
- [nfa](#) (const [nfa](#) &other)
- [nfa](#) & [operator=](#) (const [nfa](#) &other)

## Private Attributes

- [is\\_map](#) [is\\_](#)
- [si\\_map](#) [si\\_](#)
- size\_t [arity\\_](#)
- std::string [name\\_](#)
- [state](#) \* [init\\_](#)
- std::set< int > [finals\\_](#)

### 7.67.1 Detailed Description

Nondeterministic Finite Automata used by automata operators. States are represented by integers. Labels are represented by formula\_tree's nodes. Currently, only one initial state is possible.

### 7.67.2 Member Typedef Documentation

**7.67.2.1** `typedef Sgi::hash_map<int, state*, Sgi::hash<int> > spot::ltl::nfa::is_map [private]`

**7.67.2.2** `typedef succ_iterator spot::ltl::nfa::iterator`

Iterator over the successors of a state.

**7.67.2.3** `typedef boost::shared_ptr<formula_tree::node> spot::ltl::nfa::label`

**7.67.2.4** `typedef boost::shared_ptr<nfa> spot::ltl::nfa::ptr`

**7.67.2.5** `typedef Sgi::hash_map<const state*, int, ptr_hash<state> > spot::ltl::nfa::si_map [private]`

**7.67.2.6** `typedef std::list<transition*> spot::ltl::nfa::state`

### 7.67.3 Constructor & Destructor Documentation

**7.67.3.1** `spot::ltl::nfa::nfa ()`

**7.67.3.2** `spot::ltl::nfa::~~nfa ()`

**7.67.3.3** `spot::ltl::nfa::nfa (const nfa & other) [private]`

Explicitly disallow use of implicitly generated member functions we don't want.

### 7.67.4 Member Function Documentation

#### 7.67.4.1 state\* spot::ltl::nfa::add\_state (int *name*) [private]

#### 7.67.4.2 void spot::ltl::nfa::add\_transition (int *src*, int *dst*, const label *lbl*)

#### 7.67.4.3 unsigned spot::ltl::nfa::arity ()

Get the ‘arity’ i.e. max t.cost, for each transition t.

#### 7.67.4.4 iterator spot::ltl::nfa::begin (const state \* *s*) const

Return an iterator on the first succesor (if any) of *state*.

The usual way to do this with a `for` loop.

```
for (nfa::iterator i = a.begin(s); i != a.end(s); ++i);
```

#### 7.67.4.5 iterator spot::ltl::nfa::end (const state \* *s*) const

Return an iterator just past the last succesor of *state*.

#### 7.67.4.6 int spot::ltl::nfa::format\_state (const state \* *s*) const

#### 7.67.4.7 const state\* spot::ltl::nfa::get\_init\_state ()

Get the initial state of the NFA.

#### 7.67.4.8 const std::string& spot::ltl::nfa::get\_name () const

#### 7.67.4.9 bool spot::ltl::nfa::is\_final (const state \* *s*)

Tell whether the given state is final or not.

**7.67.4.10 bool spot::ltl::nfa::is\_loop ()**

Tell whether the NFA is ‘loop’, i.e. without any final state.

**7.67.4.11 nfa& spot::ltl::nfa::operator= (const nfa & *other*) [private]****7.67.4.12 void spot::ltl::nfa::set\_final (int *name*)****7.67.4.13 void spot::ltl::nfa::set\_init\_state (int *name*)****7.67.4.14 void spot::ltl::nfa::set\_name (const std::string &)****7.67.5 Member Data Documentation****7.67.5.1 size\_t spot::ltl::nfa::arity\_ [private]****7.67.5.2 std::set<int> spot::ltl::nfa::finals\_ [private]****7.67.5.3 state\* spot::ltl::nfa::init\_ [private]****7.67.5.4 is\_map spot::ltl::nfa::is\_ [private]****7.67.5.5 std::string spot::ltl::nfa::name\_ [private]**

#### 7.67.5.6 si\_map spot::ltl::nfa::si\_ [private]

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

- [ltlast/nfa.hh](#)

## 7.68 spot::nips\_exception Class Reference

An exception used to forward NIPS errors.

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

### Public Member Functions

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

### Private Attributes

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

#### 7.68.1 Detailed Description

An exception used to forward NIPS errors.

#### 7.68.2 Constructor & Destructor Documentation

**7.68.2.1** spot::nips\_exception::nips\_exception (const std::string & *where*, int *err*) [inline]

**7.68.2.2** spot::nips\_exception::nips\_exception (const std::string & *where*) [inline]

#### 7.68.3 Member Function Documentation

**7.68.3.1** int spot::nips\_exception::get\_err () const [inline]

References [err\\_](#).

### 7.68.3.2 int spot::nips\_exception::get\_err\_defined () const [inline]

References `err_defined_`.

### 7.68.3.3 std::string spot::nips\_exception::get\_where () const [inline]

References `where_`.

## 7.68.4 Member Data Documentation

### 7.68.4.1 int spot::nips\_exception::err\_ [private]

Referenced by `get_err()`.

### 7.68.4.2 bool spot::nips\_exception::err\_defined\_ [private]

Referenced by `get_err_defined()`.

### 7.68.4.3 std::string spot::nips\_exception::where\_ [private]

Referenced by `get_where()`.

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

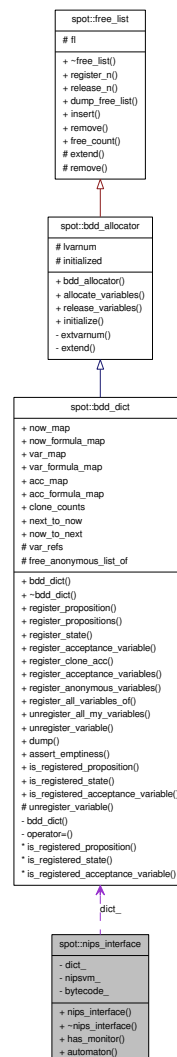
- [nips/common.hh](#)

## 7.69 spot::nips\_interface Class Reference

An interface to provide a PROMELA front-end.

```
#include <nips/nips.hh>
```

Collaboration diagram for spot::nips\_interface:



## Public Member Functions

- [nips\\_interface](#) ([bdd\\_dict](#) \*dict, const std::string &filename)
- [~nips\\_interface](#) ()
- bool [has\\_monitor](#) () const
- [tgba](#) \* [automaton](#) ()

## Private Attributes

- [bdd\\_dict](#) \* [dict\\_](#)
- [nipsvm\\_t](#) \* [nipsvm\\_](#)
- [nipsvm\\_bytecode\\_t](#) \* [bytecode\\_](#)



### 7.69.1 Detailed Description

An interface to provide a PROMELA front-end. This interface let to use a Promela model as a BÄ¼chi automata. It uses the NIPS library, which provided a virtual machine for the state-space exploration of a Promela model, therefore, models must be compiled with the NIPS compiler (<http://wwwhome.cs.utwente.nl/~michaelw/nips/>).

With this interface, properties to check aren't defined with the Spot LTL representation, but in defining correctness claims (a monitor) in the Promela model (see chapter 4, The Spin Model Checker: Primer and reference manual, Gerard J.Holzmann).

### 7.69.2 Constructor & Destructor Documentation

**7.69.2.1** `spot::nips_interface::nips_interface (bdd_dict * dict, const std::string & filename)`

**7.69.2.2** `spot::nips_interface::~~nips_interface ()`

### 7.69.3 Member Function Documentation

**7.69.3.1** `tgba* spot::nips_interface::automaton ()`

**7.69.3.2** `bool spot::nips_interface::has_monitor () const`

### 7.69.4 Member Data Documentation

**7.69.4.1** `nipsvm_bytecode_t* spot::nips_interface::bytecode_ [private]`

**7.69.4.2** `bdd_dict* spot::nips_interface::dict_ [private]`

**7.69.4.3** `nipsvm_t* spot::nips_interface::nipsvm_ [private]`

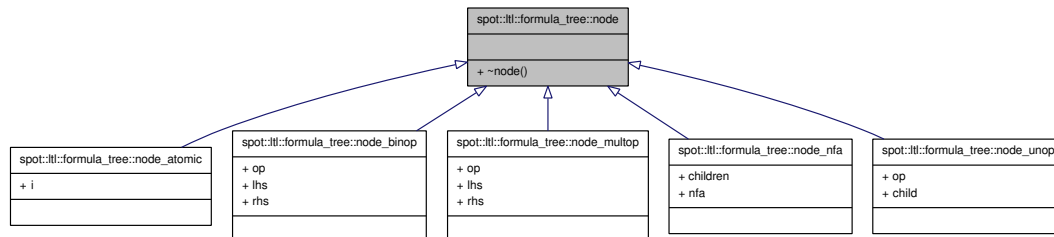
The documentation for this class was generated from the following file:

- [nips/nips.hh](#)

## 7.70 spot::ltl::formula\_tree::node Struct Reference

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

Inheritance diagram for spot::ltl::formula\_tree::node:



### Public Member Functions

- virtual [~node\(\)](#)

#### 7.70.1 Constructor & Destructor Documentation

##### 7.70.1.1 virtual spot::ltl::formula\_tree::node::~~node() [inline, virtual]

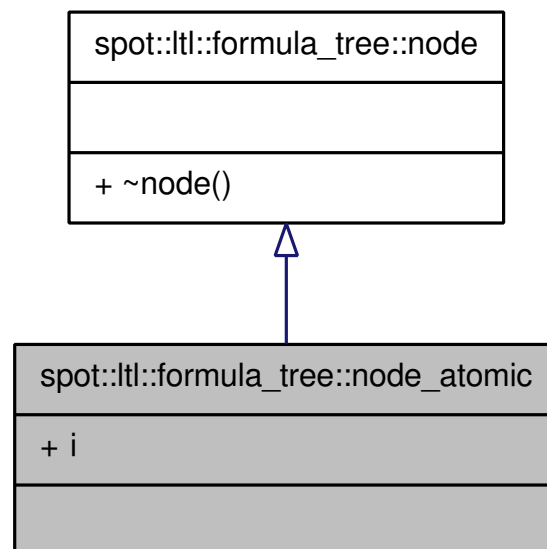
The documentation for this struct was generated from the following file:

- ltlast/[formula\\_tree.hh](#)

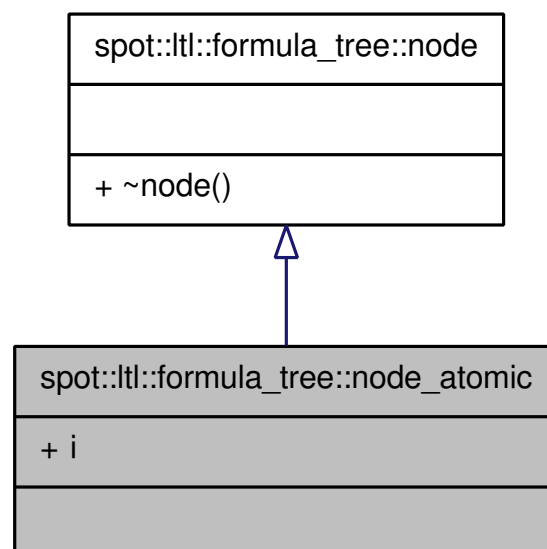
## 7.71 spot::ltl::formula\_tree::node\_atomic Struct Reference

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

Inheritance diagram for spot::ltl::formula\_tree::node\_atomic:



Collaboration diagram for spot::ltl::formula\_tree::node\_atomic:



#### Public Attributes

- `int i`

### 7.71.1 Member Data Documentation

#### 7.71.1.1 int spot::ltl::formula\_tree::node\_atomic::i

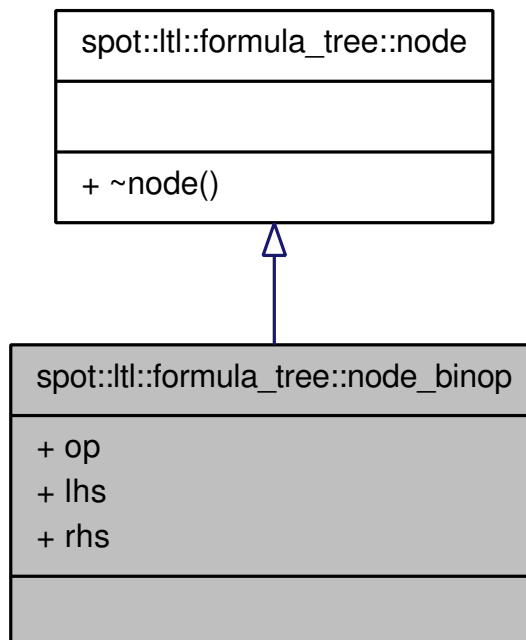
The documentation for this struct was generated from the following file:

- [ltlast/formula\\_tree.hh](#)

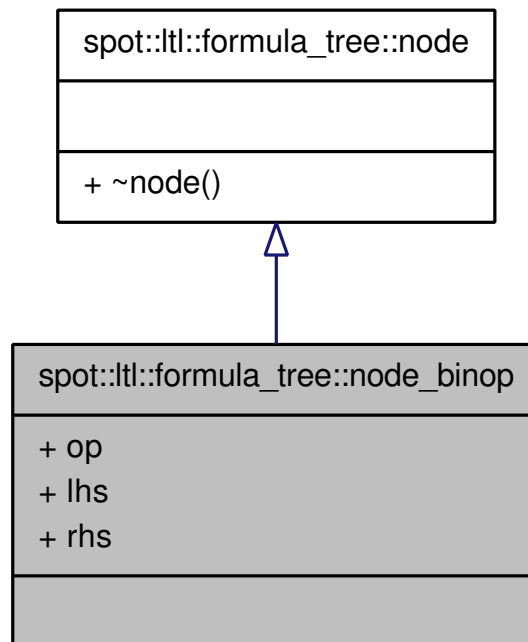
## 7.72 spot::ltl::formula\_tree::node\_binop Struct Reference

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

Inheritance diagram for spot::ltl::formula\_tree::node\_binop:



Collaboration diagram for spot::ltl::formula\_tree::node\_binop:



### Public Attributes

- [binop::type op](#)
- [node\\_ptr lhs](#)
- [node\\_ptr rhs](#)

### 7.72.1 Member Data Documentation

#### 7.72.1.1 node\_ptr spot::ltl::formula\_tree::node\_binop::lhs

#### 7.72.1.2 binop::type spot::ltl::formula\_tree::node\_binop::op

#### 7.72.1.3 node\_ptr spot::ltl::formula\_tree::node\_binop::rhs

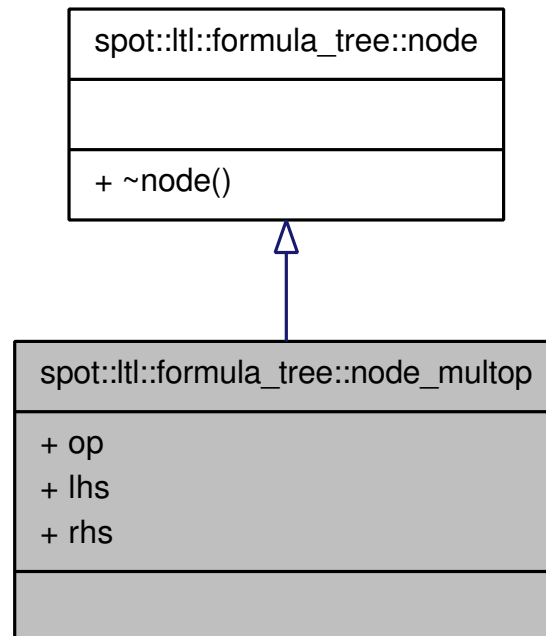
The documentation for this struct was generated from the following file:

- [ltlast/formula\\_tree.hh](#)

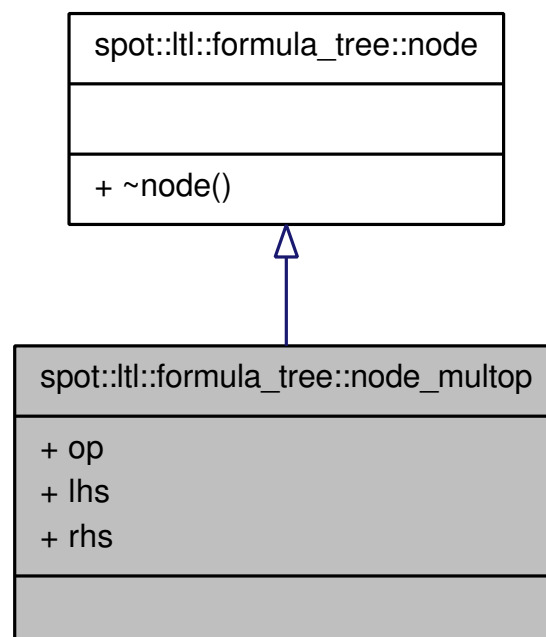
### 7.73 spot::ltl::formula\_tree::node\_multop Struct Reference

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

Inheritance diagram for spot::ltl::formula\_tree::node\_multop:



Collaboration diagram for spot::ltl::formula\_tree::node\_multop:



### Public Attributes

- [multop::type op](#)
- [node\\_ptr lhs](#)
- [node\\_ptr rhs](#)

#### 7.73.1 Member Data Documentation

##### 7.73.1.1 node\_ptr spot::ltl::formula\_tree::node\_multop::lhs

##### 7.73.1.2 multop::type spot::ltl::formula\_tree::node\_multop::op

##### 7.73.1.3 node\_ptr spot::ltl::formula\_tree::node\_multop::rhs

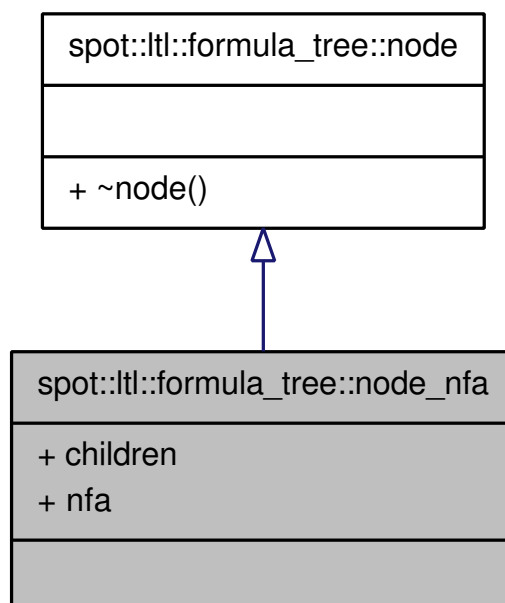
The documentation for this struct was generated from the following file:

- [ltlast/formula\\_tree.hh](#)

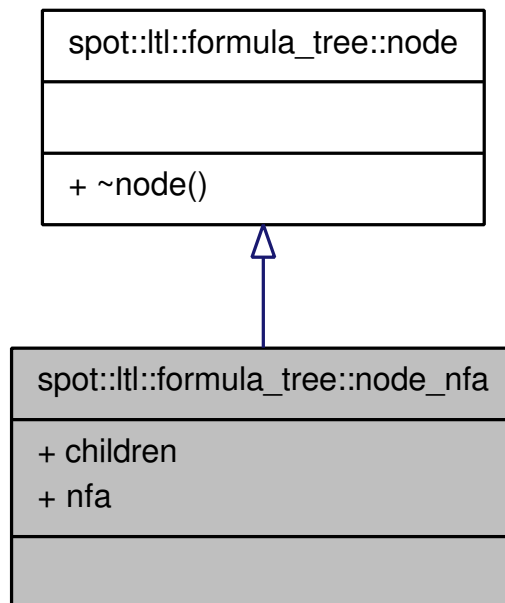
## 7.74 spot::ltl::formula\_tree::node\_nfa Struct Reference

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

Inheritance diagram for spot::ltl::formula\_tree::node\_nfa:



Collaboration diagram for spot::ltl::formula\_tree::node\_nfa:



#### Public Attributes

- `std::vector< node\_ptr > children`
- `spot::ltl::nfa::ptr nfa`

#### 7.74.1 Member Data Documentation

**7.74.1.1** `std::vector<node_ptr> spot::ltl::formula_tree::node_nfa::children`

**7.74.1.2** `spot::ltl::nfa::ptr spot::ltl::formula_tree::node_nfa::nfa`

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

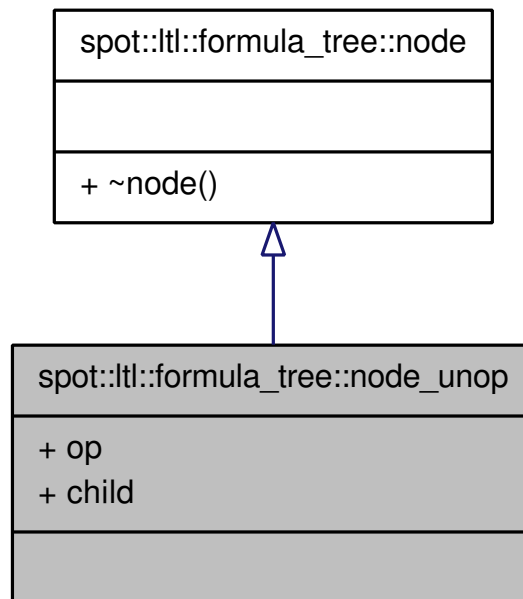
- `ltlast/formula\_tree.hh`

## 7.75 spot::ltl::formula\_tree::node\_unop Struct Reference

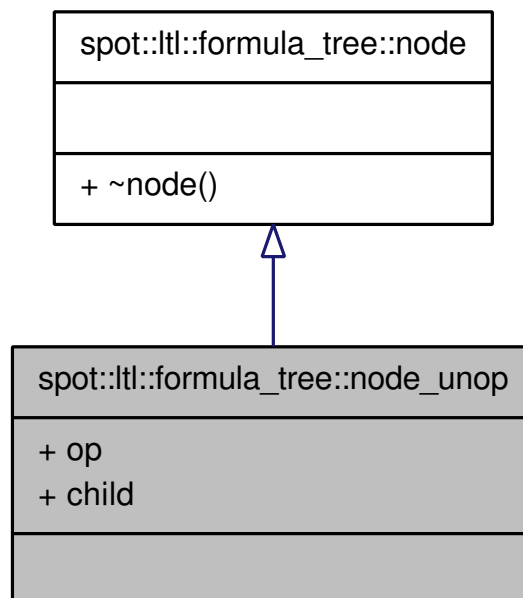
```
#include <ltlast/formula_tree.hh>
```



Inheritance diagram for spot::ltl::formula\_tree::node\_unop:



Collaboration diagram for spot::ltl::formula\_tree::node\_unop:



#### Public Attributes

- `unop::type op`
- `node_ptr child`

### 7.75.1 Member Data Documentation

#### 7.75.1.1 node\_ptr spot::ltl::formula\_tree::node\_unop::child

#### 7.75.1.2 unop::type spot::ltl::formula\_tree::node\_unop::op

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

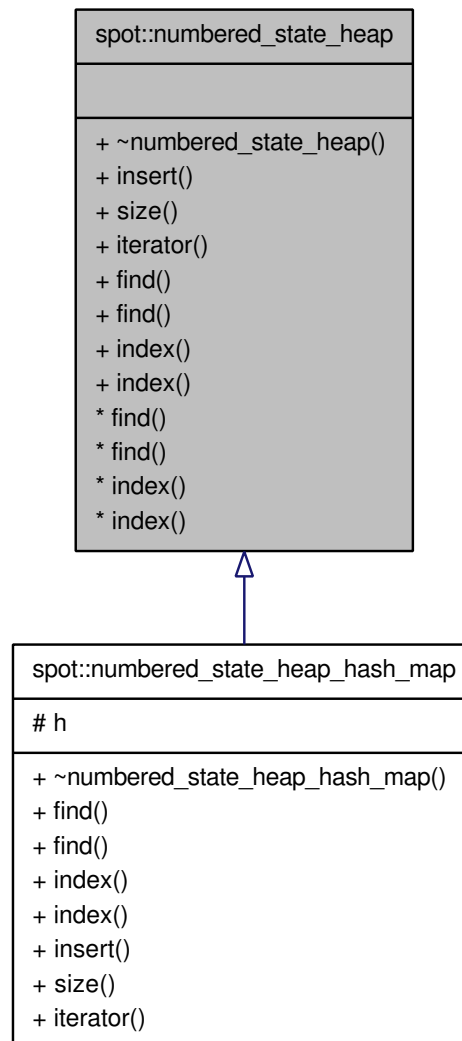
- [ltlast/formula\\_tree.hh](#)

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

Keep track of a large quantity of indexed states.

```
#include <tgbaaalgo/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 [size](#) () const =0  
*The number of stored states.*
- virtual [numbered\\_state\\_heap\\_const\\_iterator](#) \* [iterator](#) () const =0

*Return an iterator on the states/indexes pairs.*

- virtual `state_index find` (const `state *s`) const =0  
*Is state in the heap?*
- virtual `state_index_p find` (const `state *s`)=0
- virtual `state_index index` (const `state *s`) const =0  
*Return the index of an existing state.*
- virtual `state_index_p index` (const `state *s`)=0

### 7.76.1 Detailed Description

Keep track of a large quantity of indexed states.

### 7.76.2 Member Typedef Documentation

**7.76.2.1** `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index`

**7.76.2.2** `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p`

### 7.76.3 Constructor & Destructor Documentation

**7.76.3.1** `virtual spot::numbered_state_heap::~~numbered_state_heap () [inline, virtual]`

### 7.76.4 Member Function Documentation

**7.76.4.1** `virtual state_index_p spot::numbered_state_heap::find (const state * s) [pure virtual]`

Implemented in `spot::numbered_state_heap_hash_map`.

**7.76.4.2** `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](#).

**7.76.4.3** `virtual state_index_p spot::numbered_state_heap::index (const state * s) [pure virtual]`

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

**7.76.4.4** `virtual state_index spot::numbered_state_heap::index (const state * s) const [pure virtual]`

Return the index of an existing state.

This is mostly similar to [find\(\)](#), except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

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

**7.76.4.5** `virtual void spot::numbered_state_heap::insert (const state * s, int index) [pure virtual]`

Add a new state *s* with index *index*.

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

**7.76.4.6** `virtual numbered_state_heap_const_iterator* spot::numbered_state_heap::iterator () const [pure virtual]`

Return an iterator on the states/indexes pairs.

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

**7.76.4.7** `virtual int spot::numbered_state_heap::size () const [pure virtual]`

The number of stored states.

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

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

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

## 7.77 spot::numbered\_state\_heap\_const\_iterator Class Reference

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

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

### Public Member Functions

- virtual [~numbered\\_state\\_heap\\_const\\_iterator](#) ()
- virtual void [first](#) ()=0  
*Iteration.*
- virtual void [next](#) ()=0
- virtual bool [done](#) () const =0
- virtual const [state](#) \* [get\\_state](#) () const =0  
*Inspection.*
- virtual int [get\\_index](#) () const =0

### 7.77.1 Detailed Description

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

### 7.77.2 Constructor & Destructor Documentation

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

### 7.77.3 Member Function Documentation

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

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

Iteration.

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

**7.77.3.4** `virtual const state* spot::numbered_state_heap_const_iterator::get_state () const`  
`[pure virtual]`

Inspection.

**7.77.3.5** `virtual void spot::numbered_state_heap_const_iterator::next ()` `[pure virtual]`

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

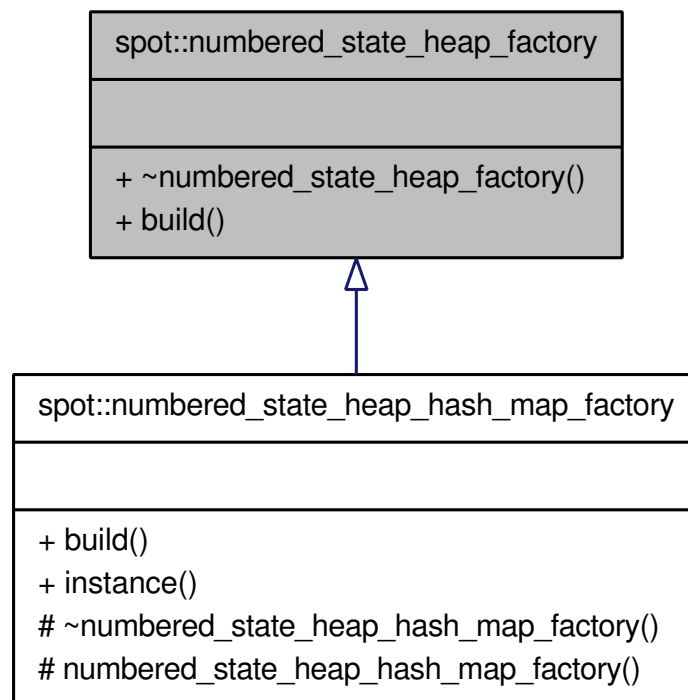
- [tgbalgorithms/gtec/nsheap.hh](#)

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

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

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

Inheritance diagram for spot::numbered\_state\_heap\_factory:



### Public Member Functions

- virtual [~numbered\\_state\\_heap\\_factory](#) ()
- virtual [numbered\\_state\\_heap](#) \* [build](#) () const =0

### 7.78.1 Detailed Description

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

### 7.78.2 Constructor & Destructor Documentation

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

### 7.78.3 Member Function Documentation

**7.78.3.1** `virtual numbered_state_heap* spot::numbered_state_heap_factory::build () const`  
[pure virtual]

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

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

- `tgbaalgos/gtec/nsheap.hh`

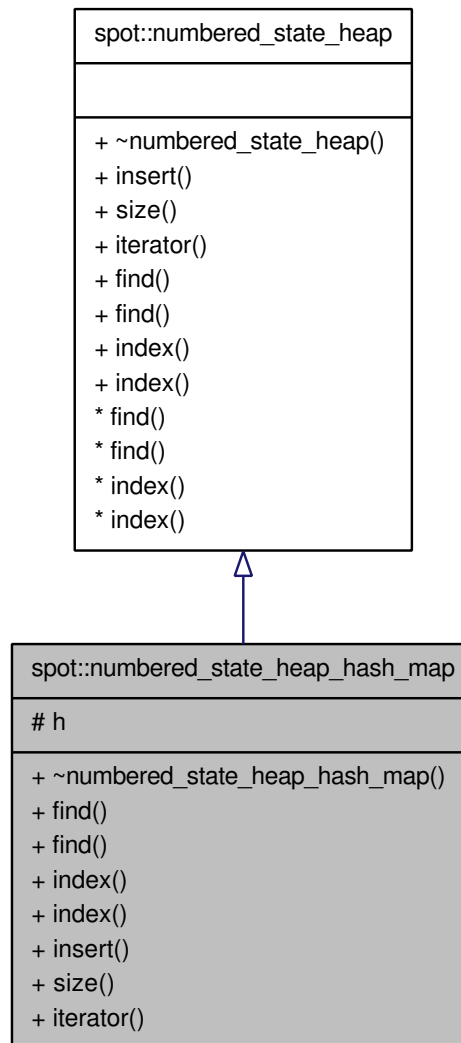
## 7.79 `spot::numbered_state_heap_hash_map` Class Reference

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

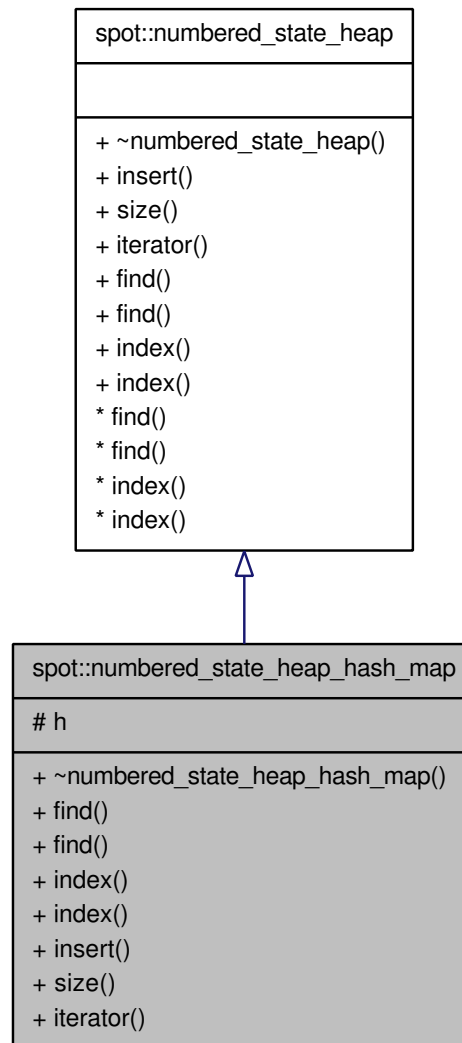
```
#include <tgbaalgos/gtec/nsheap.hh>
```



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 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.*

### 7.79.1 Detailed Description

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

### 7.79.2 Member Typedef Documentation

**7.79.2.1** `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal>  
spot::numbered_state_heap_hash_map::hash_type`

**7.79.2.2** `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index  
[inherited]`

**7.79.2.3** `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p  
[inherited]`

### 7.79.3 Constructor & Destructor Documentation

**7.79.3.1** `virtual spot::numbered_state_heap_hash_map::~~numbered_state_heap_hash_map ()  
[virtual]`

#### 7.79.4 Member Function Documentation

##### 7.79.4.1 `virtual state_index_p spot::numbered_state_heap_hash_map::find (const state * s)` [`virtual`]

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

##### 7.79.4.2 `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](#).

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

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

##### 7.79.4.4 `virtual state_index spot::numbered_state_heap_hash_map::index (const state * s) const` [`virtual`]

Return the index of an existing state.

This is mostly similar to [find\(\)](#), except it will be called for state which we know are already in the heap, or for state which may not be in the heap but for which it is always OK to do equality checks.

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

##### 7.79.4.5 `virtual void spot::numbered_state_heap_hash_map::insert (const state * s, int index)` [`virtual`]

Add a new state *s* with index *index*.

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

#### 7.79.4.6 `virtual numbered_state_heap_const_iterator* spot::numbered_state_heap_hash_map::iterator () const` `[virtual]`

Return an iterator on the states/indexes pairs.

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

#### 7.79.4.7 `virtual int spot::numbered_state_heap_hash_map::size () const` `[virtual]`

The number of stored states.

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

### 7.79.5 Member Data Documentation

#### 7.79.5.1 `hash_type spot::numbered_state_heap_hash_map::h` `[protected]`

Map of visited states.

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

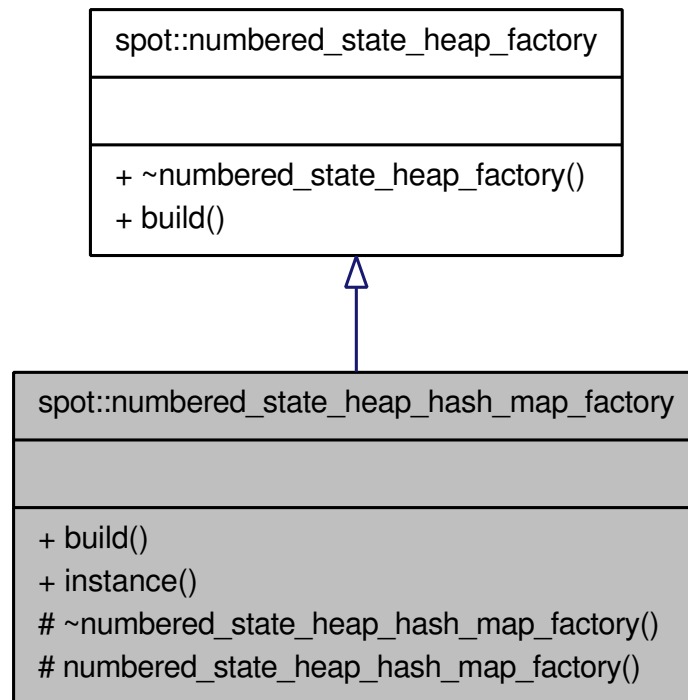
- [tgbaalgos/gtec/nsheap.hh](#)

## 7.80 `spot::numbered_state_heap_hash_map_factory` Class Reference

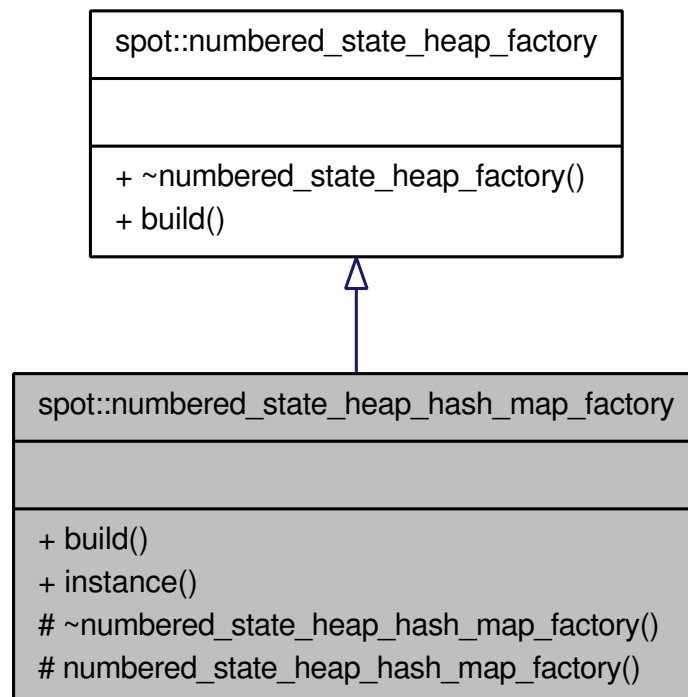
Factory for [numbered\\_state\\_heap\\_hash\\_map](#).

```
#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 ()`

#### 7.80.1 Detailed Description

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

#### 7.80.2 Constructor & Destructor Documentation

**7.80.2.1** `virtual spot::numbered_state_heap_hash_map_factory::~~numbered_state_heap_hash_map_factory ()` [`inline`, `protected`, `virtual`]

**7.80.2.2** `spot::numbered_state_heap_hash_map_factory::numbered_state_heap_hash_map_factory ()` [`protected`]

#### 7.80.3 Member Function Documentation

**7.80.3.1** `virtual numbered_state_heap_hash_map* spot::numbered_state_heap_hash_map_factory::build ()` const [`virtual`]

Implements `spot::numbered_state_heap_factory`.

**7.80.3.2** `static const numbered_state_heap_hash_map_factory* spot::numbered_state_heap_hash_map_factory::instance ()` [`static`]

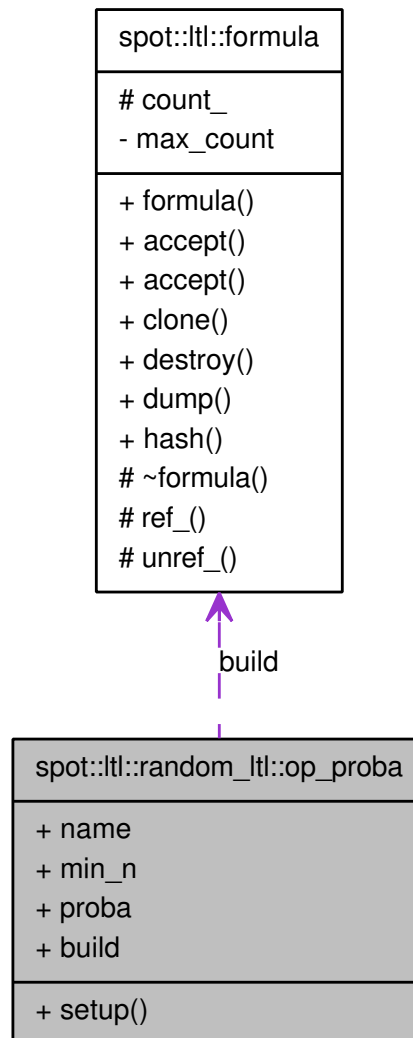
Get the unique instance of this class.

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

- `tgbaalgos/gtec/nsheap.hh`

## 7.81 spot::ltl::random\_ltl::op\_proba Struct Reference

Collaboration diagram for spot::ltl::random\_ltl::op\_proba:



### 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](#)

### 7.81.1 Member Typedef Documentation

7.81.1.1 `typedef formula*(* spot::ltl::random_ltl::op_proba::builder)(const random_ltl *rl, int n)`

### 7.81.2 Member Function Documentation

7.81.2.1 `void spot::ltl::random_ltl::op_proba::setup (const char * name, int min_n, builder build)`

### 7.81.3 Member Data Documentation

7.81.3.1 `builder spot::ltl::random_ltl::op_proba::build`

7.81.3.2 `int spot::ltl::random_ltl::op_proba::min_n`

7.81.3.3 `const char* spot::ltl::random_ltl::op_proba::name`

7.81.3.4 `double spot::ltl::random_ltl::op_proba::proba`

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

- [ltlvisit/randomltl.hh](#)

## 7.82 spot::option\_map Class Reference

Manage a map of options.

Each option is defined by a string and is associated to an integer value.

```
#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.*

## 7.82.1 Detailed Description

Manage a map of options.

Each option is defined by a string and is associated to an integer value.

## 7.82.2 Member Function Documentation

### 7.82.2.1 `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[]()`

### 7.82.2.2 int& spot::option\_map::operator[ ] (const char \* *option*)

Get a reference to the current value of *option*.

### 7.82.2.3 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\(\)](#)

### 7.82.2.4 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.

### 7.82.2.5 void spot::option\_map::set (const option\_map & *o*)

Acquire all the settings of *o*.

### 7.82.2.6 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.

### 7.82.3 Friends And Related Function Documentation

#### 7.82.3.1 std::ostream& operator<< (std::ostream & os, const option\_map & m) [friend]

Print the [option\\_map](#) *m*.

### 7.82.4 Member Data Documentation

#### 7.82.4.1 std::map<std::string, int> spot::option\_map::options\_ [private]

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

- [misc/optionmap.hh](#)

## 7.83 spot::ltl::multop::pairecmp 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

#### 7.83.1 Detailed Description

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

#### 7.83.2 Member Function Documentation

##### 7.83.2.1 bool spot::ltl::multop::pairecmp::operator() (const pair & p1, const pair & p2) const [inline]

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

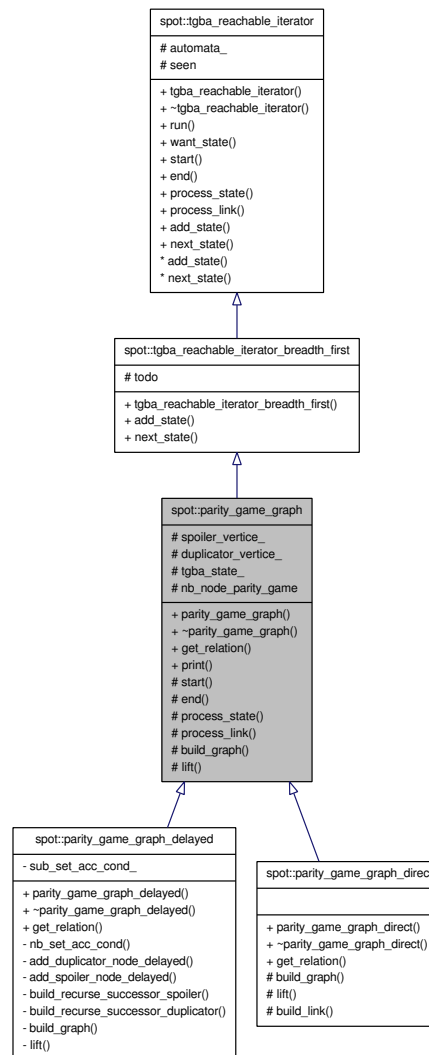
- [ltlast/multop.hh](#)

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

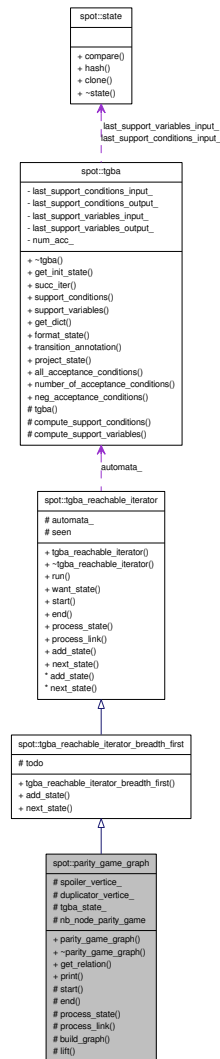
Parity game graph which compute a simulation relation.

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

Inheritance diagram for spot::parity\_game\_graph:



Collaboration diagram for spot::parity\_game\_graph:



## Public Member Functions

- [parity\\_game\\_graph](#) (const [tgba](#) \*a)
- virtual [~parity\\_game\\_graph](#) ()
- virtual [simulation\\_relation](#) \* [get\\_relation](#) ()=0
- void [print](#) (std::ostream &os)
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()

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

- void [run](#) ()  
Iterate over all reachable states of a [spot::tgba](#).
- virtual bool [want\\_state](#) (const [state](#) \*s) const

- 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.*

#### 7.84.1 Detailed Description

Parity game graph which compute a simulation relation.

### 7.84.2 Member Typedef Documentation

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

### 7.84.3 Constructor & Destructor Documentation

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

**7.84.3.2** `virtual spot::parity_game_graph::~~parity_game_graph () [virtual]`

### 7.84.4 Member Function Documentation

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

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

**7.84.4.2** `virtual void spot::parity_game_graph::build_graph () [protected, pure  
virtual]`

Compute each node of the graph.

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

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

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

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

**7.84.4.4** `virtual simulation_relation* spot::parity_game_graph::get_relation () [pure  
virtual]`

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



**7.84.4.5 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](#).

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

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

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

**7.84.4.7 void spot::parity\_game\_graph::print (std::ostream & os)****7.84.4.8 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.

**7.84.4.9 void spot::parity\_game\_graph::process\_link (int in, int out, const tgba\_succ\_iterator \* si) [protected]****7.84.4.10 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](#).

#### 7.84.4.11 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 iterates over states.

#### 7.84.4.12 void spot::parity\_game\_graph::start () [protected, virtual]

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

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

#### 7.84.4.13 virtual bool spot::tgba\_reachable\_iterator::want\_state (const state \* s) const [virtual, inherited]

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

### 7.84.5 Member Data Documentation

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

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

#### 7.84.5.2 dn\_v spot::parity\_game\_graph::duplicator\_vertice\_ [protected]

#### 7.84.5.3 int spot::parity\_game\_graph::nb\_node\_parity\_game [protected]

#### 7.84.5.4 seen\_map spot::tgba\_reachable\_iterator::seen [protected, inherited]

States already seen.

7.84.5.5 sn\_v spot::parity\_game\_graph::spoiler\_vertice\_ [protected]

7.84.5.6 s\_v spot::parity\_game\_graph::tgba\_state\_ [protected]

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

A queue of states yet to explore.

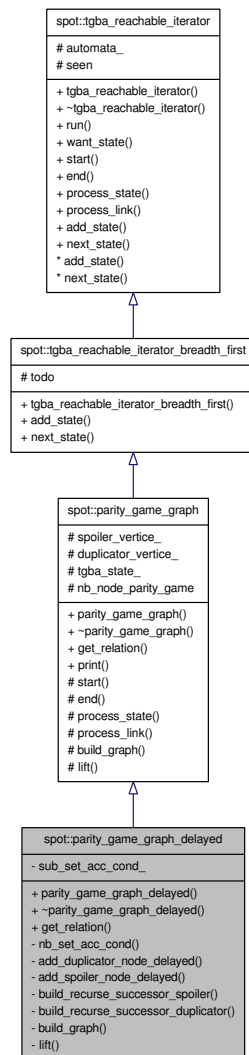
The documentation for this class was generated from the following file:

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

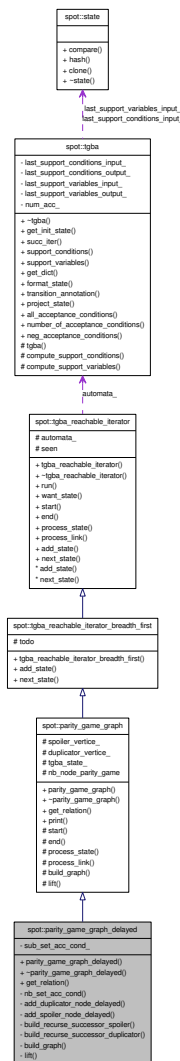
## 7.85 spot::parity\_game\_graph\_delayed Class Reference

```
#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 next state to process.*

- void [run](#) ()

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

- virtual bool [want\\_state](#) (const [state](#) \*s) const

### 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\\_](#)

### 7.85.1 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}
/// }
```

### 7.85.2 Member Typedef Documentation

#### 7.85.2.1 `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.

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

### 7.85.3 Constructor & Destructor Documentation

#### 7.85.3.1 `spot::parity_game_graph_delayed::parity_game_graph_delayed (const tgba * a)`

#### 7.85.3.2 `spot::parity_game_graph_delayed::~~parity_game_graph_delayed ()`

### 7.85.4 Member Function Documentation

**7.85.4.1** duplicator\_node\_delayed\* spot::parity\_game\_graph\_delayed::add\_duplicator\_node\_delayed (const spot::state \* *sn*, const spot::state \* *dn*, bdd *acc*, bdd *label*, int *nb*) [private]

**7.85.4.2** spoiler\_node\_delayed\* spot::parity\_game\_graph\_delayed::add\_spoiler\_node\_delayed (const spot::state \* *sn*, const spot::state \* *dn*, bdd *acc*, int *nb*) [private]

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

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

**7.85.4.4** virtual void spot::parity\_game\_graph\_delayed::build\_graph () [private, virtual]

Compute the couple as for direct simulation,.

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

**7.85.4.5** void spot::parity\_game\_graph\_delayed::build\_recurse\_successor\_duplicator (duplicator\_node \* *dn*, spoiler\_node \* *sn*, std::ostream & *os*) [private]

**7.85.4.6** void spot::parity\_game\_graph\_delayed::build\_recurse\_successor\_spoiler (spoiler\_node \* *sn*, std::ostream & *os*) [private]

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

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

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

**7.85.4.8** virtual delayed\_simulation\_relation\* spot::parity\_game\_graph\_delayed::get\_relation () [virtual]



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

#### 7.85.4.9 virtual void spot::parity\_game\_graph\_delayed::lift () [private, virtual]

The Jurdzinski's lifting algorithm.

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

#### 7.85.4.10 int spot::parity\_game\_graph\_delayed::nb\_set\_acc\_cond () [private]

Return the number of acceptance condition.

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

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

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

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

#### 7.85.4.13 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.

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

**7.85.4.15** 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](#).

**7.85.4.16** 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 iterates over states.

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

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

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

**7.85.4.18** virtual bool spot::tgba\_reachable\_iterator::want\_state (const state \* *s*) const [**virtual**, **inherited**]

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

### 7.85.5 Member Data Documentation

**7.85.5.1** const tgba\* spot::tgba\_reachable\_iterator::automata\_ [**protected**, **inherited**]

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

**7.85.5.2** dn\_v spot::parity\_game\_graph::duplicator\_vertice\_ [**protected**, **inherited**]

**7.85.5.3** int spot::parity\_game\_graph::nb\_node\_parity\_game [**protected**, **inherited**]

#### 7.85.5.4 seen\_map spot::tgba\_reachable\_iterator::seen [protected, inherited]

States already seen.

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

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

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

#### 7.85.5.8 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:

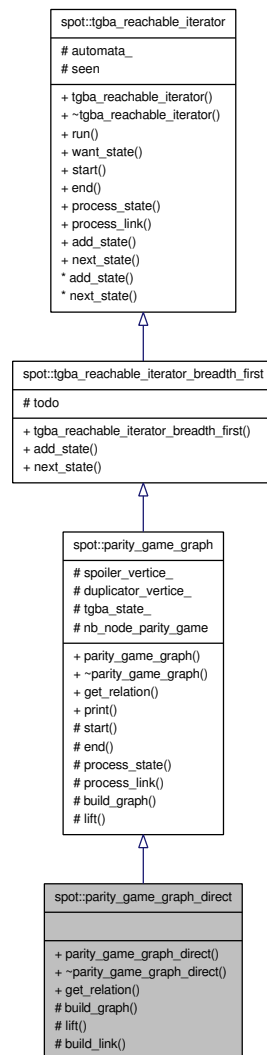
- tgbaalgorithms/[reductgba\\_sim.hh](#)

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

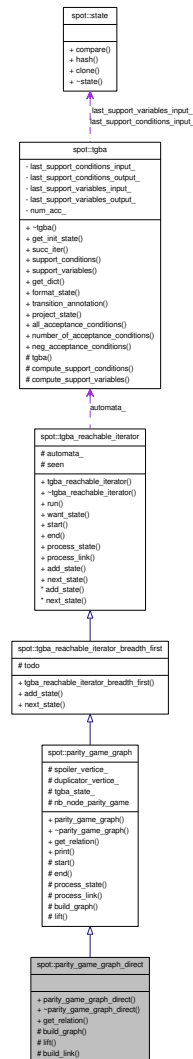
Parity game graph which compute the direct simulation relation.

```
#include <tgbaalgorithms/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 next state to process.*

- void [run](#) ()

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

- virtual bool `want_state` (const `state` \*s) const

### 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.*

#### 7.86.1 Detailed Description

Parity game graph which compute the direct simulation relation.

## 7.86.2 Member Typedef Documentation

**7.86.2.1** typedef Sgi::hash\_map<const state\*, int, state\_ptr\_hash, state\_ptr\_equal>  
spot::tgba\_reachable\_iterator::seen\_map [protected, inherited]

## 7.86.3 Constructor & Destructor Documentation

**7.86.3.1** spot::parity\_game\_graph\_direct::parity\_game\_graph\_direct (const tgba \* a)

**7.86.3.2** spot::parity\_game\_graph\_direct::~~parity\_game\_graph\_direct ()

## 7.86.4 Member Function Documentation

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

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

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

Compute each node of the graph.

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

**7.86.4.3** void spot::parity\_game\_graph\_direct::build\_link () [protected]

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

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

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

**7.86.4.5** virtual direct\_simulation\_relation\* spot::parity\_game\_graph\_direct::get\_relation ()  
[virtual]

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

#### 7.86.4.6 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](#).

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

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

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

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

#### 7.86.4.9 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.

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

#### 7.86.4.11 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](#).

**7.86.4.12 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 iterates over states.

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

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

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

**7.86.4.14 virtual bool spot::tgba\_reachable\_iterator::want\_state (const state \* s) const [virtual, inherited]**

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

**7.86.5 Member Data Documentation****7.86.5.1 const tgba\* spot::tgba\_reachable\_iterator::automata\_ [protected, inherited]**

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

**7.86.5.2 dn\_v spot::parity\_game\_graph::duplicator\_vertice\_ [protected, inherited]****7.86.5.3 int spot::parity\_game\_graph::nb\_node\_parity\_game [protected, inherited]****7.86.5.4 seen\_map spot::tgba\_reachable\_iterator::seen [protected, inherited]**

States already seen.

7.86.5.5 `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

7.86.5.6 `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

7.86.5.7 `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](#)

## 7.87 sauty::position Class Reference

Abstract a position.

```
#include <sautyparse/position.hh>
```

### Public Member Functions

- [position](#) ()  
*Construct a position.*
- void [initialize](#) (std::string \*fn)  
*Initialization.*

### Line and Column related manipulators

- void [lines](#) (int count=1)  
*(line related) Advance to the COUNT next lines.*
- void [columns](#) (int count=1)  
*(column related) Advance to the COUNT next columns.*

### Public Attributes

- std::string \* [filename](#)  
*File name to which this position refers.*
- unsigned int [line](#)  
*Current line number.*

- unsigned int `column`  
*Current column number.*

### 7.87.1 Detailed Description

Abstract a position.

### 7.87.2 Constructor & Destructor Documentation

#### 7.87.2.1 sautyy::position::position () [inline]

Construct a position.

### 7.87.3 Member Function Documentation

#### 7.87.3.1 void sautyy::position::columns (int *count* = 1) [inline]

(column related) Advance to the COUNT next columns.

References `column`.

Referenced by `sautyy::operator+=()`.

#### 7.87.3.2 void sautyy::position::initialize (std::string \**fn*) [inline]

Initialization.

References `column`, `filename`, and `line`.

#### 7.87.3.3 void sautyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References `column`, and `line`.

### 7.87.4 Member Data Documentation

#### 7.87.4.1 unsigned int sautyy::position::column

Current column number.

Referenced by `columns()`, `initialize()`, `lines()`, and `sautyy::operator<<()`.

#### 7.87.4.2 `std::string* sautyy::position::filename`

File name to which this position refers.

Referenced by `initialize()`, and `sautyy::operator<<()`.

#### 7.87.4.3 `unsigned int sautyy::position::line`

Current line number.

Referenced by `initialize()`, `lines()`, and `sautyy::operator<<()`.

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

- `sautparse/position.hh`

## 7.88 `eltly::position` Class Reference

Abstract a position.

```
#include <eltlparse/position.hh>
```

### Public Member Functions

- `position ()`  
*Construct a position.*
- `void initialize (std::string *fn)`  
*Initialization.*

### Line and Column related manipulators

- `void lines (int count=1)`  
*(line related) Advance to the COUNT next lines.*
- `void columns (int count=1)`  
*(column related) Advance to the COUNT next columns.*

### Public Attributes

- `std::string * filename`  
*File name to which this position refers.*
- `unsigned int line`  
*Current line number.*
- `unsigned int column`  
*Current column number.*

### 7.88.1 Detailed Description

Abstract a position.

### 7.88.2 Constructor & Destructor Documentation

#### 7.88.2.1 eltlyy::position::position () [inline]

Construct a position.

### 7.88.3 Member Function Documentation

#### 7.88.3.1 void eltlyy::position::columns (int *count* = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by eltlyy::operator+=().

#### 7.88.3.2 void eltlyy::position::initialize (std::string \**fn*) [inline]

Initialization.

References column, filename, and line.

Referenced by eltlyy::location::step().

#### 7.88.3.3 void eltlyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References column, and line.

### 7.88.4 Member Data Documentation

#### 7.88.4.1 unsigned int eltlyy::position::column

Current column number.

Referenced by columns(), initialize(), lines(), eltlyy::operator<<(), and eltlyy::operator==().

#### 7.88.4.2 std::string\* eltlyy::position::filename

File name to which this position refers.

Referenced by `initialize()`, `eltlyy::operator<<()`, and `eltlyy::operator==()`.

#### 7.88.4.3 unsigned int eltlyy::position::line

Current line number.

Referenced by `initialize()`, `lines()`, `eltlyy::operator<<()`, and `eltlyy::operator==()`.

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

- `eltlparse/position.hh`

## 7.89 Itlly::position Class Reference

Abstract a position.

```
#include <ltlparse/position.hh>
```

### Public Member Functions

- `position ()`  
*Construct a position.*
- `void initialize (std::string *fn)`  
*Initialization.*

### Line and Column related manipulators

- `void lines (int count=1)`  
*(line related) Advance to the COUNT next lines.*
- `void columns (int count=1)`  
*(column related) Advance to the COUNT next columns.*

### Public Attributes

- `std::string * filename`  
*File name to which this position refers.*
- `unsigned int line`  
*Current line number.*
- `unsigned int column`  
*Current column number.*

### 7.89.1 Detailed Description

Abstract a position.

### 7.89.2 Constructor & Destructor Documentation

#### 7.89.2.1 Itlyy::position::position () [inline]

Construct a position.

### 7.89.3 Member Function Documentation

#### 7.89.3.1 void Itlyy::position::columns (int *count* = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by Itlyy::operator+=().

#### 7.89.3.2 void Itlyy::position::initialize (std::string \**fn*) [inline]

Initialization.

References column, filename, and line.

Referenced by Itlyy::location::step().

#### 7.89.3.3 void Itlyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References column, and line.

### 7.89.4 Member Data Documentation

#### 7.89.4.1 unsigned int Itlyy::position::column

Current column number.

Referenced by columns(), initialize(), lines(), Itlyy::operator<<(), and Itlyy::operator==().

#### 7.89.4.2 std::string\* Itlyy::position::filename

File name to which this position refers.

Referenced by initialize(), ltlyy::operator<<(), and ltlyy::operator==().

#### 7.89.4.3 unsigned int ltlyy::position::line

Current line number.

Referenced by initialize(), lines(), ltlyy::operator<<(), and ltlyy::operator==().

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

- ltlparse/[position.hh](#)

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

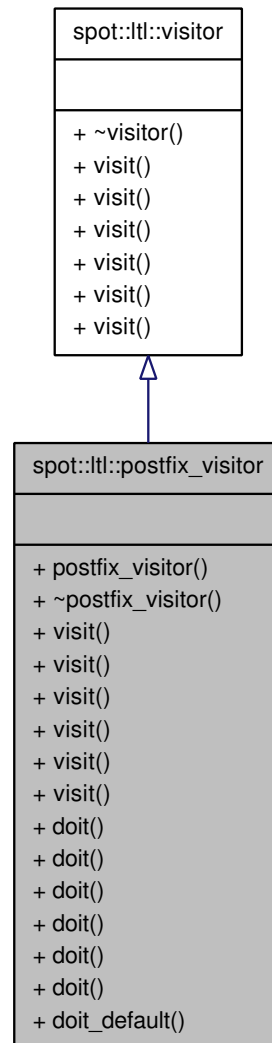
Apply an algorithm on each node of an AST, during a postfix traversal.

Override one or more of the postfix\_visitor::doit methods with the algorithm to apply.

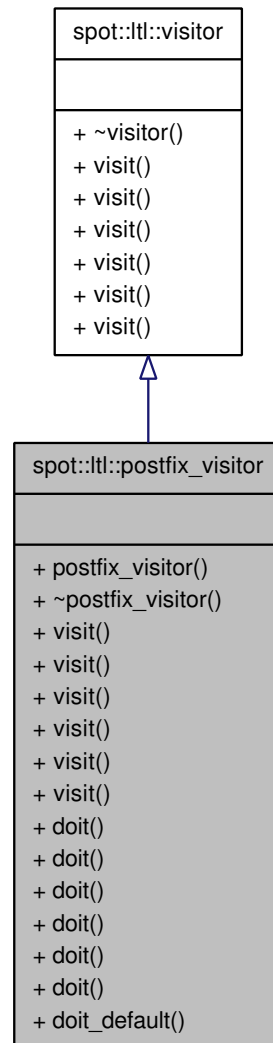
```
#include <ltlvisit/postfix.hh>
```



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\(automatop \\*c\)](#)
- 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\(automatop \\*mo\)](#)

- virtual void [doit](#) (constant \*c)
- virtual void [doit\\_default](#) (formula \*f)

### 7.90.1 Detailed Description

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

Override one or more of the postfix\_visitor::doit methods with the algorithm to apply.

### 7.90.2 Constructor & Destructor Documentation

#### 7.90.2.1 spot::ltl::postfix\_visitor::postfix\_visitor ()

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

### 7.90.3 Member Function Documentation

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

**7.90.3.2** virtual void spot::ltl::postfix\_visitor::doit (automatop \* mo) [virtual]

**7.90.3.3** virtual void spot::ltl::postfix\_visitor::doit (multop \* mo) [virtual]

**7.90.3.4** virtual void spot::ltl::postfix\_visitor::doit (binop \* bo) [virtual]

**7.90.3.5** virtual void spot::ltl::postfix\_visitor::doit (unop \* uo) [virtual]

**7.90.3.6** virtual void spot::ltl::postfix\_visitor::doit (atomic\_prop \* ap) [virtual]

**7.90.3.7** virtual void spot::ltl::postfix\_visitor::doit\_default (formula \* *f*) [virtual]

**7.90.3.8** void spot::ltl::postfix\_visitor::visit (constant \* *c*) [virtual]

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

**7.90.3.9** void spot::ltl::postfix\_visitor::visit (automatop \* *c*) [virtual]

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

**7.90.3.10** void spot::ltl::postfix\_visitor::visit (multop \* *mo*) [virtual]

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

**7.90.3.11** void spot::ltl::postfix\_visitor::visit (binop \* *bo*) [virtual]

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

**7.90.3.12** void spot::ltl::postfix\_visitor::visit (unop \* *uo*) [virtual]

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

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

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

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

- [ltlvisit/postfix.hh](#)

## 7.91 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

### 7.91.1 Detailed Description

**template<class T> struct spot::ptr\_hash< T >**

A hash function for pointers.

### 7.91.2 Member Function Documentation

**7.91.2.1 template<class T > size\_t spot::ptr\_hash< T >::operator() (const T \* *p*) const [inline]**

Referenced by spot::string\_hash::operator()().

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

- misc/[hash.hh](#)

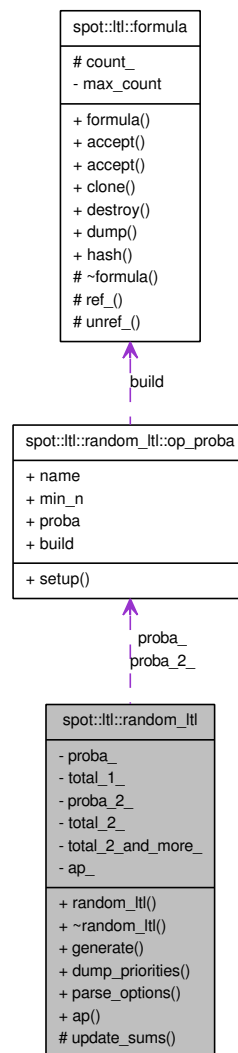
## 7.92 spot::ltl::random\_ltl Class Reference

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.

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

Collaboration diagram for spot::ltl::random\_ltl:



## Classes

- struct [op\\_proba](#)

## 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_`

#### 7.92.1 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()`.

#### 7.92.2 Constructor & Destructor Documentation

##### 7.92.2.1 `spot::ltl::random_ltl::random_ltl (const atomic_prop_set * ap)`

Create a random LTL generator using atomic propositions from *ap*.

##### 7.92.2.2 `spot::ltl::random_ltl::~~random_ltl ()`

#### 7.92.3 Member Function Documentation

##### 7.92.3.1 `const atomic_prop_set* spot::ltl::random_ltl::ap () const [inline]`

Return the set of atomic proposition used to build formulae.

References `ap_`.

### 7.92.3.2 `std::ostream& spot::ltl::random_ltl::dump_priorities (std::ostream & os) const`

Print the priorities of each operator, constants, and atomic propositions.

### 7.92.3.3 `formula* spot::ltl::random_ltl::generate (int n) const`

Generate a formula of size *n*.

It is possible to obtain formulae that are smaller than *n*, because some simple simplifications are performed by the AST. (For instance the formula `a | a` is automatically reduced to `a` by [spot::ltl::multop](#).)

Furthermore, for the purpose of this generator, `a | b | c` has length 5, while it has length 4 for [spot::ltl::length\(\)](#).

### 7.92.3.4 `const char* spot::ltl::random_ltl::parse_options (char * options)`

Update the priorities used to generate LTL formulae.

The initial priorities are as follows.

```

/// ap      n
/// false   1
/// true    1
/// not      1
/// F        1
/// G        1
/// X        1
/// equiv    1
/// implies  1
/// xor      1
/// R        1
/// U        1
/// W        1
/// M        1
/// and      1
/// or       1
///

```

Where *n* is the number of atomic propositions in the set passed to the constructor.

This means that each operator has equal chance to be selected. Also, each atomic proposition has as much chance as each constant (i.e., true and false) to be picked. This can be

These priorities can be altered using this function. *options* should be comma-separated list of KEY=VALUE assignments, using keys from the above list. For instance `"xor=0, F=3"` will prevent `xor` from being used, and will raise the relative probability of occurrences of the `F` operator.



**7.92.3.5** void spot::ltl::random\_ltl::update\_sums () [protected]

#### **7.92.4 Member Data Documentation**

**7.92.4.1** const atomic\_prop\_set\* spot::ltl::random\_ltl::ap\_ [private]

Referenced by ap().

**7.92.4.2** op\_proba\* spot::ltl::random\_ltl::proba\_ [private]

**7.92.4.3** op\_proba\* spot::ltl::random\_ltl::proba\_2\_ [private]

**7.92.4.4** double spot::ltl::random\_ltl::total\_1\_ [private]

**7.92.4.5** double spot::ltl::random\_ltl::total\_2\_ [private]

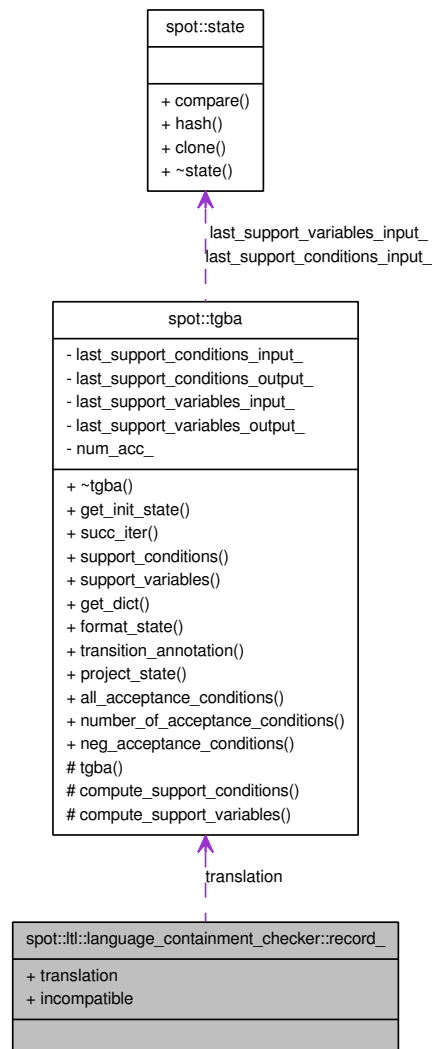
**7.92.4.6** double spot::ltl::random\_ltl::total\_2\_and\_more\_ [private]

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

- ltlvisit/[randomltl.hh](#)

## 7.93 spot::ltl::language\_containment\_checker::record\_ Struct Reference

Collaboration diagram for spot::ltl::language\_containment\_checker::record\_:



### Public Types

- typedef std::map< const [record\\_](#) \*, bool > [incomp\\_map](#)

### Public Attributes

- const [tgba](#) \* [translation](#)
- [incomp\\_map](#) [incompatible](#)

### 7.93.1 Member Typedef Documentation

**7.93.1.1** typedef std::map<const record\_\*, bool> spot::ltl::language\_containment\_checker::record\_::incomp\_map

### 7.93.2 Member Data Documentation

**7.93.2.1** incomp\_map spot::ltl::language\_containment\_checker::record\_::incompatible

**7.93.2.2** const tgba\* spot::ltl::language\_containment\_checker::record\_::translation

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

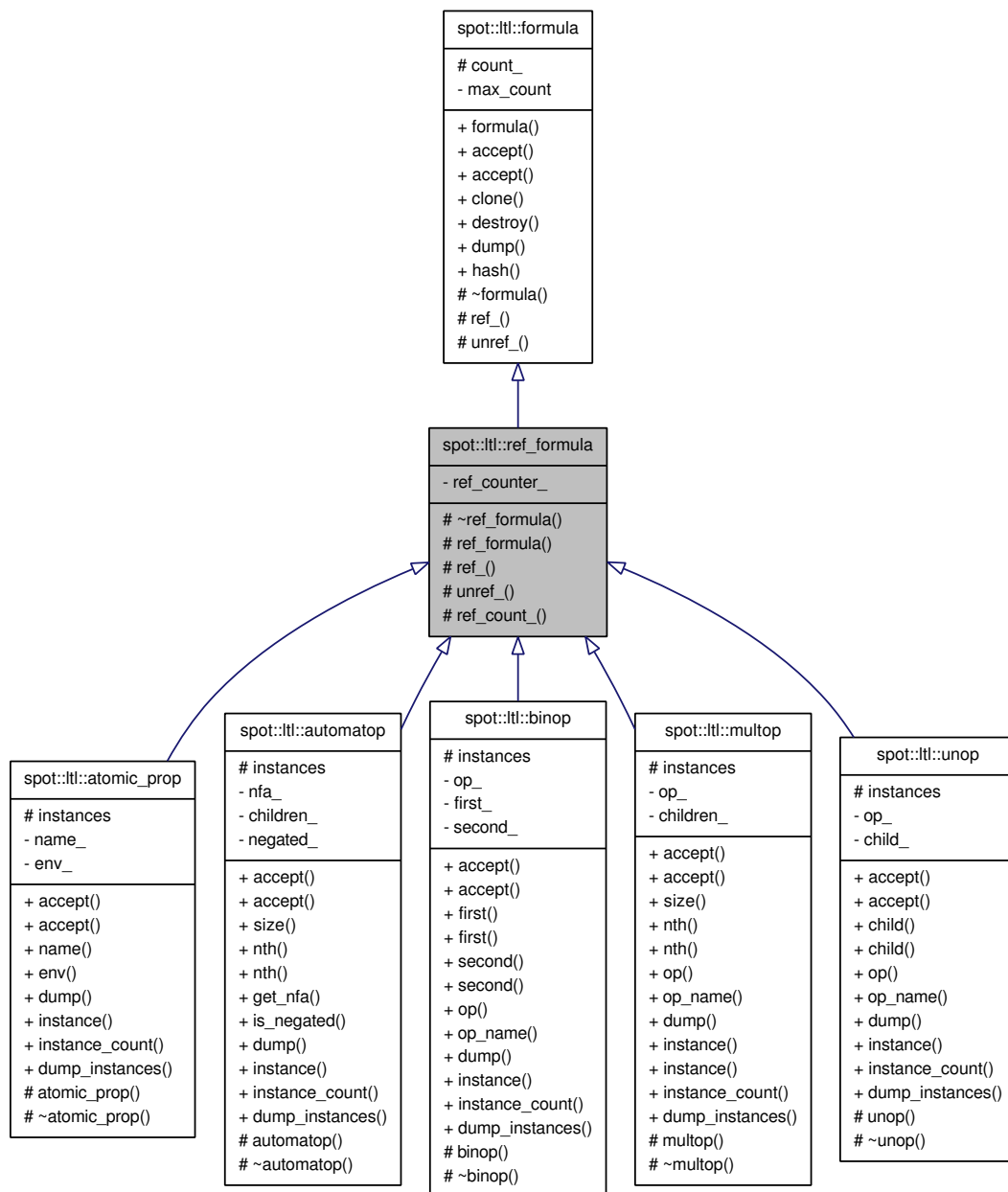
- ltlvisit/[contain.hh](#)

## 7.94 spot::ltl::ref\_formula Class Reference

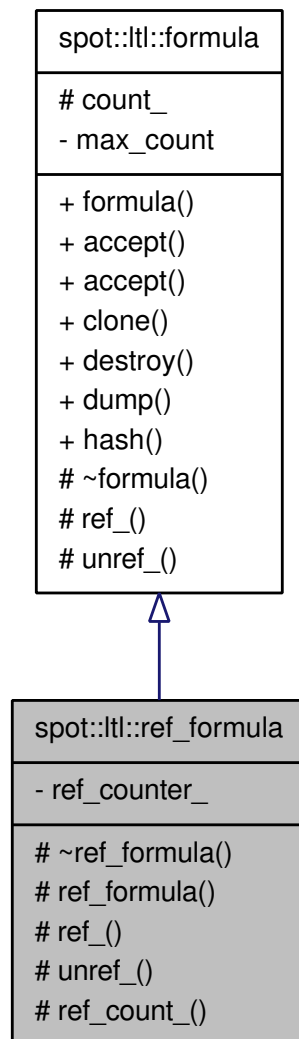
A reference-counted LTL formula.

```
#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 * clone () const`  
*clone this node*
- void `destroy () const`  
*release this node*
- virtual std::string `dump () const =0`

*Return a canonic representation of the formula.*

- `size_t hash () const`

*Return a hash key for the formula.*

### 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.*

### Protected Attributes

- `size_t count_`

*The hash key of this formula.*

### Private Attributes

- `unsigned ref_counter_`

#### 7.94.1 Detailed Description

A reference-counted LTL formula.

#### 7.94.2 Constructor & Destructor Documentation

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

**7.94.2.2** `spot::ltl::ref_formula::ref_formula ()` `[protected]`

### 7.94.3 Member Function Documentation

#### 7.94.3.1 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::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

#### 7.94.3.2 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::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

#### 7.94.3.3 formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

#### 7.94.3.4 void spot::ltl::formula::destroy () const [inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >::add\\_acceptance\\_condition\(\)](#), and [spot::tgba\\_explicit\\_labelled< std::string, string\\_hash >::declare\\_acceptance\\_condition\(\)](#).

#### 7.94.3.5 virtual std::string spot::ltl::formula::dump () const [pure virtual, inherited]

Return a canonic representation of the formula.

Implemented in [spot::ltl::atomic\\_prop](#), [spot::ltl::automatop](#), [spot::ltl::binop](#), [spot::ltl::constant](#), [spot::ltl::multop](#), and [spot::ltl::unop](#).

#### 7.94.3.6 size\_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash key for the formula.

References [spot::ltl::formula::count\\_](#).

**7.94.3.7 void spot::ltl::ref\_formula::ref\_() [protected, virtual]**

increment reference counter if any

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

**7.94.3.8 unsigned spot::ltl::ref\_formula::ref\_count\_() [protected]**

Number of references to this formula.

**7.94.3.9 bool spot::ltl::ref\_formula::unref\_() [protected, virtual]**

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

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

**7.94.4 Member Data Documentation****7.94.4.1 size\_t spot::ltl::formula::count\_ [protected, inherited]**

The hash key of this formula.

Referenced by [spot::ltl::formula::hash\(\)](#).

**7.94.4.2 unsigned spot::ltl::ref\_formula::ref\_counter\_ [private]**

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

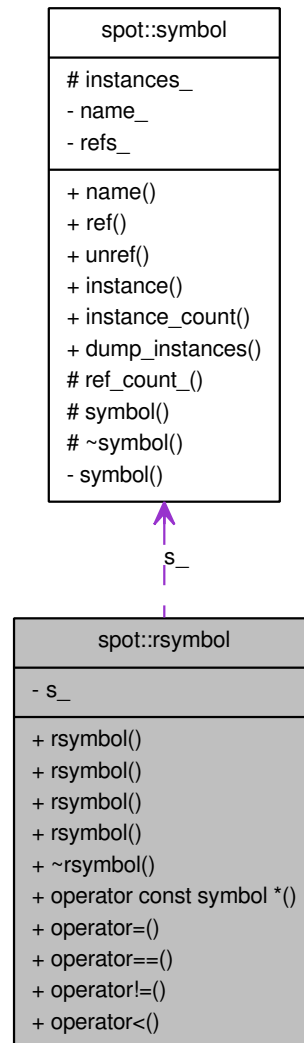
- [ltlast/refformula.hh](#)

**7.95 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_`

### 7.95.1 Constructor & Destructor Documentation

#### 7.95.1.1 `spot::rsymbol::rsymbol (const symbol * s)` [\[inline\]](#)

Referenced by `operator=()`.

#### 7.95.1.2 `spot::rsymbol::rsymbol (const std::string & s)` [\[inline\]](#)

#### 7.95.1.3 `spot::rsymbol::rsymbol (const char * s)` [\[inline\]](#)

#### 7.95.1.4 `spot::rsymbol::rsymbol (const rsymbol & rs)` [\[inline\]](#)

References `spot::symbol::ref()`, and `s_`.

#### 7.95.1.5 `spot::rsymbol::~rsymbol ()` [\[inline\]](#)

References `s_`, and `spot::symbol::unref()`.

Referenced by `operator=()`.

### 7.95.2 Member Function Documentation

#### 7.95.2.1 `spot::rsymbol::operator const symbol * () const` [\[inline\]](#)

References `s_`.

#### 7.95.2.2 `bool spot::rsymbol::operator!= (const rsymbol & rs) const` [\[inline\]](#)

References `s_`.

### 7.95.2.3 bool spot::rsymbol::operator< (const rsymbol & *rs*) const [inline]

References `s_`.

### 7.95.2.4 const rsymbol& spot::rsymbol::operator= (const rsymbol & *rs*) [inline]

References `rsymbol()`, and `~rsymbol()`.

### 7.95.2.5 bool spot::rsymbol::operator== (const rsymbol & *rs*) const [inline]

References `s_`.

## 7.95.3 Member Data Documentation

### 7.95.3.1 const symbol\* spot::rsymbol::s\_ [private]

Referenced by `operator const symbol *()`, `operator!=()`, `operator<()`, `operator==()`, `rsymbol()`, and `~rsymbol()`.

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

- [evtgba/symbol.hh](#)

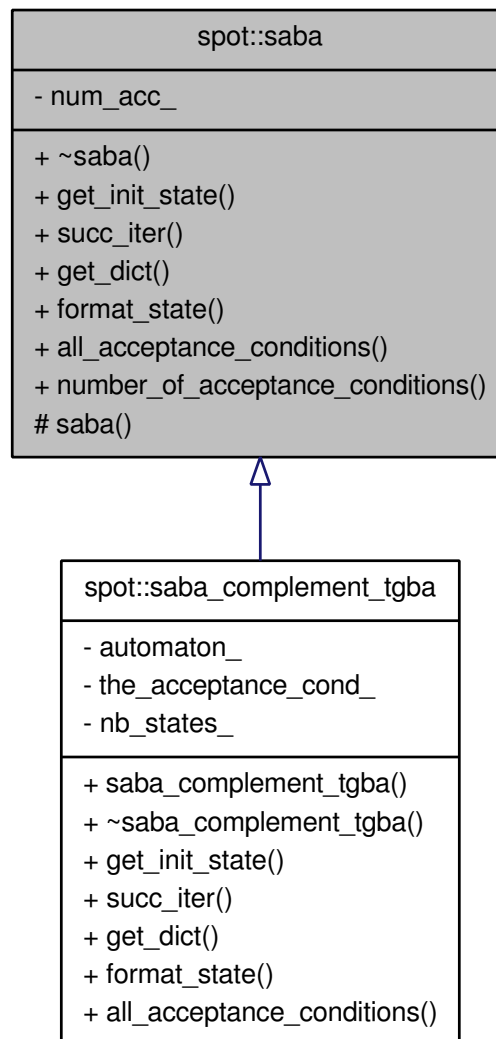
## 7.96 spot::saba Class Reference

A State-based Alternating (Generalized) Büchi Automaton.

Browsing such automaton can be achieved using two functions: `get_init_state`, and `succ_iter`. The former returns the initial state while the latter lists the successor states of any state.

```
#include <saba/saba.hh>
```

Inheritance diagram for spot::saba:



## Public Member Functions

- virtual `~saba()`
- virtual `saba_state * get_init_state()` const =0  
*Get the initial state of the automaton.*
- virtual `saba_succ_iterator * succ_iter` (const `saba_state *local_state`) const =0  
*Get an iterator over the successors of local\_state.*
- virtual `bdd_dict * get_dict()` const =0  
*Get the dictionary associated to the automaton.*
- virtual `std::string format_state` (const `saba_state *state`) const =0  
*Format the state as a string for printing.*

- 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.*

### Protected Member Functions

- [saba](#) ()

### Private Attributes

- int [num\\_acc\\_](#)

## 7.96.1 Detailed Description

A State-based Alternating (Generalized) Büchi Automaton.

Browsing such automaton can be achieved using two functions: `get_init_state`, and `succ_iter`. The former returns the initial state while the latter lists the successor states of any state. Note that although this is a transition-based automata, we never represent transitions! Transition informations are obtained by querying the iterator over the successors of a state.

## 7.96.2 Constructor & Destructor Documentation

### 7.96.2.1 spot::saba::saba () [protected]

### 7.96.2.2 virtual spot::saba::~~saba () [virtual]

## 7.96.3 Member Function Documentation

### 7.96.3.1 virtual bdd spot::saba::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::saba\\_complement\\_tgba](#).

### 7.96.3.2 virtual std::string spot::saba::format\_state (const saba\_state \* *state*) const [pure virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implemented in [spot::saba\\_complement\\_tgba](#).

### 7.96.3.3 virtual bdd\_dict\* spot::saba::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::saba\\_complement\\_tgba](#).

### 7.96.3.4 virtual saba\_state\* spot::saba::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::saba\\_complement\\_tgba](#).

### 7.96.3.5 virtual unsigned int spot::saba::number\_of\_acceptance\_conditions () const [virtual]

The number of acceptance conditions.

### 7.96.3.6 virtual saba\_succ\_iterator\* spot::saba::succ\_iter (const saba\_state \* *local\_state*) 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.

#### 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).

Implemented in [spot::saba\\_complement\\_tgba](#).

### 7.96.4 Member Data Documentation

#### 7.96.4.1 int spot::saba::num\_acc\_ [mutable, private]

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

- [saba/saba.hh](#)

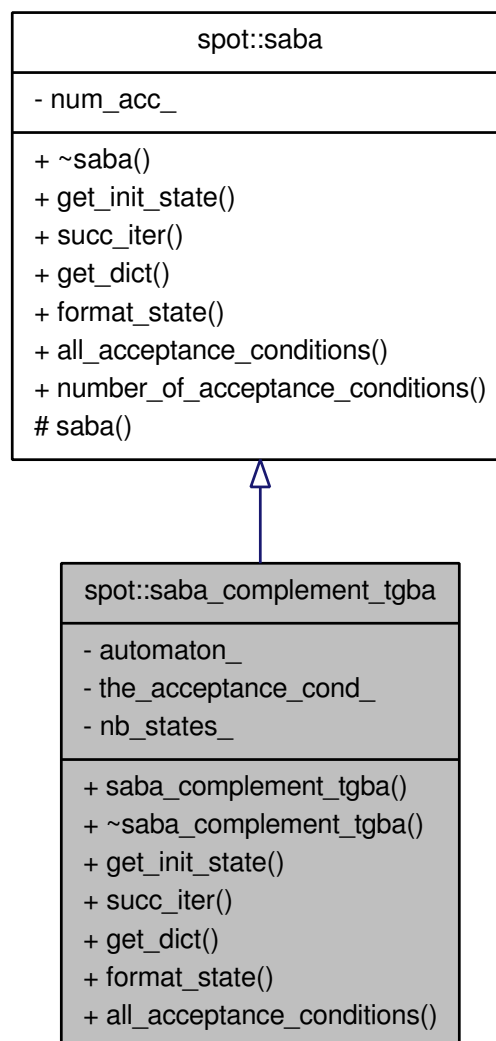
## 7.97 spot::saba\_complement\_tgba Class Reference

Complement a TGBA and produce a SABA.

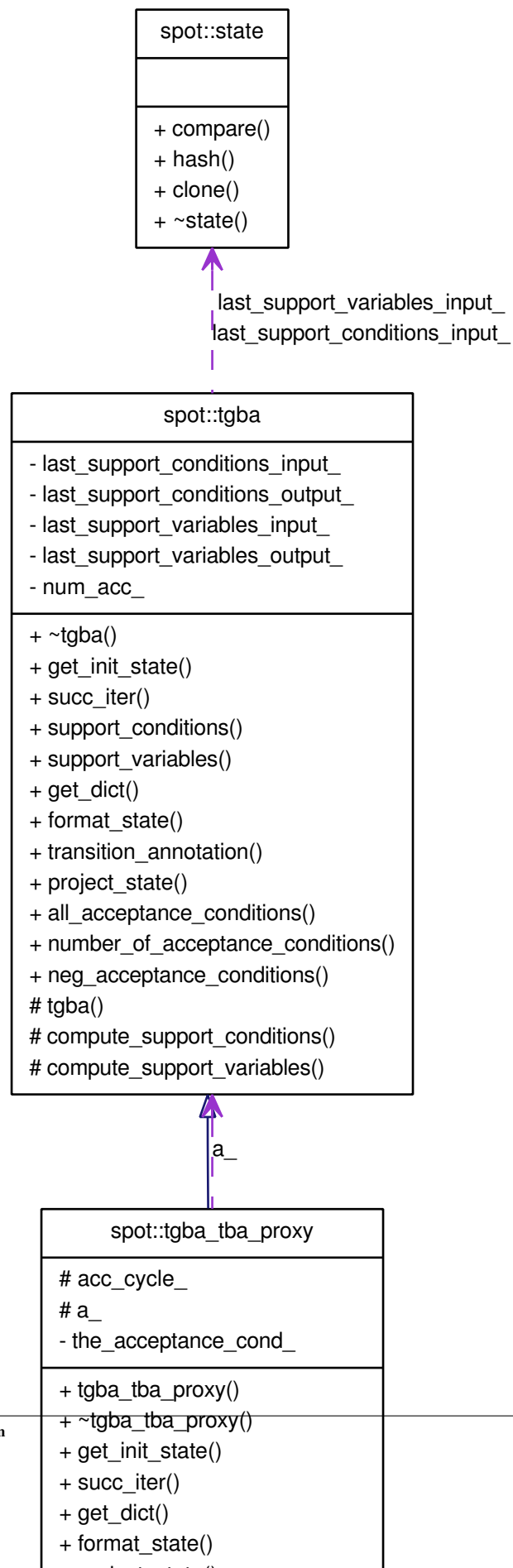
The original TGBA is transformed into a States-based Büchi Automaton.

```
#include <saba/sabacomplementtgba.hh>
```

Inheritance diagram for spot::saba\_complement\_tgba:



Collaboration diagram for spot::saba\_complement\_tgba:





### Public Member Functions

- [saba\\_complement\\_tgba](#) (const [tgba](#) \*a)
- virtual [~saba\\_complement\\_tgba](#) ()
- virtual [saba\\_state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [saba\\_succ\\_iterator](#) \* [succ\\_iter](#) (const [saba\\_state](#) \*local\_state) 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 [saba\\_state](#) \*state) const  
*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 unsigned int [number\\_of\\_acceptance\\_conditions](#) () const  
*The number of acceptance conditions.*

### Private Attributes

- const [tgba\\_sba\\_proxy](#) \* [automaton\\_](#)
- [bdd](#) [the\\_acceptance\\_cond\\_](#)
- unsigned [nb\\_states\\_](#)

#### 7.97.1 Detailed Description

Complement a TGBA and produce a SABA.

The original TGBA is transformed into a States-based Büchi Automaton. Several techniques are supposed to be applied on the resulting automaton before its transformation into a TGBA. Those techniques are expected to reduce the size of the automaton.

This algorithm comes from:

```

/// @Article{          gurumurthy.03.lncs,
///   title            = {On complementing nondeterministic {B\"uchi} automata},
///   author           = {Gurumurthy, S. and Kupferman, O. and Somenzi, F. and
///                      Vardi, M.Y.},
///   journal          = {Lecture Notes in Computer Science},
///   pages             = {96--110},
///   year             = {2003},
///   publisher        = {Springer-Verlag}
/// }
///

```

The construction is done on-the-fly, by the `saba_complement_succ_iterator` class.

#### See also

`saba_complement_succ_iterator`

## 7.97.2 Constructor & Destructor Documentation

**7.97.2.1** `spot::saba_complement_tgba::saba_complement_tgba (const tgba * a)`

**7.97.2.2** `virtual spot::saba_complement_tgba::~~saba_complement_tgba ()` **[virtual]**

## 7.97.3 Member Function Documentation

**7.97.3.1** `virtual bdd spot::saba_complement_tgba::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::saba](#).

**7.97.3.2** `virtual std::string spot::saba_complement_tgba::format_state (const saba_state * state)`  
`const` **[virtual]**

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::saba](#).

**7.97.3.3** `virtual bdd_dict* spot::saba_complement_tgba::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::saba](#).

**7.97.3.4** `virtual saba_state* spot::saba_complement_tgba::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::saba](#).

**7.97.3.5** `virtual unsigned int spot::saba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

**7.97.3.6** `virtual saba_succ_iterator* spot::saba_complement_tgba::succ_iter (const saba_state * local_state) 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.

#### 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).

Implements [spot::saba](#).

#### 7.97.4 Member Data Documentation

**7.97.4.1** `const tgba_sba_proxy* spot::saba_complement_tgba::automaton_ [private]`

**7.97.4.2** `unsigned spot::saba_complement_tgba::nb_states_ [private]`

**7.97.4.3** `bdd spot::saba_complement_tgba::the_acceptance_cond_ [private]`

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

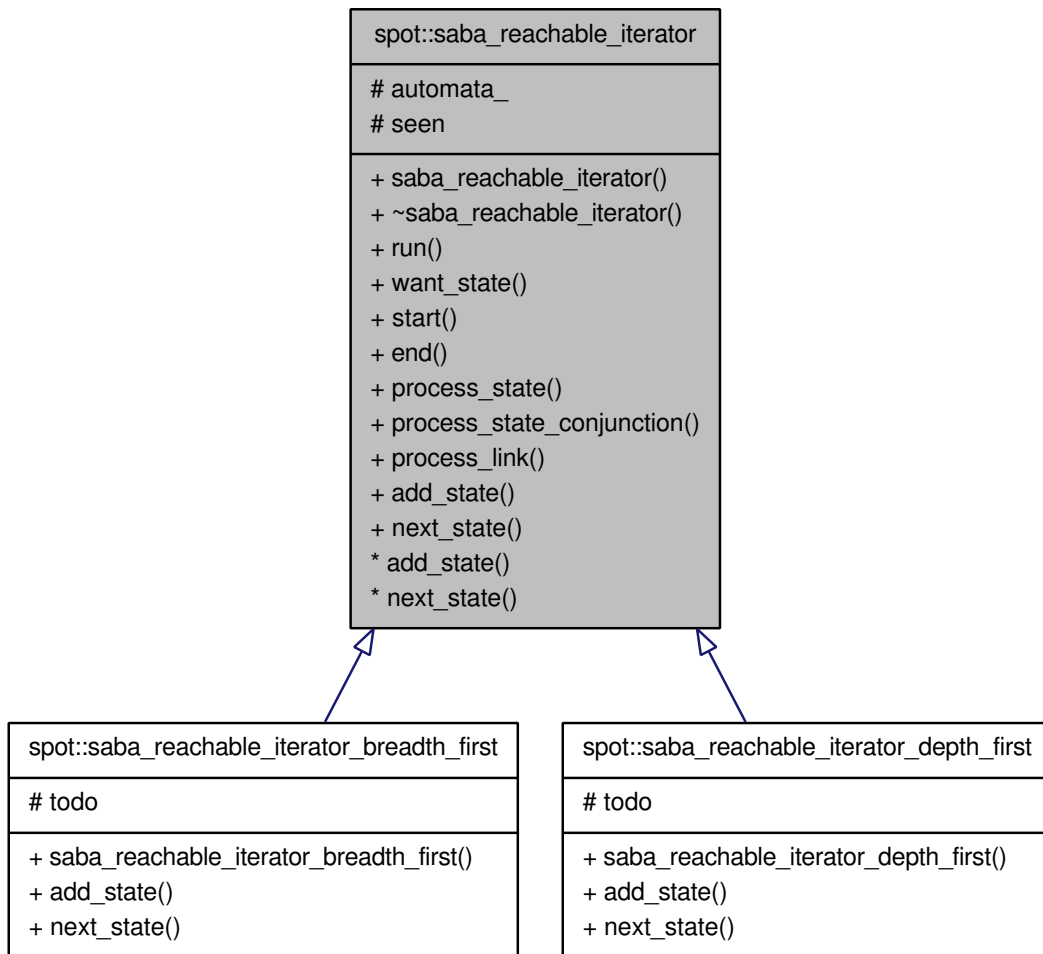
- [saba/sabacomplementtgba.hh](#)

## 7.98 spot::saba\_reachable\_iterator Class Reference

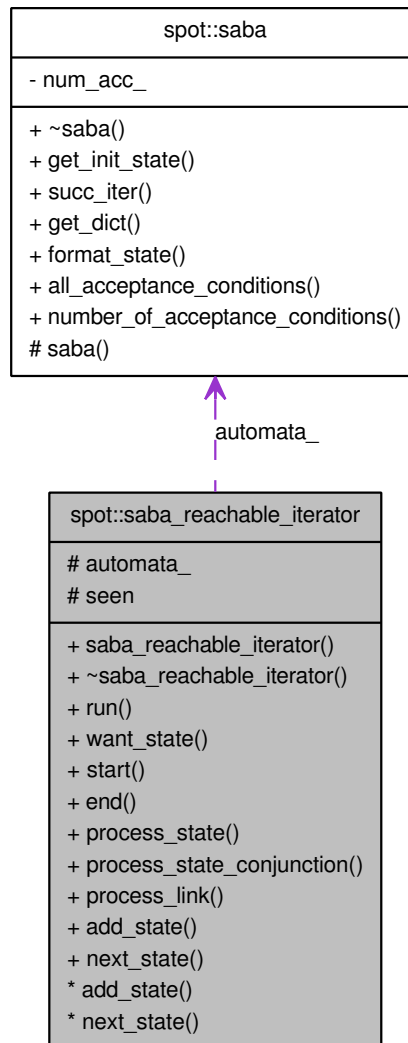
Iterate over all reachable states of a [spot::saba](#).

```
#include <sabaalgos/sabareachiter.hh>
```

Inheritance diagram for spot::saba\_reachable\_iterator:



Collaboration diagram for spot::saba\_reachable\_iterator:



## Public Member Functions

- `saba_reachable_iterator` (const `saba` \*a)
- virtual `~saba_reachable_iterator` ()
- void `run` ()  
*Iterate over all reachable states of a `spot::saba`.*
- virtual bool `want_state` (const `saba_state` \*s) const
- 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 `saba_state` \*s, int n)

- virtual void [process\\_state\\_conjunction](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)
- virtual void [process\\_link](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state](#) \*out\_s, int out, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)

#### Todo list management.

Called by [run\(\)](#) to register newly discovered states.

[spot::saba\\_reachable\\_iterator\\_depth\\_first](#) and [spot::saba\\_reachable\\_iterator\\_breadth\\_first](#) offer two precanned implementations for these functions.

- virtual void [add\\_state](#) (const [saba\\_state](#) \*s)=0
  - virtual const [saba\\_state](#) \* [next\\_state](#) ()=0
- Called by [run\(\)](#) to obtain the next state to process.

#### Protected Types

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

#### Protected Attributes

- const [saba](#) \* [automata\\_](#)  
The [spot::saba](#) to explore.
- [seen\\_map](#) [seen](#)  
States already seen.

### 7.98.1 Detailed Description

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

### 7.98.2 Member Typedef Documentation

- 7.98.2.1** typedef Sgi::hash\_map<const [saba\\_state](#)\*, int, [saba\\_state\\_ptr\\_hash](#), [saba\\_state\\_ptr\\_equal](#)> [spot::saba\\_reachable\\_iterator::seen\\_map](#) [protected]

### 7.98.3 Constructor & Destructor Documentation

- 7.98.3.1** [spot::saba\\_reachable\\_iterator::saba\\_reachable\\_iterator](#) (const [saba](#) \* a)

- 7.98.3.2** virtual [spot::saba\\_reachable\\_iterator::~~saba\\_reachable\\_iterator](#) () [virtual]

### 7.98.4 Member Function Documentation

#### 7.98.4.1 `virtual void spot::saba_reachable_iterator::add_state (const saba_state * s) [pure virtual]`

Implemented in [spot::saba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::saba\\_reachable\\_iterator\\_breadth\\_first](#).

#### 7.98.4.2 `virtual void spot::saba_reachable_iterator::end () [virtual]`

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

#### 7.98.4.3 `virtual const saba_state* spot::saba_reachable_iterator::next_state () [pure virtual]`

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

Implemented in [spot::saba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::saba\\_reachable\\_iterator\\_breadth\\_first](#).

#### 7.98.4.4 `virtual void spot::saba_reachable_iterator::process_link (const saba_state * in_s, int in, const saba_state * out_s, int out, const saba_state_conjunction * sc, int sc_id, const saba_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.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

The *in\_s* and *out\_s* states are owned by the [spot::saba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

#### 7.98.4.5 `virtual void spot::saba_reachable_iterator::process_state (const saba_state * s, int n) [virtual]`

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

##### Parameters

*s* The current state.

*n* A unique number assigned to *s*.

**7.98.4.6** `virtual void spot::saba_reachable_iterator::process_state_conjunction (const saba_state * in_s, int in, const saba_state_conjunction * sc, int sc_id, const saba_succ_iterator * si) [virtual]`

Called by [run\(\)](#) to process a conjunction of states.

#### Parameters

*in\_s* The current state.

*in* An unique number assigned to *in\_s*.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

**7.98.4.7** `void spot::saba_reachable_iterator::run ()`

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

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), [process\\_state\\_conjunction\(\)](#) and [process\\_link\(\)](#), while it iterates over states.

**7.98.4.8** `virtual void spot::saba_reachable_iterator::start () [virtual]`

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

**7.98.4.9** `virtual bool spot::saba_reachable_iterator::want_state (const saba_state * s) const [virtual]`

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

#### 7.98.5 Member Data Documentation

**7.98.5.1** `const saba* spot::saba_reachable_iterator::automata_ [protected]`

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

**7.98.5.2** `seen_map spot::saba_reachable_iterator::seen [protected]`

States already seen.



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

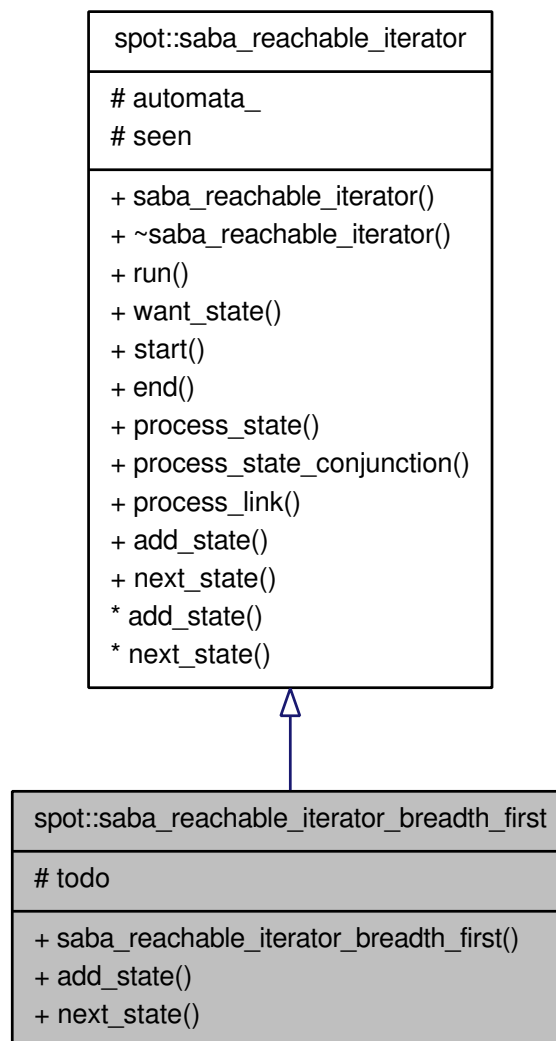
- sabaalgorithms/[sabareachiter.hh](#)

## 7.99 spot::saba\_reachable\_iterator\_breadth\_first Class Reference

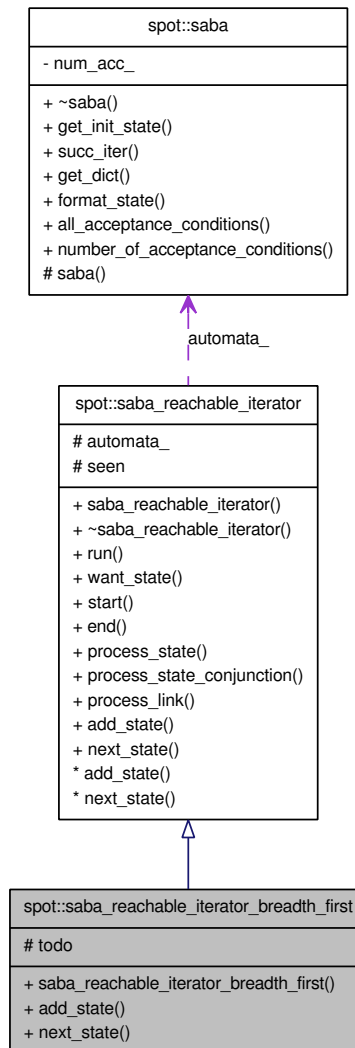
An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states breadth first.

```
#include <sabaalgorithms/sabareachiter.hh>
```

Inheritance diagram for spot::saba\_reachable\_iterator\_breadth\_first:



Collaboration diagram for spot::saba\_reachable\_iterator\_breadth\_first:



## Public Member Functions

- `saba_reachable_iterator_breadth_first` (const `saba` \*a)
- virtual void `add_state` (const `saba_state` \*s)
- virtual const `saba_state` \* `next_state` ()

*Called by `run()` to obtain the next state to process.*

- void `run` ()

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

- virtual bool `want_state` (const `saba_state` \*s) const
- 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 [saba\\_state](#) \*s, int n)
- virtual void [process\\_state\\_conjunction](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)
- virtual void [process\\_link](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state](#) \*out\_s, int out, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)

## Protected Types

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

## Protected Attributes

- std::deque< const [saba\\_state](#) \* > [todo](#)  
*A queue of states yet to explore.*
- const [saba](#) \* [automata\\_](#)  
*The [spot::saba](#) to explore.*
- [seen\\_map](#) [seen](#)  
*States already seen.*

### 7.99.1 Detailed Description

An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states breadth first.

### 7.99.2 Member Typedef Documentation

- 7.99.2.1** typedef Sgi::hash\_map<const [saba\\_state](#)\*, int, [saba\\_state\\_ptr\\_hash](#), [saba\\_state\\_ptr\\_equal](#)> [spot::saba\\_reachable\\_iterator::seen\\_map](#) [[protected](#), [inherited](#)]

### 7.99.3 Constructor & Destructor Documentation

- 7.99.3.1** [spot::saba\\_reachable\\_iterator\\_breadth\\_first::saba\\_reachable\\_iterator\\_breadth\\_first](#) (const [saba](#) \* *a*)

### 7.99.4 Member Function Documentation

- 7.99.4.1** virtual void [spot::saba\\_reachable\\_iterator\\_breadth\\_first::add\\_state](#) (const [saba\\_state](#) \* *s*) [[virtual](#)]

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

#### 7.99.4.2 virtual void spot::saba\_reachable\_iterator::end () [virtual, inherited]

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

#### 7.99.4.3 virtual const saba\_state\* spot::saba\_reachable\_iterator\_breadth\_first::next\_state () [virtual]

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

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

#### 7.99.4.4 virtual void spot::saba\_reachable\_iterator::process\_link (const saba\_state \* *in\_s*, int *in*, const saba\_state \* *out\_s*, int *out*, const saba\_state\_conjunction \* *sc*, int *sc\_id*, const saba\_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.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

The *in\_s* and *out\_s* states are owned by the [spot::saba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

#### 7.99.4.5 virtual void spot::saba\_reachable\_iterator::process\_state (const saba\_state \* *s*, int *n*) [virtual, inherited]

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

##### Parameters

*s* The current state.

*n* A unique number assigned to *s*.

**7.99.4.6** `virtual void spot::saba_reachable_iterator::process_state_conjunction (const saba_state * in_s, int in, const saba_state_conjunction * sc, int sc_id, const saba_succ_iterator * si) [virtual, inherited]`

Called by [run\(\)](#) to process a conjunction of states.

#### Parameters

*in\_s* The current state.

*in* An unique number assigned to *in\_s*.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

**7.99.4.7** `void spot::saba_reachable_iterator::run () [inherited]`

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

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), [process\\_state\\_conjunction\(\)](#) and [process\\_link\(\)](#), while it iterates over states.

**7.99.4.8** `virtual void spot::saba_reachable_iterator::start () [virtual, inherited]`

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

**7.99.4.9** `virtual bool spot::saba_reachable_iterator::want_state (const saba_state * s) const [virtual, inherited]`

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

## 7.99.5 Member Data Documentation

**7.99.5.1** `const saba* spot::saba_reachable_iterator::automata_ [protected, inherited]`

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

**7.99.5.2** `seen_map spot::saba_reachable_iterator::seen [protected, inherited]`

States already seen.

### 7.99.5.3 std::deque<const saba\_state\*> spot::saba\_reachable\_iterator\_breadth\_first::todo [protected]

A queue of states yet to explore.

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

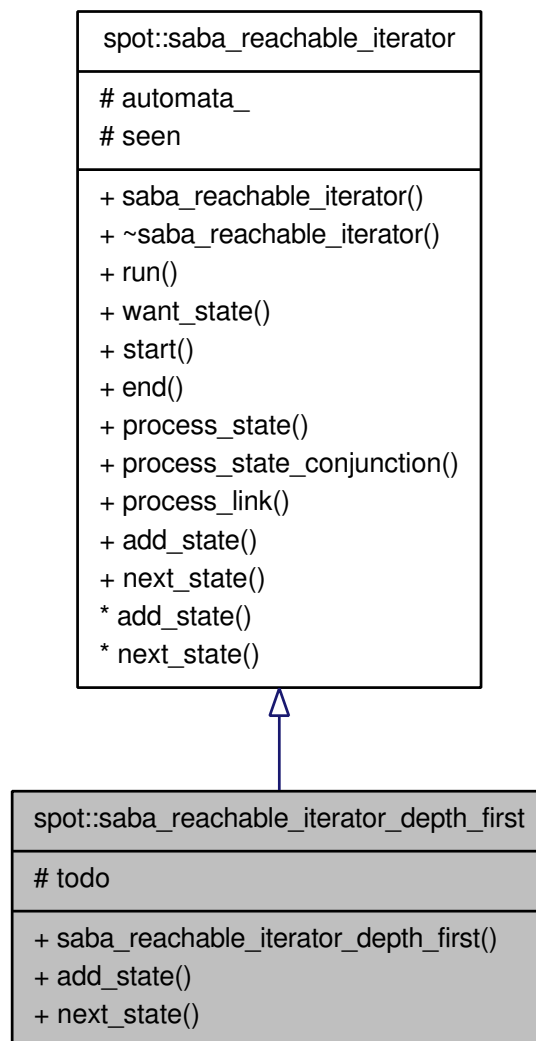
- sabaalgorithms/sabareachiter.hh

## 7.100 spot::saba\_reachable\_iterator\_depth\_first Class Reference

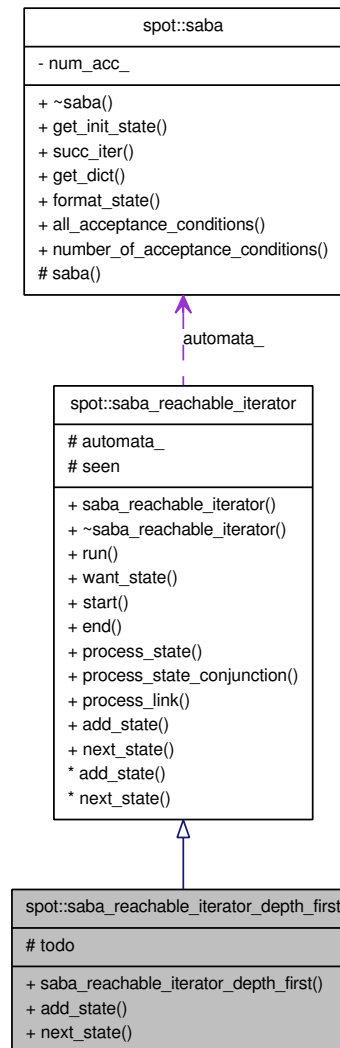
An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states depth first.

```
#include <sabaalgorithms/sabareachiter.hh>
```

Inheritance diagram for spot::saba\_reachable\_iterator\_depth\_first:



Collaboration diagram for spot::saba\_reachable\_iterator\_depth\_first:



## Public Member Functions

- `saba_reachable_iterator_depth_first` (const `saba` \*a)
- virtual void `add_state` (const `saba_state` \*s)
- virtual const `saba_state` \* `next_state` ()

*Called by `run()` to obtain the next state to process.*

- void `run` ()

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

- virtual bool `want_state` (const `saba_state` \*s) const
- 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 [saba\\_state](#) \*s, int n)
- virtual void [process\\_state\\_conjunction](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)
- virtual void [process\\_link](#) (const [saba\\_state](#) \*in\_s, int in, const [saba\\_state](#) \*out\_s, int out, const [saba\\_state\\_conjunction](#) \*sc, int sc\_id, const [saba\\_succ\\_iterator](#) \*si)

### Protected Types

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

### Protected Attributes

- std::stack< const [saba\\_state](#) \* > [todo](#)  
*A stack of states yet to explore.*
- const [saba](#) \* [automata\\_](#)  
*The [spot::saba](#) to explore.*
- [seen\\_map](#) [seen](#)  
*States already seen.*

## 7.100.1 Detailed Description

An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states depth first.

## 7.100.2 Member Typedef Documentation

- 7.100.2.1** typedef Sgi::hash\_map<const [saba\\_state](#)\*, int, [saba\\_state\\_ptr\\_hash](#), [saba\\_state\\_ptr\\_equal](#)> [spot::saba\\_reachable\\_iterator::seen\\_map](#) [[protected](#), [inherited](#)]

## 7.100.3 Constructor & Destructor Documentation

- 7.100.3.1** [spot::saba\\_reachable\\_iterator\\_depth\\_first::saba\\_reachable\\_iterator\\_depth\\_first](#) (const [saba](#) \* a)

## 7.100.4 Member Function Documentation

- 7.100.4.1** virtual void [spot::saba\\_reachable\\_iterator\\_depth\\_first::add\\_state](#) (const [saba\\_state](#) \* s) [[virtual](#)]



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

**7.100.4.2** `virtual void spot::saba_reachable_iterator::end () [virtual, inherited]`

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

**7.100.4.3** `virtual const saba_state* spot::saba_reachable_iterator_depth_first::next_state () [virtual]`

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

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

**7.100.4.4** `virtual void spot::saba_reachable_iterator::process_link (const saba_state * in_s, int in, const saba_state * out_s, int out, const saba_state_conjunction * sc, int sc_id, const saba_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.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

The *in\_s* and *out\_s* states are owned by the [spot::saba\\_reachable\\_iterator](#) instance and destroyed when the instance is destroyed.

**7.100.4.5** `virtual void spot::saba_reachable_iterator::process_state (const saba_state * s, int n) [virtual, inherited]`

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

#### Parameters

*s* The current state.

*n* A unique number assigned to *s*.

**7.100.4.6** virtual void spot::saba\_reachable\_iterator::process\_state\_conjunction (const saba\_state \* *in\_s*, int *in*, const saba\_state\_conjunction \* *sc*, int *sc\_id*, const saba\_succ\_iterator \* *si*) [**virtual**, **inherited**]

Called by [run\(\)](#) to process a conjunction of states.

#### Parameters

*in\_s* The current state.

*in* An unique number assigned to *in\_s*.

*sc* The [spot::saba\\_state\\_conjunction](#) positionned on the current conjunction.

*sc\_id* An unique number for the this transition assigned to *sc*.

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

**7.100.4.7** void spot::saba\_reachable\_iterator::run () [**inherited**]

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

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), [process\\_state\\_conjunction\(\)](#) and [process\\_link\(\)](#), while it iterates over states.

**7.100.4.8** virtual void spot::saba\_reachable\_iterator::start () [**virtual**, **inherited**]

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

**7.100.4.9** virtual bool spot::saba\_reachable\_iterator::want\_state (const saba\_state \* *s*) const [**virtual**, **inherited**]

Called by [add\\_state](#) or [next\\_states](#) implementations to filter states. Default implementation always return true.

### 7.100.5 Member Data Documentation

**7.100.5.1** const saba\* spot::saba\_reachable\_iterator::automata\_ [**protected**, **inherited**]

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

**7.100.5.2** seen\_map spot::saba\_reachable\_iterator::seen [**protected**, **inherited**]

States already seen.

### 7.100.5.3 std::stack<const saba\_state\*> spot::saba\_reachable\_iterator\_depth\_first::todo [protected]

A stack of states yet to explore.

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

- sabaalgs/[sabareachiter.hh](#)

## 7.101 spot::saba\_state Class Reference

Abstract class for saba states.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

- virtual int [compare](#) (const [saba\\_state](#) \*other) const =0  
*Compares two states (that come from the same automaton).*
- virtual size\_t [hash](#) () const =0  
*Hash a state.*
- virtual [saba\\_state](#) \* [clone](#) () const =0  
*Duplicate a state.*
- virtual bdd [acceptance\\_conditions](#) () const =0  
*Get the acceptance condition.*
- virtual ~[saba\\_state](#) ()

#### 7.101.1 Detailed Description

Abstract class for saba states.

#### 7.101.2 Constructor & Destructor Documentation

##### 7.101.2.1 virtual spot::saba\_state::~~saba\_state () [inline, virtual]

#### 7.101.3 Member Function Documentation

##### 7.101.3.1 virtual bdd spot::saba\_state::acceptance\_conditions () const [pure virtual]

Get the acceptance condition.

saba are state-labeled automata, then their acceptance conditions are labeled on states.

**7.101.3.2** `virtual saba_state* spot::saba_state::clone () const [pure virtual]`

Duplicate a state.

**7.101.3.3** `virtual int spot::saba_state::compare (const saba_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::saba\\_state\\_ptr\\_less\\_than](#)

Referenced by `spot::saba_state_ptr_equal::operator()()`, and `spot::saba_state_ptr_less_than::operator()()`.

**7.101.3.4** `virtual size_t spot::saba_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::saba\\_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.

Referenced by `spot::saba_state_ptr_hash::operator()()`.

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

- [saba/sabastate.hh](#)

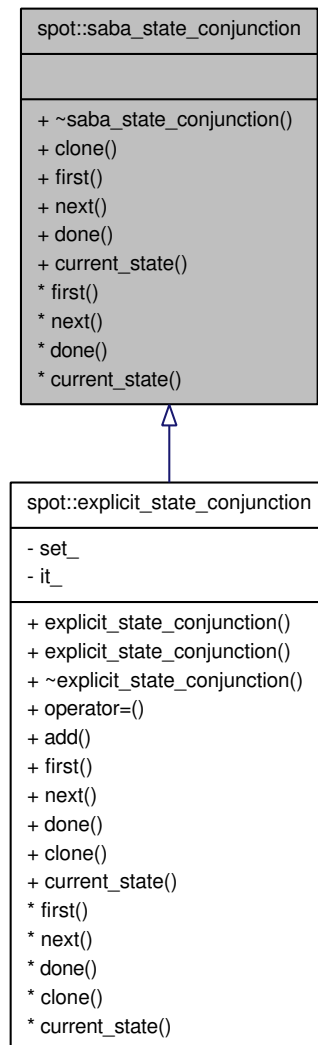
**7.102** `spot::saba_state_conjunction` Class Reference

Iterate over a conjunction of [saba\\_state](#).

This class provides the basic functionalities required to iterate over a conjunction of states of a saba.

```
#include <saba/sabasucciter.hh>
```

Inheritance diagram for spot::saba\_state\_conjunction:



## Public Member Functions

- virtual `~saba_state_conjunction ()`
- virtual `saba_state_conjunction * clone () const =0`  
Duplicate a *saba\_state* conjunction.

## Iteration

- virtual void `first ()=0`  
Position the iterator on the first successor of the conjunction (if any).
- virtual void `next ()=0`  
Jump to the next successor (if any).
- virtual bool `done () const =0`

*Check whether the iteration over a conjunction of states is finished.*

### Inspection

- virtual [saba\\_state](#) \* [current\\_state](#) () const =0

*Get the state of the current successor.*

## 7.102.1 Detailed Description

Iterate over a conjunction of [saba\\_state](#).

This class provides the basic functionalities required to iterate over a conjunction of states of a saba.

## 7.102.2 Constructor & Destructor Documentation

**7.102.2.1** virtual spot::saba\_state\_conjunction::~saba\_state\_conjunction () [inline, virtual]

## 7.102.3 Member Function Documentation

**7.102.3.1** virtual saba\_state\_conjunction\* spot::saba\_state\_conjunction::clone () const [pure virtual]

Duplicate a [saba\\_state](#) conjunction.

Implemented in [spot::explicit\\_state\\_conjunction](#).

**7.102.3.2** virtual saba\_state\* spot::saba\_state\_conjunction::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.

### Warning

the state is allocated with new, its deletion is the responsibility of the caller.

Implemented in [spot::explicit\\_state\\_conjunction](#).

**7.102.3.3** virtual bool spot::saba\_state\_conjunction::done () const [pure virtual]

Check whether the iteration over a conjunction of states is finished.

This function should be called after any call to `first()` or `next()` and before any enquiry about the current state.

Implemented in `spot::explicit_state_conjunction`.

#### 7.102.3.4 `virtual void spot::saba_state_conjunction::first()` `[pure virtual]`

Position the iterator on the first successor of the conjunction (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::explicit_state_conjunction`.

#### 7.102.3.5 `virtual void spot::saba_state_conjunction::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::explicit_state_conjunction`.

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

- `saba/sabasucciter.hh`

## 7.103 `spot::saba_state_ptr_equal` Struct Reference

An Equivalence Relation for `saba_state*`.

This is meant to be used as a comparison functor for `Sgi hash_map` whose key are of type `saba_state*`.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

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

#### 7.103.1 Detailed Description

An Equivalence Relation for `saba_state*`.

This is meant to be used as a comparison functor for `Sgi hash_map` whose key are of type `saba_state*`. For instance here is how one could declare a map of `saba_state*`.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::saba_state*, int, spot::saba_state_ptr_hash,
              spot::saba_state_ptr_equal> seen;
```

### 7.103.2 Member Function Documentation

#### 7.103.2.1 bool spot::saba\_state\_ptr\_equal::operator() (const saba\_state \* *left*, const saba\_state \* *right*) const [inline]

References spot::saba\_state::compare().

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

- [saba/sabastate.hh](#)

## 7.104 spot::saba\_state\_ptr\_hash Struct Reference

Hash Function for saba\_state\*.

This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type saba\_state\*.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

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

#### 7.104.1 Detailed Description

Hash Function for saba\_state\*.

This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type saba\_state\*. For instance here is how one could declare a map of saba\_state\*.

```
// Remember how many times each state has been visited.
Sgi::hash_map<spot::saba_state*, int, spot::saba_state_ptr_hash,
              spot::saba_state_ptr_equal> seen;
```

### 7.104.2 Member Function Documentation

#### 7.104.2.1 size\_t spot::saba\_state\_ptr\_hash::operator() (const saba\_state \* *that*) const [inline]

References spot::saba\_state::hash().

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

- [saba/sabastate.hh](#)



## 7.105 spot::saba\_state\_ptr\_less\_than Struct Reference

Strict Weak Ordering for saba\_state\*.

This is meant to be used as a comparison functor for STL map whose key are of type saba\_state\*.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

- bool [operator\(\)](#) (const [saba\\_state](#) \*left, const [saba\\_state](#) \*right) const

#### 7.105.1 Detailed Description

Strict Weak Ordering for saba\_state\*.

This is meant to be used as a comparison functor for STL map whose key are of type saba\_state\*. For instance here is how one could declare a map of saba\_state\*.

```
// Remember how many times each state has been visited.  
std::map<spot::saba_state*, int, spot::saba_state_ptr_less_than> seen;
```

#### 7.105.2 Member Function Documentation

**7.105.2.1** bool [spot::saba\\_state\\_ptr\\_less\\_than::operator\(\)](#) (const [saba\\_state](#) \* *left*, const [saba\\_state](#) \* *right*) const [\[inline\]](#)

References [spot::saba\\_state::compare\(\)](#).

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

- [saba/sabastate.hh](#)

## 7.106 spot::saba\_state\_shared\_ptr\_equal Struct Reference

An Equivalence Relation for shared\_saba\_state (shared\_ptr<const saba\_state\*>).

This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type shared\_saba\_state.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

- bool [operator\(\)](#) ([shared\\_saba\\_state](#) left, [shared\\_saba\\_state](#) right) const

#### 7.106.1 Detailed Description

An Equivalence Relation for shared\_saba\_state (shared\_ptr<const saba\_state\*>).

This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type shared\_saba\_state. For instance here is how one could declare a map of shared\_saba\_state

```
// Remember how many times each state has been visited.
Sgi::hash_map<shared_saba_state, int,
    spot::saba_state_shared_ptr_hash,
    spot::saba_state_shared_ptr_equal> seen;
```

## 7.106.2 Member Function Documentation

### 7.106.2.1 bool spot::saba\_state\_shared\_ptr\_equal::operator() (shared\_saba\_state *left*, shared\_saba\_state *right*) const [inline]

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

- [saba/sabastate.hh](#)

## 7.107 spot::saba\_state\_shared\_ptr\_hash Struct Reference

Hash Function for shared\_saba\_state (shared\_ptr<const saba\_state\*>).

This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type shared\_saba\_state.

```
#include <saba/sabastate.hh>
```

### Public Member Functions

- `size_t operator() (shared_saba_state that) const`

### 7.107.1 Detailed Description

Hash Function for shared\_saba\_state (shared\_ptr<const saba\_state\*>).

This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type shared\_saba\_state. For instance here is how one could declare a map of shared\_saba\_state.

```
// Remember how many times each state has been visited.
Sgi::hash_map<shared_saba_state, int,
    spot::saba_state_shared_ptr_hash,
    spot::saba_state_shared_ptr_equal> seen;
```

## 7.107.2 Member Function Documentation

### 7.107.2.1 size\_t spot::saba\_state\_shared\_ptr\_hash::operator() (shared\_saba\_state *that*) const [inline]

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

- [saba/sabastate.hh](#)

## 7.108 `spot::saba_state_shared_ptr_less_than` Struct Reference

Strict Weak Ordering for `shared_saba_state` (`shared_ptr<const saba_state*>`).

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

```
#include <saba/sabastate.hh>
```

### Public Member Functions

- `bool operator() (shared_saba_state left, shared_saba_state right) const`

#### 7.108.1 Detailed Description

Strict Weak Ordering for `shared_saba_state` (`shared_ptr<const saba_state*>`).

This is meant to be used as a comparison functor for STL map whose key are of type `shared_saba_state`. For instance here is how one could declare a map of `shared_saba_state`.

```
// Remember how many times each state has been visited.
std::map<shared_saba_state, int, spot::saba_state_shared_ptr_less_than>
seen;
```

#### 7.108.2 Member Function Documentation

##### 7.108.2.1 `bool spot::saba_state_shared_ptr_less_than::operator() (shared_saba_state left, shared_saba_state right) const` [inline]

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

- [saba/sabastate.hh](#)

## 7.109 `spot::saba_succ_iterator` Class Reference

Iterate over the successors of a [saba\\_state](#).

This class provides the basic functionalities required to iterate over the successors of a state of a saba. Since transitions of an alternating automaton are defined as a boolean function with conjunctions (universal) and disjunctions (non-deterministic),.

```
#include <saba/sabasucciter.hh>
```

### Public Member Functions

- `virtual ~saba_succ_iterator ()`

#### Iteration

- `virtual void first ()=0`  
*Position the iterator on the first conjunction of successors (if any).*

- virtual void `next ()`=0  
*Jump to the next conjunction of successors (if any).*
- virtual bool `done ()` const =0  
*Check whether the iteration is finished.*

### Inspection

- virtual `saba_state_conjunction * current_conjunction ()` const =0  
*Get current conjunction of successor states.*
- virtual bdd `current_condition ()` const =0  
*Get the condition on the transition leading to this successor.*

## 7.109.1 Detailed Description

Iterate over the successors of a `saba_state`.

This class provides the basic functionalities required to iterate over the successors of a state of a saba. Since transitions of an alternating automaton are defined as a boolean function with conjunctions (universal) and disjunctions (non-deterministic),.

## 7.109.2 Constructor & Destructor Documentation

**7.109.2.1** virtual `spot::saba_succ_iterator::~saba_succ_iterator ()` [`inline`, `virtual`]

## 7.109.3 Member Function Documentation

**7.109.3.1** virtual bdd `spot::saba_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.

**7.109.3.2** virtual `saba_state_conjunction* spot::saba_succ_iterator::current_conjunction ()` const [`pure virtual`]

Get current conjunction of successor states.

**7.109.3.3** virtual bool `spot::saba_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())
    ...
```

#### 7.109.3.4 `virtual void spot::saba_succ_iterator::first()` `[pure virtual]`

Position the iterator on the first conjunction of successors (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.

#### 7.109.3.5 `virtual void spot::saba_succ_iterator::next()` `[pure virtual]`

Jump to the next conjunction of successors (if any).

##### Warning

Again, one should always call `done()` to ensure there is a successor.

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

- [saba/sabasucciter.hh](#)

## 7.110 `spot::scc_map::scc` Struct Reference

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

### Public Member Functions

- `scc` (int `index`)

### Public Attributes

- int `index`  
*Index of the SCC.*
- bdd `acc`
- `std::list< const state * >` `states`  
*States of the component.*

- `cond_set` `conds`  
*Set of conditions used in the SCC.*
- `bdd` `supp`  
*Conjunction of atomic propositions used in the SCC.*
- `bdd` `supp_rec`  
*Conjunction of atomic propositions used in the SCC.*
- `succ_type` `succ`  
*Successor SCC.*
- `bool` `trivial`  
*Trivial SCC have one state and no self-loops.*
- `bdd` `useful_acc`  
*Useful acceptance conditions.*

### 7.110.1 Constructor & Destructor Documentation

#### 7.110.1.1 `spot::scc_map::scc::scc (int index) [inline]`

### 7.110.2 Member Data Documentation

#### 7.110.2.1 `bdd` `spot::scc_map::scc::acc`

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

#### 7.110.2.2 `cond_set` `spot::scc_map::scc::conds`

Set of conditions used in the SCC.

#### 7.110.2.3 `int` `spot::scc_map::scc::index`

Index of the SCC.

#### 7.110.2.4 `std::list<const state*>` `spot::scc_map::scc::states`

States of the component.

**7.110.2.5 `succ_type spot::scc_map::scc::succ`**

Successor SCC.

**7.110.2.6 `bdd spot::scc_map::scc::supp`**

Conjunction of atomic propositions used in the SCC.

**7.110.2.7 `bdd spot::scc_map::scc::supp_rec`**

Conjunction of atomic propositions used in the SCC.

**7.110.2.8 `bool spot::scc_map::scc::trivial`**

Trivial SCC have one state and no self-loops.

**7.110.2.9 `bdd spot::scc_map::scc::useful_acc`**

Useful acceptance conditions.

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

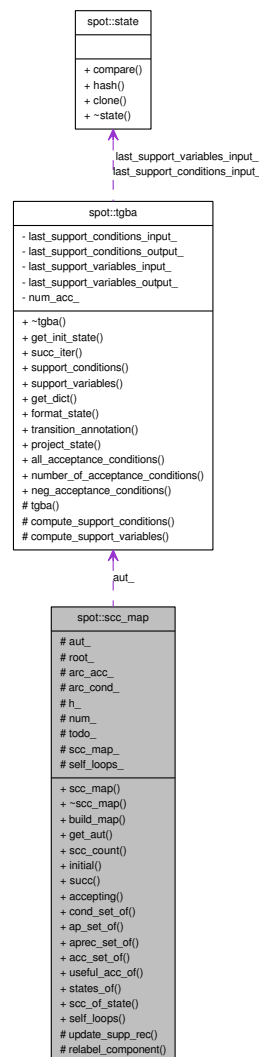
- `tgbalgorithms/scc.hh`

**7.111 `spot::scc_map` Class Reference**

Build a map of Strongly Connected components in in a TGBA.

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

Collaboration diagram for spot::scc\_map:



## Classes

- struct [scc](#)

## Public Types

- typedef std::map< unsigned, bdd > [succ\\_type](#)
- typedef std::set< bdd, [bdd\\_less\\_than](#) > [cond\\_set](#)

## Public Member Functions

- [scc\\_map](#) (const [tgba](#) \*aut)

*Constructor.*



- `~scc_map ()`
- `void build_map ()`  
*Actually compute the graph of strongly connected components.*
- `const tgba * get_aut () const`  
*Get the automaton for which the map has been constructed.*
- `unsigned scc_count () const`  
*Get the number of SCC in the automaton.*
- `unsigned initial () const`  
*Get number of the SCC containing the initial state.*
- `const succ_type & succ (unsigned n) const`  
*Successor SCCs of a SCC.*
- `bool accepting (unsigned n) const`  
*Return whether an SCC is accepting.*
- `const cond_set & cond_set_of (unsigned n) const`  
*Return the set of conditions occurring in an SCC.*
- `bdd ap_set_of (unsigned n) const`  
*Return the set of atomic properties occurring in an SCC.*
- `bdd aprec_set_of (unsigned n) const`  
*Return the set of atomic properties reachable from this SCC.*
- `bdd acc_set_of (unsigned n) const`  
*Return the set of acceptance conditions occurring in an SCC.*
- `bdd useful_acc_of (unsigned n) const`  
*Return the set of useful acceptance conditions if SCC n.*
- `const std::list< const state * > & states_of (unsigned n) const`  
*Return the set of states of an SCC.*
- `unsigned scc_of_state (const state *s) const`  
*Return the number of the SCC a state belongs too.*
- `unsigned self_loops () const`  
*Return the number of self loops in the automaton.*

### Protected Types

- `typedef std::list< scc > stack_type`
- `typedef Sgi::hash_map< const state *, int, state_ptr_hash, state_ptr_equal > hash_type`
- `typedef std::pair< const spot::state *, tgba_succ_iterator * > pair_state_iter`
- `typedef std::vector< scc > scc_map_type`

### Protected Member Functions

- bdd [update\\_supp\\_rec](#) (unsigned [state](#))
- int [relabel\\_component](#) ()

### Protected Attributes

- const [tgba](#) \* [aut\\_](#)
- [stack\\_type](#) [root\\_](#)
- std::stack< bdd > [arc\\_acc\\_](#)
- std::stack< bdd > [arc\\_cond\\_](#)
- [hash\\_type](#) [h\\_](#)
- int [num\\_](#)
- std::stack< [pair\\_state\\_iter](#) > [todo\\_](#)
- [scc\\_map\\_type](#) [scc\\_map\\_](#)
- unsigned [self\\_loops\\_](#)

#### 7.111.1 Detailed Description

Build a map of Strongly Connected components in in a TGBA.

#### 7.111.2 Member Typedef Documentation

**7.111.2.1** `typedef std::set<bdd, bdd_less_than> spot::scc_map::cond_set`

**7.111.2.2** `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal>  
spot::scc_map::hash_type [protected]`

**7.111.2.3** `typedef std::pair<const spot::state*, tgba_succ_iterator*>  
spot::scc_map::pair_state_iter [protected]`

**7.111.2.4** `typedef std::vector<scc> spot::scc_map::scc_map_type [protected]`

**7.111.2.5** `typedef std::list<scc> spot::scc_map::stack_type [protected]`

**7.111.2.6** `typedef std::map<unsigned, bdd> spot::scc_map::succ_type`

### 7.111.3 Constructor & Destructor Documentation

#### 7.111.3.1 `spot::scc_map::scc_map (const tgba * aut)`

Constructor.

This will not compute the map initially. You should call `build_map()` to do so.

#### 7.111.3.2 `spot::scc_map::~~scc_map ()`

### 7.111.4 Member Function Documentation

#### 7.111.4.1 `bdd spot::scc_map::acc_set_of (unsigned n) const`

Return the set of acceptance conditions occurring in an SCC.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.2 `bool spot::scc_map::accepting (unsigned n) const`

Return whether an SCC is accepting.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.3 `bdd spot::scc_map::ap_set_of (unsigned n) const`

Return the set of atomic properties occurring in an SCC.

##### Returns

a BDD that is a conjunction of all atomic properties occurring on the transitions in the SCC *n*.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.4 `bdd spot::scc_map::aprec_set_of (unsigned n) const`

Return the set of atomic properties reachable from this SCC.

##### Returns

a BDD that is a conjunction of all atomic properties occurring on the transitions reachable from this SCC *n*.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.5 `void spot::scc_map::build_map ()`

Actually compute the graph of strongly connected components.

#### 7.111.4.6 `const cond_set& spot::scc_map::cond_set_of (unsigned n) const`

Return the set of conditions occurring in an SCC.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.7 `const tgba* spot::scc_map::get_aut () const`

Get the automaton for which the map has been constructed.

#### 7.111.4.8 `unsigned spot::scc_map::initial () const`

Get number of the SCC containing the initial state.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.9 `int spot::scc_map::relabel_component () [protected]`

#### 7.111.4.10 `unsigned spot::scc_map::scc_count () const`

Get the number of SCC in the automaton.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.11 `unsigned spot::scc_map::scc_of_state (const state * s) const`

Return the number of the SCC a state belongs too.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.12 `unsigned spot::scc_map::self_loops () const`

Return the number of self loops in the automaton.

#### 7.111.4.13 `const std::list<const state*>& spot::scc_map::states_of (unsigned n) const`

Return the set of states of an SCC.

The states in the returned list are still owned by the `scc_map` instance. They should NOT be deleted by the client code.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.14 `const succ_type& spot::scc_map::succ (unsigned n) const`

Successor SCCs of a SCC.

##### Precondition

This should only be called once `build_map()` has run.

#### 7.111.4.15 `bdd spot::scc_map::update_supp_rec (unsigned state) [protected]`

#### 7.111.4.16 `bdd spot::scc_map::useful_acc_of (unsigned n) const`

Return the set of useful acceptance conditions if SCC *n*.

Useless acceptances conditions are always implied by other acceptances conditions. This returns all the other acceptance conditions.

### 7.111.5 Member Data Documentation

**7.111.5.1** `std::stack<bdd> spot::scc_map::arc_acc_` `[protected]`

**7.111.5.2** `std::stack<bdd> spot::scc_map::arc_cond_` `[protected]`

**7.111.5.3** `const tgba* spot::scc_map::aut_` `[protected]`

**7.111.5.4** `hash_type spot::scc_map::h_` `[protected]`

**7.111.5.5** `int spot::scc_map::num_` `[protected]`

**7.111.5.6** `stack_type spot::scc_map::root_` `[protected]`

**7.111.5.7** `scc_map_type spot::scc_map::scc_map_` `[protected]`

**7.111.5.8** `unsigned spot::scc_map::self_loops_` `[protected]`

**7.111.5.9** `std::stack<pair_state_iter> spot::scc_map::todo_` `[protected]`

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

- [tgbaalgos/scc.hh](#)

## 7.112 spot::scc\_stack Class Reference

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

### Classes

- struct [connected\\_component](#)

### 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

#### 7.112.1 Member Typedef Documentation

##### 7.112.1.1 typedef std::list<connected\_component> spot::scc\_stack::stack\_type

## 7.112.2 Member Function Documentation

### 7.112.2.1 unsigned spot::scc\_stack::clear\_rem ()

Purge all `rem` members.

#### Returns

the number of elements cleared.

### 7.112.2.2 bool spot::scc\_stack::empty () const

Is the stack empty?

### 7.112.2.3 void spot::scc\_stack::pop ()

Pop the top SCC.

### 7.112.2.4 void spot::scc\_stack::push (int *index*)

Stack a new SCC with index *index*.

### 7.112.2.5 std::list<const state\*>& spot::scc\_stack::rem ()

The `rem` member of the top SCC.

### 7.112.2.6 size\_t spot::scc\_stack::size () const

How many SCC are in stack.

### 7.112.2.7 const connected\_component& spot::scc\_stack::top () const

Access the top SCC.

### 7.112.2.8 connected\_component& spot::scc\_stack::top ()

Access the top SCC.



### 7.112.3 Member Data Documentation

#### 7.112.3.1 `stack_type spot::scc_stack::s`

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

- `tgbaalgos/gtec/sccstack.hh`

## 7.113 `spot::scc_stats` Struct Reference

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

### Public Member Functions

- `std::ostream & dump (std::ostream &out) const`

### Public Attributes

- unsigned `scc_total`  
*Total number of SCCs.*
- unsigned `acc_scc`  
*Total number of accepting SCC.*
- unsigned `dead_scc`
- unsigned `acc_paths`
- unsigned `dead_paths`
- unsigned `self_loops`
- `std::vector< bool > useless_scc_map`  
*A map of the useless SCCs.*
- bdd `useful_acc`

### 7.113.1 Member Function Documentation

#### 7.113.1.1 `std::ostream& spot::scc_stats::dump (std::ostream & out) const`

### 7.113.2 Member Data Documentation

#### 7.113.2.1 unsigned `spot::scc_stats::acc_paths`

Number of maximal accepting paths.

An path is maximal and accepting if it ends in an accepting SCC that is only dead (i.e. non accepting) successors, or no successors at all.

**7.113.2.2 `unsigned spot::scc_stats::acc_scc`**

Total number of accepting SCC.

**7.113.2.3 `unsigned spot::scc_stats::dead_paths`**

Number of paths to a terminal dead SCC.

A terminal dead SCC is a dead SCC without successors.

**7.113.2.4 `unsigned spot::scc_stats::dead_scc`**

Total number of dead SCC.

An SCC is dead if no accepting SCC is reachable from it. Note that an SCC can be neither dead nor accepting.

**7.113.2.5 `unsigned spot::scc_stats::scc_total`**

Total number of SCCs.

**7.113.2.6 `unsigned spot::scc_stats::self_loops`****7.113.2.7 `bdd spot::scc_stats::useful_acc`**

The set of useful acceptance conditions (i.e. acceptance conditions that are not always implied by other acceptance conditions).

**7.113.2.8 `std::vector<bool> spot::scc_stats::useless_scc_map`**

A map of the useless SCCs.

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

- `tgbaalgos/scc.hh`

**7.114 `spot::sccs_set` Struct Reference**

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

**Public Attributes**

- `std::set< unsigned > sccs`

- unsigned [size](#)

#### 7.114.1 Member Data Documentation

##### 7.114.1.1 `std::set<unsigned> spot::sccs_set::sccs`

##### 7.114.1.2 `unsigned spot::sccs_set::size`

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

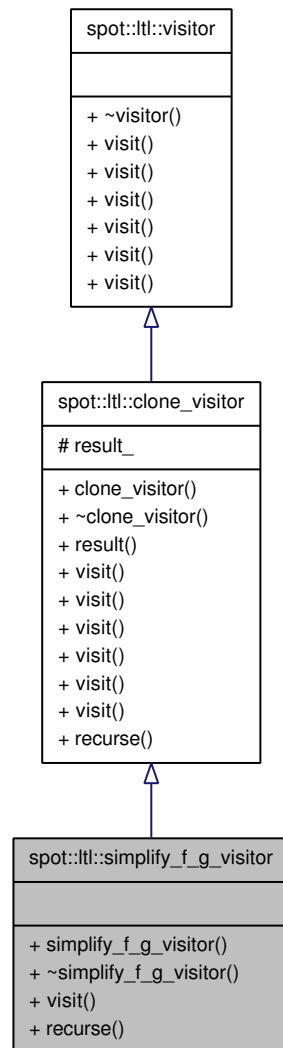
- `tgbaalgos/cutscc.hh`

#### 7.115 `spot::ltl::simplify_f_g_visitor` Class Reference

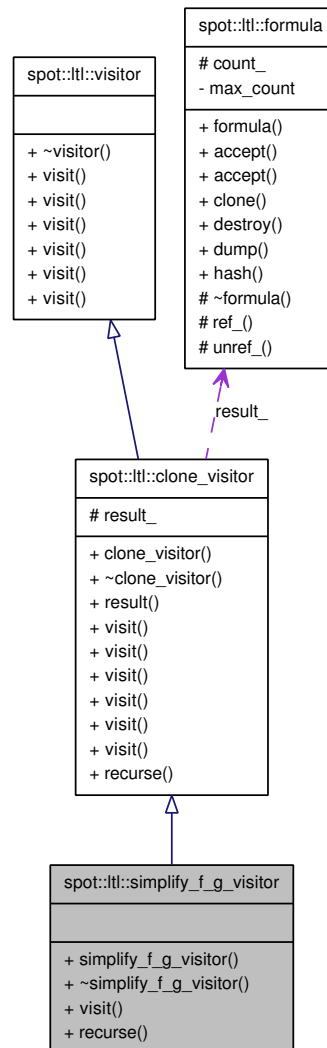
Replace `true` `U f` and `false` `R g` by `F f` and `G g`.

```
#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 (automatop *mo)`
- `void visit (multop *mo)`
- `void visit (constant *c)`

**Protected Attributes**

- `formula * result_`

**Private Types**

- `typedef clone_visitor super`

**7.115.1 Detailed Description**

Replace `true U f` and `false R g` by `F f` and `G g`. Perform the following rewriting (from left to right):

- `true U a = F a`
- `a M true = F a`
- `false R a = G a`
- `a W false = G a`

**7.115.2 Member Typedef Documentation**

**7.115.2.1** `typedef clone_visitor spot::ltl::simplify_f_g_visitor::super` `[private]`

**7.115.3 Constructor & Destructor Documentation**

**7.115.3.1** `spot::ltl::simplify_f_g_visitor::simplify_f_g_visitor ()`

**7.115.3.2** `virtual spot::ltl::simplify_f_g_visitor::~~simplify_f_g_visitor ()` `[virtual]`

**7.115.4 Member Function Documentation**

**7.115.4.1** `virtual formula* spot::ltl::simplify_f_g_visitor::recurse (formula *f)` `[virtual]`

Reimplemented from `spot::ltl::clone_visitor`.

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

**7.115.4.3** `void spot::ltl::clone_visitor::visit (constant * c)` `[virtual, inherited]`

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

**7.115.4.4** `void spot::ltl::clone_visitor::visit (multop * mo)` `[virtual, inherited]`

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

**7.115.4.5** `void spot::ltl::clone_visitor::visit (automatop * mo)` `[virtual, inherited]`

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

**7.115.4.6** `void spot::ltl::clone_visitor::visit (unop * uo)` `[virtual, inherited]`

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

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

**7.115.4.7** `void spot::ltl::clone_visitor::visit (atomic_prop * ap)` `[virtual, inherited]`

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

**7.115.4.8** `void spot::ltl::simplify_f_g_visitor::visit (binop * bo)` `[virtual]`

Reimplemented from [spot::ltl::clone\\_visitor](#).

## 7.115.5 Member Data Documentation

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

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

- [ltlvisit/simpfg.hh](#)

## 7.116 `eltlyy::slice< T, S >` Class Template Reference

Present a slice of the top of a stack.

```
#include <eltlparse/stack.hh>
```

### Public Member Functions

- [slice](#) (const S & [stack](#), unsigned int range)
- const T & [operator\[\]](#) (unsigned int i) const

### Private Attributes

- const S & [stack\\_](#)
- unsigned int [range\\_](#)

#### 7.116.1 Detailed Description

**template<class T, class S = stack<T>> class eltlyy::slice< T, S >**

Present a slice of the top of a stack.

#### 7.116.2 Constructor & Destructor Documentation

**7.116.2.1** **template<class T, class S = stack<T>> eltlyy::slice< T, S >::slice** (const S & *stack*, unsigned int *range*) [**inline**]

#### 7.116.3 Member Function Documentation

**7.116.3.1** **template<class T, class S = stack<T>> const T& eltlyy::slice< T, S >::operator[]** (unsigned int *i*) const [**inline**]

References `eltlyy::slice< T, S >::range_`, and `eltlyy::slice< T, S >::stack_`.

#### 7.116.4 Member Data Documentation

**7.116.4.1** **template<class T, class S = stack<T>> unsigned int eltlyy::slice< T, S >::range\_** [**private**]

Referenced by `eltlyy::slice< T, S >::operator[]()`.

**7.116.4.2** **template<class T, class S = stack<T>> const S& eltlyy::slice< T, S >::stack\_** [**private**]

Referenced by `eltlyy::slice< T, S >::operator[]()`.

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

- `eltlparse/stack.hh`



## 7.117 ltlyy::slice< T, S > Class Template Reference

Present a slice of the top of a stack.

```
#include <ltlparse/stack.hh>
```

### Public Member Functions

- [slice](#) (const S &[stack](#), unsigned int range)
- const T & [operator\[\]](#) (unsigned int i) const

### Private Attributes

- const S & [stack\\_](#)
- unsigned int [range\\_](#)

#### 7.117.1 Detailed Description

**template<class T, class S = stack<T>> class ltlyy::slice< T, S >**

Present a slice of the top of a stack.

#### 7.117.2 Constructor & Destructor Documentation

**7.117.2.1** **template<class T, class S = stack<T>> ltlyy::slice< T, S >::slice (const S & *stack*, unsigned int *range*) [inline]**

#### 7.117.3 Member Function Documentation

**7.117.3.1** **template<class T, class S = stack<T>> const T& ltlyy::slice< T, S >::operator[] (unsigned int *i*) const [inline]**

References [ltlyy::slice< T, S >::range\\_](#), and [ltlyy::slice< T, S >::stack\\_](#).

#### 7.117.4 Member Data Documentation

**7.117.4.1** **template<class T, class S = stack<T>> unsigned int ltlyy::slice< T, S >::range\_ [private]**

Referenced by [ltlyy::slice< T, S >::operator\[\]\(\)](#).

**7.117.4.2** `template<class T , class S = stack<T>> const S& ltlyy::slice< T, S >::stack_  
[private]`

Referenced by `ltlyy::slice< T, S >::operator[ ]()`.

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

- [ltlparse/stack.hh](#)

## 7.118 sautyy::slice< T, S > Class Template Reference

Present a slice of the top of a stack.

```
#include <sautyparse/stack.hh>
```

### Public Member Functions

- [slice](#) (const S & [stack](#), unsigned int range)
- const T & [operator\[ \]](#) (unsigned int i) const

### Private Attributes

- const S & [stack\\_](#)
- unsigned int [range\\_](#)

### 7.118.1 Detailed Description

`template<class T, class S = stack<T>> class sautyy::slice< T, S >`

Present a slice of the top of a stack.

### 7.118.2 Constructor & Destructor Documentation

**7.118.2.1** `template<class T , class S = stack<T>> sautyy::slice< T, S >::slice (const S & stack,  
unsigned int range) [inline]`

### 7.118.3 Member Function Documentation

**7.118.3.1** `template<class T , class S = stack<T>> const T& sautyy::slice< T, S >::operator[ ]  
(unsigned int i) const [inline]`

References `sautyy::slice< T, S >::range_`, and `sautyy::slice< T, S >::stack_`.

### 7.118.4 Member Data Documentation

#### 7.118.4.1 `template<class T , class S = stack<T>> unsigned int sautyy::slice< T, S >::range_ [private]`

Referenced by `sautyy::slice< T, S >::operator[]()`.

#### 7.118.4.2 `template<class T , class S = stack<T>> const S& sautyy::slice< T, S >::stack_ [private]`

Referenced by `sautyy::slice< T, S >::operator[]()`.

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

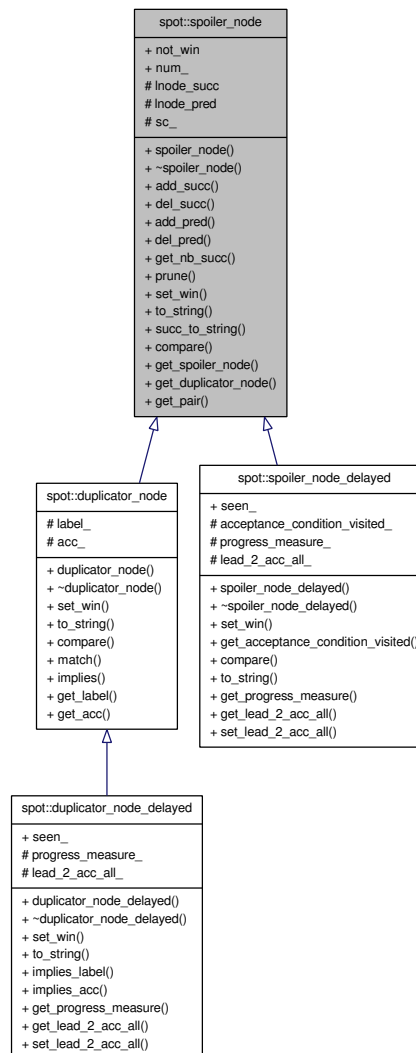
- `sautparse/stack.hh`

## 7.119 spot::spoiler\_node Class Reference

Spoiler node of parity game graph.

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

Inheritance diagram for spot::spoiler\_node:



## Public Member Functions

- `spoiler_node` (const `state` \*d\_node, const `state` \*s\_node, int num)
- virtual `~spoiler_node` ()
- bool `add_succ` (spoiler\_node \*n)

*Add a successor. Return true if n wasn't yet in the list of successor, false otherwise.*

- void `del_succ` (spoiler\_node \*n)
- virtual void `add_pred` (spoiler\_node \*n)
- virtual void `del_pred` ()
- int `get_nb_succ` ()
- bool `prune` ()
- virtual bool `set_win` ()
- virtual std::string `to_string` (const `tgba` \*a)
- virtual std::string `succ_to_string` ()

- virtual bool `compare` (`spoiler_node *n`)
- const `state * get_spoiler_node` ()
- const `state * get_duplicator_node` ()
- `state_couple * get_pair` ()

#### Public Attributes

- bool `not_win`
- int `num_`

#### Protected Attributes

- `sn_v * lnode_succ`
- `sn_v * lnode_pred`
- `state_couple * sc_`

### 7.119.1 Detailed Description

Spoiler node of parity game graph.

### 7.119.2 Constructor & Destructor Documentation

**7.119.2.1** `spot::spoiler_node::spoiler_node (const state * d_node, const state * s_node, int num)`

**7.119.2.2** `virtual spot::spoiler_node::~~spoiler_node ()` **[virtual]**

### 7.119.3 Member Function Documentation

**7.119.3.1** `virtual void spot::spoiler_node::add_pred (spoiler_node * n)` **[virtual]**

**7.119.3.2** `bool spot::spoiler_node::add_succ (spoiler_node * n)`

Add a successor. Return true if *n* wasn't yet in the list of successor, false otherwise.

**7.119.3.3** `virtual bool spot::spoiler_node::compare (spoiler_node * n)` **[virtual]**

Reimplemented in `spot::duplicator_node`, and `spot::spoiler_node_delayed`.

**7.119.3.4** `virtual void spot::spoiler_node::del_pred ()` **[virtual]**

**7.119.3.5** `void spot::spoiler_node::del_succ (spoiler_node * n)`

**7.119.3.6** `const state* spot::spoiler_node::get_duplicator_node ()`

**7.119.3.7** `int spot::spoiler_node::get_nb_succ ()`

**7.119.3.8** `state_couple* spot::spoiler_node::get_pair ()`

**7.119.3.9** `const state* spot::spoiler_node::get_spoiler_node ()`

**7.119.3.10** `bool spot::spoiler_node::prune ()`

**7.119.3.11** `virtual bool spot::spoiler_node::set_win ()` **[virtual]**

Reimplemented in [spot::duplicator\\_node](#), [spot::spoiler\\_node\\_delayed](#), and [spot::duplicator\\_node\\_delayed](#).

**7.119.3.12** `virtual std::string spot::spoiler_node::succ_to_string ()` **[virtual]**

**7.119.3.13** `virtual std::string spot::spoiler_node::to_string (const tgba * a)` **[virtual]**

Reimplemented in [spot::duplicator\\_node](#), [spot::spoiler\\_node\\_delayed](#), and [spot::duplicator\\_node\\_delayed](#).

### 7.119.4 Member Data Documentation

**7.119.4.1** `sn_v* spot::spoiler_node::lnode_pred` `[protected]`

**7.119.4.2** `sn_v* spot::spoiler_node::lnode_succ` `[protected]`

**7.119.4.3** `bool spot::spoiler_node::not_win`

**7.119.4.4** `int spot::spoiler_node::num_`

**7.119.4.5** `state_couple* spot::spoiler_node::sc_` `[protected]`

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

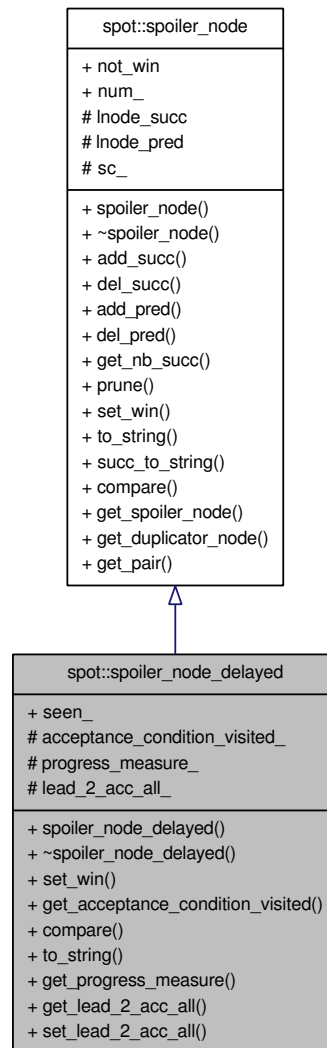
- `tgbaalgos/reductgba_sim.hh`

## 7.120 `spot::spoiler_node_delayed` Class Reference

Spoiler node of parity game graph for delayed simulation.

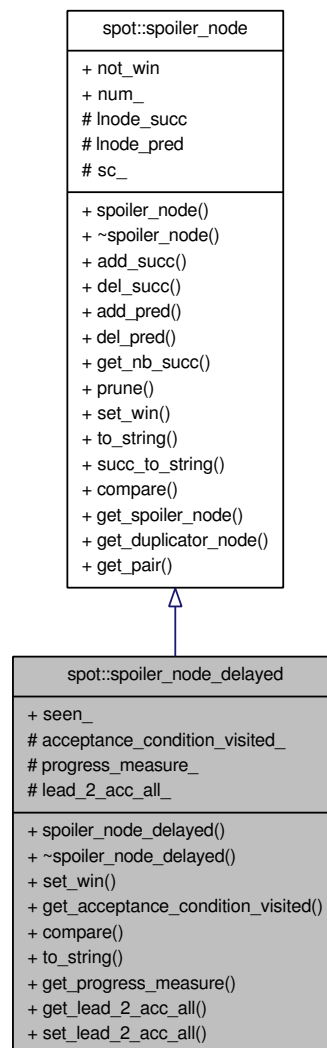
```
#include <tgbaalgos/reductgba_sim.hh>
```

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_`

#### 7.120.1 Detailed Description

Spoiler node of parity game graph for delayed simulation.

#### 7.120.2 Constructor & Destructor Documentation

**7.120.2.1** `spot::spoiler_node_delayed::spoiler_node_delayed (const state * d_node, const state * s_node, bdd a, int num)`

**7.120.2.2** `spot::spoiler_node_delayed::~spoiler_node_delayed ()`

#### 7.120.3 Member Function Documentation

**7.120.3.1** virtual void `spot::spoiler_node::add_pred` (`spoiler_node * n`) [`virtual`, `inherited`]

**7.120.3.2** `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 eitherwise.

**7.120.3.3** `virtual bool spot::spoiler_node_delayed::compare (spoiler_node * n)` `[virtual]`

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

**7.120.3.4** `virtual void spot::spoiler_node::del_pred ()` `[virtual, inherited]`**7.120.3.5** `void spot::spoiler_node::del_succ (spoiler_node * n)` `[inherited]`**7.120.3.6** `bdd spot::spoiler_node_delayed::get_acceptance_condition_visited ()` `const`**7.120.3.7** `const state* spot::spoiler_node::get_duplicator_node ()` `[inherited]`**7.120.3.8** `bool spot::spoiler_node_delayed::get_lead_2_acc_all ()`**7.120.3.9** `int spot::spoiler_node::get_nb_succ ()` `[inherited]`**7.120.3.10** `state_couple* spot::spoiler_node::get_pair ()` `[inherited]`**7.120.3.11** `int spot::spoiler_node_delayed::get_progress_measure ()` `const`**7.120.3.12** `const state* spot::spoiler_node::get_spoiler_node ()` `[inherited]`

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

**7.120.3.14** `bool spot::spoiler_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

**7.120.3.15** `bool spot::spoiler_node_delayed::set_win ()` `[virtual]`

Return true if the `progress_measure` has changed.

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

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

**7.120.3.17** `virtual std::string spot::spoiler_node_delayed::to_string (const tgba * a)` `[virtual]`

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

## 7.120.4 Member Data Documentation

**7.120.4.1** `bdd spot::spoiler_node_delayed::acceptance_condition_visited_` `[protected]`

a Bdd for retain all the acceptance condition that a node has visited.

**7.120.4.2** `bool spot::spoiler_node_delayed::lead_2_acc_all_` `[protected]`

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

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

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

7.120.4.6 int spot::spoiler\_node::num\_ [inherited]

7.120.4.7 int spot::spoiler\_node\_delayed::progress\_measure\_ [protected]

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

7.120.4.9 bool spot::spoiler\_node\_delayed::seen\_

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

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

## 7.121 sautyy::stack< T, S > Class Template Reference

```
#include <sautparse/stack.hh>
```

### Public Types

- typedef S::reverse\_iterator [iterator](#)
- typedef S::const\_reverse\_iterator [const\\_iterator](#)

### Public Member Functions

- [stack](#) ()
- [stack](#) (unsigned int n)
- T & [operator\[\]](#) (unsigned int i)
- const T & [operator\[\]](#) (unsigned int i) const
- void [push](#) (const T &t)
- void [pop](#) (unsigned int n=1)
- unsigned int [height](#) () const
- [const\\_iterator](#) [begin](#) () const
- [const\\_iterator](#) [end](#) () const

### Private Attributes

- S [seq\\_](#)

```
template<class T, class S = std::deque<T>> class sautyy::stack< T, S >
```

#### 7.121.1 Member Typedef Documentation

**7.121.1.1** `template<class T , class S = std::deque<T>> typedef S::const_reverse_iterator sautyy::stack< T, S >::const_iterator`

**7.121.1.2** `template<class T , class S = std::deque<T>> typedef S::reverse_iterator sautyy::stack< T, S >::iterator`

#### 7.121.2 Constructor & Destructor Documentation

**7.121.2.1** `template<class T , class S = std::deque<T>> sautyy::stack< T, S >::stack () [inline]`

**7.121.2.2** `template<class T , class S = std::deque<T>> sautyy::stack< T, S >::stack (unsigned int n) [inline]`

#### 7.121.3 Member Function Documentation

**7.121.3.1** `template<class T , class S = std::deque<T>> const_iterator sautyy::stack< T, S >::begin () const [inline]`

References sautyy::stack< T, S >::seq\_.

**7.121.3.2** `template<class T , class S = std::deque<T>> const_iterator sautyy::stack< T, S >::end () const [inline]`

References sautyy::stack< T, S >::seq\_.

**7.121.3.3** `template<class T , class S = std::deque<T>> unsigned int sautyy::stack< T, S >::height () const [inline]`

References sautyy::stack< T, S >::seq\_.

**7.121.3.4** `template<class T , class S = std::deque<T>> const T& sautyy::stack< T, S >::operator[] (unsigned int i) const [inline]`

References `sautyy::stack< T, S >::seq_`.

**7.121.3.5** `template<class T , class S = std::deque<T>> T& sautyy::stack< T, S >::operator[] (unsigned int i) [inline]`

References `sautyy::stack< T, S >::seq_`.

**7.121.3.6** `template<class T , class S = std::deque<T>> void sautyy::stack< T, S >::pop (unsigned int n = 1) [inline]`

References `sautyy::stack< T, S >::seq_`.

**7.121.3.7** `template<class T , class S = std::deque<T>> void sautyy::stack< T, S >::push (const T & t) [inline]`

References `sautyy::stack< T, S >::seq_`.

## 7.121.4 Member Data Documentation

**7.121.4.1** `template<class T , class S = std::deque<T>> S sautyy::stack< T, S >::seq_ [private]`

Referenced by `sautyy::stack< T, S >::begin()`, `sautyy::stack< T, S >::end()`, `sautyy::stack< T, S >::height()`, `sautyy::stack< T, S >::operator[]()`, `sautyy::stack< T, S >::pop()`, and `sautyy::stack< T, S >::push()`.

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

- `sautparse/stack.hh`

## 7.122 `ltlty::stack< T, S >` Class Template Reference

```
#include <ltlparse/stack.hh>
```

### Public Types

- `typedef S::reverse_iterator iterator`
- `typedef S::const_reverse_iterator const_iterator`

**Public Member Functions**

- [stack](#) ()
- [stack](#) (unsigned int n)
- T & [operator\[\]](#) (unsigned int i)
- const T & [operator\[\]](#) (unsigned int i) const
- void [push](#) (const T &t)
- void [pop](#) (unsigned int n=1)
- unsigned int [height](#) () const
- [const\\_iterator begin](#) () const
- [const\\_iterator end](#) () const

**Private Attributes**

- S [seq\\_](#)

**template<class T, class S = std::deque<T>> class ltlyy::stack< T, S >**

**7.122.1 Member Typedef Documentation**

**7.122.1.1** **template<class T , class S = std::deque<T>> typedef S::const\_reverse\_iterator ltlyy::stack< T, S >::const\_iterator**

**7.122.1.2** **template<class T , class S = std::deque<T>> typedef S::reverse\_iterator ltlyy::stack< T, S >::iterator**

**7.122.2 Constructor & Destructor Documentation**

**7.122.2.1** **template<class T , class S = std::deque<T>> ltlyy::stack< T, S >::stack () [inline]**

**7.122.2.2** **template<class T , class S = std::deque<T>> ltlyy::stack< T, S >::stack (unsigned int n) [inline]**

**7.122.3 Member Function Documentation**

**7.122.3.1** **template<class T , class S = std::deque<T>> const\_iterator ltlyy::stack< T, S >::begin () const [inline]**

References [ltlyy::stack< T, S >::seq\\_](#).



**7.122.3.2** `template<class T, class S = std::deque<T>> const_iterator ltlyy::stack< T, S >::end ()  
const [inline]`

References `ltlyy::stack< T, S >::seq_`.

**7.122.3.3** `template<class T, class S = std::deque<T>> unsigned int ltlyy::stack< T, S >::height  
() const [inline]`

References `ltlyy::stack< T, S >::seq_`.

**7.122.3.4** `template<class T, class S = std::deque<T>> const T& ltlyy::stack< T, S >::operator[]  
(unsigned int i) const [inline]`

References `ltlyy::stack< T, S >::seq_`.

**7.122.3.5** `template<class T, class S = std::deque<T>> T& ltlyy::stack< T, S >::operator[]  
(unsigned int i) [inline]`

References `ltlyy::stack< T, S >::seq_`.

**7.122.3.6** `template<class T, class S = std::deque<T>> void ltlyy::stack< T, S >::pop (unsigned  
int n = 1) [inline]`

References `ltlyy::stack< T, S >::seq_`.

**7.122.3.7** `template<class T, class S = std::deque<T>> void ltlyy::stack< T, S >::push (const T  
& t) [inline]`

References `ltlyy::stack< T, S >::seq_`.

## 7.122.4 Member Data Documentation

**7.122.4.1** `template<class T, class S = std::deque<T>> S ltlyy::stack< T, S >::seq_ [private]`

Referenced by `ltlyy::stack< T, S >::begin()`, `ltlyy::stack< T, S >::end()`, `ltlyy::stack< T, S >::height()`, `ltlyy::stack< T, S >::operator[]()`, `ltlyy::stack< T, S >::pop()`, and `ltlyy::stack< T, S >::push()`.

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

- [ltlparse/stack.hh](#)

## 7.123 `eltlyy::stack< T, S >` Class Template Reference

```
#include <eltlparse/stack.hh>
```

### Public Types

- typedef `S::reverse_iterator` `iterator`
- typedef `S::const_reverse_iterator` `const_iterator`

### Public Member Functions

- `stack` ()
- `stack` (unsigned int n)
- `T & operator[]` (unsigned int i)
- `const T & operator[]` (unsigned int i) const
- void `push` (const T &t)
- void `pop` (unsigned int n=1)
- unsigned int `height` () const
- `const_iterator begin` () const
- `const_iterator end` () const

### Private Attributes

- `S seq_`

```
template<class T, class S = std::deque<T>> class eltlyy::stack< T, S >
```

#### 7.123.1 Member Typedef Documentation

**7.123.1.1** `template<class T , class S = std::deque<T>> typedef S::const_reverse_iterator eltlyy::stack< T, S >::const_iterator`

**7.123.1.2** `template<class T , class S = std::deque<T>> typedef S::reverse_iterator eltlyy::stack< T, S >::iterator`

#### 7.123.2 Constructor & Destructor Documentation

**7.123.2.1** `template<class T , class S = std::deque<T>> eltlyy::stack< T, S >::stack () [inline]`

**7.123.2.2** `template<class T , class S = std::deque<T>> eltlyy::stack< T, S >::stack (unsigned int n) [inline]`

### 7.123.3 Member Function Documentation

**7.123.3.1** `template<class T , class S = std::deque<T>> const_iterator eltlyy::stack< T, S >::begin () const [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.2** `template<class T , class S = std::deque<T>> const_iterator eltlyy::stack< T, S >::end () const [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.3** `template<class T , class S = std::deque<T>> unsigned int eltlyy::stack< T, S >::height () const [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.4** `template<class T , class S = std::deque<T>> const T& eltlyy::stack< T, S >::operator[] (unsigned int i) const [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.5** `template<class T , class S = std::deque<T>> T& eltlyy::stack< T, S >::operator[] (unsigned int i) [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.6** `template<class T , class S = std::deque<T>> void eltlyy::stack< T, S >::pop (unsigned int n = 1) [inline]`

References `eltlyy::stack< T, S >::seq_`.

**7.123.3.7** `template<class T, class S = std::deque<T>> void eltlyy::stack< T, S >::push (const T & t) [inline]`

References `eltlyy::stack< T, S >::seq_`.

#### 7.123.4 Member Data Documentation

**7.123.4.1** `template<class T, class S = std::deque<T>> S eltlyy::stack< T, S >::seq_ [private]`

Referenced by `eltlyy::stack< T, S >::begin()`, `eltlyy::stack< T, S >::end()`, `eltlyy::stack< T, S >::height()`, `eltlyy::stack< T, S >::operator[]()`, `eltlyy::stack< T, S >::pop()`, and `eltlyy::stack< T, S >::push()`.

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

- `eltlparse/stack.hh`

## 7.124 spot::evtgba\_explicit::state Struct Reference

```
#include <evtgba/explicit.hh>
```

### Public Attributes

- `transition_list` in
- `transition_list` out

#### 7.124.1 Member Data Documentation

**7.124.1.1** `transition_list spot::evtgba_explicit::state::in`

**7.124.1.2** `transition_list spot::evtgba_explicit::state::out`

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

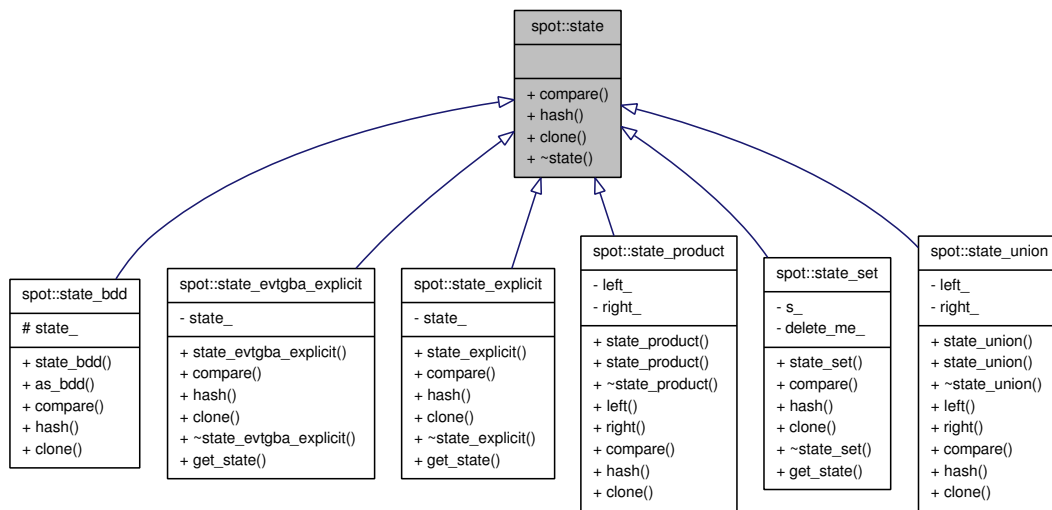
- `evtgba/explicit.hh`

## 7.125 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](#) ()

### 7.125.1 Detailed Description

Abstract class for states.

### 7.125.2 Constructor & Destructor Documentation

#### 7.125.2.1 virtual spot::state::~~state () [inline, virtual]

### 7.125.3 Member Function Documentation

#### 7.125.3.1 virtual state\* spot::state::clone () const [pure virtual]

Duplicate a state.

Implemented in [spot::state\\_evtgba\\_explicit](#), [spot::state\\_bdd](#), [spot::state\\_set](#), [spot::state\\_explicit](#), [spot::state\\_product](#), and [spot::state\\_union](#).

### 7.125.3.2 `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\\_evtgba\\_explicit](#), [spot::state\\_bdd](#), [spot::state\\_set](#), [spot::state\\_explicit](#), [spot::state\\_product](#), and [spot::state\\_union](#).

Referenced by [spot::state\\_ptr\\_equal::operator\(\)\(\)](#), and [spot::state\\_ptr\\_less\\_than::operator\(\)\(\)](#).

### 7.125.3.3 `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\\_set](#), [spot::state\\_explicit](#), [spot::state\\_product](#), and [spot::state\\_union](#).

Referenced by [spot::state\\_ptr\\_hash::operator\(\)\(\)](#).

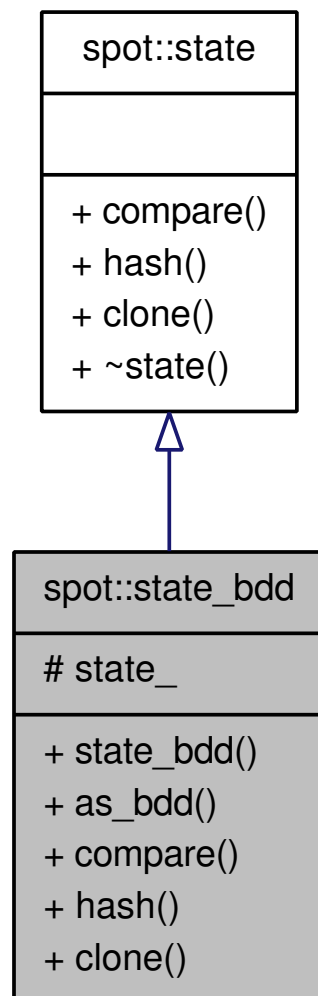
The documentation for this class was generated from the following file:

- [tgba/state.hh](#)

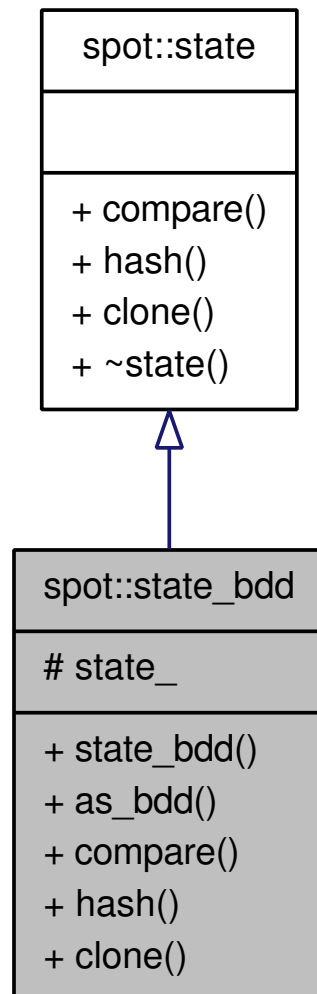
## 7.126 `spot::state_bdd` Class Reference

```
#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.*

#### 7.126.1 Detailed Description

A state whose representation is a BDD.

#### 7.126.2 Constructor & Destructor Documentation

##### 7.126.2.1 `spot::state_bdd::state_bdd (bdd s) [inline]`

#### 7.126.3 Member Function Documentation

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

Return the BDD part of the state.

References `state_`.

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

Duplicate a state.

Implements [spot::state](#).

##### 7.126.3.3 `virtual int spot::state_bdd::compare (const state * other) const [virtual]`

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

### See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

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

Hash a state.

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

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

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

Implements `spot::state`.

### 7.126.4 Member Data Documentation

#### 7.126.4.1 `bdd spot::state_bdd::state_` `[protected]`

BDD representation of the state.

Referenced by `as_bdd()`.

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

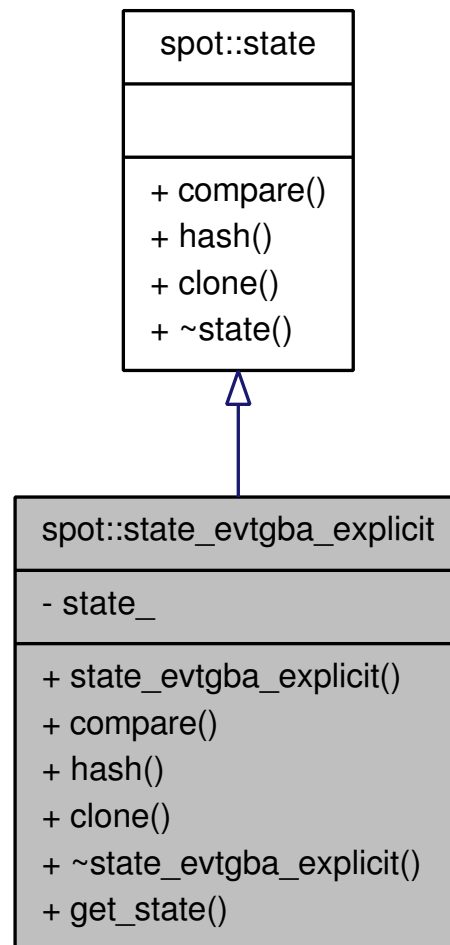
- `tgba/statebdd.hh`

### 7.127 `spot::state_evtgba_explicit` Class Reference

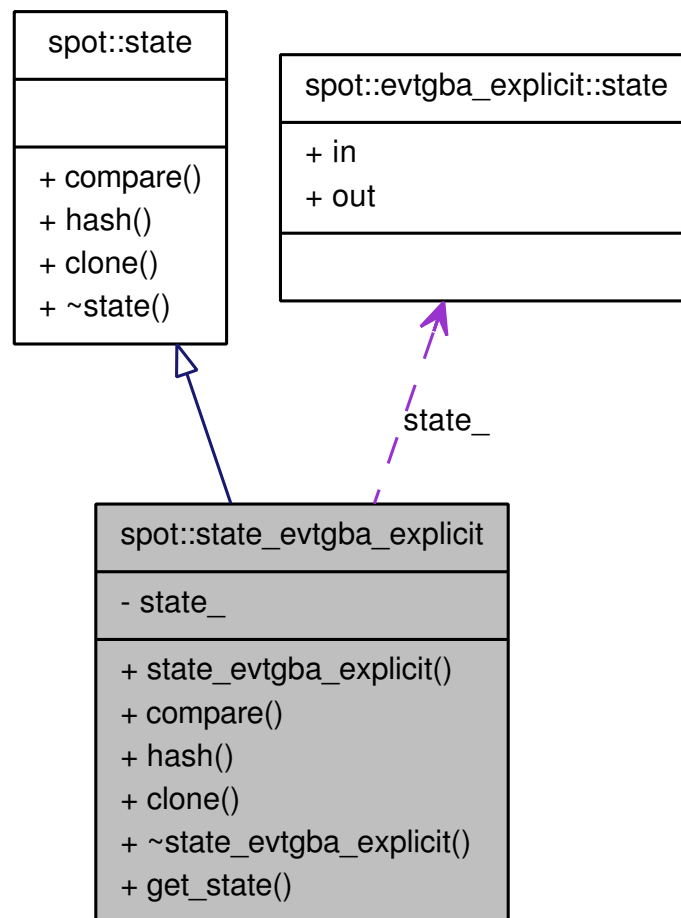
States used by `spot::tgba_evtgba_explicit`.

```
#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  
*Compares two states (that come from the same automaton).*
- 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

### Private Attributes

- const [evtgba\\_explicit::state](#) \* [state\\_](#)

### 7.127.1 Detailed Description

States used by `spot::tgba_evtgba_explicit`.

### 7.127.2 Constructor & Destructor Documentation

**7.127.2.1** `spot::state_evtgba_explicit::state_evtgba_explicit (const evtgba_explicit::state * s)`  
[**inline**]

**7.127.2.2** `virtual spot::state_evtgba_explicit::~~state_evtgba_explicit ()` [**inline**, **virtual**]

### 7.127.3 Member Function Documentation

**7.127.3.1** `virtual state_evtgba_explicit* spot::state_evtgba_explicit::clone () const` [**virtual**]

Duplicate a state.

Implements [spot::state](#).

**7.127.3.2** `virtual int spot::state_evtgba_explicit::compare (const spot::state * other) const`  
[**virtual**]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

**7.127.3.3** `const evtgba_explicit::state* spot::state_evtgba_explicit::get_state () const`

**7.127.3.4** `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`.

#### 7.127.4 Member Data Documentation

##### 7.127.4.1 `const evtgba_explicit::state* spot::state_evtgba_explicit::state_ [private]`

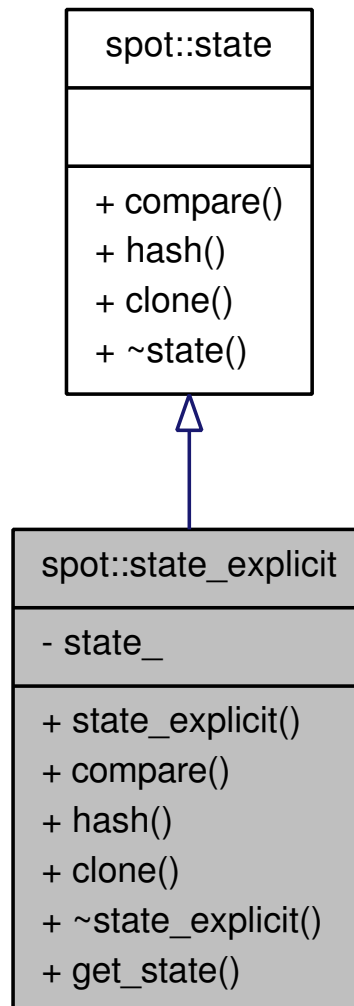
The documentation for this class was generated from the following file:

- `evtgba/explicit.hh`

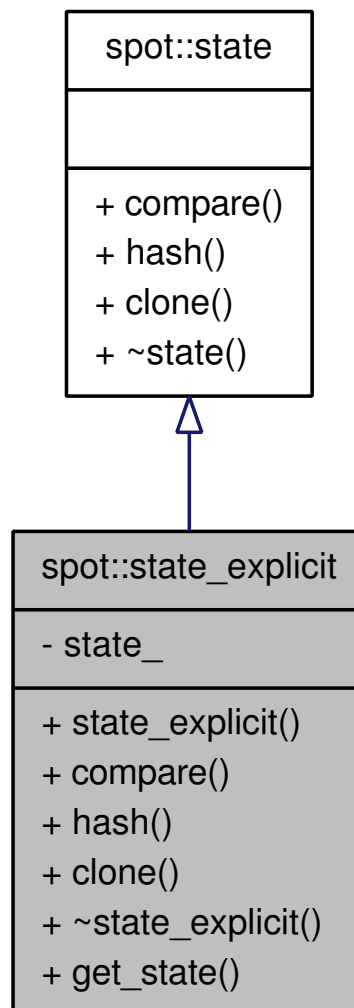
## 7.128 `spot::state_explicit` Class Reference

```
#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  
*Compares two states (that come from the same automaton).*
- 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



### Private Attributes

- const [tgba\\_explicit::state](#) \* `state_`

### 7.128.1 Detailed Description

States used by [spot::tgba\\_explicit](#).

### 7.128.2 Constructor & Destructor Documentation

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

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

### 7.128.3 Member Function Documentation

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

Duplicate a state.

Implements [spot::state](#).

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

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

### See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

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

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::get_label()`.

#### 7.128.3.4 `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 as long as one of these states exists. So it's OK to use a `spot::state` as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

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

Implements `spot::state`.

#### 7.128.4 Member Data Documentation

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

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

- `tgba/tgbaexplicit.hh`

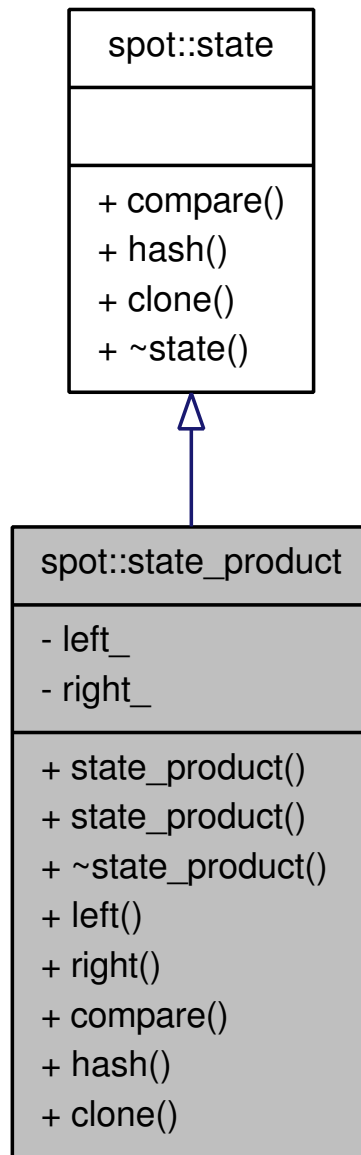
## 7.129 `spot::state_product` Class Reference

A state for `spot::tgba_product`.

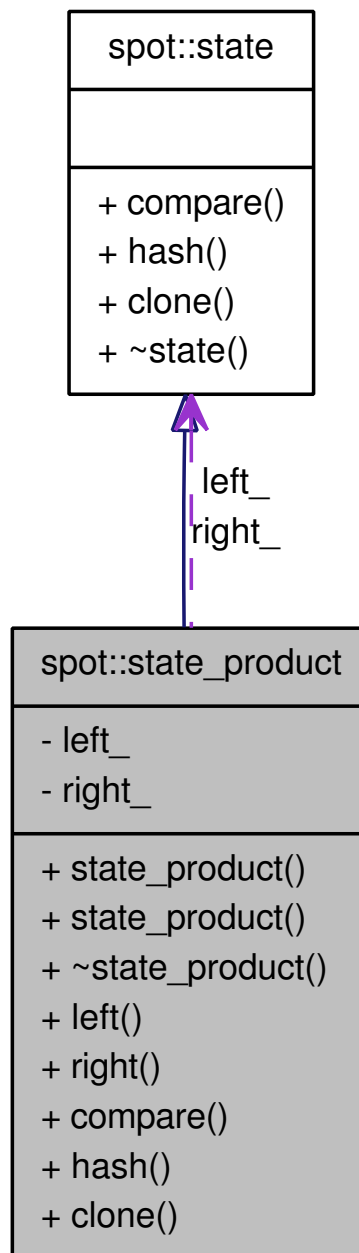
This state is in fact a pair of state: the state from the left automaton and that of the right.

```
#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.*

#### 7.129.1 Detailed Description

A state for `spot::tgba_product`.

This state is in fact a pair of state: the state from the left automaton and that of the right.

#### 7.129.2 Constructor & Destructor Documentation

##### 7.129.2.1 `spot::state_product::state_product (state * left, state * right)` [`inline`]

Constructor.

### Parameters

*left* The state from the left automaton.

*right* The state from the right automaton. These states are acquired by `spot::state_product`, and will be deleted on destruction.

##### 7.129.2.2 `spot::state_product::state_product (const state_product & o)`

Copy constructor.

##### 7.129.2.3 `virtual spot::state_product::~~state_product()` [`virtual`]

### 7.129.3 Member Function Documentation

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

Duplicate a state.

Implements [spot::state](#).

#### 7.129.3.2 `virtual int spot::state_product::compare (const state * other) const [virtual]`

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

#### 7.129.3.3 `virtual size_t spot::state_product::hash () const [virtual]`

Hash a state.

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

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

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

Implements [spot::state](#).

#### 7.129.3.4 `state* spot::state_product::left () const [inline]`

References `left_`.

#### 7.129.3.5 `state* spot::state_product::right () const [inline]`

References `right_`.

### 7.129.4 Member Data Documentation

#### 7.129.4.1 state\* spot::state\_product::left\_ [private]

State from the left automaton.

Referenced by left().

#### 7.129.4.2 state\* spot::state\_product::right\_ [private]

State from the right automaton.

Referenced by right().

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

- [tgba/tgbaproduct.hh](#)

## 7.130 spot::state\_ptr\_equal Struct Reference

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\*.

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

### Public Member Functions

- bool operator() (const state \*left, const state \*right) const

#### 7.130.1 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;
```

### 7.130.2 Member Function Documentation

#### 7.130.2.1 bool spot::state\_ptr\_equal::operator() (const state \* left, const state \* right) const [inline]

References spot::state::compare().

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

- [tgba/state.hh](#)

## 7.131 spot::state\_ptr\_hash Struct Reference

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\*.

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

### Public Member Functions

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

#### 7.131.1 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;
```

#### 7.131.2 Member Function Documentation

##### 7.131.2.1 size\_t spot::state\_ptr\_hash::operator() (const state \*that) const [inline]

References spot::state::hash().

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

- [tgba/state.hh](#)

## 7.132 spot::state\_ptr\_less\_than Struct Reference

Strict Weak Ordering for state\*.

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

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

### Public Member Functions

- `bool operator() (const state *left, const state *right) const`

#### 7.132.1 Detailed Description

Strict Weak Ordering for state\*.

This is meant to be used as a comparison functor for STL map whose key are of type state\*. For instance here is how one could declare a map of state\*.

```
// Remember how many times each state has been visited.
std::map<spot::state*, int, spot::state_ptr_less_than> seen;
```



### 7.132.2 Member Function Documentation

#### 7.132.2.1 bool spot::state\_ptr\_less\_than::operator() (const state \* *left*, const state \* *right*) const [inline]

References spot::state::compare().

The documentation for this struct was generated from the following file:

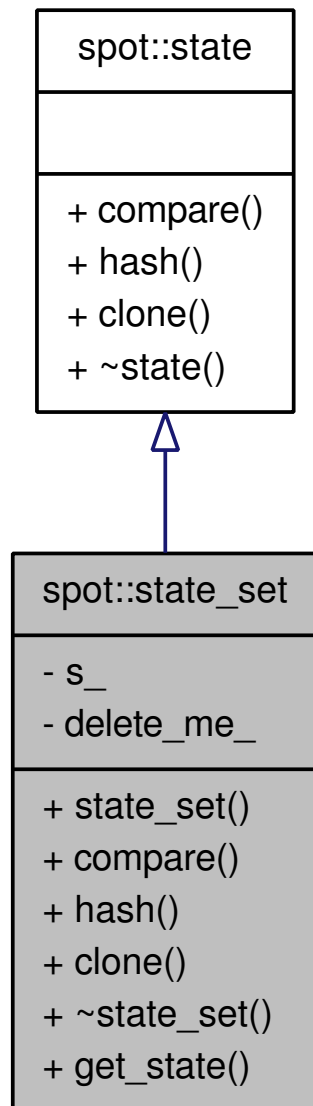
- [tgba/state.hh](#)

### 7.133 spot::state\_set Class Reference

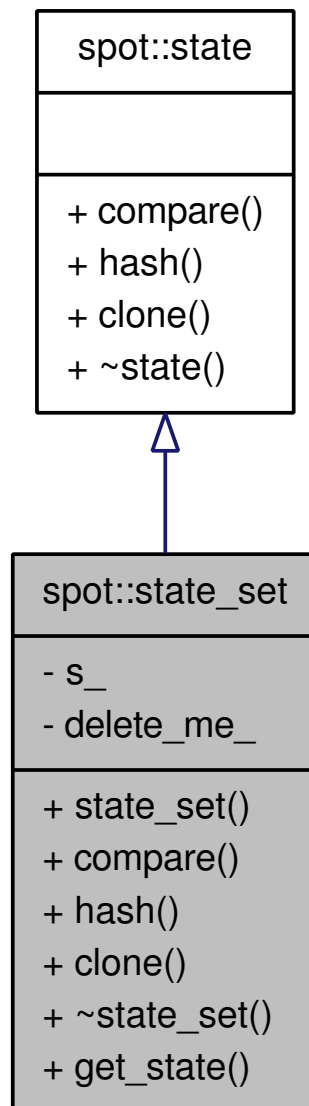
Set of states deriving from [spot::state](#).

```
#include <tgba/taatgba.hh>
```

Inheritance diagram for spot::state\_set:



Collaboration diagram for spot::state\_set:



### Public Member Functions

- `state_set` (const `taa_tgba::state_set` \*s, bool delete\_me=false)
- virtual int `compare` (const `spot::state` \*) const  
*Compares two states (that come from the same automaton).*
- virtual size\_t `hash` () const  
*Hash a state.*
- virtual `state_set` \* `clone` () const  
*Duplicate a state.*
- virtual `~state_set` ()

- const [taa\\_tgba::state\\_set](#) \* `get_state` () const

#### Private Attributes

- const [taa\\_tgba::state\\_set](#) \* `s_`
- bool `delete_me_`

#### 7.133.1 Detailed Description

Set of states deriving from [spot::state](#).

#### 7.133.2 Constructor & Destructor Documentation

**7.133.2.1** `spot::state_set::state_set (const taa_tgba::state_set * s, bool delete_me = false)`  
[inline]

**7.133.2.2** `virtual spot::state_set::~~state_set ()` [inline, virtual]

References `delete_me_`, and `s_`.

#### 7.133.3 Member Function Documentation

**7.133.3.1** `virtual state_set* spot::state_set::clone () const` [virtual]

Duplicate a state.

Implements [spot::state](#).

**7.133.3.2** `virtual int spot::state_set::compare (const spot::state * other) const` [virtual]

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

#### See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

**7.133.3.3 `const taa_tgba::state_set* spot::state_set::get_state () const`**

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::format_state()`.

**7.133.3.4 `virtual size_t spot::state_set::hash () const` `[virtual]`**

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in `compare()`'s sense) for only has one of these states exists. So it's OK to use a `spot::state` as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements `spot::state`.

**7.133.4 Member Data Documentation****7.133.4.1 `bool spot::state_set::delete_me_` `[private]`**

Referenced by `~state_set()`.

**7.133.4.2 `const taa_tgba::state_set* spot::state_set::s_` `[private]`**

Referenced by `~state_set()`.

The documentation for this class was generated from the following file:

- `tgba/taatgba.hh`

**7.134 `spot::state_shared_ptr_equal` Struct Reference**

An Equivalence Relation for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a comparison functor for `Sgi hash_map` whose key are of type `shared_state`.

```
#include <tgba/state.hh>
```

**Public Member Functions**

- `bool operator() (shared_state left, shared_state right) const`

### 7.134.1 Detailed Description

An Equivalence Relation for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a comparison functor for `Sgi hash_map` whose key are of type `shared_state`. For instance here is how one could declare a map of `shared_state`

```
// Remember how many times each state has been visited.
Sgi::hash_map<shared_state, int,
             spot::state_shared_ptr_hash,
             spot::state_shared_ptr_equal> seen;
```

### 7.134.2 Member Function Documentation

#### 7.134.2.1 `bool spot::state_shared_ptr_equal::operator() (shared_state left, shared_state right) const [inline]`

The documentation for this struct was generated from the following file:

- [tgba/state.hh](#)

## 7.135 spot::state\_shared\_ptr\_hash Struct Reference

Hash Function for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a hash functor for `Sgi's hash_map` whose key are of type `shared_state`.

```
#include <tgba/state.hh>
```

### Public Member Functions

- `size_t operator() (shared_state that) const`

### 7.135.1 Detailed Description

Hash Function for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a hash functor for `Sgi's hash_map` whose key are of type `shared_state`. For instance here is how one could declare a map of `shared_state`.

```
// Remember how many times each state has been visited.
Sgi::hash_map<shared_state, int,
             spot::state_shared_ptr_hash,
             spot::state_shared_ptr_equal> seen;
```

### 7.135.2 Member Function Documentation

#### 7.135.2.1 `size_t spot::state_shared_ptr_hash::operator() (shared_state that) const [inline]`

The documentation for this struct was generated from the following file:

- [tgba/state.hh](#)

## 7.136 `spot::state_shared_ptr_less_than` Struct Reference

Strict Weak Ordering for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a comparison functor for STL map whose key are of type `shared_state`.

```
#include <tgba/state.hh>
```

### Public Member Functions

- `bool operator() (shared_state left, shared_state right) const`

#### 7.136.1 Detailed Description

Strict Weak Ordering for `shared_state` (`shared_ptr<const state*>`).

This is meant to be used as a comparison functor for STL map whose key are of type `shared_state`. For instance here is how one could declare a map of `shared_state`.

```
// Remember how many times each state has been visited.  
std::map<shared_state, int, spot::state_shared_ptr_less_than> seen;
```

#### 7.136.2 Member Function Documentation

**7.136.2.1** `bool spot::state_shared_ptr_less_than::operator() (shared_state left, shared_state right) const [inline]`

The documentation for this struct was generated from the following file:

- `tgba/state.hh`

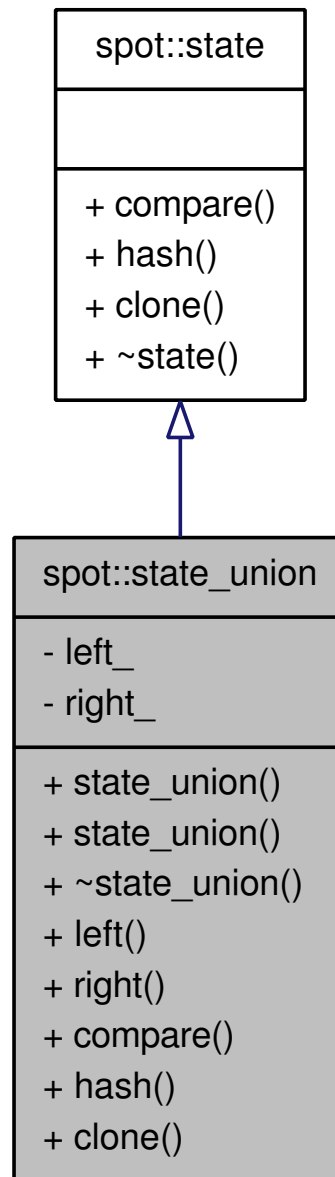
## 7.137 `spot::state_union` Class Reference

A state for `spot::tgba_union`.

This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.

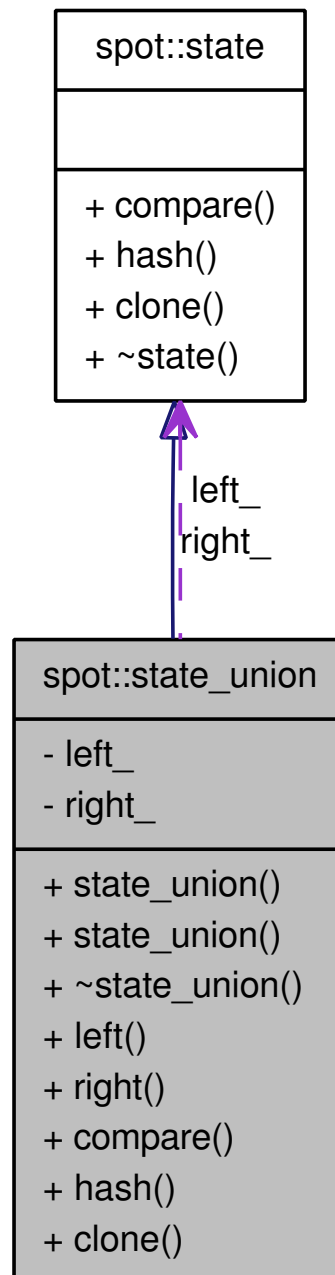
```
#include <tgba/tgbaunion.hh>
```

Inheritance diagram for spot::state\_union:





Collaboration diagram for spot::state\_union:



### Public Member Functions

- `state_union` (`state` \*left, `state` \*right)  
*Constructor.*
- `state_union` (const `state_union` &o)  
*Copy constructor.*

- virtual `~state_union ()`
- `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_union * clone () const`  
*Duplicate a state.*

### Private Attributes

- `state * left_`
- `state * right_`

#### 7.137.1 Detailed Description

A state for `spot::tgba_union`.

This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.

#### 7.137.2 Constructor & Destructor Documentation

##### 7.137.2.1 `spot::state_union::state_union (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_union`, and will be deleted on destruction.

##### 7.137.2.2 `spot::state_union::state_union (const state_union & o)`

Copy constructor.

##### 7.137.2.3 `virtual spot::state_union::~~state_union () [virtual]`

### 7.137.3 Member Function Documentation

#### 7.137.3.1 `virtual state_union* spot::state_union::clone () const [virtual]`

Duplicate a state.

Implements [spot::state](#).

#### 7.137.3.2 `virtual int spot::state_union::compare (const state * other) const [virtual]`

Compares two states (that come from the same automaton).

This method returns an integer less than, equal to, or greater than zero if *this* is found, respectively, to be less than, equal to, or greater than *other* according to some implicit total order.

This method should not be called to compare states from different automata.

See also

[spot::state\\_ptr\\_less\\_than](#)

Implements [spot::state](#).

#### 7.137.3.3 `virtual size_t spot::state_union::hash () const [virtual]`

Hash a state.

This method returns an integer that can be used as a hash value for this state.

Note that the hash value is guaranteed to be unique for all equal states (in [compare\(\)](#)'s sense) for only has long has one of these states exists. So it's OK to use a [spot::state](#) as a key in a `hash_map` because the mere use of the state as a key in the hash will ensure the state continues to exist.

However if you create the state, get its hash key, delete the state, recreate the same state, and get its hash key, you may obtain two different hash keys if the same state were not already used elsewhere. In practice this weird situation can occur only when the state is BDD-encoded, because BDD numbers (used to build the hash value) can be reused for other formulas. That probably doesn't matter, since the hash value is meant to be used in a `hash_map`, but it had to be noted.

Implements [spot::state](#).

#### 7.137.3.4 `state* spot::state_union::left () const [inline]`

References `left_`.

#### 7.137.3.5 `state* spot::state_union::right () const [inline]`

References `right_`.

### 7.137.4 Member Data Documentation

#### 7.137.4.1 state\* spot::state\_union::left\_ [private]

Does the state belongs to the left automaton ?

Referenced by left().

#### 7.137.4.2 state\* spot::state\_union::right\_ [private]

Does the state belongs to the right automaton ?

Referenced by right().

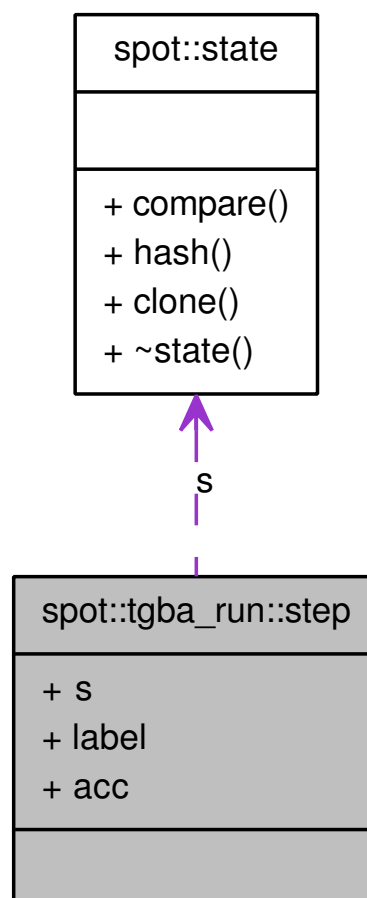
The documentation for this class was generated from the following file:

- [tgba/tgbaunion.hh](#)

### 7.138 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](#)

#### 7.138.1 Member Data Documentation

##### 7.138.1.1 bdd spot::tgba\_run::step::acc

##### 7.138.1.2 bdd spot::tgba\_run::step::label

##### 7.138.1.3 const state\* spot::tgba\_run::step::s

The documentation for this struct was generated from the following file:

- [tgbaalgos/emptiness.hh](#)

## 7.139 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

#### 7.139.1 Detailed Description

A hash function for strings.

#### 7.139.2 Member Function Documentation

##### 7.139.2.1 [size\\_t spot::string\\_hash::operator\(\)](#) (const std::string &s) const [\[inline\]](#)

References [spot::knuth32\\_hash\(\)](#), and [spot::ptr\\_hash< T >::operator\(\)](#).

The documentation for this struct was generated from the following file:

- [misc/hash.hh](#)

## 7.140 `spot::ltl::succ_iterator` Class Reference

```
#include <ltlast/nfa.hh>
```

### Public Member Functions

- `succ_iterator` (const `nfa::state::const_iterator` `s`)
- void `operator++` ()
- bool `operator!=` (const `succ_iterator` &`rhs`) const
- const `nfa::transition` \* `operator*` () const

### Private Attributes

- `nfa::state::const_iterator` `i_`

### 7.140.1 Constructor & Destructor Documentation

**7.140.1.1** `spot::ltl::succ_iterator::succ_iterator` (const `nfa::state::const_iterator` `s`) [`inline`]

### 7.140.2 Member Function Documentation

**7.140.2.1** `bool spot::ltl::succ_iterator::operator!=` (const `succ_iterator` &`rhs`) const [`inline`]

References `i_`.

**7.140.2.2** `const nfa::transition* spot::ltl::succ_iterator::operator*` () const [`inline`]

References `i_`.

**7.140.2.3** `void spot::ltl::succ_iterator::operator++` () [`inline`]

References `i_`.

### 7.140.3 Member Data Documentation

**7.140.3.1** `nfa::state::const_iterator spot::ltl::succ_iterator::i_` [`private`]

Referenced by `operator!=()`, `operator*()`, and `operator++()`.

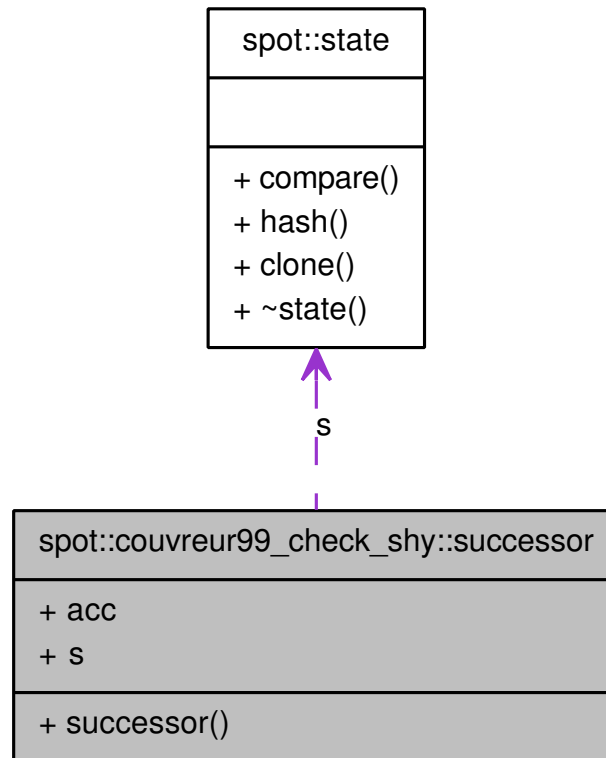
The documentation for this class was generated from the following file:

- `ltlast/nfa.hh`

## 7.141 spot::couvreur99\_check\_shy::successor Struct Reference

```
#include <tgbalgorithms/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`

### 7.141.1 Constructor & Destructor Documentation

**7.141.1.1** `spot::couvreur99_check_shy::successor::successor` (bdd `acc`, const `spot::state * s`)  
`[inline]`

### 7.141.2 Member Data Documentation

#### 7.141.2.1 bdd spot::couvreur99\_check\_shy::successor::acc

#### 7.141.2.2 const spot::state\* spot::couvreur99\_check\_shy::successor::s

The documentation for this struct was generated from the following file:

- [tgbalgos/gtec/gtec.hh](#)

## 7.142 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\\_](#)

## 7.142.1 Member Typedef Documentation

**7.142.1.1** `typedef std::map<const std::string, const symbol*> spot::symbol::map` `[protected]`

## 7.142.2 Constructor & Destructor Documentation

**7.142.2.1** `spot::symbol::symbol (const std::string * name)` `[protected]`

**7.142.2.2** `spot::symbol::~symbol ()` `[protected]`

**7.142.2.3** `spot::symbol::symbol (const symbol &)` `[private]`

## 7.142.3 Member Function Documentation

**7.142.3.1** `static std::ostream& spot::symbol::dump_instances (std::ostream & os)` `[static]`

List all instances of atomic propositions. For debugging.

**7.142.3.2** `static const symbol* spot::symbol::instance (const std::string & name)` `[static]`

**7.142.3.3** `static unsigned spot::symbol::instance_count ()` `[static]`

Number of instantiated atomic propositions. For debugging.

**7.142.3.4** `const std::string& spot::symbol::name () const`

**7.142.3.5** `void spot::symbol::ref () const`

Referenced by `spot::rsymbol::rsymbol()`.

**7.142.3.6** `int spot::symbol::ref_count_ () const` **[protected]**

**7.142.3.7** `void spot::symbol::unref () const`

Referenced by `spot::rsymbol::~rsymbol()`.

#### **7.142.4 Member Data Documentation**

**7.142.4.1** `map spot::symbol::instances_` **[static, protected]**

**7.142.4.2** `const std::string* spot::symbol::name_` **[private]**

Undefined.

**7.142.4.3** `int spot::symbol::refs_` **[mutable, private]**

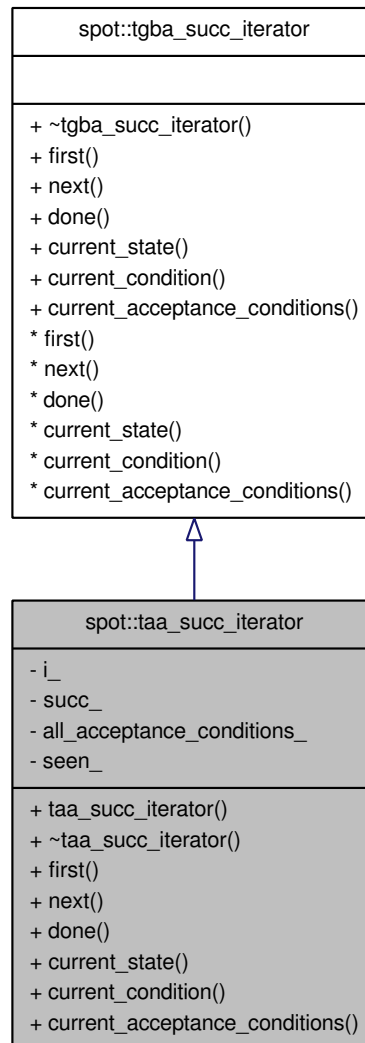
The documentation for this class was generated from the following file:

- [evtgba/symbol.hh](#)

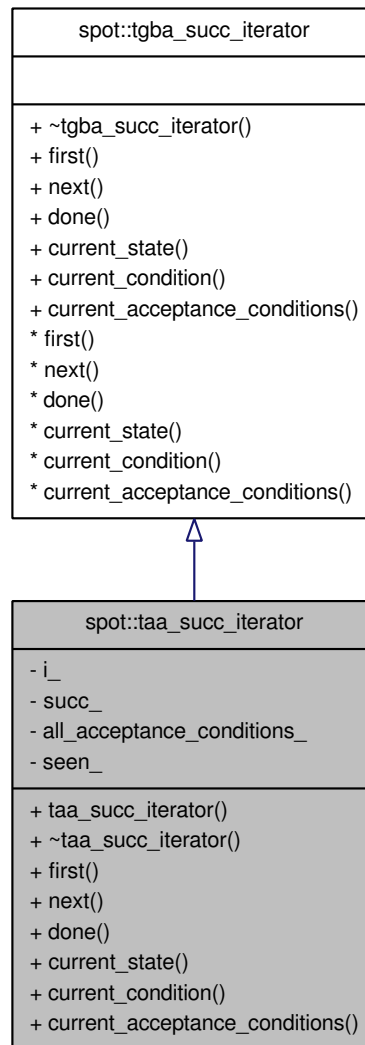
### **7.143 spot::taa\_succ\_iterator Class Reference**

```
#include <tgba/taatgba.hh>
```

Inheritance diagram for spot::taa\_succ\_iterator:



Collaboration diagram for spot::taa\_succ\_iterator:



## Classes

- struct [distance\\_sort](#)

## Public Member Functions

- [taa\\_succ\\_iterator](#) (const [taa\\_tgba::state\\_set](#) \*s, bdd all\_acc)
- virtual [~taa\\_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_set * 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 Types

- `typedef taa_tgba::state::const_iterator iterator`
- `typedef std::pair< iterator, iterator > iterator_pair`
- `typedef std::vector< iterator_pair > bounds_t`
- `typedef Sgi::hash_map< const spot::state_set *, std::vector< taa_tgba::transition * >, state_ptr_hash, state_ptr_equal > seen_map`

### Private Attributes

- `std::vector< taa_tgba::transition * >::const_iterator i_`
- `std::vector< taa_tgba::transition * > succ_`
- `bdd all_acceptance_conditions_`
- `seen_map seen_`

#### 7.143.1 Member Typedef Documentation

**7.143.1.1** `typedef std::vector<iterator_pair> spot::taa_succ_iterator::bounds_t [private]`

**7.143.1.2** `typedef taa_tgba::state::const_iterator spot::taa_succ_iterator::iterator [private]`

Those typedefs are used to generate all possible successors in the constructor using a cartesian product.

**7.143.1.3** `typedef std::pair<iterator, iterator> spot::taa_succ_iterator::iterator_pair [private]`

**7.143.1.4** `typedef Sgi::hash_map< const spot::state_set*, std::vector<taa_tgba::transition*>, state_ptr_hash, state_ptr_equal> spot::taa_succ_iterator::seen_map [private]`

### 7.143.2 Constructor & Destructor Documentation

**7.143.2.1** `spot::taa_succ_iterator::taa_succ_iterator (const taa_tgba::state_set * s, bdd all_acc)`

**7.143.2.2** `virtual spot::taa_succ_iterator::~~taa_succ_iterator ()` **[virtual]**

### 7.143.3 Member Function Documentation

**7.143.3.1** `virtual bdd spot::taa_succ_iterator::current_acceptance_conditions () const`  
**[virtual]**

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.143.3.2** `virtual bdd spot::taa_succ_iterator::current_condition () const` **[virtual]**

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.143.3.3** `virtual state_set* spot::taa_succ_iterator::current_state () const` **[virtual]**

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.143.3.4** `virtual bool spot::taa_succ_iterator::done () const` **[virtual]**

Check whether the iteration is finished.

This function should be called after any call to [first\(\)](#) or [next\(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())  
...
```

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.143.3.5 virtual void spot::taa\_succ\_iterator::first () [virtual]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

##### Warning

One should always call [done \(\)](#) to ensure there is a successor, even after [first \(\)](#). A common trap is to assume that there is at least one successor: this is wrong.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.143.3.6 virtual void spot::taa\_succ\_iterator::next () [virtual]

Jump to the next successor (if any).

##### Warning

Again, one should always call [done \(\)](#) to ensure there is a successor.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.143.4 Member Data Documentation

#### 7.143.4.1 bdd spot::taa\_succ\_iterator::all\_acceptance\_conditions\_ [private]

#### 7.143.4.2 std::vector<taa\_tgba::transition\*>::const\_iterator spot::taa\_succ\_iterator::i\_ [private]

#### 7.143.4.3 seen\_map spot::taa\_succ\_iterator::seen\_ [private]

#### 7.143.4.4 std::vector<taa\_tgba::transition\*> spot::taa\_succ\_iterator::succ\_ [private]

The documentation for this class was generated from the following file:

- [tgba/taatgba.hh](#)

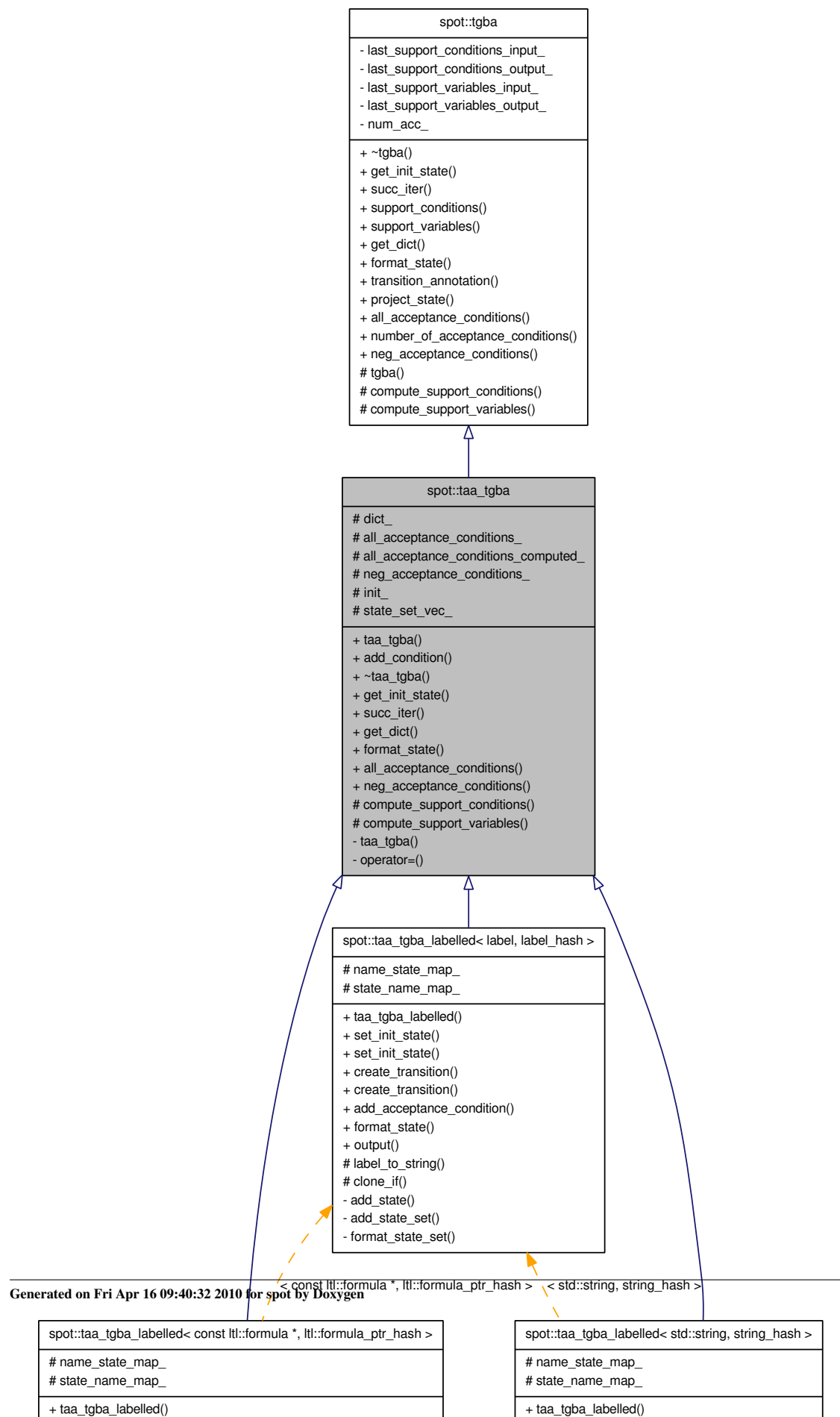
## 7.144 spot::taa\_tgba Class Reference

A self-loop Transition-based Alternating Automaton (TAA) which is seen as a TGBA (abstract class, see below).

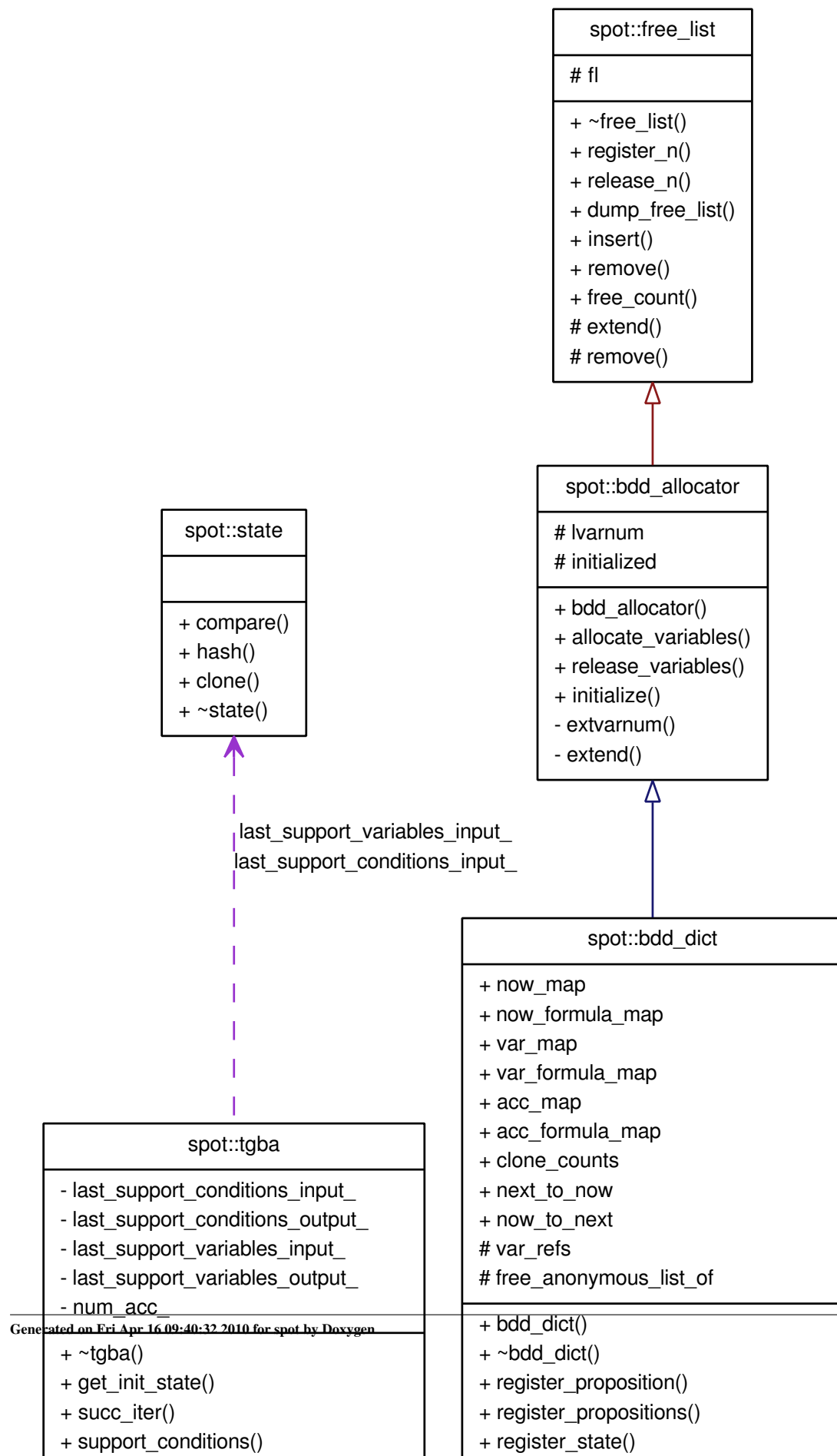
```
#include <tgba/taatgba.hh>
```



Inheritance diagram for spot::taa\_tgba:



Collaboration diagram for spot::taa\_tgba:



## Classes

- struct [transition](#)  
*Explicit transitions.*

## Public Types

- typedef std::list< [transition](#) \* > [state](#)
- typedef std::set< [state](#) \* > [state\\_set](#)

## Public Member Functions

- [taa\\_tgba](#) ([bdd\\_dict](#) \*dict)
- void [add\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- virtual [~taa\\_tgba](#) ()  
*TGBA interface.*
- 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  
*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 [spot::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.*

### Protected Types

- typedef std::vector< taa\_tgba::state\_set \* > ss\_vec

### Protected Member Functions

- virtual bdd compute\_support\_conditions (const spot::state \*state) const  
*Do the actual computation of tgba::support\_conditions().*
- virtual bdd compute\_support\_variables (const spot::state \*state) const  
*Do the actual computation of tgba::support\_variables().*

### Protected Attributes

- bdd\_dict \* dict\_
- bdd all\_acceptance\_conditions\_
- bool all\_acceptance\_conditions\_computed\_
- bdd neg\_acceptance\_conditions\_
- taa\_tgba::state\_set \* init\_
- ss\_vec state\_set\_vec\_

### Private Member Functions

- taa\_tgba (const taa\_tgba &other)
- taa\_tgba & operator= (const taa\_tgba &other)

#### 7.144.1 Detailed Description

A self-loop Transition-based Alternating Automaton (TAA) which is seen as a TGBA (abstract class, see below).

#### 7.144.2 Member Typedef Documentation

**7.144.2.1** typedef std::vector<taa\_tgba::state\_set\*> spot::taa\_tgba::ss\_vec [protected]

**7.144.2.2** typedef std::list<transition\*> spot::taa\_tgba::state

**7.144.2.3** typedef std::set<state\*> spot::taa\_tgba::state\_set

### 7.144.3 Constructor & Destructor Documentation

**7.144.3.1** `spot::taa_tgba::taa_tgba (bdd_dict * dict)`

**7.144.3.2** `virtual spot::taa_tgba::~~taa_tgba ()` **[virtual]**

TGBA interface.

**7.144.3.3** `spot::taa_tgba::taa_tgba (const taa_tgba & other)` **[private]**

### 7.144.4 Member Function Documentation

**7.144.4.1** `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f)`

**7.144.4.2** `virtual bdd spot::taa_tgba::all_acceptance_conditions () const` **[virtual]**

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.144.4.3** `virtual bdd spot::taa_tgba::compute_support_conditions (const spot::state * state) const` **[protected, virtual]**

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.144.4.4** `virtual bdd spot::taa_tgba::compute_support_variables (const spot::state * state) const` **[protected, virtual]**

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.144.4.5 virtual std::string spot::taa\_tgba::format\_state (const spot::state \* state) const [pure virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

Implemented in [spot::taa\\_tgba\\_labelled< label, label\\_hash >](#), [spot::taa\\_tgba\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#), and [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >](#).

#### 7.144.4.6 virtual bdd\_dict\* spot::taa\_tgba::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

#### 7.144.4.7 virtual spot::state\* spot::taa\_tgba::get\_init\_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

#### 7.144.4.8 virtual bdd spot::taa\_tgba::neg\_acceptance\_conditions () const [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

#### 7.144.4.9 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

**7.144.4.10** `taa_tgba& spot::taa_tgba::operator= (const taa_tgba & other) [private]`

**7.144.4.11** `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`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

**7.144.4.12** `virtual tgba_succ_iterator* spot::taa_tgba::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const [virtual]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

#### Parameters

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

***global\_automaton*** In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**7.144.4.13** `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.144.4.14 bdd spot::tgba::support\_variables (const state \* state) const [inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.144.4.15 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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.144.5 Member Data Documentation****7.144.5.1 bdd spot::taa\_tgba::all\_acceptance\_conditions\_ [mutable, protected]****7.144.5.2 bool spot::taa\_tgba::all\_acceptance\_conditions\_computed\_ [mutable, protected]**

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.144.5.3 bdd\_dict\* spot::taa\_tgba::dict\_ [protected]**

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.



#### 7.144.5.4 taa\_tgba::state\_set\* spot::taa\_tgba::init\_ [protected]

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::set\_init\_state().

#### 7.144.5.5 bdd spot::taa\_tgba::neg\_acceptance\_conditions\_ [protected]

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::add\_acceptance\_condition().

#### 7.144.5.6 ss\_vec spot::taa\_tgba::state\_set\_vec\_ [protected]

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::add\_state\_set().

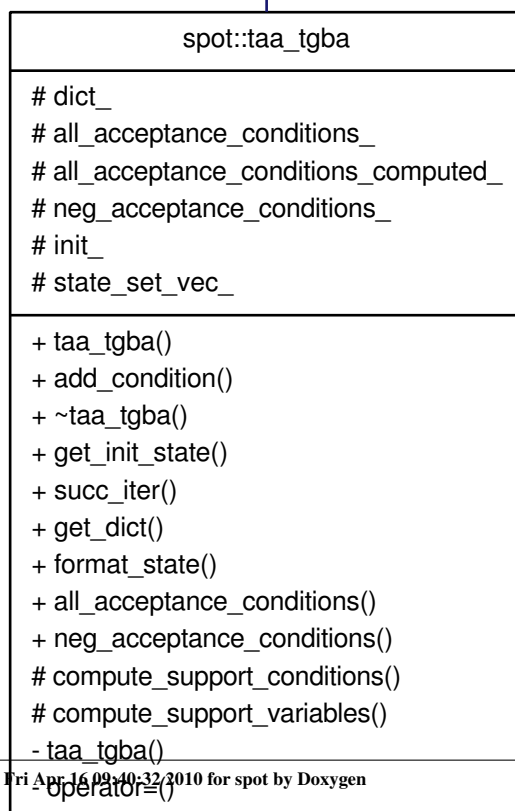
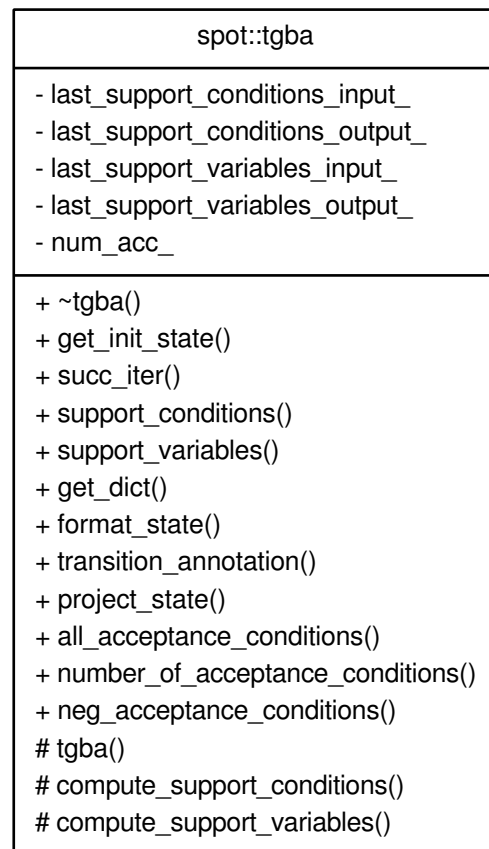
The documentation for this class was generated from the following file:

- tgba/[taatgba.hh](#)

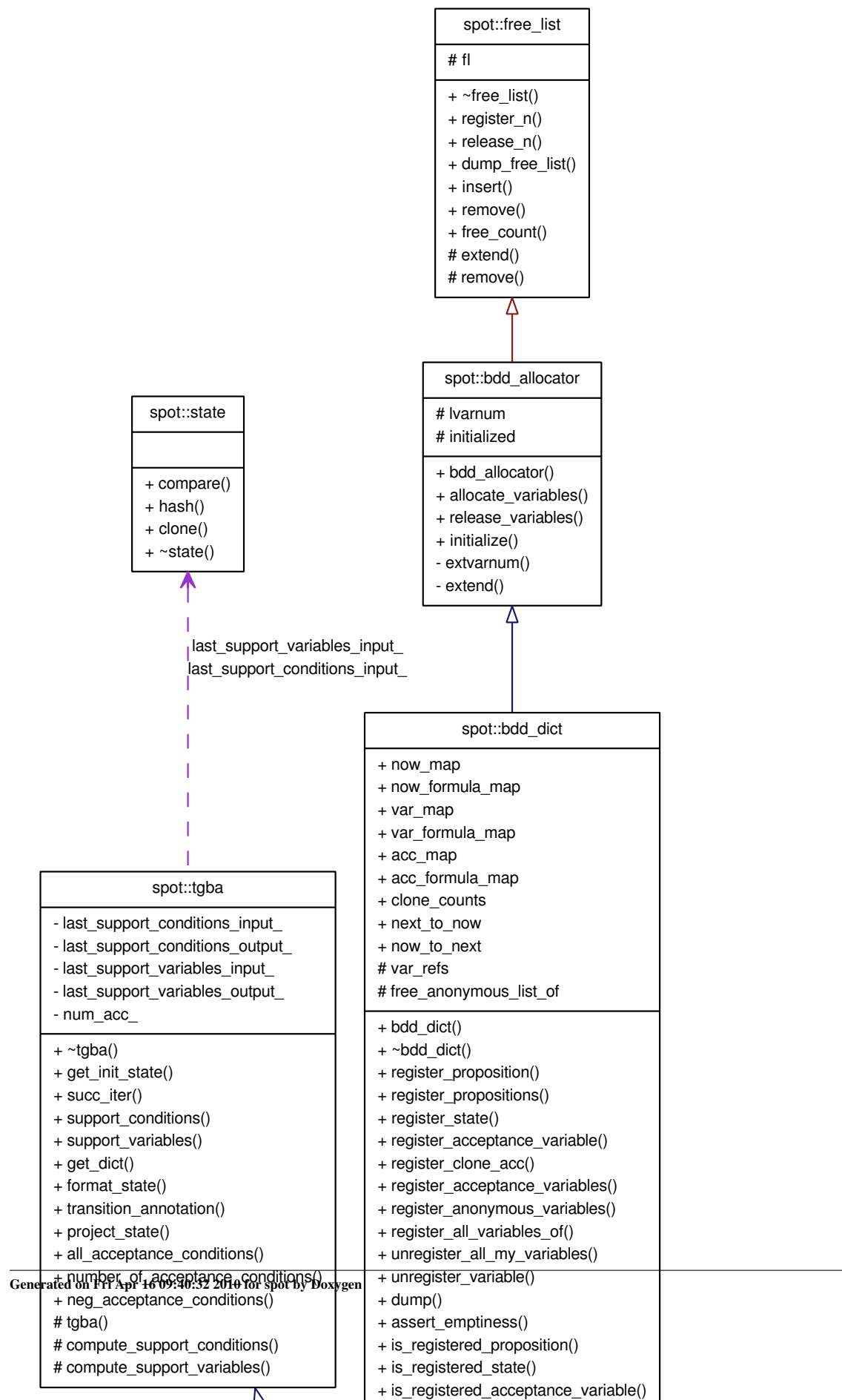
### 7.145 spot::taa\_tgba\_formula Class Reference

```
#include <tgba/taatgba.hh>
```

Inheritance diagram for spot::taa\_tgba\_formula:



Collaboration diagram for spot::taa\_tgba\_formula:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)
- typedef std::set< [state](#) \* > [state\\_set](#)

## Public Member Functions

- [taa\\_tgba\\_formula](#) ([bdd\\_dict](#) \*dict)
- [~taa\\_tgba\\_formula](#) ()
- void [set\\_init\\_state](#) (const const [ltl::formula](#) \*&s)
- void [set\\_init\\_state](#) (const std::vector< const [ltl::formula](#) \* > &s)
- [transition](#) \* [create\\_transition](#) (const const [ltl::formula](#) \*&s, const std::vector< const [ltl::formula](#) \* > &d)
- [transition](#) \* [create\\_transition](#) (const const [ltl::formula](#) \*&s, const const [ltl::formula](#) \*&d)
- void [add\\_acceptance\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const  
*Format the state as a string for printing.*
- void [output](#) (std::ostream &os) const  
*Output a TAA in a stream.*
- void [add\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- 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  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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 Types

- typedef const [ltl::formula](#) \* [label\\_t](#)
- typedef Sgi::hash\_map< const const [ltl::formula](#) \*, [taa\\_tgba::state](#) \*, [ltl::formula\\_ptr\\_hash](#) > [ns\\_map](#)
- typedef Sgi::hash\_map< const [taa\\_tgba::state](#) \*, const [ltl::formula](#) \*, [ptr\\_hash](#)< [taa\\_tgba::state](#) > > [sn\\_map](#)
- typedef std::vector< [taa\\_tgba::state\\_set](#) \* > [ss\\_vec](#)

### Protected Member Functions

- virtual std::string [label\\_to\\_string](#) (const [label\\_t](#) &label) const  
*Return a label as a string.*
- virtual [ltl::formula](#) \* [clone\\_if](#) (const [label\\_t](#) &label) const  
*Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.*
- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Protected Attributes

- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- bdd\_dict \* [dict\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bool [all\\_acceptance\\_conditions\\_computed\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)
- [taa\\_tgba::state\\_set](#) \* [init\\_](#)
- [ss\\_vec](#) [state\\_set\\_vec\\_](#)

## 7.145.1 Member Typedef Documentation

- 7.145.1.1** typedef const [ltl::formula](#) \* [spot::taa\\_tgba\\_labelled](#)< const [ltl::formula](#) \* , [ltl::formula\\_ptr\\_hash](#) >::[label\\_t](#) [[protected](#), [inherited](#)]

7.145.1.2 `typedef Sgi::hash_map< const const ltl::formula * , taa_tgba::state*,  
ltl::formula_ptr_hash > spot::taa_tgba_labelled< const ltl::formula * ,  
ltl::formula_ptr_hash >::ns_map [protected, inherited]`

7.145.1.3 `typedef Sgi::hash_map< const taa_tgba::state*, const ltl::formula * ,  
ptr_hash<taa_tgba::state> > spot::taa_tgba_labelled< const ltl::formula * ,  
ltl::formula_ptr_hash >::sn_map [protected, inherited]`

7.145.1.4 `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,  
inherited]`

7.145.1.5 `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

7.145.1.6 `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

## 7.145.2 Constructor & Destructor Documentation

7.145.2.1 `spot::taa_tgba_formula::taa_tgba_formula (bdd_dict * dict) [inline]`

7.145.2.2 `spot::taa_tgba_formula::~~taa_tgba_formula ()`

## 7.145.3 Member Function Documentation

7.145.3.1 `void spot::taa_tgba_labelled< const ltl::formula * , ltl::formula_ptr_hash  
>::add_acceptance_condition (transition * t, const ltl::formula * f) [inline,  
inherited]`

7.145.3.2 `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f)  
[inherited]`

**7.145.3.3** `virtual bdd spot::taa_tgba::all_acceptance_conditions () const [virtual, inherited]`

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.145.3.4** `virtual ltl::formula* spot::taa_tgba_formula::clone_if (const label_t & lbl) const [protected, virtual]`

Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.

Implements [spot::taa\\_tgba\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#).

**7.145.3.5** `virtual bdd spot::taa_tgba::compute_support_conditions (const spot::state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.145.3.6** `virtual bdd spot::taa_tgba::compute_support_variables (const spot::state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.145.3.7** `transition* spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::create_transition (const const ltl::formula * & s, const const ltl::formula * & d) [inline, inherited]`

**7.145.3.8** `transition* spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::create_transition (const const ltl::formula * & s, const std::vector< const ltl::formula * > & d) [inline, inherited]`

**7.145.3.9** `virtual std::string spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::format_state (const spot::state * s) const` `[inline, virtual, inherited]`

Format the state as a string for printing.

If state is a [spot::state\\_set](#) of only one element, then the string corresponding to state->get\_state() is returned.

Otherwise a string composed of each string corresponding to each state->get\_state() in the [spot::state\\_set](#) is returned, e.g. like {string\_1,...,string\_n}.

Implements [spot::taa\\_tgba](#).

**7.145.3.10** `virtual bdd_dict* spot::taa_tgba::get_dict () const` `[virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.145.3.11** `virtual spot::state* spot::taa_tgba::get_init_state () const` `[virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.145.3.12** `virtual std::string spot::taa_tgba_formula::label_to_string (const label_t & lbl) const` `[protected, virtual]`

Return a label as a string.

Implements [spot::taa\\_tgba\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#).

**7.145.3.13** `virtual bdd spot::taa_tgba::neg_acceptance_conditions () const` `[virtual, inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.



Implements [spot::tgba](#).

**7.145.3.14** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
`[virtual, inherited]`

The number of acceptance conditions.

**7.145.3.15** `void spot::taa_tgba_labelled< const ltl::formula * , ltl::formula_ptr_hash >::output`  
`(std::ostream & os) const [inline, inherited]`

Output a TAA in a stream.

**7.145.3.16** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const`  
`[virtual, inherited]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

#### Returns

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.145.3.17** `void spot::taa_tgba_labelled< const ltl::formula * , ltl::formula_ptr_hash`  
`>::set_init_state (const std::vector< const ltl::formula * > & s) [inline,`  
`inherited]`

**7.145.3.18** `void spot::taa_tgba_labelled< const ltl::formula * , ltl::formula_ptr_hash`  
`>::set_init_state (const const ltl::formula * & s) [inline, inherited]`

**7.145.3.19** `virtual tgba_succ_iterator* spot::taa_tgba::succ_iter (const spot::state * local_state,`  
`const spot::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`.

#### 7.145.3.20 `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.145.3.21 `bdd spot::tgba::support_variables (const state * state) const [inherited]`

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.145.3.22 `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 it the empty string.

**Parameters**

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

**7.145.4 Member Data Documentation**

**7.145.4.1** `bdd spot::taa_tgba::all_acceptance_conditions_ [mutable, protected, inherited]`

**7.145.4.2** `bool spot::taa_tgba::all_acceptance_conditions_computed_ [mutable, protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.145.4.3** `bdd_dict* spot::taa_tgba::dict_ [protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.145.4.4** `taa_tgba::state_set* spot::taa_tgba::init_ [protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::set_init_state()`.

**7.145.4.5** `ns_map spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::name_state_map_ [protected, inherited]`

**7.145.4.6** `bdd spot::taa_tgba::neg_acceptance_conditions_ [protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.145.4.7** `sn_map spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::state_name_map_ [protected, inherited]`

**7.145.4.8** `ss_vec spot::taa_tgba::state_set_vec_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_state_set()`.

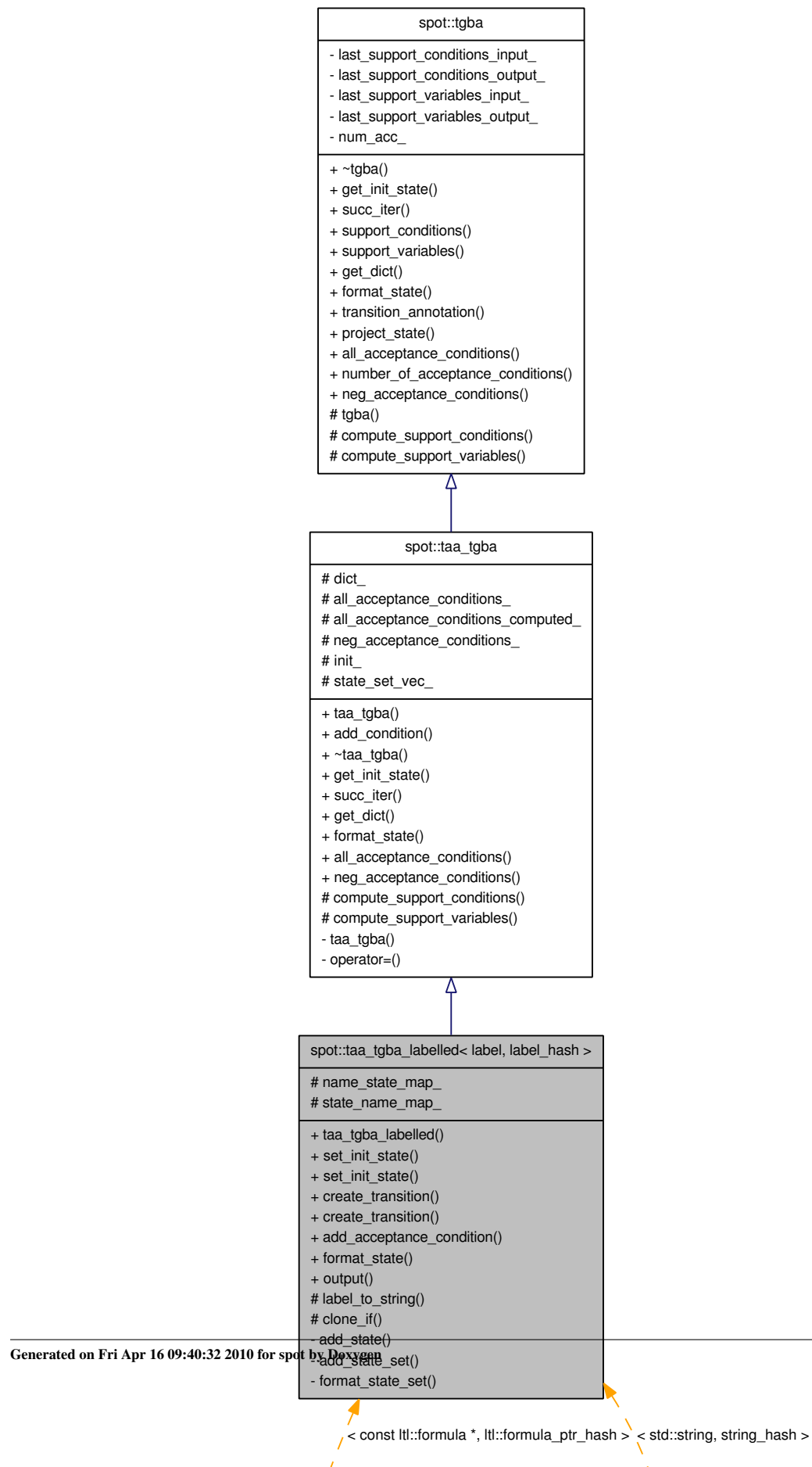
The documentation for this class was generated from the following file:

- `tgba/taatgba.hh`

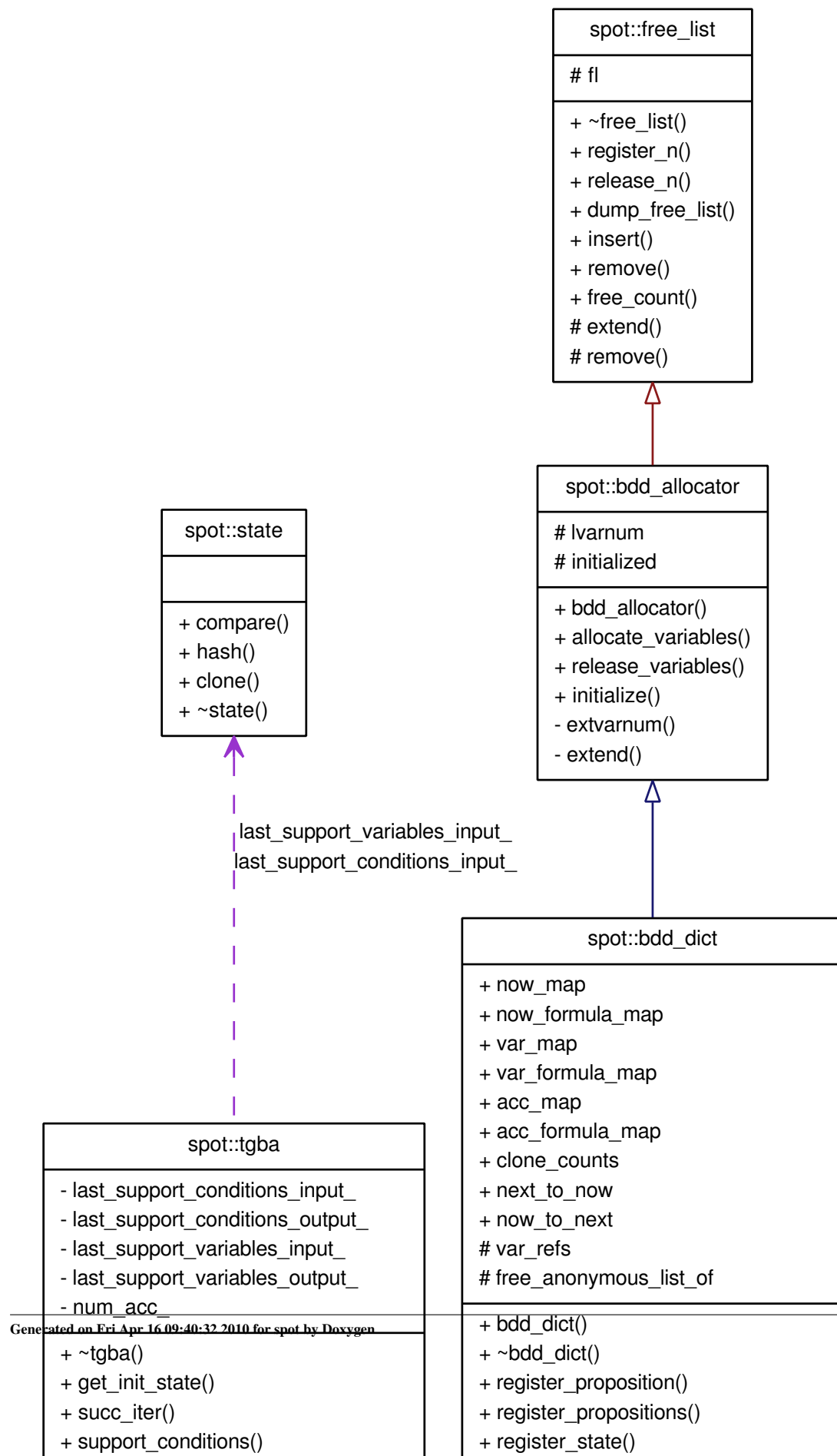
**7.146** `spot::taa_tgba_labelled< label, label_hash >` Class Template Reference

```
#include <tgba/taatgba.hh>
```

Inheritance diagram for spot::taa\_tgba\_labelled< label, label\_hash >:



Collaboration diagram for spot::taa\_tgba\_labelled< label, label\_hash >:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)
- typedef std::set< [state](#) \* > [state\\_set](#)

## Public Member Functions

- [taa\\_tgba\\_labelled](#) ([bdd\\_dict](#) \*dict)
- void [set\\_init\\_state](#) (const label &s)
- void [set\\_init\\_state](#) (const std::vector< label > &s)
- [transition](#) \* [create\\_transition](#) (const label &s, const std::vector< label > &d)
- [transition](#) \* [create\\_transition](#) (const label &s, const label &d)
- void [add\\_acceptance\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const  
*Format the state as a string for printing.*
- void [output](#) (std::ostream &os) const  
*Output a TAA in a stream.*
- void [add\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- 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  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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 Types

- typedef label [label\\_t](#)
- typedef Sgi::hash\_map< const label, [taa\\_tgba::state](#) \*, label\_hash > [ns\\_map](#)
- typedef Sgi::hash\_map< const [taa\\_tgba::state](#) \*, label, [ptr\\_hash](#)< [taa\\_tgba::state](#) > > [sn\\_map](#)
- typedef std::vector< [taa\\_tgba::state\\_set](#) \* > [ss\\_vec](#)

### Protected Member Functions

- virtual std::string [label\\_to\\_string](#) (const [label\\_t](#) &lbl) const =0  
*Return a label as a string.*
- virtual [label\\_t](#) [clone\\_if](#) (const [label\\_t](#) &lbl) const =0  
*Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.*
- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Protected Attributes

- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- bdd\_dict \* [dict\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bool [all\\_acceptance\\_conditions\\_computed\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)
- [taa\\_tgba::state\\_set](#) \* [init\\_](#)
- [ss\\_vec](#) [state\\_set\\_vec\\_](#)

### Private Member Functions

- [taa\\_tgba::state](#) \* [add\\_state](#) (const label &name)  
*Return the [taa\\_tgba::state](#) for name, creating it when it does not exist already.*
- [taa\\_tgba::state\\_set](#) \* [add\\_state\\_set](#) (const std::vector< label > &names)  
*Return the [taa::state\\_set](#) for names.*
- std::string [format\\_state\\_set](#) (const [taa\\_tgba::state\\_set](#) \*ss) const

### 7.146.1 Detailed Description

template<typename label, typename label\_hash> class spot::taa\_tgba\_labelled< label, label\_hash >

A [taa\\_tgba](#) instance with states labeled by a given type. Still an abstract class, see below.



### 7.146.2 Member Typedef Documentation

**7.146.2.1** `template<typename label, typename label_hash> typedef label  
spot::taa_tgba_labelled< label, label_hash >::label_t [protected]`

**7.146.2.2** `template<typename label, typename label_hash> typedef Sgi::hash_map< const label,  
taa_tgba::state*, label_hash > spot::taa_tgba_labelled< label, label_hash >::ns_map  
[protected]`

**7.146.2.3** `template<typename label, typename label_hash> typedef Sgi::hash_map< const  
taa_tgba::state*, label, ptr_hash<taa_tgba::state> > spot::taa_tgba_labelled< label,  
label_hash >::sn_map [protected]`

**7.146.2.4** `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,  
inherited]`

**7.146.2.5** `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

**7.146.2.6** `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

### 7.146.3 Constructor & Destructor Documentation

**7.146.3.1** `template<typename label, typename label_hash> spot::taa_tgba_labelled< label,  
label_hash >::taa_tgba_labelled (bdd_dict * dict) [inline]`

### 7.146.4 Member Function Documentation

**7.146.4.1** `template<typename label, typename label_hash> void spot::taa_tgba_labelled<  
label, label_hash >::add_acceptance_condition (transition * t, const ltl::formula * f)  
[inline]`

**7.146.4.2** void spot::taa\_tgba::add\_condition (transition \* *t*, const ltl::formula \* *f*)  
[inherited]

**7.146.4.3** template<typename label, typename label\_hash> taa\_tgba::state\*  
spot::taa\_tgba\_labelled< label, label\_hash >::add\_state (const label & *name*)  
[inline, private]

Return the [taa\\_tgba::state](#) for *name*, creating it when it does not exist already.

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::add\_state\_set(), and spot::taa\_tgba\_labelled< std::string, string\_hash >::create\_transition().

**7.146.4.4** template<typename label, typename label\_hash> taa\_tgba::state\_set\*  
spot::taa\_tgba\_labelled< label, label\_hash >::add\_state\_set (const std::vector< label >  
& *names*) [inline, private]

Return the taa::state\_set for *names*.

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::create\_transition(), and spot::taa\_tgba\_labelled< std::string, string\_hash >::set\_init\_state().

**7.146.4.5** virtual bdd spot::taa\_tgba::all\_acceptance\_conditions () const [virtual,  
inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.146.4.6** template<typename label, typename label\_hash> virtual label\_t  
spot::taa\_tgba\_labelled< label, label\_hash >::clone\_if (const label\_t & *lbl*) const  
[protected, pure virtual]

Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.

Implemented in [spot::taa\\_tgba\\_string](#), and [spot::taa\\_tgba\\_formula](#).

Referenced by spot::taa\_tgba\_labelled< std::string, string\_hash >::add\_state().

**7.146.4.7** `virtual bdd spot::taa_tgba::compute_support_conditions (const spot::state * state) const`  
`[protected, virtual, inherited]`

Do the actual computation of `tgba::support_conditions()`.

Implements `spot::tgba`.

**7.146.4.8** `virtual bdd spot::taa_tgba::compute_support_variables (const spot::state * state) const`  
`[protected, virtual, inherited]`

Do the actual computation of `tgba::support_variables()`.

Implements `spot::tgba`.

**7.146.4.9** `template<typename label, typename label_hash> transition* spot::taa_tgba_labelled<`  
`label, label_hash >::create_transition (const label & s, const label & d) [inline]`

**7.146.4.10** `template<typename label, typename label_hash> transition* spot::taa_tgba_labelled<`  
`label, label_hash >::create_transition (const label & s, const std::vector< label > & d)`  
`[inline]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::create_transition()`.

**7.146.4.11** `template<typename label, typename label_hash> virtual std::string`  
`spot::taa_tgba_labelled< label, label_hash >::format_state (const spot::state * s) const`  
`[inline, virtual]`

Format the state as a string for printing.

If state is a `spot::state_set` of only one element, then the string corresponding to `state->get_state()` is returned.

Otherwise a string composed of each string corresponding to each `state->get_state()` in the `spot::state_set` is returned, e.g. like `{string_1,...,string_n}`.

Implements `spot::taa_tgba`.

**7.146.4.12** `template<typename label, typename label_hash> std::string spot::taa_tgba_labelled<`  
`label, label_hash >::format_state_set (const taa_tgba::state_set * ss) const [inline,`  
`private]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::format_state()`, and `spot::taa_tgba_labelled< std::string, string_hash >::output()`.

**7.146.4.13** `virtual bdd_dict* spot::taa_tgba::get_dict () const [virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.146.4.14** `virtual spot::state* spot::taa_tgba::get_init_state () const [virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.146.4.15** `template<typename label, typename label_hash> virtual std::string  
spot::taa_tgba_labelled< label, label_hash >::label_to_string (const label_t & lbl)  
const [protected, pure virtual]`

Return a label as a string.

Implemented in [spot::taa\\_tgba\\_string](#), and [spot::taa\\_tgba\\_formula](#).

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::format_state_set()`, and `spot::taa_tgba_labelled< std::string, string_hash >::output()`.

**7.146.4.16** `virtual bdd spot::taa_tgba::neg_acceptance_conditions () const [virtual, inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.146.4.17** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const  
[virtual, inherited]`

The number of acceptance conditions.

**7.146.4.18** `template<typename label, typename label_hash> void spot::taa_tgba_labelled< label, label_hash >::output (std::ostream & os) const [inline]`

Output a TAA in a stream.

**7.146.4.19** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const [virtual, inherited]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

#### Returns

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.146.4.20** `template<typename label, typename label_hash> void spot::taa_tgba_labelled< label, label_hash >::set_init_state (const std::vector< label > & s) [inline]`

**7.146.4.21** `template<typename label, typename label_hash> void spot::taa_tgba_labelled< label, label_hash >::set_init_state (const label & s) [inline]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::set_init_state()`.

**7.146.4.22** `virtual tgba_succ_iterator* spot::taa_tgba::succ_iter (const spot::state * local_state, const spot::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 *local\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

**7.146.4.23 `bdd spot::tgba::support_conditions (const state * state) const [inherited]`**

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.146.4.24 `bdd spot::tgba::support_variables (const state * state) const [inherited]`**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.146.4.25 `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](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

### 7.146.5 Member Data Documentation

**7.146.5.1** `bdd spot::taa_tgba::all_acceptance_conditions_` `[mutable, protected, inherited]`

**7.146.5.2** `bool spot::taa_tgba::all_acceptance_conditions_computed_` `[mutable, protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.146.5.3** `bdd_dict* spot::taa_tgba::dict_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.146.5.4** `taa_tgba::state_set* spot::taa_tgba::init_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::set_init_state()`.

**7.146.5.5** `template<typename label, typename label_hash> ns_map spot::taa_tgba_labelled< label, label_hash >::name_state_map_` `[protected]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, `spot::taa_tgba_labelled< std::string, string_hash >::add_state()`, and `spot::taa_tgba_labelled< std::string, string_hash >::output()`.

**7.146.5.6** `bdd spot::taa_tgba::neg_acceptance_conditions_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.146.5.7** `template<typename label, typename label_hash> sn_map spot::taa_tgba_labelled< label, label_hash >::state_name_map_` `[protected]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_state()`, and `spot::taa_tgba_labelled< std::string, string_hash >::format_state_set()`.

**7.146.5.8** `ss_vec spot::taa_tgba::state_set_vec_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_state_set()`.

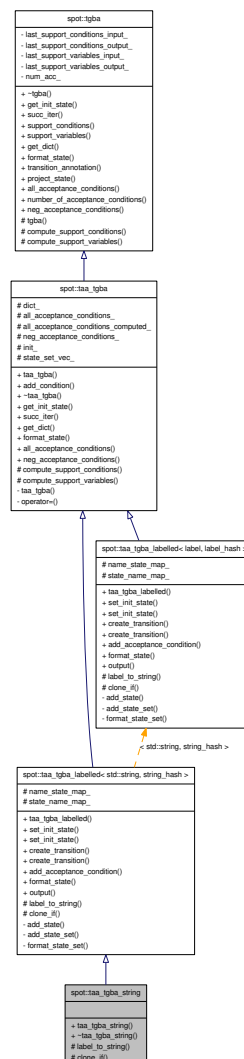
The documentation for this class was generated from the following file:

- [tgba/taatgba.hh](#)

## 7.147 spot::taa\_tgba\_string Class Reference

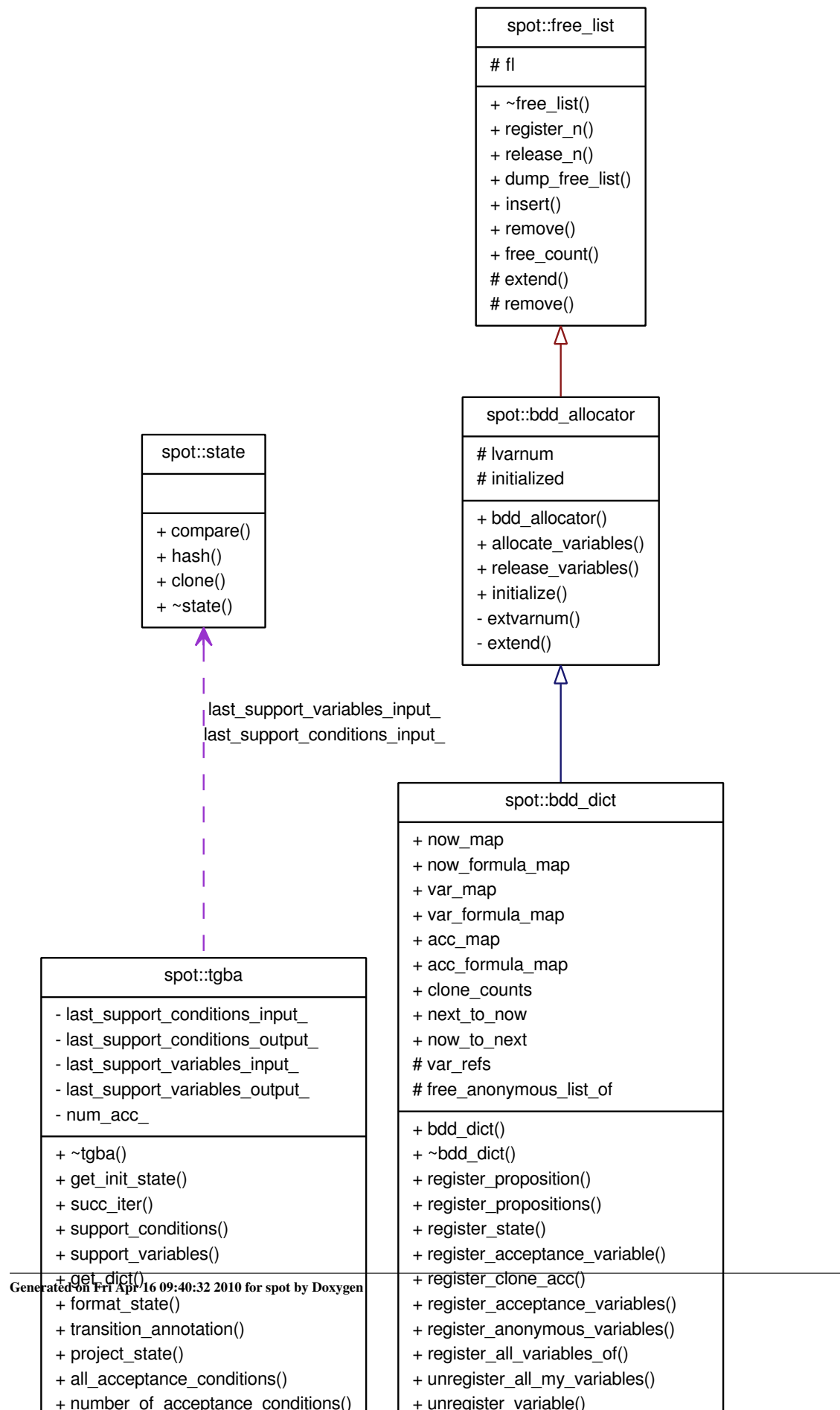
```
#include <tgba/taatgba.hh>
```

Inheritance diagram for `spot::taa_tgba_string`:





Collaboration diagram for spot::taa\_tgba\_string:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)
- typedef std::set< [state](#) \* > [state\\_set](#)

## Public Member Functions

- [taa\\_tgba\\_string](#) (bdd\_dict \*dict)
- [~taa\\_tgba\\_string](#) ()
- void [set\\_init\\_state](#) (const std::string &s)
- void [set\\_init\\_state](#) (const std::vector< std::string > &s)
- [transition](#) \* [create\\_transition](#) (const std::string &s, const std::vector< std::string > &d)
- [transition](#) \* [create\\_transition](#) (const std::string &s, const std::string &d)
- void [add\\_acceptance\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const  
*Format the state as a string for printing.*
- void [output](#) (std::ostream &os) const  
*Output a TAA in a stream.*
- void [add\\_condition](#) ([transition](#) \*t, const [ltl::formula](#) \*f)
- 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  
*Get an iterator over the successors of local\_state.*
- virtual bdd\_dict \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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 Types

- typedef std::string [label\\_t](#)
- typedef Sgi::hash\_map< const std::string, [taa\\_tgba::state](#) \*, [string\\_hash](#) > [ns\\_map](#)
- typedef Sgi::hash\_map< const [taa\\_tgba::state](#) \*, std::string, [ptr\\_hash](#)< [taa\\_tgba::state](#) > > [sn\\_map](#)
- typedef std::vector< [taa\\_tgba::state\\_set](#) \* > [ss\\_vec](#)

### Protected Member Functions

- virtual std::string [label\\_to\\_string](#) (const std::string &label) const  
*Return a label as a string.*
- virtual std::string [clone\\_if](#) (const std::string &label) const  
*Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.*
- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*

### Protected Attributes

- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- [bdd\\_dict](#) \* [dict\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bool [all\\_acceptance\\_conditions\\_computed\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)
- [taa\\_tgba::state\\_set](#) \* [init\\_](#)
- [ss\\_vec](#) [state\\_set\\_vec\\_](#)

#### 7.147.1 Member Typedef Documentation

**7.147.1.1** typedef std::string spot::taa\_tgba\_labelled< std::string , [string\\_hash](#) >::label\_t  
[protected, inherited]

**7.147.1.2** typedef Sgi::hash\_map< const std::string , [taa\\_tgba::state](#)\*, [string\\_hash](#) >  
spot::taa\_tgba\_labelled< std::string , [string\\_hash](#) >::ns\_map [protected,  
inherited]

**7.147.1.3** `typedef Sgi::hash_map< const taa_tgba::state*, std::string, ptr_hash<taa_tgba::state>  
> spot::taa_tgba_labelled< std::string, string_hash >::sn_map [protected,  
inherited]`

**7.147.1.4** `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,  
inherited]`

**7.147.1.5** `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

**7.147.1.6** `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

## 7.147.2 Constructor & Destructor Documentation

**7.147.2.1** `spot::taa_tgba_string::taa_tgba_string(bdd_dict * dict) [inline]`

**7.147.2.2** `spot::taa_tgba_string::~~taa_tgba_string()`

## 7.147.3 Member Function Documentation

**7.147.3.1** `void spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition  
(transition * t, const ltl::formula * f) [inline, inherited]`

References `spot::bdd_dict::acc_map`, `spot::taa_tgba::transition::acceptance_conditions`, `spot::taa_tgba::all_acceptance_conditions_computed_`, `spot::ltl::formula::destroy()`, `spot::taa_tgba::dict_`, `spot::taa_tgba_labelled< label, label_hash >::name_state_map_`, `spot::taa_tgba::neg_acceptance_conditions_`, and `spot::bdd_dict::register_acceptance_variable()`.

**7.147.3.2** `void spot::taa_tgba::add_condition(transition * t, const ltl::formula * f)  
[inherited]`

**7.147.3.3** `virtual bdd spot::taa_tgba::all_acceptance_conditions () const` [`virtual`, `inherited`]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.147.3.4** `virtual std::string spot::taa_tgba_string::clone_if (const std::string & lbl) const` [`protected`, `virtual`]

Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.

Implements [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >](#).

**7.147.3.5** `virtual bdd spot::taa_tgba::compute_support_conditions (const spot::state * state) const` [`protected`, `virtual`, `inherited`]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.147.3.6** `virtual bdd spot::taa_tgba::compute_support_variables (const spot::state * state) const` [`protected`, `virtual`, `inherited`]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.147.3.7** `transition* spot::taa_tgba_labelled< std::string, string_hash >::create_transition (const std::string & s, const std::string & d)` [`inline`, `inherited`]

References [spot::taa\\_tgba\\_labelled< label, label\\_hash >::create\\_transition\(\)](#).

**7.147.3.8** `transition* spot::taa_tgba_labelled< std::string, string_hash >::create_transition (const std::string & s, const std::vector< std::string > & d)` [`inline`, `inherited`]

References [spot::taa\\_tgba::transition::acceptance\\_conditions](#), [spot::taa\\_tgba\\_labelled< label, label\\_hash >::add\\_state\(\)](#), [spot::taa\\_tgba\\_labelled< label, label\\_hash >::add\\_state\\_set\(\)](#), [spot::taa\\_tgba::transition::condition](#), and [spot::taa\\_tgba::transition::dst](#).

**7.147.3.9** `virtual std::string spot::taa_tgba_labelled< std::string, string_hash >::format_state  
(const spot::state * s) const [inline, virtual, inherited]`

Format the state as a string for printing.

If state is a [spot::state\\_set](#) of only one element, then the string corresponding to state->get\_state() is returned.

Otherwise a string composed of each string corresponding to each state->get\_state() in the [spot::state\\_set](#) is returned, e.g. like {string\_1,...,string\_n}.

Implements [spot::taa\\_tgba](#).

References [spot::taa\\_tgba\\_labelled< label, label\\_hash >::format\\_state\\_set\(\)](#), and [spot::state\\_set::get\\_state\(\)](#).

**7.147.3.10** `virtual bdd_dict* spot::taa_tgba::get_dict () const [virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.147.3.11** `virtual spot::state* spot::taa_tgba::get_init_state () const [virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.147.3.12** `virtual std::string spot::taa_tgba_string::label_to_string (const std::string & lbl) const  
[protected, virtual]`

Return a label as a string.

Implements [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >](#).

**7.147.3.13** `virtual bdd spot::taa_tgba::neg_acceptance_conditions () const [virtual,  
inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.147.3.14** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
**[virtual, inherited]**

The number of acceptance conditions.

**7.147.3.15** `void spot::taa_tgba_labelled< std::string , string_hash >::output (std::ostream & os)`  
**const [inline, inherited]**

Output a TAA in a stream.

References [spot::taa\\_tgba\\_labelled< label, label\\_hash >::format\\_state\\_set\(\)](#), [spot::taa\\_tgba\\_labelled< label, label\\_hash >::label\\_to\\_string\(\)](#), and [spot::taa\\_tgba\\_labelled< label, label\\_hash >::name\\_state\\_map\\_](#).

**7.147.3.16** `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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.147.3.17** `void spot::taa_tgba_labelled< std::string , string_hash >::set_init_state (const std::vector< std::string > & s)` **[inline, inherited]**

References [spot::taa\\_tgba\\_labelled< label, label\\_hash >::add\\_state\\_set\(\)](#), and [spot::taa\\_tgba::init\\_](#).

**7.147.3.18** `void spot::taa_tgba_labelled< std::string , string_hash >::set_init_state (const std::string & s)` **[inline, inherited]**

References [spot::taa\\_tgba\\_labelled< label, label\\_hash >::set\\_init\\_state\(\)](#).

**7.147.3.19** `virtual tgba_succ_iterator* spot::taa_tgba::succ_iter (const spot::state * local_state,  
const spot::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`.

**7.147.3.20** `bdd spot::tgba::support_conditions (const state * state) const` [**inherited**]

Get a formula that must hold whatever successor is taken.

#### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.147.3.21** `bdd spot::tgba::support_variables (const state * state) const` [**inherited**]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.



**7.147.3.22** `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](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

### 7.147.4 Member Data Documentation

**7.147.4.1** `bdd spot::taa_tgba::all_acceptance_conditions_` `[mutable, protected, inherited]`

**7.147.4.2** `bool spot::taa_tgba::all_acceptance_conditions_computed_` `[mutable, protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.147.4.3** `bdd_dict* spot::taa_tgba::dict_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.147.4.4** `taa_tgba::state_set* spot::taa_tgba::init_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::set_init_state()`.

**7.147.4.5** `ns_map spot::taa_tgba_labelled< std::string, string_hash >::name_state_map_`  
`[protected, inherited]`

**7.147.4.6** `bdd spot::taa_tgba::neg_acceptance_conditions_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`.

**7.147.4.7** `sn_map spot::taa_tgba_labelled< std::string , string_hash >::state_name_map_`  
`[protected, inherited]`

**7.147.4.8** `ss_vec spot::taa_tgba::state_set_vec_` `[protected, inherited]`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_state_set()`.

The documentation for this class was generated from the following file:

- [tgba/taatgba.hh](#)

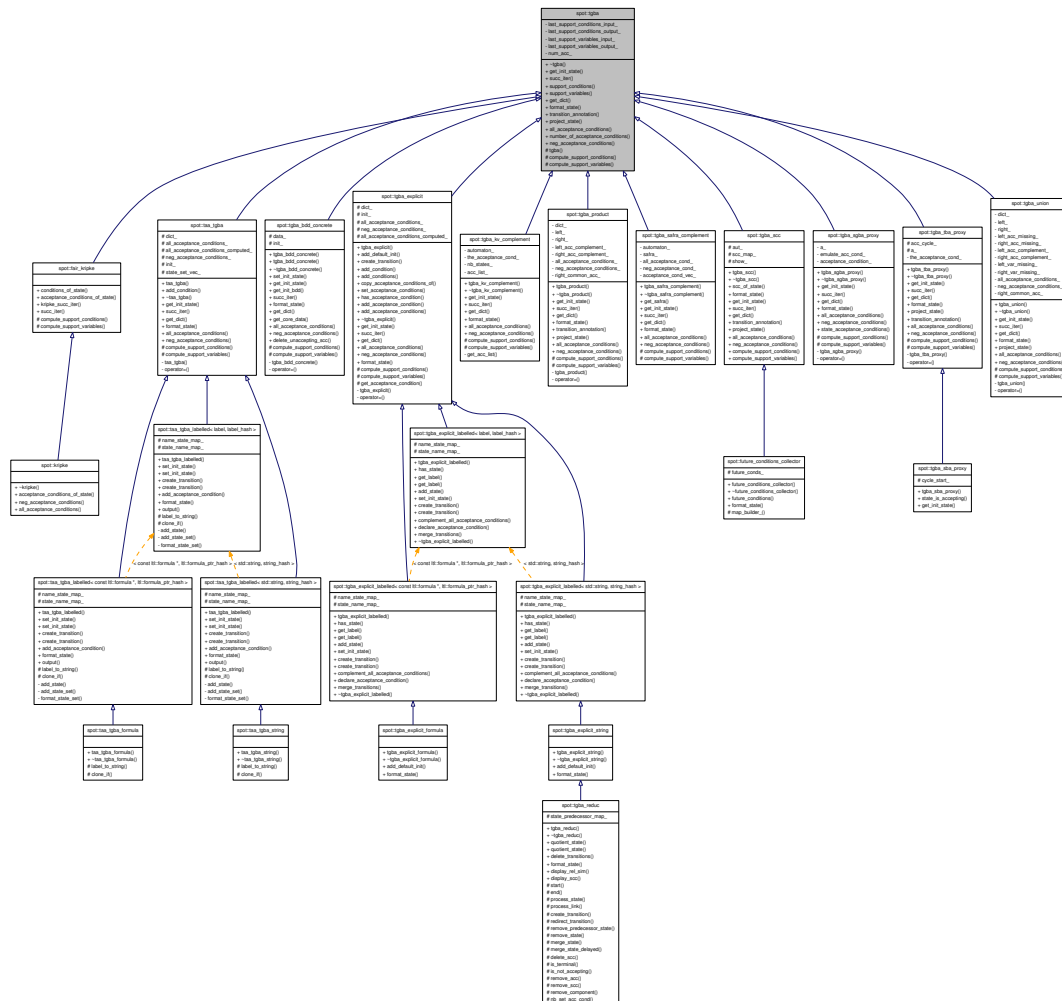
## 7.148 spot::tgba Class Reference

A Transition-based Generalized Büchi Automaton.

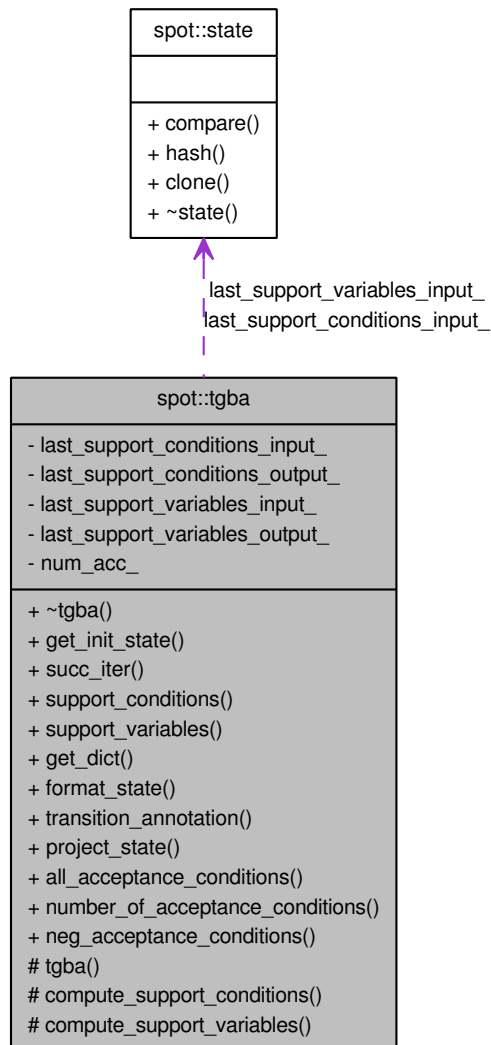
The acronym TGBA (Transition-based Generalized Büchi Automaton) was coined by Dimitra Gianakopoulou and Flavio Lerda in "From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata". (FORTE'02).

```
#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_`

### 7.148.1 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 traverses 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 lists the successor states of any state.

Note that although this is a transition-based automata, we never represent transitions! Transition informations are obtained by querying the iterator over the successors of a state.

## 7.148.2 Constructor & Destructor Documentation

### 7.148.2.1 `spot::tgba::tgba ()` **[protected]**

### 7.148.2.2 `virtual spot::tgba::~~tgba ()` **[virtual]**

## 7.148.3 Member Function Documentation

### 7.148.3.1 `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::kripke`, `spot::taa_tgba`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

### 7.148.3.2 `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::fair_kripke`, `spot::taa_tgba`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

### 7.148.3.3 `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::fair_kripke`, `spot::taa_tgba`, `spot::tgba_bdd_concrete`, `spot::tgba_explicit`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

### 7.148.3.4 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 that owns the state.

Implemented in [spot::future\\_conditions\\_collector](#), [spot::taa\\_tgba](#), [spot::taa\\_tgba\\_labelled< label, label\\_hash >](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_explicit\\_string](#), [spot::tgba\\_explicit\\_formula](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_reduc](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), [spot::tgba\\_union](#), [spot::taa\\_tgba\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#), and [spot::taa\\_tgba\\_labelled< std::string, string\\_hash >](#).

### 7.148.3.5 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::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.148.3.6 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::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), [spot::tgba\\_sba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.148.3.7 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::kripke](#), [spot::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safracomplement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.148.3.8** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
**[virtual]**

The number of acceptance conditions.

**7.148.3.9** `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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.148.3.10** `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::taa\\_tgba](#), [spot::tgba\\_bdd\\_concrete](#), [spot::tgba\\_explicit](#), [spot::tgba\\_kv\\_complement](#), [spot::tgba\\_product](#), [spot::tgba\\_safr\\_complement](#), [spot::tgba\\_scc](#), [spot::tgba\\_sgba\\_proxy](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).



**7.148.3.11 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.

**7.148.3.12 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.

**7.148.3.13 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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.148.4 Member Data Documentation****7.148.4.1 const state\* spot::tgba::last\_support\_conditions\_input\_ [mutable, private]****7.148.4.2 bdd spot::tgba::last\_support\_conditions\_output\_ [mutable, private]**

7.148.4.3 `const state* spot::tgba::last_support_variables_input_` [mutable, private]

7.148.4.4 `bdd spot::tgba::last_support_variables_output_` [mutable, private]

7.148.4.5 `int spot::tgba::num_acc_` [mutable, private]

The documentation for this class was generated from the following file:

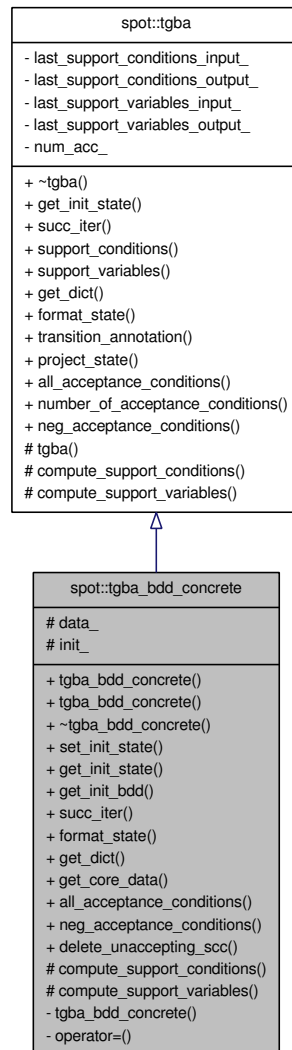
- [tgba/tgba.hh](#)

## 7.149 `spot::tgba_bdd_concrete` Class Reference

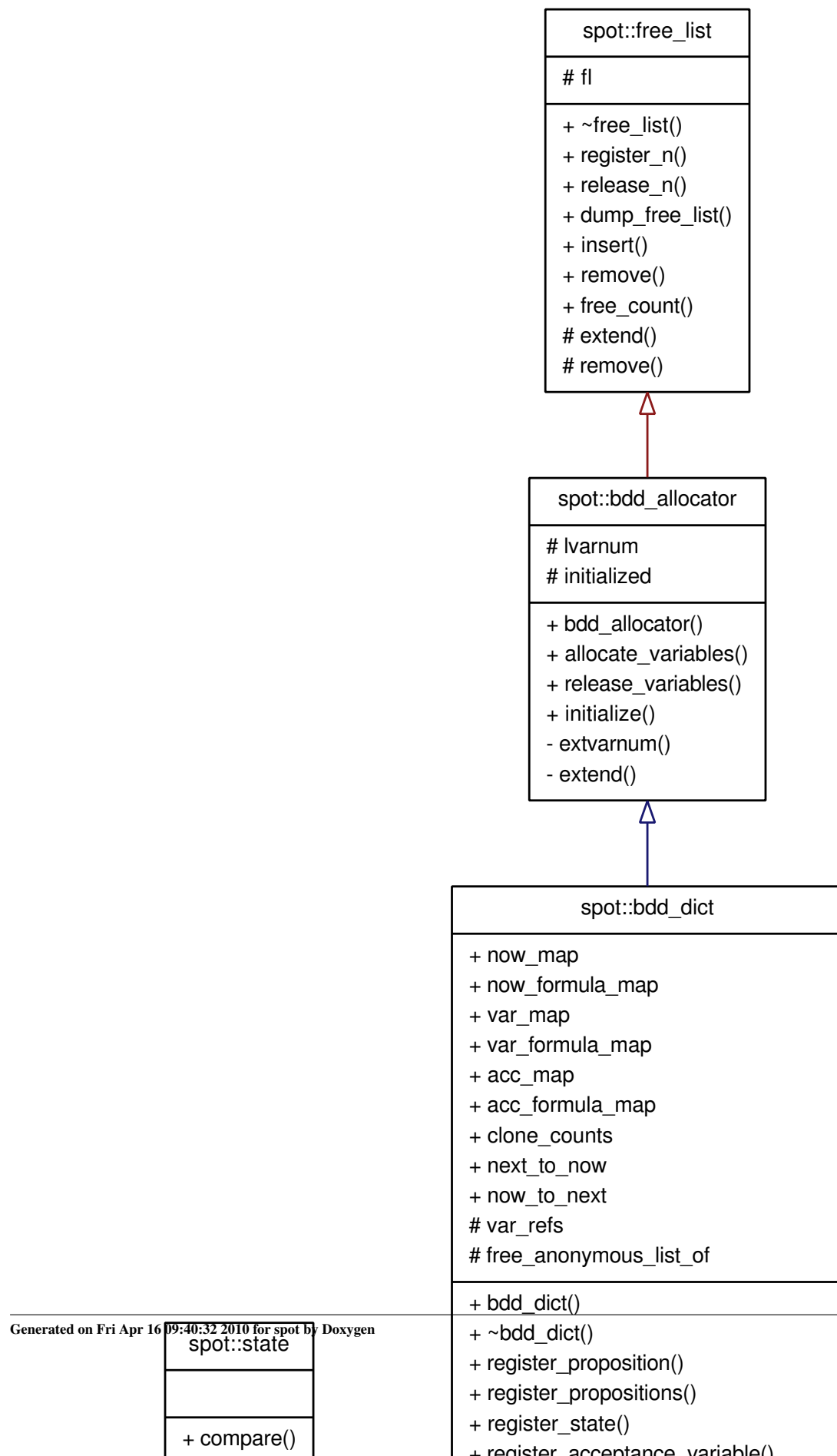
A concrete [spot::tgba](#) implemented using BDDs.

```
#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.*
- void [delete\\_unaccepting\\_scc](#) ()  
*Delete SCCs (Strongly Connected Components) from the TGBA which cannot be accepting.*
- bdd [support\\_conditions](#) (const [state](#) \*state) const  
*Get a formula that must hold whatever successor is taken.*
- bdd [support\\_variables](#) (const [state](#) \*state) const  
*Get the conjunctions of variables tested by the outgoing transitions of state.*
- 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](#) & operator= (const [tgba\\_bdd\\_concrete](#) &)

#### 7.149.1 Detailed Description

A concrete [spot::tgba](#) implemented using BDDs.

#### 7.149.2 Constructor & Destructor Documentation

##### 7.149.2.1 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const tgba\_bdd\_factory & fact)

Construct a [tgba\\_bdd\\_concrete](#) with unknown initial state.  
[set\\_init\\_state\(\)](#) should be called later.

##### 7.149.2.2 spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const tgba\_bdd\_factory & fact, bdd init)

Construct a [tgba\\_bdd\\_concrete](#) with known initial state.

**7.149.2.3** virtual spot::tgba\_bdd\_concrete::~~tgba\_bdd\_concrete () [virtual]

**7.149.2.4** spot::tgba\_bdd\_concrete::tgba\_bdd\_concrete (const tgba\_bdd\_concrete &)  
[private]

### 7.149.3 Member Function Documentation

**7.149.3.1** virtual bdd spot::tgba\_bdd\_concrete::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.149.3.2** virtual bdd spot::tgba\_bdd\_concrete::compute\_support\_conditions (const state \* state)  
const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.149.3.3** virtual bdd spot::tgba\_bdd\_concrete::compute\_support\_variables (const state \* state)  
const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.149.3.4** void spot::tgba\_bdd\_concrete::delete\_unaccepting\_scc ()

Delete SCCs (Strongly Connected Components) from the TGBA which cannot be accepting.

**7.149.3.5** 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 that owns the state.

Implements [spot::tgba](#).

### 7.149.3.6 const tgba\_bdd\_core\_data& spot::tgba\_bdd\_concrete::get\_core\_data () const

Get the core data associated to this automaton.

These data includes the various BDD used to represent the relation, encode variable sets, Next-to-Now rewrite rules, etc.

### 7.149.3.7 virtual bdd\_dict\* spot::tgba\_bdd\_concrete::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

### 7.149.3.8 bdd spot::tgba\_bdd\_concrete::get\_init\_bdd () const

Get the initial state directly as a BDD.

The sole point of this method is to prevent writing horrors such as

```
state_bdd* s = automata.get_init_state();
some_class some_instance(s->as_bdd());
delete s;
```

### 7.149.3.9 virtual state\_bdd\* spot::tgba\_bdd\_concrete::get\_init\_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

### 7.149.3.10 virtual bdd spot::tgba\_bdd\_concrete::neg\_acceptance\_conditions () const [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.



This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.149.3.11** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
`[virtual, inherited]`

The number of acceptance conditions.

**7.149.3.12** `tgba_bdd_concrete& spot::tgba_bdd_concrete::operator= (const tgba_bdd_concrete &) [private]`

**7.149.3.13** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const`  
`[virtual, inherited]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

#### Returns

0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.149.3.14** `virtual void spot::tgba_bdd_concrete::set_init_state (bdd s) [virtual]`

Set the initial state.

**7.149.3.15** `virtual tgba_succ_iterator_concrete* spot::tgba_bdd_concrete::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const`  
`[virtual]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand.

*global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by [succ\\_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by [succ\\_iter](#).

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

#### 7.149.3.16 bdd spot::tgba::support\_conditions (const state \* state) const [inherited]

Get a formula that must hold whatever successor is taken.

#### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by [succ\\_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute\\_support\\_conditions\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

#### 7.149.3.17 bdd spot::tgba::support\_variables (const state \* state) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some [succ\\_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute\\_support\\_variables\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

#### 7.149.3.18 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 it the empty string.

#### Parameters

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

#### 7.149.4 Member Data Documentation

##### 7.149.4.1 `tgba_bdd_core_data` `spot::tgba_bdd_concrete::data_` `[protected]`

Core data associated to the automaton.

##### 7.149.4.2 `bdd` `spot::tgba_bdd_concrete::init_` `[protected]`

Initial state.

The documentation for this class was generated from the following file:

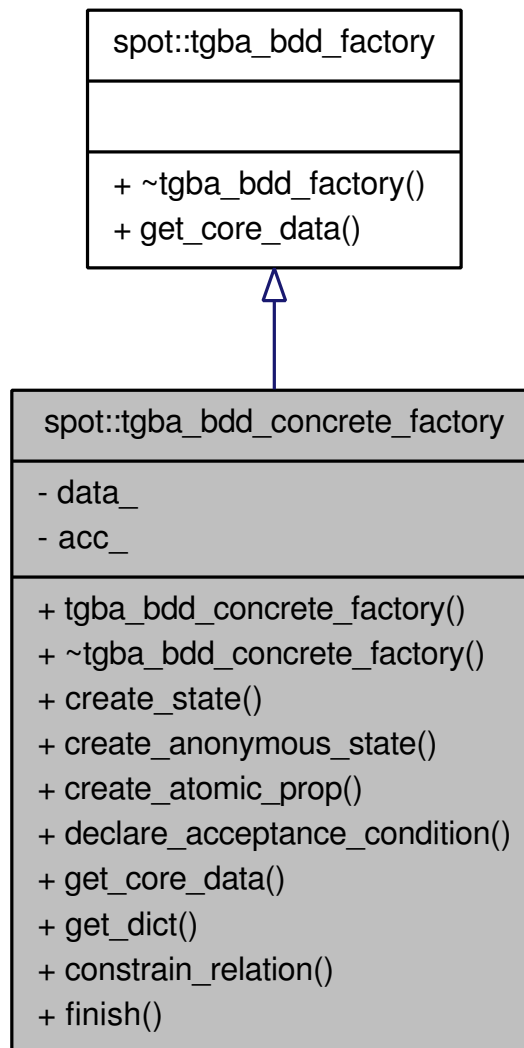
- [tgba/tgbabddconcrete.hh](#)

#### 7.150 `spot::tgba_bdd_concrete_factory` Class Reference

Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.

```
#include <tgba/tgbabddconcretefactory.hh>
```

Inheritance diagram for spot::tgba\_bdd\_concrete\_factory:



spot.free_list
# 11



- `tgba_bdd_concrete_factory (bdd_dict *dict)`
- `virtual ~tgba_bdd_concrete_factory ()`
- `int create_state (const ltl::formula *f)`
- `int create_anonymous_state ()`
- `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`

- `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.*

### 7.150.1 Detailed Description

Helper class to build a [spot::tgba\\_bdd\\_concrete](#) object.

### 7.150.2 Member Typedef Documentation

- 7.150.2.1** `typedef Sgi::hash_map<const ltl::formula*, bdd, ltl::formula_ptr_hash>  
spot::tgba_bdd_concrete_factory::acc_map_ [private]`

### 7.150.3 Constructor & Destructor Documentation

- 7.150.3.1** `spot::tgba_bdd_concrete_factory::tgba_bdd_concrete_factory (bdd_dict * dict)`

- 7.150.3.2** `virtual spot::tgba_bdd_concrete_factory::~tgba_bdd_concrete_factory () [virtual]`

### 7.150.4 Member Function Documentation

- 7.150.4.1** `void spot::tgba_bdd_concrete_factory::constrain_relation (bdd new_rel)`

Add a new constraint to the relation.

**7.150.4.2 int spot::tgba\_bdd\_concrete\_factory::create\_anonymous\_state ()**

Create a anonymous Now/Next variables.

**Returns**

The BDD variable number  $v$  for the Now state. The Next BDD corresponds to  $v+1$ .

**7.150.4.3 int spot::tgba\_bdd\_concrete\_factory::create\_atomic\_prop (const ltl::formula \*  $f$ )**

Create an atomic proposition variable for formula  $f$ .

**Parameters**

$f$  The formula to create an atomic proposition for.

**Returns**

The variable number for this state.

The atomic proposition is not created if it already exists. Instead its existing variable number is returned. Variable numbers can be turned into BDD using `ithvar()`.

**7.150.4.4 int spot::tgba\_bdd\_concrete\_factory::create\_state (const ltl::formula \*  $f$ )**

Create a Now/Next variables for formula  $f$ .

**Parameters**

$f$  The formula to create a state for.

**Returns**

The BDD variable number  $v$  for the Now state. The Next BDD corresponds to  $v+1$ .

The state variables are not created if they already exist. Instead their existing variable numbers are returned. Variable numbers can be turned into BDD using `ithvar()`.

**7.150.4.5 void spot::tgba\_bdd\_concrete\_factory::declare\_acceptance\_condition (bdd  $b$ , const ltl::formula \*  $a$ )**

Declare an acceptance condition.

Formula such as ' $\text{f U g}$ ' or ' $\text{F g}$ ' make the promise that ' $\text{g}$ ' 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 ' $\text{g}$ '.

**Parameters**

$b$  a BDD indicating which variables are in the acceptance set

$a$  the formula associated

#### 7.150.4.6 `void spot::tgba_bdd_concrete_factory::finish ()`

Perform final computations before the relation can be used.

This function should be called after all propositions, state, acceptance conditions, and constraints have been declared, and before calling `get_code_data()` or `get_dict()`.

#### 7.150.4.7 `const tgba_bdd_core_data& spot::tgba_bdd_concrete_factory::get_core_data () const [virtual]`

Get the core data for the new automata.

Implements `spot::tgba_bdd_factory`.

#### 7.150.4.8 `bdd_dict* spot::tgba_bdd_concrete_factory::get_dict () const`

### 7.150.5 Member Data Documentation

#### 7.150.5.1 `acc_map_ spot::tgba_bdd_concrete_factory::acc_ [private]`

BDD associated to each acceptance condition.

#### 7.150.5.2 `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/tgababddconcretfactory.hh](#)

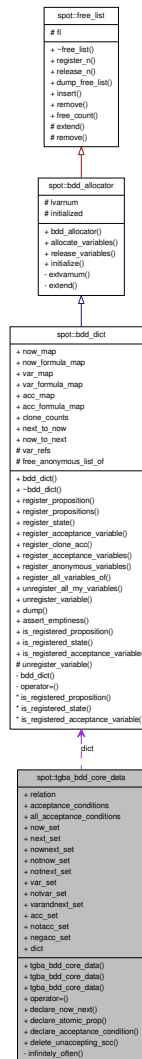
### 7.151 `spot::tgba_bdd_core_data` Struct Reference

Core data for a TGBA encoded using BDDs.

```
#include <tgba/tgababddcoredata.hh>
```



Collaboration diagram for spot::tgba\_bdd\_core\_data:



## Public Member Functions

- [tgba\\_bdd\\_core\\_data](#) ([bdd\\_dict](#) \*dict)

*Default constructor.*

- [tgba\\_bdd\\_core\\_data](#) (const [tgba\\_bdd\\_core\\_data](#) &copy)

*Copy constructor.*

- [tgba\\_bdd\\_core\\_data](#) (const [tgba\\_bdd\\_core\\_data](#) &left, const [tgba\\_bdd\\_core\\_data](#) &right)

*Merge two [tgba\\_bdd\\_core\\_data](#).*

- const [tgba\\_bdd\\_core\\_data](#) & operator= (const [tgba\\_bdd\\_core\\_data](#) &copy)
- void [declare\\_now\\_next](#) (bdd now, bdd next)

*Update the variable sets to take a new pair of variables into account.*

- void [declare\\_atomic\\_prop](#) (bdd var)  
*Update the variable sets to take a new atomic proposition into account.*
- void [declare\\_acceptance\\_condition](#) (bdd prom)  
*Update the variable sets to take a new acceptance condition into account.*
- void [delete\\_unaccepting\\_scc](#) (bdd init)  
*Delete SCCs (Strongly Connected Components) from the relation which cannot be accepting.*

### 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.*

### Private Member Functions

- `bdd infinitely_often` (`bdd s`, `bdd acc`, `bdd er`)

#### 7.151.1 Detailed Description

Core data for a TGBA encoded using BDDs.

#### 7.151.2 Constructor & Destructor Documentation

##### 7.151.2.1 `spot::tgba_bdd_core_data::tgba_bdd_core_data (bdd_dict * dict)`

Default constructor.

Initially all variable set are empty and the `relation` is true.

##### 7.151.2.2 `spot::tgba_bdd_core_data::tgba_bdd_core_data (const tgba_bdd_core_data & copy)`

Copy constructor.

##### 7.151.2.3 `spot::tgba_bdd_core_data::tgba_bdd_core_data (const tgba_bdd_core_data & left, const tgba_bdd_core_data & right)`

Merge two `tgba_bdd_core_data`.

This is used when building a product of two automata.

#### 7.151.3 Member Function Documentation

##### 7.151.3.1 `void spot::tgba_bdd_core_data::declare_acceptance_condition (bdd prom)`

Update the variable sets to take a new acceptance condition into account.

**7.151.3.2 void spot::tgba\_bdd\_core\_data::declare\_atomic\_prop (bdd *var*)**

Update the variable sets to take a new atomic proposition into account.

**7.151.3.3 void spot::tgba\_bdd\_core\_data::declare\_now\_next (bdd *now*, bdd *next*)**

Update the variable sets to take a new pair of variables into account.

**7.151.3.4 void spot::tgba\_bdd\_core\_data::delete\_unaccepting\_scc (bdd *init*)**

Delete SCCs (Strongly Connected Components) from the relation which cannot be accepting.

**7.151.3.5 bdd spot::tgba\_bdd\_core\_data::infinitely\_often (bdd *s*, bdd *acc*, bdd *er*) [private]****7.151.3.6 const tgba\_bdd\_core\_data& spot::tgba\_bdd\_core\_data::operator= (const tgba\_bdd\_core\_data & *copy*)****7.151.4 Member Data Documentation****7.151.4.1 bdd spot::tgba\_bdd\_core\_data::acc\_set**

The (positive) conjunction of all variables which are acceptance conditions.

**7.151.4.2 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  $F \cup b$ .

To summarize, `acceptance_conditions` contains three kinds of variables:

- "Next" variables, that encode the destination state,
- atomic propositions, which are things to verify before going on to the next state,
- "Acc" variables.

#### 7.151.4.3 `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.

#### 7.151.4.4 `bdd_dict* spot::tgba_bdd_core_data::dict`

The dictionary used by the automata.

#### 7.151.4.5 `bdd spot::tgba_bdd_core_data::negacc_set`

The negative conjunction of all variables which are acceptance conditions.

#### 7.151.4.6 `bdd spot::tgba_bdd_core_data::next_set`

The conjunction of all Next variables, in their positive form.

#### 7.151.4.7 `bdd spot::tgba_bdd_core_data::notacc_set`

The (positive) conjunction of all variables which are not acceptance conditions.

#### 7.151.4.8 `bdd spot::tgba_bdd_core_data::notnext_set`

The (positive) conjunction of all variables which are not Next variables.

**7.151.4.9 `bdd spot::tgba_bdd_core_data::notnow_set`**

The (positive) conjunction of all variables which are not Now variables.

**7.151.4.10 `bdd spot::tgba_bdd_core_data::notvar_set`**

The (positive) conjunction of all variables which are not atomic propositions.

**7.151.4.11 `bdd spot::tgba_bdd_core_data::now_set`**

The conjunction of all Now variables, in their positive form.

**7.151.4.12 `bdd spot::tgba_bdd_core_data::nownext_set`**

The conjunction of all Now and Next variables, in their positive form.

**7.151.4.13 `bdd spot::tgba_bdd_core_data::relation`**

encodes the transition relation of the TGBA.

`relation` uses three kinds of variables:

- "Now" variables, that encode the current state
- "Next" variables, that encode the destination state
- atomic propositions, which are things to verify before going on to the next state

**7.151.4.14 `bdd spot::tgba_bdd_core_data::var_set`**

The (positive) conjunction of all variables which are atomic propositions.

**7.151.4.15 `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:

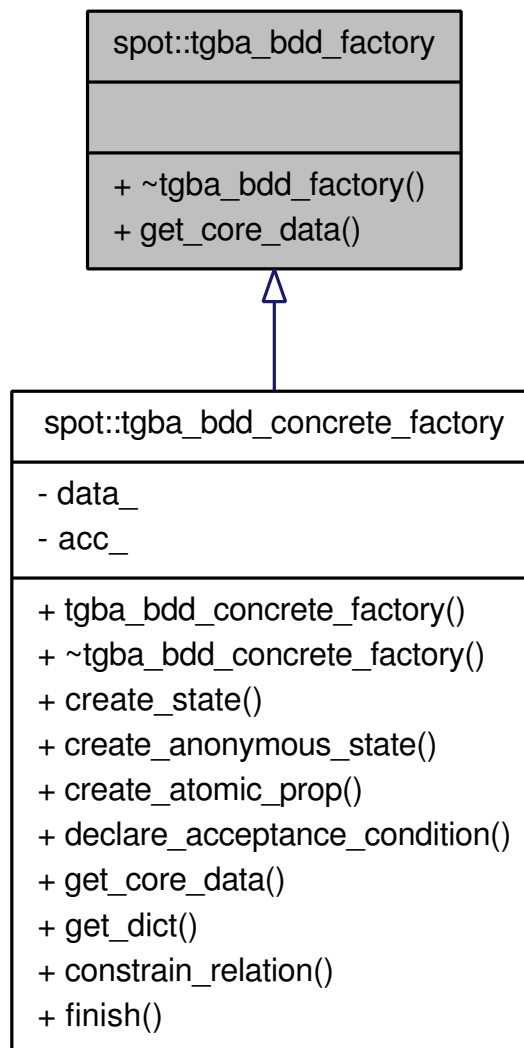
- [tgba/tgabddcoredata.hh](#)

## 7.152 spot::tgba\_bdd\_factory Class Reference

Abstract class for [spot::tgba\\_bdd\\_concrete](#) factories.

```
#include <tgba/tgbabddfactory.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.*

### 7.152.1 Detailed Description

Abstract class for [spot::tgba\\_bdd\\_concrete](#) factories. A [spot::tgba\\_bdd\\_concrete](#) can be constructed from anything that supplies core data and their associated dictionary.

### 7.152.2 Constructor & Destructor Documentation

**7.152.2.1** virtual spot::tgba\_bdd\_factory::~~tgba\_bdd\_factory () [inline, virtual]

### 7.152.3 Member Function Documentation

**7.152.3.1** 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:

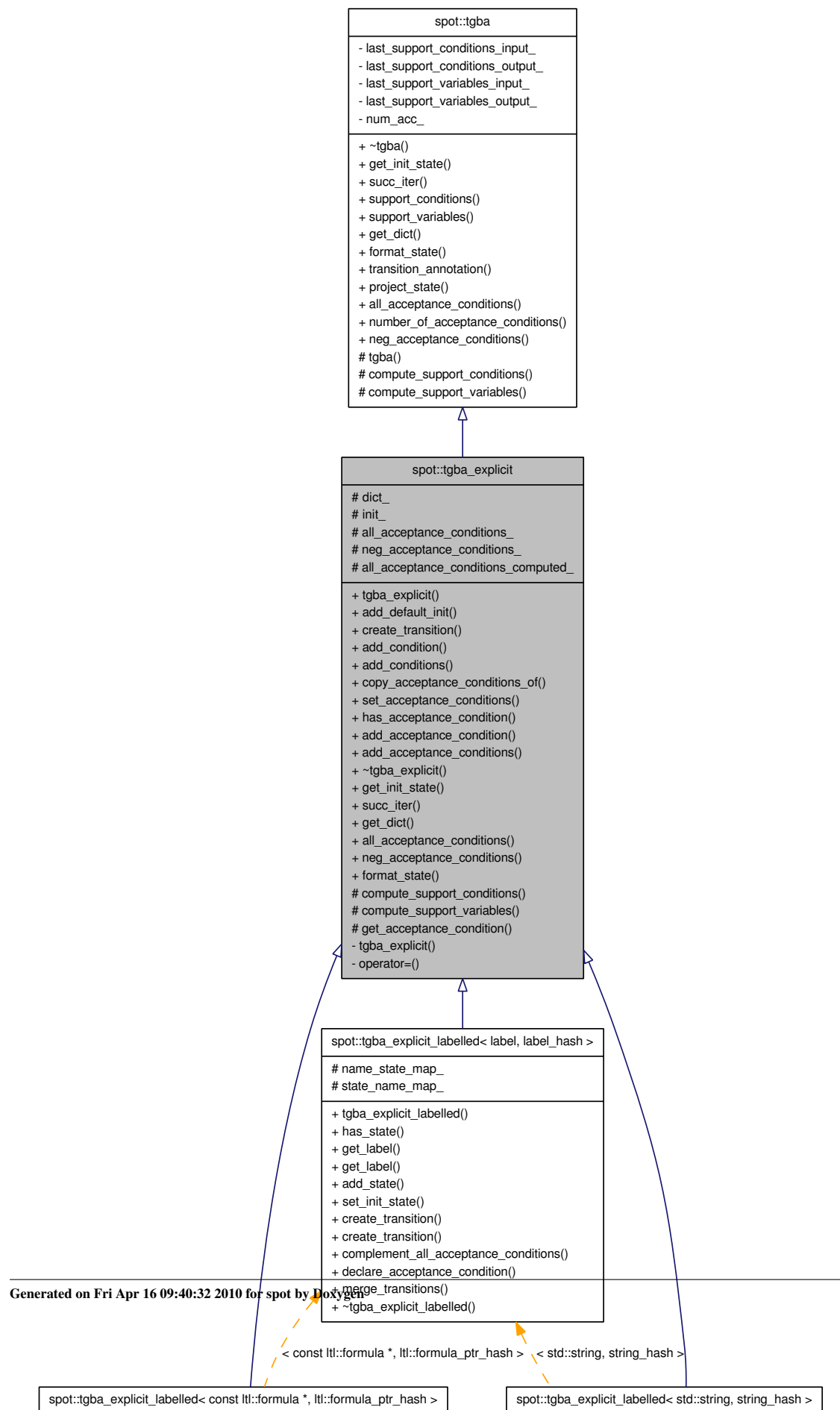
- [tgba/tgabddfactory.hh](#)

## 7.153 spot::tgba\_explicit Class Reference

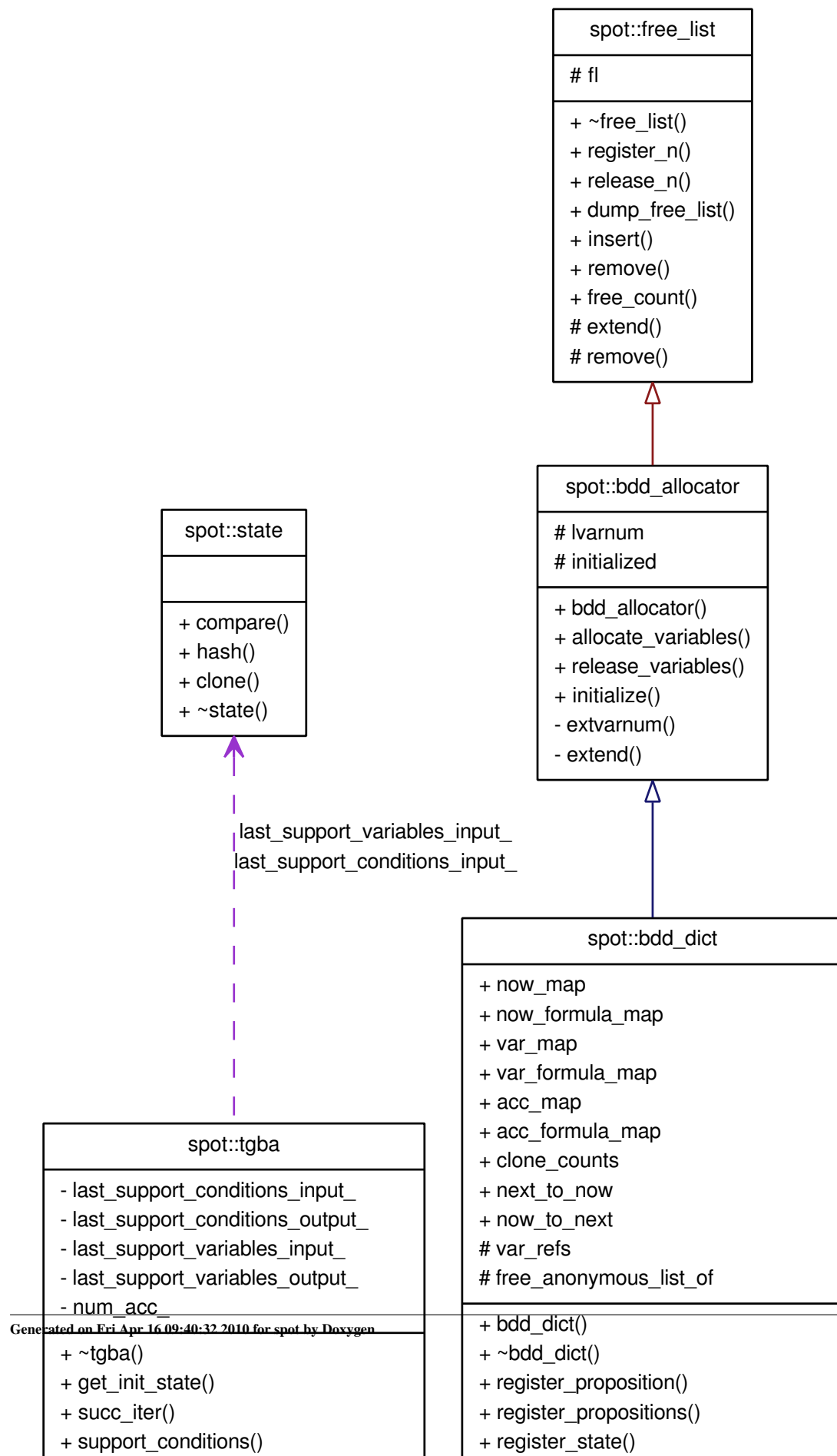
```
#include <tgba/tgbaexplicit.hh>
```



Inheritance diagram for spot::tgba\_explicit:



Collaboration diagram for spot::tgba\_explicit:



## Classes

- struct [transition](#)  
*Explicit transitions (used by [spot::tgba\\_explicit](#)).*

## Public Types

- typedef std::list< [transition](#) \* > [state](#)

## Public Member Functions

- [tgba\\_explicit](#) ([bdd\\_dict](#) \*dict)
- virtual [state](#) \* [add\\_default\\_init](#) ()=0  
*Add a default initial state.*
- [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 [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- void [set\\_acceptance\\_conditions](#) (bdd acc)  
*Set the acceptance conditions.*
- 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.*
- 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  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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.*
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const =0

*Format the state as a string for printing.*

- `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 spot::state *state) const`  
*Do the actual computation of `tgba::support_conditions()`.*
- `virtual bdd compute_support_variables (const spot::state *state) const`  
*Do the actual computation of `tgba::support_variables()`.*
- `bdd get_acceptance_condition (const ltl::formula *f)`

### Protected Attributes

- `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 & operator= (const tgba_explicit &other)`

#### 7.153.1 Detailed Description

Explicit representation of a `spot::tgba`.

#### 7.153.2 Member Typedef Documentation

##### 7.153.2.1 `typedef std::list<transition*> spot::tgba_explicit::state`

### 7.153.3 Constructor & Destructor Documentation

**7.153.3.1** `spot::tgba_explicit::tgba_explicit (bdd_dict * dict)`

**7.153.3.2** `virtual spot::tgba_explicit::~~tgba_explicit ()` **[virtual]**

**7.153.3.3** `spot::tgba_explicit::tgba_explicit (const tgba_explicit & other)` **[private]**

### 7.153.4 Member Function Documentation

**7.153.4.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f)`

**7.153.4.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)`

This assumes that all acceptance conditions in *f* are known from dict.

**7.153.4.3** `void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f)`

**7.153.4.4** `void spot::tgba_explicit::add_conditions (transition * t, bdd f)`

This assumes that all variables in *f* are known from dict.

**7.153.4.5** `virtual state* spot::tgba_explicit::add_default_init ()` **[pure virtual]**

Add a default initial state.

Implemented in [spot::tgba\\_explicit\\_string](#), and [spot::tgba\\_explicit\\_formula](#).

**7.153.4.6** `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](#).

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::complement_all_acceptance_conditions()`.

**7.153.4.7** `virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.153.4.8** `virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.153.4.9** `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.

**7.153.4.10** `transition* spot::tgba_explicit::create_transition (state * source, const state * dest)`

Reimplemented in [spot::tgba\\_explicit\\_labelled< label, label\\_hash >](#), [spot::tgba\\_explicit\\_labelled< const ltl::formula \\*, ltl::formula\\_ptr\\_hash >](#), and [spot::tgba\\_explicit\\_labelled< std::string, string\\_hash >](#).

**7.153.4.11** `virtual std::string spot::tgba_explicit::format_state (const spot::state * state) const [pure virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

Implemented in [spot::tgba\\_explicit\\_string](#), [spot::tgba\\_explicit\\_formula](#), and [spot::tgba\\_reduc](#).

**7.153.4.12** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula *f)`  
[protected]

**7.153.4.13** `virtual bdd_dict* spot::tgba_explicit::get_dict () const` [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.153.4.14** `virtual spot::state* spot::tgba_explicit::get_init_state () const` [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.153.4.15** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula *f) const`

**7.153.4.16** `virtual bdd spot::tgba_explicit::neg_acceptance_conditions () const` [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.153.4.17** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
[virtual, inherited]

The number of acceptance conditions.

**7.153.4.18** tgba\_explicit& spot::tgba\_explicit::operator= (const tgba\_explicit & *other*)  
[private]

**7.153.4.19** 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, spot::tgba\_scc, spot::tgba\_tba\_proxy, and spot::tgba\_union.

**7.153.4.20** void spot::tgba\_explicit::set\_acceptance\_conditions (bdd *acc*)

The the acceptance conditions.

**7.153.4.21** 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]

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.

Implements spot::tgba.



**7.153.4.22 `bdd spot::tgba::support_conditions (const state * state) const` [inherited]**

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.153.4.23 `bdd spot::tgba::support_variables (const state * state) const` [inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.153.4.24 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.153.5 Member Data Documentation****7.153.5.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` [mutable, protected]****7.153.5.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` [mutable, protected]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

### 7.153.5.3 bdd\_dict\* spot::tgba\_explicit::dict\_ [protected]

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::declare\_acceptance\_condition().

### 7.153.5.4 tgba\_explicit::state\* spot::tgba\_explicit::init\_ [protected]

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::add\_state(), and spot::tgba\_explicit\_labelled< std::string, string\_hash >::set\_init\_state().

### 7.153.5.5 bdd spot::tgba\_explicit::neg\_acceptance\_conditions\_ [protected]

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::declare\_acceptance\_condition().

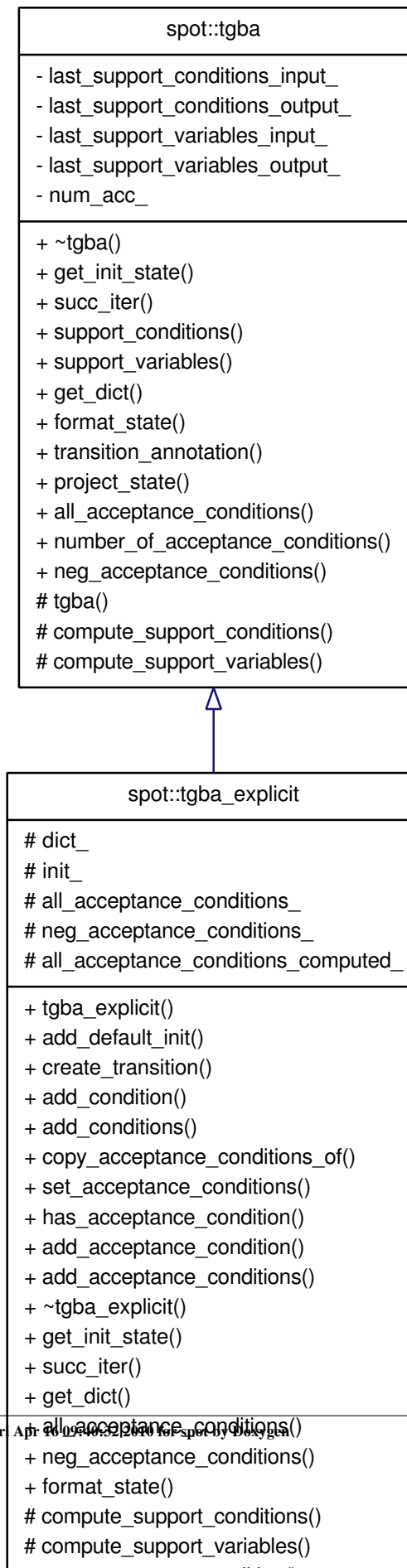
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

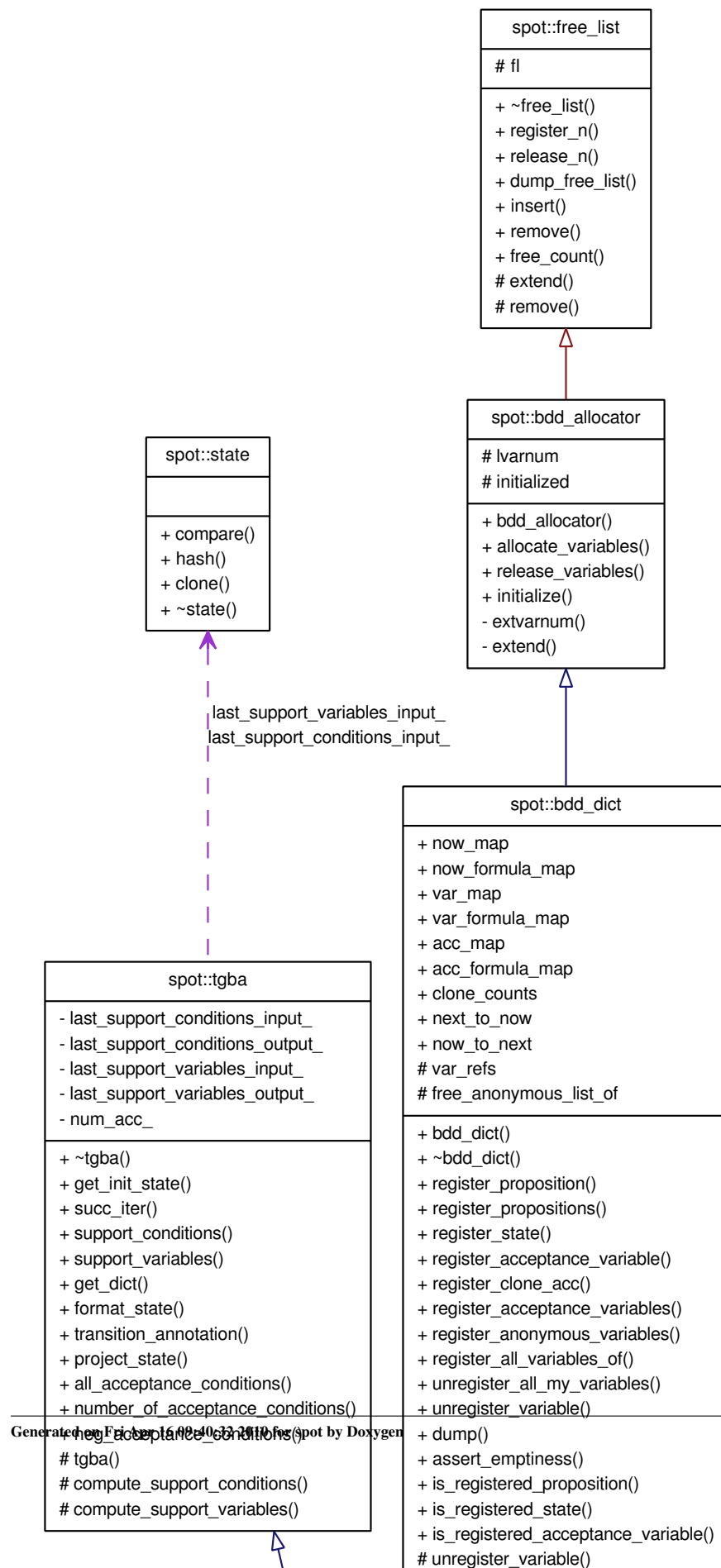
## 7.154 spot::tgba\_explicit\_formula Class Reference

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for spot::tgba\_explicit\_formula:



Collaboration diagram for spot::tgba\_explicit\_formula:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)

## Public Member Functions

- [tgba\\_explicit\\_formula](#) (bdd\_dict \*dict)
- virtual [~tgba\\_explicit\\_formula](#) ()
- virtual [state](#) \* [add\\_default\\_init](#) ()  
*Add a default initial state.*
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const  
*Format the state as a string for printing.*
- bool [has\\_state](#) (const const [ltl::formula](#) \*&name)
- const const [ltl::formula](#) \*& [get\\_label](#) (const [tgba\\_explicit::state](#) \*s) const
- const const [ltl::formula](#) \*& [get\\_label](#) (const [spot::state](#) \*s) const
- [state](#) \* [add\\_state](#) (const const [ltl::formula](#) \*&name)
- [state](#) \* [set\\_init\\_state](#) (const const [ltl::formula](#) \*&state)
- [transition](#) \* [create\\_transition](#) ([state](#) \*source, const [state](#) \*dest)
- [transition](#) \* [create\\_transition](#) (const const [ltl::formula](#) \*&source, const const [ltl::formula](#) \*&dest)
- void [complement\\_all\\_acceptance\\_conditions](#) ()
- void [declare\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- void [merge\\_transitions](#) ()
- 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 [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- void [set\\_acceptance\\_conditions](#) (bdd acc)  
*The the acceptance conditions.*
- 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.*
- 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  
*Get an iterator over the successors of local\_state.*
- virtual bdd\_dict \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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 Types

- typedef const [ltl::formula](#) \* [label\\_t](#)
- typedef Sgi::hash\_map< const [ltl::formula](#) \*, [tgba\\_explicit::state](#) \*, [ltl::formula\\_ptr\\_hash](#) > [ns\\_map](#)
- typedef Sgi::hash\_map< const [tgba\\_explicit::state](#) \*, const [ltl::formula](#) \*, [ptr\\_hash](#)< [tgba\\_explicit::state](#) > > [sn\\_map](#)

### Protected Member Functions

- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*
- bdd [get\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)

### 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\\_](#)

### 7.154.1 Member Typedef Documentation

- 7.154.1.1** `typedef const ltl::formula * spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::label_t [protected, inherited]`
- 7.154.1.2** `typedef Sgi::hash_map<const ltl::formula * , tgba_explicit::state*, ltl::formula_ptr_hash > spot::tgba_explicit_labelled< const ltl::formula * , ltl::formula_ptr_hash >::ns_map [protected, inherited]`
- 7.154.1.3** `typedef Sgi::hash_map<const tgba_explicit::state*, const ltl::formula * , ptr_hash<tgba_explicit::state> > spot::tgba_explicit_labelled< const ltl::formula * , ltl::formula_ptr_hash >::sn_map [protected, inherited]`
- 7.154.1.4** `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

### 7.154.2 Constructor & Destructor Documentation

- 7.154.2.1** `spot::tgba_explicit_formula::tgba_explicit_formula (bdd_dict * dict) [inline]`
- 7.154.2.2** `virtual spot::tgba_explicit_formula::~~tgba_explicit_formula () [virtual]`

### 7.154.3 Member Function Documentation

- 7.154.3.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`
- 7.154.3.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]`

This assumes that all acceptance conditions in  $f$  are known from dict.

**7.154.3.3** void spot::tgba\_explicit::add\_condition (transition \* *t*, const ltl::formula \* *f*)  
[inherited]

**7.154.3.4** void spot::tgba\_explicit::add\_conditions (transition \* *t*, bdd *f*) [inherited]

This assumes that all variables in *f* are known from dict.

**7.154.3.5** virtual state\* spot::tgba\_explicit\_formula::add\_default\_init () [virtual]

Add a default initial state.

Implements [spot::tgba\\_explicit](#).

**7.154.3.6** state\* spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash  
>::add\_state (const const ltl::formula \* & *name*) [inline, inherited]

Return the tgba\_explicit::state for *name*, creating the state if it does not exist.

**7.154.3.7** virtual bdd spot::tgba\_explicit::all\_acceptance\_conditions () const [virtual,  
inherited]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::complement\_all\_acceptance\_conditions().

**7.154.3.8** void spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash  
>::complement\_all\_acceptance\_conditions () [inline, inherited]

**7.154.3.9** virtual bdd spot::tgba\_explicit::compute\_support\_conditions (const spot::state \* *state*)  
const [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).



**7.154.3.10** `virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state)  
const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.154.3.11** `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.

**7.154.3.12** `transition* spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash  
>::create_transition (const const ltl::formula * & source, const const ltl::formula * &  
dest) [inline, inherited]`

**7.154.3.13** `transition* spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash  
>::create_transition (state * source, const state * dest) [inline, inherited]`

Reimplemented from [spot::tgba\\_explicit](#).

**7.154.3.14** `void spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash  
>::declare_acceptance_condition (const ltl::formula * f) [inline, inherited]`

**7.154.3.15** `virtual std::string spot::tgba_explicit_formula::format_state (const spot::state * state)  
const [virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba\\_explicit](#).

**7.154.3.16** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)  
[protected, inherited]`

**7.154.3.17 virtual bdd\_dict\* spot::tgba\_explicit::get\_dict () const [virtual, inherited]**

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.154.3.18 virtual spot::state\* spot::tgba\_explicit::get\_init\_state () const [virtual, inherited]**

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.154.3.19 const const ltl::formula \* & spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash >::get\_label (const spot::state \* s) const [inline, inherited]****7.154.3.20 const const ltl::formula \* & spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash >::get\_label (const tgba\_explicit::state \* s) const [inline, inherited]****7.154.3.21 bool spot::tgba\_explicit::has\_acceptance\_condition (const ltl::formula \* f) const [inherited]****7.154.3.22 bool spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash >::has\_state (const const ltl::formula \* & name) [inline, inherited]****7.154.3.23 void spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash >::merge\_transitions () [inline, inherited]**

### 7.154.3.24 virtual bdd spot::tgba\_explicit::neg\_acceptance\_conditions () const [virtual, inherited]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

### 7.154.3.25 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

### 7.154.3.26 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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.154.3.27 void spot::tgba\_explicit::set\_acceptance\_conditions (bdd acc) [inherited]

The the acceptance conditions.

### 7.154.3.28 state\* spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash >::set\_init\_state (const const ltl::formula \* & state) [inline, inherited]

**7.154.3.29** `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]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

#### Parameters

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**7.154.3.30** `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

#### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.154.3.31** `bdd spot::tgba::support_variables (const state * state) const [inherited]`

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.154.3.32** `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](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

### 7.154.4 Member Data Documentation

**7.154.4.1** `bdd spot::tgba_explicit::all_acceptance_conditions_` **[mutable, protected, inherited]**

**7.154.4.2** `bool spot::tgba_explicit::all_acceptance_conditions_computed_` **[mutable, protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.154.4.3** `bdd_dict* spot::tgba_explicit::dict_` **[protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.154.4.4** `tgba_explicit::state* spot::tgba_explicit::init_` **[protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::set_init_state()`.

**7.154.4.5** `ns_map spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::name_state_map_` **[protected, inherited]**

**7.154.4.6** `bdd spot::tgba_explicit::neg_acceptance_conditions_` **[protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

7.154.4.7 sn\_map spot::tgba\_explicit\_labelled< const ltl::formula \*, ltl::formula\_ptr\_hash  
>::state\_name\_map\_ [protected, inherited]

The documentation for this class was generated from the following file:

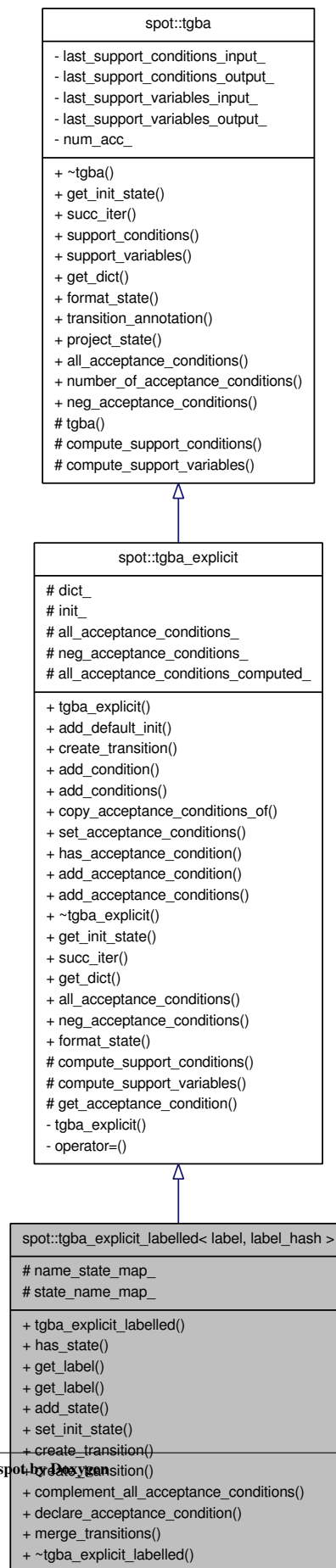
- [tgba/tgbaexplicit.hh](#)

## 7.155 spot::tgba\_explicit\_labelled< label, label\_hash > Class Template Reference

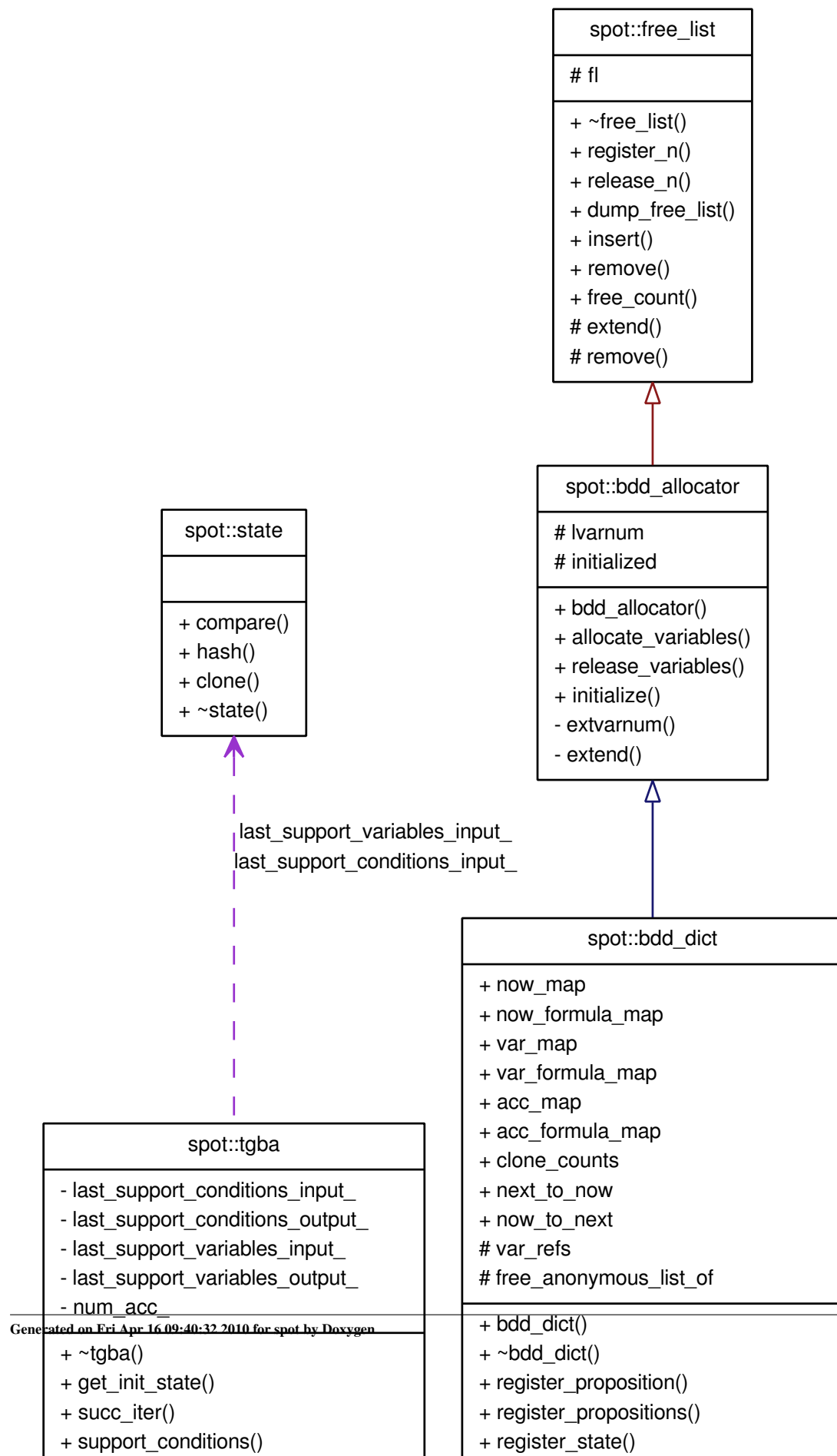
A [tgba\\_explicit](#) instance with states labeled by a given type.

```
#include <tgba/tgbaexplicit.hh>
```

Inheritance diagram for spot::tgba\_explicit\_labelled< label, label\_hash >:



Collaboration diagram for spot::tgba\_explicit\_labelled< label, label\_hash >:





## Public Types

- `typedef std::list< transition * > state`

## Public Member Functions

- `tgba\_explicit\_labelled (bdd_dict *dict)`
- `bool has\_state (const label &name)`
- `const label & get\_label (const tgba\_explicit::state *s) const`
- `const label & get\_label (const spot::state *s) const`
- `state * add\_state (const label &name)`
- `state * set\_init\_state (const label &state)`
- `transition * create\_transition (state *source, const state *dest)`
- `transition * create\_transition (const label &source, const label &dest)`
- `void complement\_all\_acceptance\_conditions ()`
- `void declare\_acceptance\_condition (const ltl::formula *f)`
- `void merge\_transitions ()`
- `virtual ~tgba\_explicit\_labelled ()`
- `virtual state * add\_default\_init ()=0`  
*Add a default initial state.*
- `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 copy\_acceptance\_conditions\_of (const tgba *a)`  
*Copy the acceptance conditions of a tgba.*
- `void set\_acceptance\_conditions (bdd acc)`  
*The the acceptance conditions.*
- `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.*
- `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`  
*Get an iterator over the successors of local\_state.*
- `virtual bdd_dict * get\_dict () const`  
*Get the dictionary associated to the 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.*

- virtual `std::string format_state` (const `spot::state *s`) const =0  
*Format the state as a string for printing.*
- 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 Types

- typedef label `label_t`
- typedef `Sgi::hash_map< label, tgba_explicit::state *, label_hash >` `ns_map`
- typedef `Sgi::hash_map< const tgba_explicit::state *, label, ptr_hash< tgba_explicit::state > >` `sn_map`

### Protected Member Functions

- virtual bdd `compute_support_conditions` (const `spot::state *state`) const  
*Do the actual computation of `tgba::support_conditions()`.*
- virtual bdd `compute_support_variables` (const `spot::state *state`) const  
*Do the actual computation of `tgba::support_variables()`.*
- bdd `get_acceptance_condition` (const `ltl::formula *f`)

### 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_`

### 7.155.1 Detailed Description

`template<typename label, typename label_hash> class spot::tgba_explicit_labelled< label, label_hash >`

A [tgba\\_explicit](#) instance with states labeled by a given type.

### 7.155.2 Member Typedef Documentation

**7.155.2.1** `template<typename label, typename label_hash> typedef label spot::tgba_explicit_labelled< label, label_hash >::label_t [protected]`

**7.155.2.2** `template<typename label, typename label_hash> typedef Sgi::hash_map<label, tgba_explicit::state*, label_hash> spot::tgba_explicit_labelled< label, label_hash >::ns_map [protected]`

**7.155.2.3** `template<typename label, typename label_hash> typedef Sgi::hash_map<const tgba_explicit::state*, label, ptr_hash<tgba_explicit::state> > spot::tgba_explicit_labelled< label, label_hash >::sn_map [protected]`

**7.155.2.4** `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

### 7.155.3 Constructor & Destructor Documentation

**7.155.3.1** `template<typename label, typename label_hash> spot::tgba_explicit_labelled< label, label_hash >::tgba_explicit_labelled (bdd_dict * dict) [inline]`

**7.155.3.2** `template<typename label, typename label_hash> virtual spot::tgba_explicit_labelled< label, label_hash >::~~tgba_explicit_labelled () [inline, virtual]`

### 7.155.4 Member Function Documentation

**7.155.4.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`

**7.155.4.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f)`  
**[inherited]**

This assumes that all acceptance conditions in *f* are known from dict.

**7.155.4.3** `void spot::tgba_explicit::add_condition (transition * t, const ltl::formula * f)`  
**[inherited]**

**7.155.4.4** `void spot::tgba_explicit::add_conditions (transition * t, bdd f)` **[inherited]**

This assumes that all variables in *f* are known from dict.

**7.155.4.5** `virtual state* spot::tgba_explicit::add_default_init ()` **[pure virtual, inherited]**

Add a default initial state.

Implemented in [spot::tgba\\_explicit\\_string](#), and [spot::tgba\\_explicit\\_formula](#).

**7.155.4.6** `template<typename label, typename label_hash> state* spot::tgba_explicit_labelled< label, label_hash >::add_state (const label & name)` **[inline]**

Return the [tgba\\_explicit::state](#) for *name*, creating the state if it does not exist.

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::create_transition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::set_init_state()`.

**7.155.4.7** `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](#).

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::complement_all_acceptance_conditions()`.

**7.155.4.8** `template<typename label, typename label_hash> void spot::tgba_explicit_labelled< label, label_hash >::complement_all_acceptance_conditions () [inline]`

**7.155.4.9** `virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.155.4.10** `virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.155.4.11** `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.

**7.155.4.12** `template<typename label, typename label_hash> transition* spot::tgba_explicit_labelled< label, label_hash >::create_transition (const label & source, const label & dest) [inline]`

**7.155.4.13** `template<typename label, typename label_hash> transition* spot::tgba_explicit_labelled< label, label_hash >::create_transition (state * source, const state * dest) [inline]`

Reimplemented from [spot::tgba\\_explicit](#).

Referenced by [spot::tgba\\_explicit\\_labelled< std::string, string\\_hash >::create\\_transition\(\)](#).

**7.155.4.14** `template<typename label, typename label_hash> void spot::tgba_explicit_labelled< label, label_hash >::declare_acceptance_condition (const ltl::formula * f) [inline]`

**7.155.4.15** `virtual std::string spot::tgba_explicit::format_state (const spot::state * state) const`  
`[pure virtual, inherited]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

Implemented in [spot::tgba\\_explicit\\_string](#), [spot::tgba\\_explicit\\_formula](#), and [spot::tgba\\_reduc](#).

**7.155.4.16** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f)`  
`[protected, inherited]`

**7.155.4.17** `virtual bdd_dict* spot::tgba_explicit::get_dict () const` `[virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.155.4.18** `virtual spot::state* spot::tgba_explicit::get_init_state () const` `[virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.155.4.19** `template<typename label, typename label_hash> const label&`  
`spot::tgba_explicit_labelled< label, label_hash >::get_label (const spot::state * s) const`  
`[inline]`

**7.155.4.20** `template<typename label, typename label_hash> const label&`  
`spot::tgba_explicit_labelled< label, label_hash >::get_label (const tgba_explicit::state`  
`* s) const` `[inline]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::get_label()`.

**7.155.4.21** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const`  
`[inherited]`

**7.155.4.22** `template<typename label, typename label_hash> bool spot::tgba_explicit_labelled<`  
`label, label_hash >::has_state (const label & name) [inline]`

**7.155.4.23** `template<typename label, typename label_hash> void spot::tgba_explicit_labelled<`  
`label, label_hash >::merge_transitions () [inline]`

**7.155.4.24** `virtual bdd spot::tgba_explicit::neg_acceptance_conditions () const [virtual,`  
`inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.155.4.25** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
`[virtual, inherited]`

The number of acceptance conditions.

**7.155.4.26** `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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.155.4.27** `void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]`

The the acceptance conditions.

**7.155.4.28** `template<typename label, typename label_hash> state* spot::tgba_explicit_labelled< label, label_hash >::set_init_state (const label & state) [inline]`**7.155.4.29** `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]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

**Parameters**

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**7.155.4.30** `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.



**7.155.4.31 `bdd spot::tgba::support_variables (const state * state) const` [inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.155.4.32 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.155.5 Member Data Documentation****7.155.5.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` [mutable, protected, inherited]****7.155.5.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` [mutable, protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.155.5.3 `bdd_dict* spot::tgba_explicit::dict_` [protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.155.5.4 `tgba_explicit::state* spot::tgba_explicit::init_` [protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::set_init_state()`.

#### 7.155.5.5 `template<typename label, typename label_hash> ns_map spot::tgba_explicit_labelled<label, label_hash>::name_state_map_` `[protected]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, `spot::tgba_explicit_labelled< std::string, string_hash >::complement_all_acceptance_conditions()`, `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`, `spot::tgba_explicit_labelled< std::string, string_hash >::has_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::merge_transitions()`.

#### 7.155.5.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_` `[protected, inherited]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.155.5.7 `template<typename label, typename label_hash> sn_map spot::tgba_explicit_labelled<label, label_hash>::state_name_map_` `[protected]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::get_label()`.

The documentation for this class was generated from the following file:

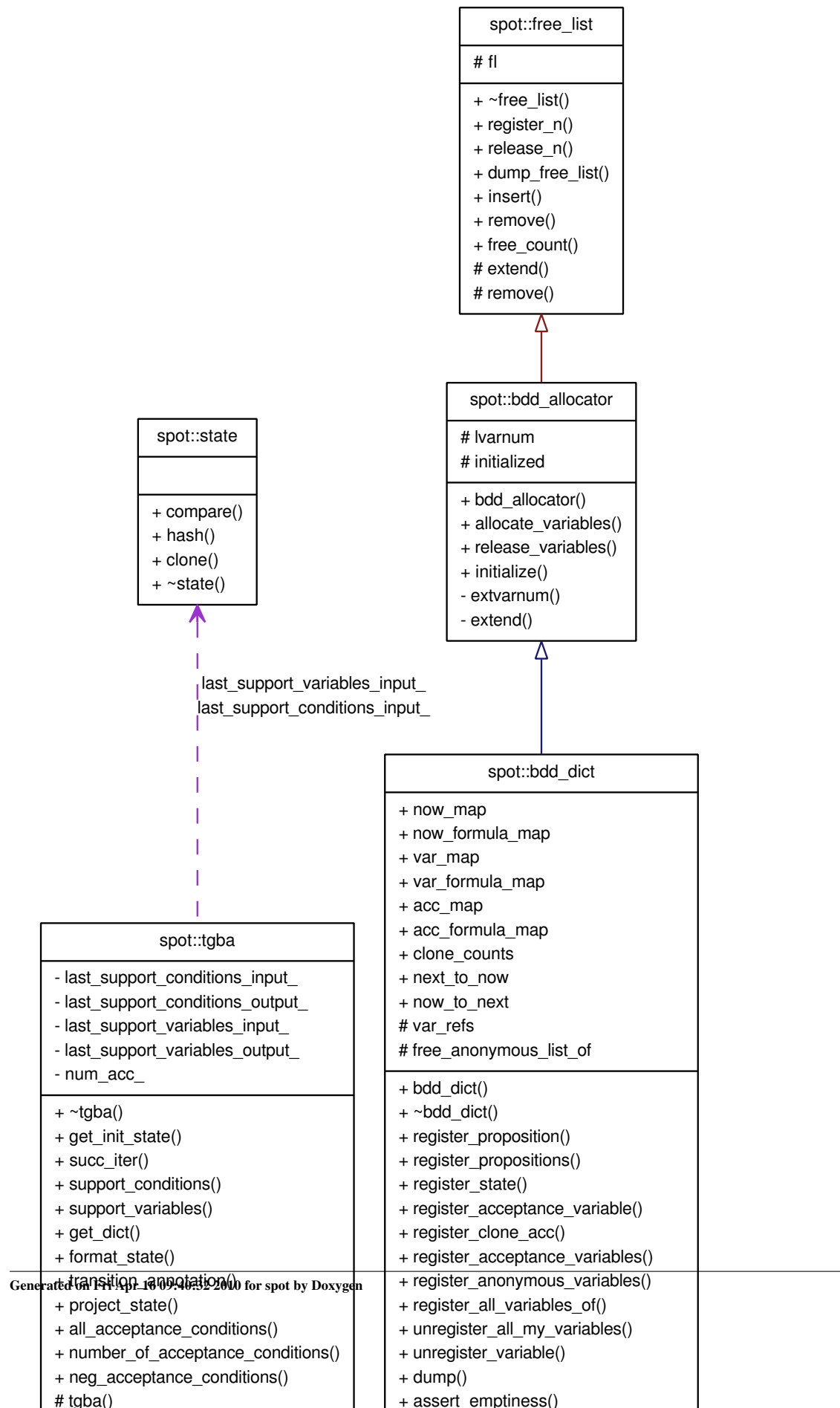
- [tgba/tgbaexplicit.hh](#)

## 7.156 spot::tgba\_explicit\_string Class Reference

```
#include <tgba/tgbaexplicit.hh>
```



Collaboration diagram for spot::tgba\_explicit\_string:



## Public Types

- typedef std::list< [transition](#) \* > [state](#)

## Public Member Functions

- [tgba\\_explicit\\_string](#) (bdd\_dict \*dict)
- virtual ~[tgba\\_explicit\\_string](#) ()
- virtual [state](#) \* [add\\_default\\_init](#) ()  
*Add a default initial state.*
- virtual std::string [format\\_state](#) (const [spot::state](#) \*s) const  
*Format the state as a string for printing.*
- bool [has\\_state](#) (const std::string &name)
- const std::string & [get\\_label](#) (const [tgba\\_explicit::state](#) \*s) const
- const std::string & [get\\_label](#) (const [spot::state](#) \*s) const
- [state](#) \* [add\\_state](#) (const std::string &name)
- [state](#) \* [set\\_init\\_state](#) (const std::string &state)
- [transition](#) \* [create\\_transition](#) ([state](#) \*source, const [state](#) \*dest)
- [transition](#) \* [create\\_transition](#) (const std::string &source, const std::string &dest)
- void [complement\\_all\\_acceptance\\_conditions](#) ()
- void [declare\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- void [merge\\_transitions](#) ()
- 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 [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- void [set\\_acceptance\\_conditions](#) (bdd acc)  
*Set the acceptance conditions.*
- 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.*
- 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  
*Get an iterator over the successors of local\_state.*
- virtual bdd\_dict \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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 Types

- typedef std::string [label\\_t](#)
- typedef Sgi::hash\_map< 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  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*
- bdd [get\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)

### 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\\_](#)

### 7.156.1 Member Typedef Documentation

**7.156.1.1** `typedef std::string spot::tgba_explicit_labelled< std::string , string_hash >::label_t [protected, inherited]`

**7.156.1.2** `typedef Sgi::hash_map<std::string , tgba_explicit::state*, string_hash > spot::tgba_explicit_labelled< std::string , string_hash >::ns_map [protected, inherited]`

**7.156.1.3** `typedef Sgi::hash_map<const tgba_explicit::state*, std::string , ptr_hash<tgba_explicit::state> > spot::tgba_explicit_labelled< std::string , string_hash >::sn_map [protected, inherited]`

**7.156.1.4** `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

### 7.156.2 Constructor & Destructor Documentation

**7.156.2.1** `spot::tgba_explicit_string::tgba_explicit_string (bdd_dict * dict) [inline]`

**7.156.2.2** `virtual spot::tgba_explicit_string::~~tgba_explicit_string () [virtual]`

### 7.156.3 Member Function Documentation

**7.156.3.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`

**7.156.3.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]`

This assumes that all acceptance conditions in  $f$  are known from dict.

**7.156.3.3** void spot::tgba\_explicit::add\_condition (transition \* *t*, const ltl::formula \* *f*)  
[*inherited*]

**7.156.3.4** void spot::tgba\_explicit::add\_conditions (transition \* *t*, bdd *f*) [*inherited*]

This assumes that all variables in *f* are known from dict.

**7.156.3.5** virtual state\* spot::tgba\_explicit\_string::add\_default\_init () [*virtual*]

Add a default initial state.

Implements [spot::tgba\\_explicit](#).

**7.156.3.6** state\* spot::tgba\_explicit\_labelled< std::string, string\_hash >::add\_state (const std::string & *name*) [*inline*, *inherited*]

Return the tgba\_explicit::state for *name*, creating the state if it does not exist.

References spot::tgba\_explicit::init\_, spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_, and spot::tgba\_explicit\_labelled< label, label\_hash >::state\_name\_map\_.

**7.156.3.7** virtual bdd spot::tgba\_explicit::all\_acceptance\_conditions () const [*virtual*, *inherited*]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::complement\_all\_acceptance\_conditions().

**7.156.3.8** void spot::tgba\_explicit\_labelled< std::string, string\_hash >::complement\_all\_acceptance\_conditions () [*inline*, *inherited*]

References spot::tgba\_explicit::all\_acceptance\_conditions(), and spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_.



**7.156.3.9** `virtual bdd spot::tgba_explicit::compute_support_conditions (const spot::state * state)  
const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.156.3.10** `virtual bdd spot::tgba_explicit::compute_support_variables (const spot::state * state)  
const [protected, virtual, inherited]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.156.3.11** `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.

**7.156.3.12** `transition* spot::tgba_explicit_labelled< std::string , string_hash >::create_transition  
(const std::string & source, const std::string & dest) [inline, inherited]`

References [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::add\\_state\(\)](#), and [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::create\\_transition\(\)](#).

**7.156.3.13** `transition* spot::tgba_explicit_labelled< std::string , string_hash >::create_transition  
(state * source, const state * dest) [inline, inherited]`

Reimplemented from [spot::tgba\\_explicit](#).

References [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::create\\_transition\(\)](#).

**7.156.3.14** `void spot::tgba_explicit_labelled< std::string , string_hash  
>::declare_acceptance_condition (const ltl::formula *f) [inline, inherited]`

References [spot::tgba\\_explicit::all\\_acceptance\\_conditions\\_computed\\_](#), [spot::ltl::formula::destroy\(\)](#), [spot::tgba\\_explicit::dict\\_](#), [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::name\\_state\\_map\\_](#), [spot::tgba\\_explicit::neg\\_acceptance\\_conditions\\_](#), and [spot::bdd\\_dict::register\\_acceptance\\_variable\(\)](#).

**7.156.3.15** `virtual std::string spot::tgba_explicit_string::format_state (const spot::state * state) const [virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba\\_explicit](#).

Reimplemented in [spot::tgba\\_reduc](#).

**7.156.3.16** `bdd spot::tgba_explicit::get_acceptance_condition (const ltl::formula * f) [protected, inherited]`

**7.156.3.17** `virtual bdd_dict* spot::tgba_explicit::get_dict () const [virtual, inherited]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.156.3.18** `virtual spot::state* spot::tgba_explicit::get_init_state () const [virtual, inherited]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.156.3.19** `const std::string & spot::tgba_explicit_labelled< std::string, string_hash >::get_label (const spot::state * s) const [inline, inherited]`

References [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::get\\_label\(\)](#), and [spot::state\\_explicit::get\\_state\(\)](#).

**7.156.3.20** `const std::string & spot::tgba_explicit_labelled< std::string, string_hash >::get_label (const tgba_explicit::state * s) const [inline, inherited]`

References [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::state\\_name\\_map\\_](#).

**7.156.3.21** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula *f) const`  
**[inherited]**

**7.156.3.22** `bool spot::tgba_explicit_labelled< std::string, string_hash >::has_state (const std::string &name) [inline, inherited]`

References `spot::tgba_explicit_labelled< label, label_hash >::name_state_map_`.

**7.156.3.23** `void spot::tgba_explicit_labelled< std::string, string_hash >::merge_transitions ()`  
**[inline, inherited]**

References `spot::tgba_explicit_labelled< label, label_hash >::name_state_map_`.

**7.156.3.24** `virtual bdd spot::tgba_explicit::neg_acceptance_conditions () const [virtual, inherited]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.156.3.25** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
**[virtual, inherited]**

The number of acceptance conditions.

**7.156.3.26** `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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.156.3.27 void spot::tgba\_explicit::set\_acceptance\_conditions (bdd acc) [inherited]

The the acceptance conditions.

### 7.156.3.28 state\* spot::tgba\_explicit\_labelled< std::string , string\_hash >::set\_init\_state (const std::string & state) [inline, inherited]

References [spot::tgba\\_explicit\\_labelled< label, label\\_hash >::add\\_state\(\)](#), and [spot::tgba\\_explicit::init\\_](#).

### 7.156.3.29 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]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by [succ\\_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by [succ\\_iter](#).

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

### 7.156.3.30 bdd spot::tgba::support\_conditions (const state \* state) const [inherited]

Get a formula that must hold whatever successor is taken.

#### Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.156.3.31 `bdd spot::tgba::support_variables (const state * state) const` **[inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

#### 7.156.3.32 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

### 7.156.4 Member Data Documentation

#### 7.156.4.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` **[mutable, protected, inherited]**

#### 7.156.4.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` **[mutable, protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.156.4.3 `bdd_dict* spot::tgba_explicit::dict_` **[protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

**7.156.4.4 tgba\_explicit::state\* spot::tgba\_explicit::init\_ [protected, inherited]**

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::add\_state(), and spot::tgba\_explicit\_labelled< std::string, string\_hash >::set\_init\_state().

**7.156.4.5 ns\_map spot::tgba\_explicit\_labelled< std::string , string\_hash >::name\_state\_map\_ [protected, inherited]****7.156.4.6 bdd spot::tgba\_explicit::neg\_acceptance\_conditions\_ [protected, inherited]**

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::declare\_acceptance\_condition().

**7.156.4.7 sn\_map spot::tgba\_explicit\_labelled< std::string , string\_hash >::state\_name\_map\_ [protected, inherited]**

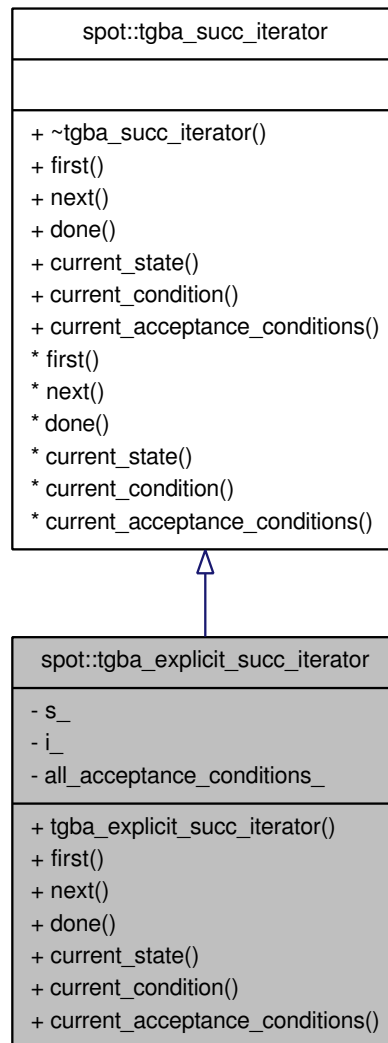
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

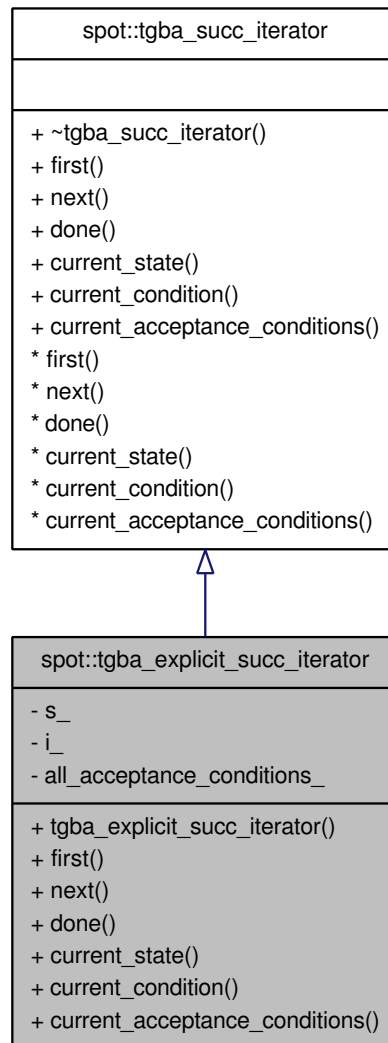
**7.157 spot::tgba\_explicit\_succ\_iterator Class Reference**

```
#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 void `first` ()  
*Position the iterator on the first successor (if any).*
- virtual void `next` ()  
*Jump to the next successor (if any).*
- virtual bool `done` () const  
*Check whether the iteration is finished.*
- virtual `state_explicit` \* `current_state` () const  
*Get the state of the current successor.*



- virtual bdd `current_condition ()` const  
*Get the condition on the transition leading to this successor.*
- virtual bdd `current_acceptance_conditions ()` const  
*Get the acceptance conditions on the transition leading to this successor.*

### Private Attributes

- const `tgba_explicit::state * s_`
- `tgba_explicit::state::const_iterator i_`
- bdd `all_acceptance_conditions_`

### 7.157.1 Detailed Description

Successor iterators used by `spot::tgba_explicit`.

### 7.157.2 Constructor & Destructor Documentation

**7.157.2.1** `spot::tgba_explicit_succ_iterator::tgba_explicit_succ_iterator (const tgba_explicit::state * s, bdd all_acc)`

### 7.157.3 Member Function Documentation

**7.157.3.1** virtual bdd `spot::tgba_explicit_succ_iterator::current_acceptance_conditions ()` const  
[**virtual**]

Get the acceptance conditions on the transition leading to this successor.

Implements `spot::tgba_succ_iterator`.

**7.157.3.2** virtual bdd `spot::tgba_explicit_succ_iterator::current_condition ()` const [virtual]

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements `spot::tgba_succ_iterator`.

**7.157.3.3** virtual `state_explicit*` `spot::tgba_explicit_succ_iterator::current_state ()` const  
[**virtual**]

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.157.3.4 `virtual bool spot::tgba_explicit_succ_iterator::done () const` **[virtual]**

Check whether the iteration is finished.

This function should be called after any call to `first ()` or `next ()` and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first (); !s->done (); s->next ())  
...
```

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.157.3.5 `virtual void spot::tgba_explicit_succ_iterator::first ()` **[virtual]**

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

##### Warning

One should always call `done ()` to ensure there is a successor, even after `first ()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.157.3.6 `virtual void spot::tgba_explicit_succ_iterator::next ()` **[virtual]**

Jump to the next successor (if any).

##### Warning

Again, one should always call `done ()` to ensure there is a successor.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.157.4 Member Data Documentation

#### 7.157.4.1 `bdd spot::tgba_explicit_succ_iterator::all_acceptance_conditions_` **[private]**

7.157.4.2 `tgba_explicit::state::const_iterator spot::tgba_explicit_succ_iterator::i_` `[private]`

7.157.4.3 `const tgba_explicit::state* spot::tgba_explicit_succ_iterator::s_` `[private]`

The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

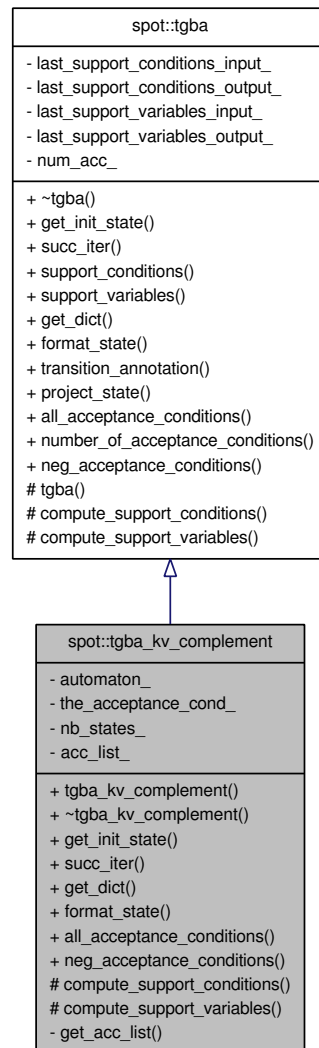
## 7.158 `spot::tgba_kv_complement` Class Reference

Build a complemented automaton.

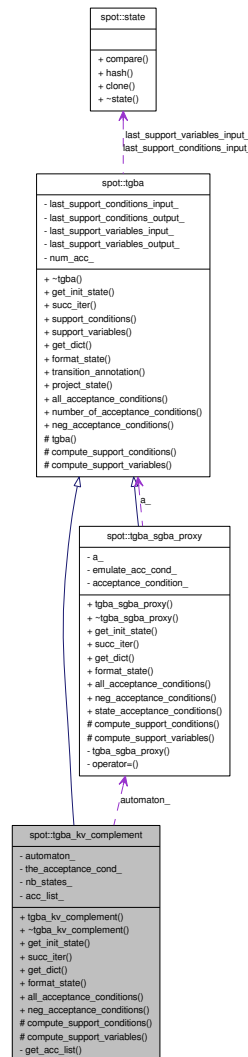
The construction comes from:

```
#include <tgba/tgbakvcomplement.hh>
```

Inheritance diagram for spot::tgba\_kv\_complement:



Collaboration diagram for spot::tgba\_kv\_complement:



## Public Member Functions

- [tgba\\_kv\\_complement](#) (const [tgba](#) \*a)
- virtual [~tgba\\_kv\\_complement](#) ()
- 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 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\(\)](#).*

### Private Member Functions

- void [get\\_acc\\_list](#) ()

### Private Attributes

- const [tgba\\_sgba\\_proxy](#) \* [automaton\\_](#)
- bdd [the\\_acceptance\\_cond\\_](#)
- unsigned [nb\\_states\\_](#)
- [acc\\_list\\_t](#) [acc\\_list\\_](#)

### 7.158.1 Detailed Description

Build a complemented automaton.

The construction comes from:

```

/// @Article{      kupferman.05.tcs,
///   title        = {From complementation to certification},
///   author       = {Kupferman, O. and Vardi, M.Y.},
///   journal      = {Theoretical Computer Science},
///   volume       = {345},
///   number       = {1},
///   pages        = {83--100},
///   year         = {2005},
///   publisher    = {Elsevier}
/// }
///

```

The original automaton is used as a States-based Generalized Büchi Automaton.

The construction is done on-the-fly, by the `tgba_kv_complement_succ_iterator` class.

See also

`tgba_kv_complement_succ_iterator`

### 7.158.2 Constructor & Destructor Documentation

**7.158.2.1** `spot::tgba_kv_complement::tgba_kv_complement (const tgba * a)`

**7.158.2.2** `virtual spot::tgba_kv_complement::~~tgba_kv_complement () [virtual]`

### 7.158.3 Member Function Documentation

**7.158.3.1** `virtual bdd spot::tgba_kv_complement::all_acceptance_conditions () const [virtual]`

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.158.3.2** `virtual bdd spot::tgba_kv_complement::compute_support_conditions (const state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.158.3.3** `virtual bdd spot::tgba_kv_complement::compute_support_variables (const state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.158.3.4** `virtual std::string spot::tgba_kv_complement::format_state (const state * state) const [virtual]`

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

**7.158.3.5** `void spot::tgba_kv_complement::get_acc_list () [private]`

Retrieve all the atomic acceptance conditions of the automaton. They are inserted into *acc\_list\_*.

**7.158.3.6** `virtual bdd_dict* spot::tgba_kv_complement::get_dict () const [virtual]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.158.3.7** `virtual state* spot::tgba_kv_complement::get_init_state () const [virtual]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.158.3.8** `virtual bdd spot::tgba_kv_complement::neg_acceptance_conditions () const [virtual]`

Return the conjunction of all negated acceptance variables.



For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

### 7.158.3.9 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

### 7.158.3.10 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](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.158.3.11 virtual tgba\_succ\_iterator\* spot::tgba\_kv\_complement::succ\_iter (const state \* local\_state, const state \* global\_state = 0, const tgba \* global\_automaton = 0) const [virtual]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

### 7.158.3.12 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.

### 7.158.3.13 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.

### 7.158.3.14 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](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

## 7.158.4 Member Data Documentation

### 7.158.4.1 acc\_list\_t spot::tgba\_kv\_complement::acc\_list\_ [private]

7.158.4.2 `const tgba_sgba_proxy* spot::tgba_kv_complement::automaton_` `[private]`

7.158.4.3 `unsigned spot::tgba_kv_complement::nb_states_` `[private]`

7.158.4.4 `bdd spot::tgba_kv_complement::the_acceptance_cond_` `[private]`

The documentation for this class was generated from the following file:

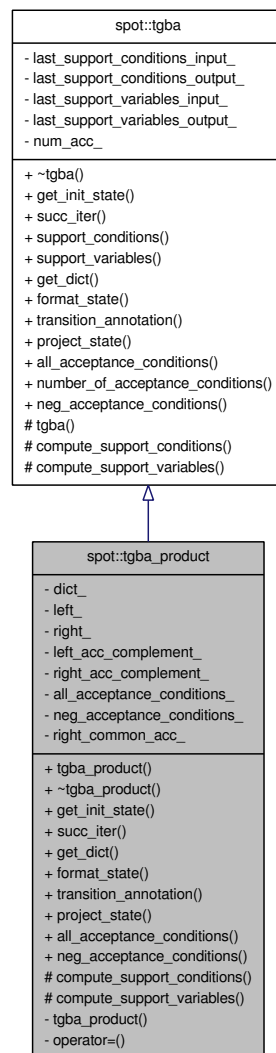
- [tgba/tgbakvcomplement.hh](#)

## 7.159 `spot::tgba_product` Class Reference

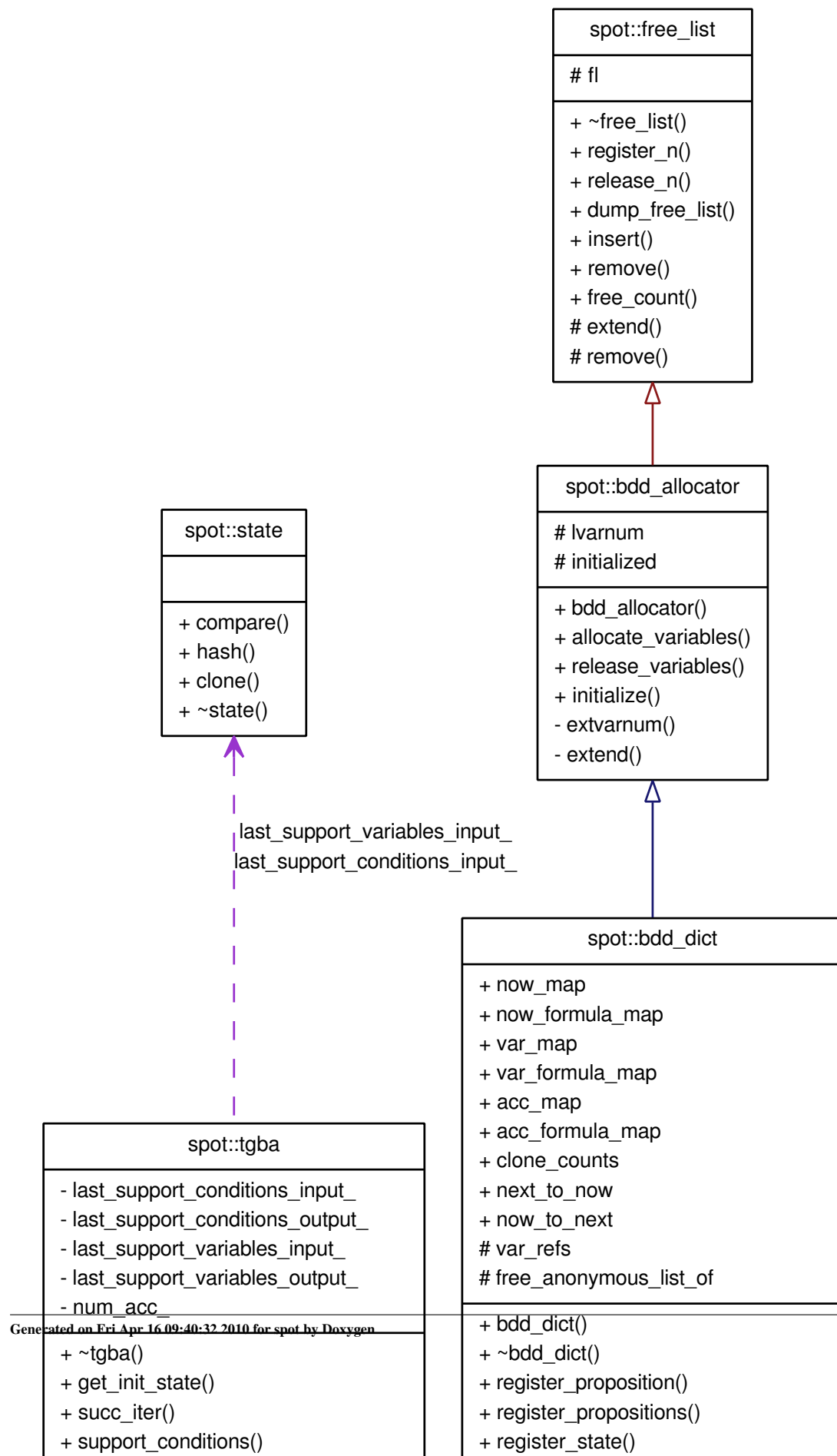
A lazy product. (States are computed on the fly.).

```
#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](#) & [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\\_](#)
- [bddPair](#) \* [right\\_common\\_acc\\_](#)

#### 7.159.1 Detailed Description

A lazy product. (States are computed on the fly.).

#### 7.159.2 Constructor & Destructor Documentation

##### 7.159.2.1 spot::tgba\_product::tgba\_product (const [tgba](#) \* *left*, const [tgba](#) \* *right*)

Constructor.

#### Parameters

*left* The left automata in the product.

*right* The right automata in the product. Do not be fooled by these arguments: a product is commutative.

##### 7.159.2.2 virtual spot::tgba\_product::~~tgba\_product () [virtual]

##### 7.159.2.3 spot::tgba\_product::tgba\_product (const [tgba\\_product](#) &) [private]

#### 7.159.3 Member Function Documentation

##### 7.159.3.1 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 accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 7.159.3.2 virtual bdd spot::tgba\_product::compute\_support\_conditions (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 7.159.3.3 virtual bdd spot::tgba\_product::compute\_support\_variables (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.159.3.4 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 that owns the state.

Implements [spot::tgba](#).

#### 7.159.3.5 virtual bdd\_dict\* spot::tgba\_product::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

#### 7.159.3.6 virtual state\* spot::tgba\_product::get\_init\_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).



**7.159.3.7 virtual bdd spot::tgba\_product::neg\_acceptance\_conditions () const [virtual]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.159.3.8 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**7.159.3.9 tgba\_product& spot::tgba\_product::operator=(const tgba\_product &) [private]****7.159.3.10 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](#).

**7.159.3.11 virtual tgba\_succ\_iterator\_product\* spot::tgba\_product::succ\_iter (const state \* local\_state, const state \* global\_state = 0, const tgba \* global\_automaton = 0) const [virtual]**

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand.

*global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by [succ\\_iter](#), and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *local\_state*, *global\_state* is not adopted by [succ\\_iter](#).

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

#### 7.159.3.12 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.

#### 7.159.3.13 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.

#### 7.159.3.14 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 is the empty string.

#### Parameters

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented from [spot::tgba](#).

#### 7.159.4 Member Data Documentation

7.159.4.1 `bdd spot::tgba_product::all_acceptance_conditions_` `[private]`

7.159.4.2 `bdd_dict* spot::tgba_product::dict_` `[private]`

7.159.4.3 `const tgba* spot::tgba_product::left_` `[private]`

7.159.4.4 `bdd spot::tgba_product::left_acc_complement_` `[private]`

7.159.4.5 `bdd spot::tgba_product::neg_acceptance_conditions_` `[private]`

7.159.4.6 `const tgba* spot::tgba_product::right_` `[private]`

7.159.4.7 `bdd spot::tgba_product::right_acc_complement_` `[private]`

7.159.4.8 `bddPair* spot::tgba_product::right_common_acc_` `[private]`

The documentation for this class was generated from the following file:

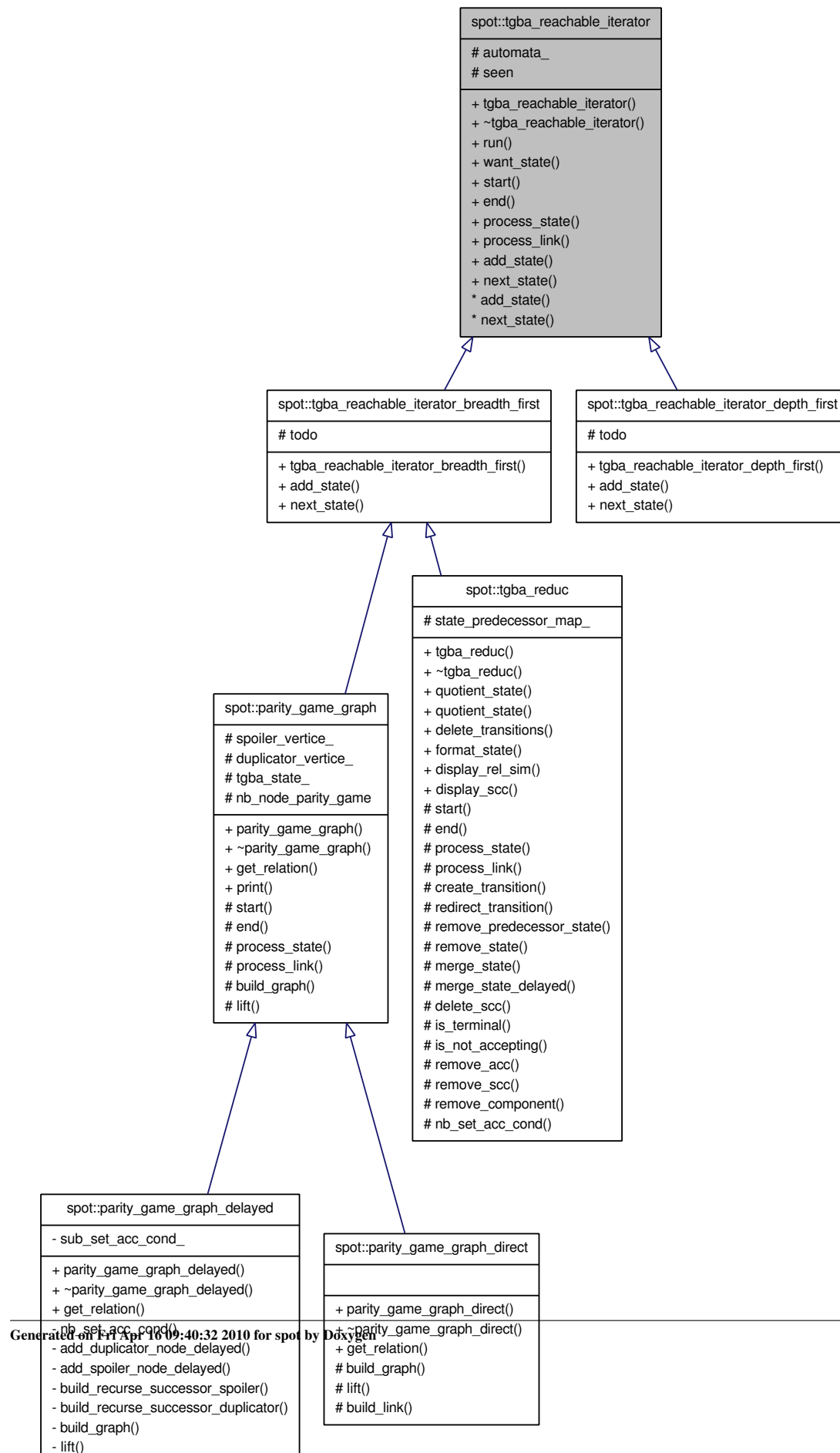
- `tgba/tgbaproduct.hh`

## 7.160 `spot::tgba_reachable_iterator` Class Reference

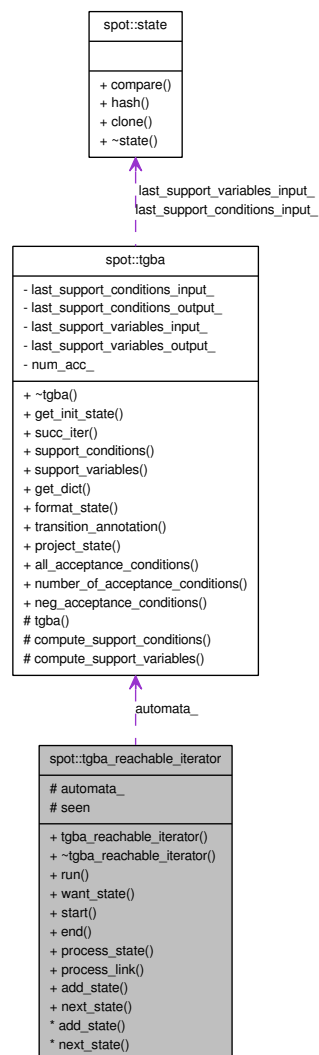
Iterate over all reachable states of a `spot::tgba`.

```
#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 bool [want\\_state](#) (const [state](#) \*s) const
- 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 next state to process.

#### Protected Types

- typedef Sgi::hash\_map< const [state](#) \*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#) > [seen\\_map](#)

#### Protected Attributes

- const [tgba](#) \* [automata\\_](#)  
The [spot::tgba](#) to explore.
- [seen\\_map](#) [seen](#)  
States already seen.

#### 7.160.1 Detailed Description

Iterate over all reachable states of a [spot::tgba](#).

#### 7.160.2 Member Typedef Documentation

- 7.160.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::tgba\\_reachable\\_iterator::seen\\_map](#) [protected]

#### 7.160.3 Constructor & Destructor Documentation

- 7.160.3.1** [spot::tgba\\_reachable\\_iterator::tgba\\_reachable\\_iterator](#) (const [tgba](#) \* a)

- 7.160.3.2** virtual [spot::tgba\\_reachable\\_iterator::~~tgba\\_reachable\\_iterator](#) () [virtual]

### 7.160.4 Member Function Documentation

#### 7.160.4.1 virtual void spot::tgba\_reachable\_iterator::add\_state (const state \* *s*) [pure virtual]

Implemented in [spot::tgba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::tgba\\_reachable\\_iterator\\_breadth\\_first](#).

#### 7.160.4.2 virtual void spot::tgba\_reachable\_iterator::end () [virtual]

Called by [run\(\)](#) once all states have been explored.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

#### 7.160.4.3 virtual const state\* spot::tgba\_reachable\_iterator::next\_state () [pure virtual]

Called by [run\(\)](#) to obtain the next state to process.

Implemented in [spot::tgba\\_reachable\\_iterator\\_depth\\_first](#), and [spot::tgba\\_reachable\\_iterator\\_breadth\\_first](#).

#### 7.160.4.4 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.

#### 7.160.4.5 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::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

#### 7.160.4.6 `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 iterates over states.

#### 7.160.4.7 `virtual void spot::tgba_reachable_iterator::start ()` `[virtual]`

Called by `run()` before starting its iteration.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

#### 7.160.4.8 `virtual bool spot::tgba_reachable_iterator::want_state (const state * s) const` `[virtual]`

Called by `add_state` or `next_states` implementations to filter states. Default implementation always return true.

### 7.160.5 Member Data Documentation

#### 7.160.5.1 `const tgba* spot::tgba_reachable_iterator::automata_` `[protected]`

The `spot::tgba` to explore.

#### 7.160.5.2 `seen_map spot::tgba_reachable_iterator::seen` `[protected]`

States already seen.

The documentation for this class was generated from the following file:

- `tgbaalgos/reachiter.hh`

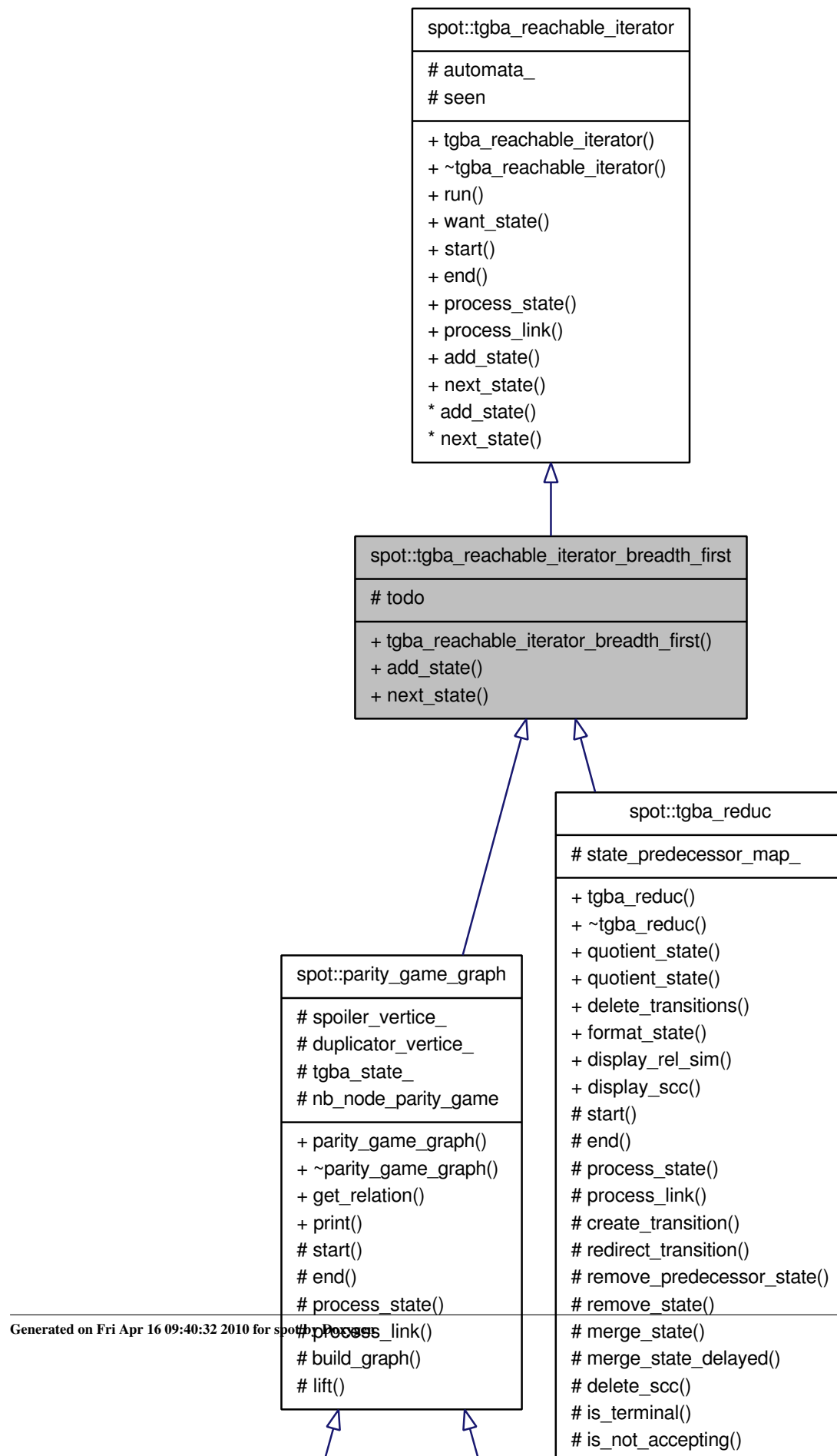
## 7.161 `spot::tgba_reachable_iterator_breadth_first` Class Reference

An implementation of `spot::tgba_reachable_iterator` that browses states breadth first.

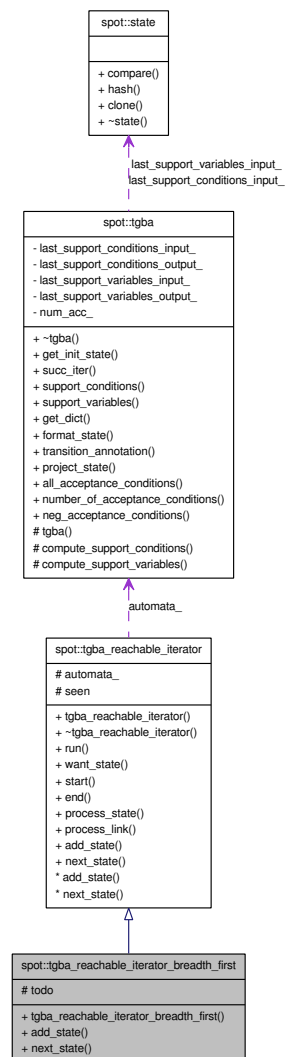
```
#include <tgbaalgos/reachiter.hh>
```



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 next state to process.*

- void [run](#) ()

*Iterate over all reachable states of a [spot::tgba](#).*

- virtual bool [want\\_state](#) (const [state](#) \*s) const
- 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.*

#### 7.161.1 Detailed Description

An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states breadth first.

#### 7.161.2 Member Typedef Documentation

- 7.161.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::tgba\\_reachable\\_iterator::seen\\_map](#) [[protected](#), [inherited](#)]

#### 7.161.3 Constructor & Destructor Documentation

- 7.161.3.1** [spot::tgba\\_reachable\\_iterator\\_breadth\\_first::tgba\\_reachable\\_iterator\\_breadth\\_first](#)  
(const [tgba](#) \* a)

#### 7.161.4 Member Function Documentation

- 7.161.4.1** virtual void [spot::tgba\\_reachable\\_iterator\\_breadth\\_first::add\\_state](#) (const [state](#) \* s)  
[[virtual](#)]

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.161.4.2 virtual void spot::tgba\_reachable\_iterator::end () [virtual, inherited]**

Called by [run\(\)](#) once all states have been explored.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**7.161.4.3 virtual const state\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state () [virtual]**

Called by [run\(\)](#) to obtain the next state to process.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.161.4.4 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.

**7.161.4.5 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::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**7.161.4.6 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 iterates over states.

#### 7.161.4.7 `virtual void spot::tgba_reachable_iterator::start ()` `[virtual, inherited]`

Called by `run()` before starting its iteration.

Reimplemented in `spot::tgba_reduc`, and `spot::parity_game_graph`.

#### 7.161.4.8 `virtual bool spot::tgba_reachable_iterator::want_state (const state * s) const` `[virtual, inherited]`

Called by `add_state` or `next_states` implementations to filter states. Default implementation always return true.

### 7.161.5 Member Data Documentation

#### 7.161.5.1 `const tgba* spot::tgba_reachable_iterator::automata_` `[protected, inherited]`

The `spot::tgba` to explore.

#### 7.161.5.2 `seen_map spot::tgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

#### 7.161.5.3 `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo` `[protected]`

A queue of states yet to explore.

The documentation for this class was generated from the following file:

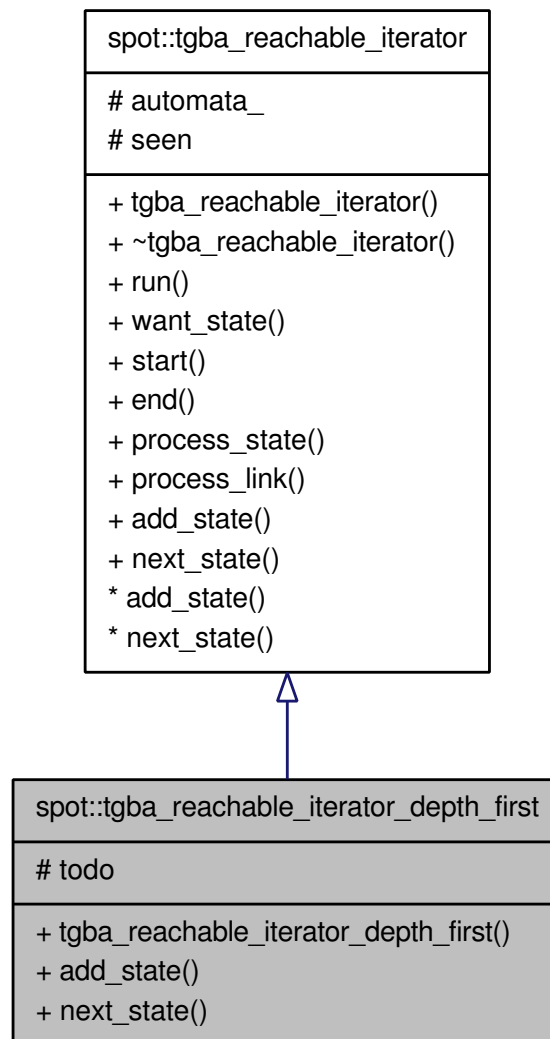
- `tgbaalgos/reachiter.hh`

## 7.162 `spot::tgba_reachable_iterator_depth_first` Class Reference

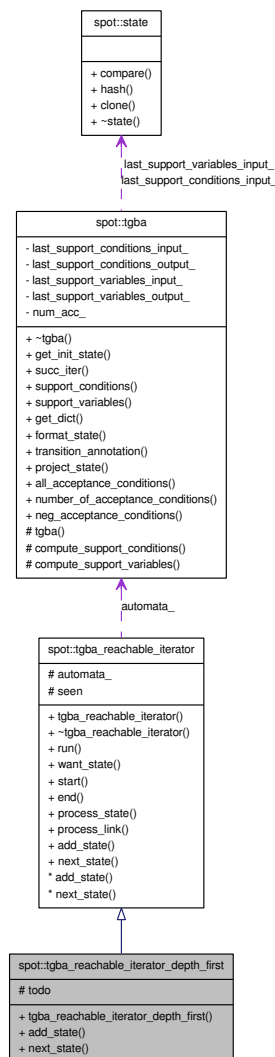
An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

```
#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 next state to process.*

- void [run](#) ()

*Iterate over all reachable states of a [spot::tgba](#).*

- virtual bool [want\\_state](#) (const [state](#) \*s) const
- 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.

## 7.162.1 Detailed Description

An implementation of [spot::tgba\\_reachable\\_iterator](#) that browses states depth first.

## 7.162.2 Member Typedef Documentation

- 7.162.2.1** typedef Sgi::hash\_map<const [state](#)\*, int, [state\\_ptr\\_hash](#), [state\\_ptr\\_equal](#)>  
[spot::tgba\\_reachable\\_iterator::seen\\_map](#) [[protected](#), [inherited](#)]

## 7.162.3 Constructor & Destructor Documentation

- 7.162.3.1** [spot::tgba\\_reachable\\_iterator\\_depth\\_first::tgba\\_reachable\\_iterator\\_depth\\_first](#) (const [tgba](#) \* a)

## 7.162.4 Member Function Documentation

- 7.162.4.1** virtual void [spot::tgba\\_reachable\\_iterator\\_depth\\_first::add\\_state](#) (const [state](#) \* s)  
[[virtual](#)]

Implements [spot::tgba\\_reachable\\_iterator](#).



**7.162.4.2 virtual void spot::tgba\_reachable\_iterator::end () [virtual, inherited]**

Called by [run\(\)](#) once all states have been explored.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**7.162.4.3 virtual const state\* spot::tgba\_reachable\_iterator\_depth\_first::next\_state () [virtual]**

Called by [run\(\)](#) to obtain the next state to process.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.162.4.4 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.

**7.162.4.5 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::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

**7.162.4.6 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 iterates over states.

#### 7.162.4.7 virtual void spot::tgba\_reachable\_iterator::start () [virtual, inherited]

Called by [run\(\)](#) before starting its iteration.

Reimplemented in [spot::tgba\\_reduc](#), and [spot::parity\\_game\\_graph](#).

#### 7.162.4.8 virtual bool spot::tgba\_reachable\_iterator::want\_state (const state \* s) const [virtual, inherited]

Called by add\_state or next\_states implementations to filter states. Default implementation always return true.

### 7.162.5 Member Data Documentation

#### 7.162.5.1 const tgba\* spot::tgba\_reachable\_iterator::automata\_ [protected, inherited]

The [spot::tgba](#) to explore.

#### 7.162.5.2 seen\_map spot::tgba\_reachable\_iterator::seen [protected, inherited]

States already seen.

#### 7.162.5.3 std::stack<const state\*> spot::tgba\_reachable\_iterator\_depth\_first::todo [protected]

A stack of states yet to explore.

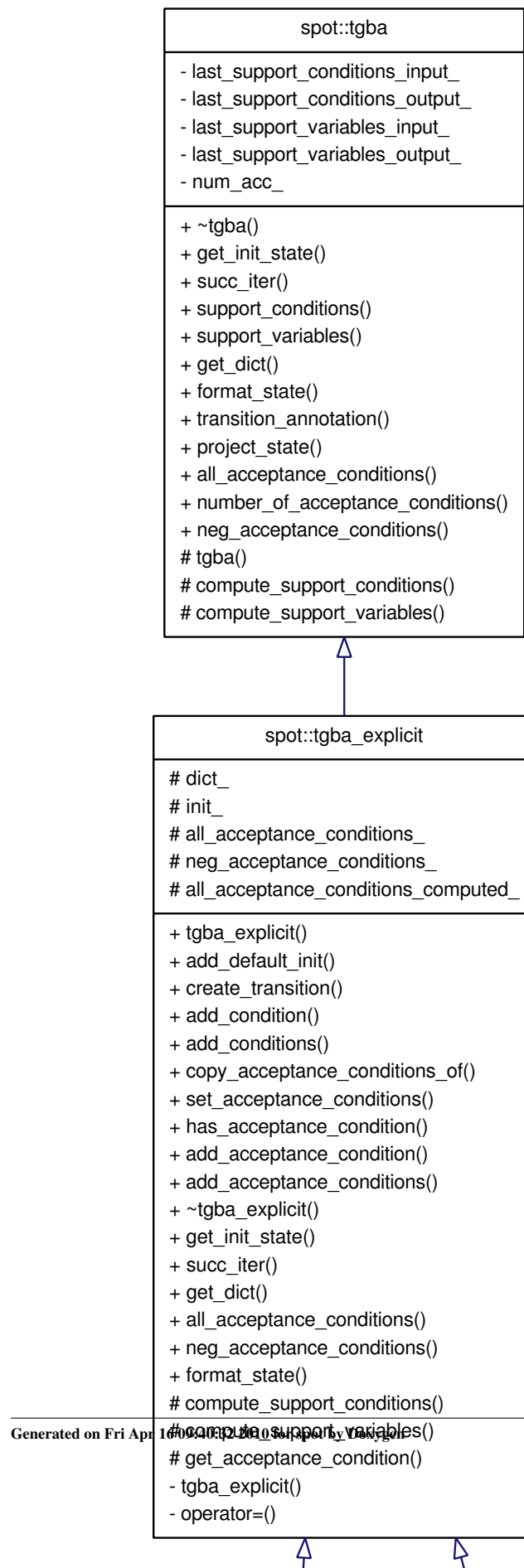
The documentation for this class was generated from the following file:

- tgbaalgos/[reachiter.hh](#)

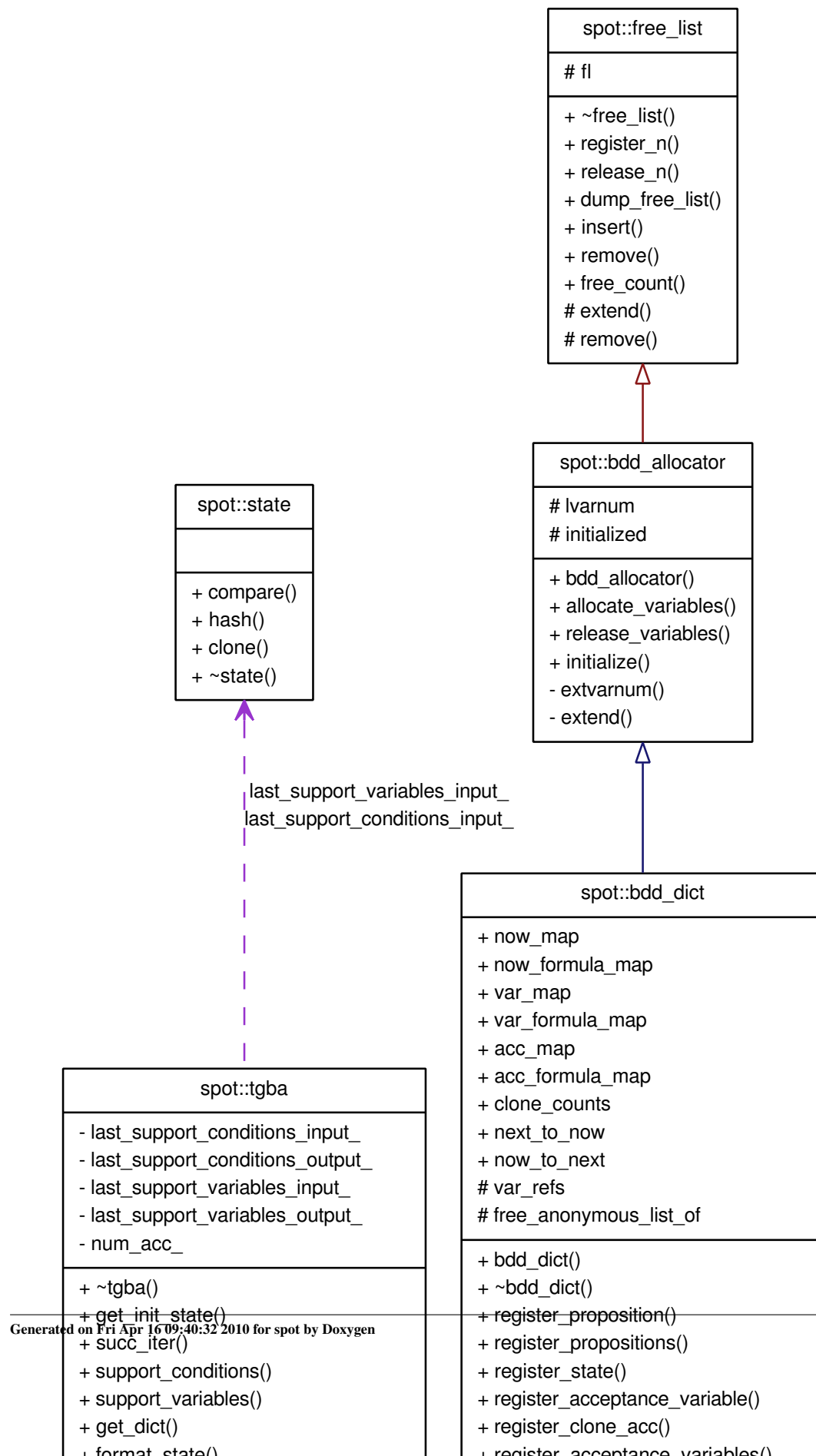
## 7.163 spot::tgba\_reduc Class Reference

```
#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)
- [~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.*
- 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)
- virtual [state](#) \* [add\\_default\\_init](#) ()  
*Add a default initial state.*
- bool [has\\_state](#) (const std::string &name)
- const std::string & [get\\_label](#) (const [tgba\\_explicit::state](#) \*s) const
- const std::string & [get\\_label](#) (const [spot::state](#) \*s) const
- [state](#) \* [add\\_state](#) (const std::string &name)
- [state](#) \* [set\\_init\\_state](#) (const std::string &state)
- [transition](#) \* [create\\_transition](#) ([state](#) \*source, const [state](#) \*dest)
- [transition](#) \* [create\\_transition](#) (const std::string &source, const std::string &dest)
- void [complement\\_all\\_acceptance\\_conditions](#) ()
- void [declare\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)
- void [merge\\_transitions](#) ()
- 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 [copy\\_acceptance\\_conditions\\_of](#) (const [tgba](#) \*a)  
*Copy the acceptance conditions of a tgba.*
- void [set\\_acceptance\\_conditions](#) (bdd acc)  
*The the acceptance conditions.*
- 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.*
- 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  
*Get an iterator over the successors of local\_state.*
- virtual [bdd\\_dict](#) \* [get\\_dict](#) () const  
*Get the dictionary associated to the 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.*
- virtual void [add\\_state](#) (const [state](#) \*s)
- virtual const [state](#) \* [next\\_state](#) ()  
*Called by [run\(\)](#) to obtain the next state to process.*
- void [run](#) ()  
*Iterate over all reachable states of a [spot::tgba](#).*
- virtual bool [want\\_state](#) (const [state](#) \*s) const
- 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 std::string [label\\_t](#)
- typedef Sgi::hash\_map< 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](#)
- 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 [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)  
*Same as [remove\\_scc](#) but more efficient.*
- int [nb\\_set\\_acc\\_cond](#) () const
- virtual bdd [compute\\_support\\_conditions](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_conditions\(\)](#).*
- virtual bdd [compute\\_support\\_variables](#) (const [spot::state](#) \*state) const  
*Do the actual computation of [tgba::support\\_variables\(\)](#).*
- bdd [get\\_acceptance\\_condition](#) (const [ltl::formula](#) \*f)

## Protected Attributes

- [sp\\_map](#) [state\\_predecessor\\_map\\_](#)
- [ns\\_map](#) [name\\_state\\_map\\_](#)
- [sn\\_map](#) [state\\_name\\_map\\_](#)
- [bdd\\_dict](#) \* [dict\\_](#)
- [tgba\\_explicit::state](#) \* [init\\_](#)
- bdd [all\\_acceptance\\_conditions\\_](#)
- bdd [neg\\_acceptance\\_conditions\\_](#)
- bool [all\\_acceptance\\_conditions\\_computed\\_](#)
- [std::deque](#)< const [state](#) \* > [todo](#)

*A queue of states yet to explore.*

- const tgba \* automata\_  
*The spot::tgba to explore.*
- seen\_map seen  
*States already seen.*

### 7.163.1 Detailed Description

Explicit automata used in reductions.

### 7.163.2 Member Typedef Documentation

**7.163.2.1** typedef std::string spot::tgba\_explicit\_labelled< std::string , string\_hash >::label\_t  
[protected, inherited]

**7.163.2.2** typedef Sgi::hash\_map<std::string , tgba\_explicit::state\*, string\_hash >  
spot::tgba\_explicit\_labelled< std::string , string\_hash >::ns\_map [protected,  
inherited]

**7.163.2.3** typedef Sgi::hash\_map<const state\*, int, state\_ptr\_hash, state\_ptr\_equal>  
spot::tgba\_reachable\_iterator::seen\_map [protected, inherited]

**7.163.2.4** typedef Sgi::hash\_map<const tgba\_explicit::state\*, std::string ,  
ptr\_hash<tgba\_explicit::state> > spot::tgba\_explicit\_labelled< std::string ,  
string\_hash >::sn\_map [protected, inherited]

**7.163.2.5** typedef Sgi::hash\_map<const tgba\_explicit::state\*, std::list<state\*>\*,  
ptr\_hash<tgba\_explicit::state> > spot::tgba\_reduc::sp\_map [protected]

**7.163.2.6** typedef std::list<transition\*> spot::tgba\_explicit::state [inherited]



### 7.163.3 Constructor & Destructor Documentation

**7.163.3.1** spot::tgba\_reduc::tgba\_reduc (const tgba \* *a*)

**7.163.3.2** spot::tgba\_reduc::~~tgba\_reduc ()

### 7.163.4 Member Function Documentation

**7.163.4.1** void spot::tgba\_explicit::add\_acceptance\_condition (transition \* *t*, const ltl::formula \* *f*) **[inherited]**

**7.163.4.2** void spot::tgba\_explicit::add\_acceptance\_conditions (transition \* *t*, bdd *f*) **[inherited]**

This assumes that all acceptance conditions in *f* are known from dict.

**7.163.4.3** void spot::tgba\_explicit::add\_condition (transition \* *t*, const ltl::formula \* *f*) **[inherited]**

**7.163.4.4** void spot::tgba\_explicit::add\_conditions (transition \* *t*, bdd *f*) **[inherited]**

This assumes that all variables in *f* are known from dict.

**7.163.4.5** virtual state\* spot::tgba\_explicit\_string::add\_default\_init () **[virtual, inherited]**

Add a default initial state.

Implements [spot::tgba\\_explicit](#).

**7.163.4.6** virtual void spot::tgba\_reachable\_iterator\_breadth\_first::add\_state (const state \* *s*) **[virtual, inherited]**

Implements [spot::tgba\\_reachable\\_iterator](#).

#### 7.163.4.7 state\* spot::tgba\_explicit\_labelled< std::string , string\_hash >::add\_state (const std::string & name) [inline, inherited]

Return the tgba\_explicit::state for *name*, creating the state if it does not exist.

References spot::tgba\_explicit::init\_, spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_, and spot::tgba\_explicit\_labelled< label, label\_hash >::state\_name\_map\_.

#### 7.163.4.8 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](#).

Referenced by spot::tgba\_explicit\_labelled< std::string, string\_hash >::complement\_all\_acceptance\_conditions().

#### 7.163.4.9 void spot::tgba\_explicit\_labelled< std::string , string\_hash >::complement\_all\_acceptance\_conditions () [inline, inherited]

References spot::tgba\_explicit::all\_acceptance\_conditions(), and spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_.

#### 7.163.4.10 virtual bdd spot::tgba\_explicit::compute\_support\_conditions (const spot::state \* state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 7.163.4.11 virtual bdd spot::tgba\_explicit::compute\_support\_variables (const spot::state \* state) const [protected, virtual, inherited]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.163.4.12 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.

**7.163.4.13** `transition* spot::tgba_explicit_labelled< std::string , string_hash >::create_transition (const std::string & source, const std::string & dest) [inline, inherited]`

References `spot::tgba_explicit_labelled< label, label_hash >::add_state()`, and `spot::tgba_explicit_labelled< label, label_hash >::create_transition()`.

**7.163.4.14** `transition* spot::tgba_explicit_labelled< std::string , string_hash >::create_transition (state * source, const state * dest) [inline, inherited]`

Reimplemented from [spot::tgba\\_explicit](#).

References `spot::tgba_explicit_labelled< label, label_hash >::create_transition()`.

**7.163.4.15** `transition* spot::tgba_reduc::create_transition (const spot::state * source, const spot::state * dest) [protected]`

Create a transition using two state of a TGBA.

**7.163.4.16** `void spot::tgba_explicit_labelled< std::string , string_hash >::declare_acceptance_condition (const ltl::formula * f) [inline, inherited]`

References `spot::tgba_explicit::all_acceptance_conditions_computed_`, `spot::ltl::formula::destroy()`, `spot::tgba_explicit::dict_`, `spot::tgba_explicit_labelled< label, label_hash >::name_state_map_`, `spot::tgba_explicit::neg_acceptance_conditions_`, and `spot::bdd_dict::register_acceptance_variable()`.

**7.163.4.17** `void spot::tgba_reduc::delete_scc () [protected]`

Remove all the scc which are terminal and doesn't contains all the acceptance conditions.

**7.163.4.18** `void spot::tgba_reduc::delete_transitions (simulation_relation * rel)`

Delete some transitions with help of a simulation relation.

**7.163.4.19** `void spot::tgba_reduc::display_rel_sim (simulation_relation * rel, std::ostream & os)`

**7.163.4.20** void spot::tgba\_reduc::display\_scc (std::ostream & *os*)

**7.163.4.21** void spot::tgba\_reduc::end () [protected, virtual]

Called by [run\(\)](#) once all states have been explored.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.163.4.22** virtual std::string spot::tgba\_reduc::format\_state (const spot::state \* *state*) const [virtual]

Add the SCC index to the display of the state *state*.

Reimplemented from [spot::tgba\\_explicit\\_string](#).

**7.163.4.23** bdd spot::tgba\_explicit::get\_acceptance\_condition (const ltl::formula \* *f*) [protected, inherited]

**7.163.4.24** virtual bdd\_dict\* spot::tgba\_explicit::get\_dict () const [virtual, inherited]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.163.4.25** virtual spot::state\* spot::tgba\_explicit::get\_init\_state () const [virtual, inherited]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.163.4.26** const std::string & spot::tgba\_explicit\_labelled< std::string, string\_hash >::get\_label (const spot::state \* *s*) const [inline, inherited]

References spot::tgba\_explicit\_labelled< label, label\_hash >::get\_label(), and spot::state\_explicit::get\_state().

**7.163.4.27** `const std::string & spot::tgba_explicit_labelled< std::string, string_hash >::get_label (const tgba_explicit::state * s) const` `[inline, inherited]`

References spot::tgba\_explicit\_labelled< label, label\_hash >::state\_name\_map\_.

**7.163.4.28** `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const` `[inherited]`

**7.163.4.29** `bool spot::tgba_explicit_labelled< std::string, string_hash >::has_state (const std::string & name) const` `[inline, inherited]`

References spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_.

**7.163.4.30** `bool spot::tgba_reduc::is_not_accepting (const spot::state * s, int n = -1)` `[protected]`

**7.163.4.31** `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}
/// }
```

**7.163.4.32** `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.

**7.163.4.33** 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.

**7.163.4.34** void spot::tgba\_explicit\_labelled< std::string , string\_hash >::merge\_transitions () **[inline, inherited]**

References spot::tgba\_explicit\_labelled< label, label\_hash >::name\_state\_map\_.

**7.163.4.35** int spot::tgba\_reduc::nb\_set\_acc\_cond () const **[protected]**

**7.163.4.36** virtual bdd spot::tgba\_explicit::neg\_acceptance\_conditions () const **[virtual, inherited]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.163.4.37** virtual const state\* spot::tgba\_reachable\_iterator\_breadth\_first::next\_state () **[virtual, inherited]**

Called by [run\(\)](#) to obtain the next state to process.

Implements [spot::tgba\\_reachable\\_iterator](#).

**7.163.4.38** virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const **[virtual, inherited]**

The number of acceptance conditions.

**7.163.4.39** 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.

**7.163.4.40** `void spot::tgba_reduc::process_link (int in, int out, const tgba_succ_iterator * si) [protected]`

**7.163.4.41** `void spot::tgba_reduc::process_state (const spot::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](#).

**7.163.4.42** `virtual state* spot::tgba::project_state (const state * s, const tgba * t) const [virtual, inherited]`

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

**Returns**

- 0 if the projection fails (*s* is unrelated to *t*), or a new `state*` (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

**7.163.4.43** `void spot::tgba_reduc::quotient_state (delayed_simulation_relation * rel)`

Build the quotient automata. Call this method when use to a delayed simulation relation.

**7.163.4.44 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.

**7.163.4.45 void spot::tgba\_reduc::redirect\_transition (const spot::state \* *s*, const spot::state \* *simul*) [protected]**

Remove all the transition from the state *q*, predecessor of both *s* and *simul*, which can be removed.

**7.163.4.46 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.

**7.163.4.47 void spot::tgba\_reduc::remove\_component (const spot::state \* *from*) [protected]**

Same as remove\_scc but more efficient.

For compute\_scc.

**7.163.4.48 void spot::tgba\_reduc::remove\_predecessor\_state (const state \* *s*, const state \* *p*) [protected]**

Remove *p* of the predecessor of *s*.

**7.163.4.49 void spot::tgba\_reduc::remove\_scc (spot::state \* *s*) [protected]**

Remove all the state which belong to the same scc that *s*.

**7.163.4.50 void spot::tgba\_reduc::remove\_state (const spot::state \* *s*) [protected]**

Remove all the transition leading to *s*. *s* is then unreachable and can be consider as remove.

**7.163.4.51 void spot::tgba\_reachable\_iterator::run () [inherited]**

Iterate over all reachable states of a [spot::tgba](#).

This is a template method that will call [add\\_state\(\)](#), [next\\_state\(\)](#), [start\(\)](#), [end\(\)](#), [process\\_state\(\)](#), and [process\\_link\(\)](#), while it iterates over states.



**7.163.4.52 void spot::tgba\_explicit::set\_acceptance\_conditions (bdd *acc*) [inherited]**

The the acceptance conditions.

**7.163.4.53 state\* spot::tgba\_explicit\_labelled< std::string, string\_hash >::set\_init\_state (const std::string & *state*) [inline, inherited]**

References spot::tgba\_explicit\_labelled< label, label\_hash >::add\_state(), and spot::tgba\_explicit::init\_.

**7.163.4.54 void spot::tgba\_reduc::start () [protected, virtual]**

Called by [run\(\)](#) before starting its iteration.

Reimplemented from [spot::tgba\\_reachable\\_iterator](#).

**7.163.4.55 virtual tgba\_succ\_iterator\* spot::tgba\_explicit::succ\_iter (const spot::state \* *local\_state*, const spot::state \* *global\_state* = 0, const tgba \* *global\_automaton* = 0) const [virtual, inherited]**

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

**Parameters**

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

***global\_automaton*** In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

**7.163.4.56 bdd spot::tgba::support\_conditions (const state \* *state*) const [inherited]**

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.163.4.57 bdd spot::tgba::support\_variables (const state \* state) const [inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.163.4.58 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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.163.4.59 virtual bool spot::tgba\_reachable\_iterator::want\_state (const state \* s) const [virtual, inherited]**

Called by `add_state` or `next_states` implementations to filter states. Default implementation always return `true`.

**7.163.5 Member Data Documentation****7.163.5.1 bdd spot::tgba\_explicit::all\_acceptance\_conditions\_ [mutable, protected, inherited]****7.163.5.2 bool spot::tgba\_explicit::all\_acceptance\_conditions\_computed\_ [mutable, protected, inherited]**

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.163.5.3 `const tgba* spot::tgba_reachable_iterator::automata_` `[protected, inherited]`

The `spot::tgba` to explore.

#### 7.163.5.4 `bdd_dict* spot::tgba_explicit::dict_` `[protected, inherited]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.163.5.5 `tgba_explicit::state* spot::tgba_explicit::init_` `[protected, inherited]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::set_init_state()`.

#### 7.163.5.6 `ns_map spot::tgba_explicit_labelled< std::string, string_hash >::name_state_map_` `[protected, inherited]`

#### 7.163.5.7 `bdd spot::tgba_explicit::neg_acceptance_conditions_` `[protected, inherited]`

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.163.5.8 `seen_map spot::tgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

#### 7.163.5.9 `sn_map spot::tgba_explicit_labelled< std::string, string_hash >::state_name_map_` `[protected, inherited]`

#### 7.163.5.10 `sp_map spot::tgba_reduc::state_predecessor_map_` `[protected]`

#### 7.163.5.11 `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo` [protected, inherited]

A queue of states yet to explore.

The documentation for this class was generated from the following file:

- [tgba/tgbareduc.hh](#)

### 7.164 spot::tgba\_run Struct Reference

An accepted run, for a tgba.

```
#include <tgbaalgos/emptiness.hh>
```

#### Classes

- struct [step](#)

#### 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

#### 7.164.1 Detailed Description

An accepted run, for a tgba.

#### 7.164.2 Member Typedef Documentation

##### 7.164.2.1 `typedef std::list<step> spot::tgba_run::steps`

### 7.164.3 Constructor & Destructor Documentation

#### 7.164.3.1 spot::tgba\_run::~~tgba\_run ()

#### 7.164.3.2 spot::tgba\_run::tgba\_run () [inline]

#### 7.164.3.3 spot::tgba\_run::tgba\_run (const tgba\_run & run)

### 7.164.4 Member Function Documentation

#### 7.164.4.1 tgba\_run& spot::tgba\_run::operator= (const tgba\_run & run)

### 7.164.5 Member Data Documentation

#### 7.164.5.1 steps spot::tgba\_run::cycle

#### 7.164.5.2 steps spot::tgba\_run::prefix

The documentation for this struct was generated from the following file:

- [tgbaalgos/emptiness.hh](#)

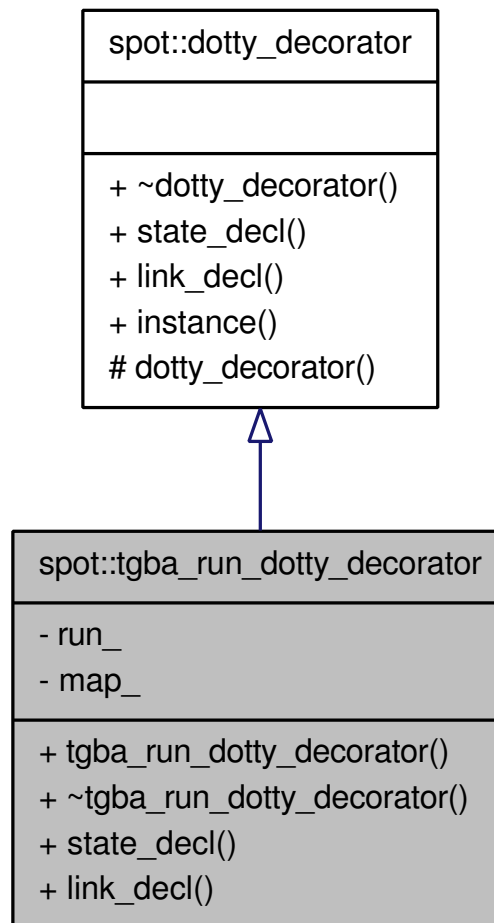
## 7.165 spot::tgba\_run\_dotty\_decorator Class Reference

Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).

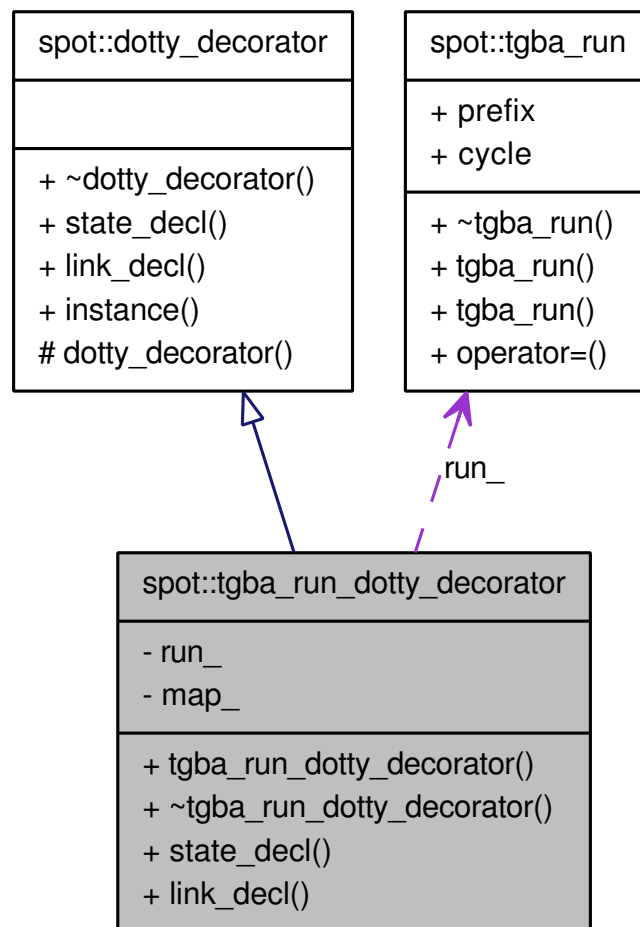
An instance of this class can be passed to [spot::dotty\\_reachable](#).

```
#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\\_](#)

### 7.165.1 Detailed Description

Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).

An instance of this class can be passed to [spot::dotty\\_reachable](#).

### 7.165.2 Member Typedef Documentation

**7.165.2.1** typedef std::map<const state\*, std::pair<step\_set, step\_set>, spot::state\_ptr\_less\_than> spot::tgba\_run\_dotty\_decorator::step\_map [private]

**7.165.2.2** typedef std::pair<tgba\_run::steps::const\_iterator, int> spot::tgba\_run\_dotty\_decorator::step\_num [private]

**7.165.2.3** typedef std::list<step\_num> spot::tgba\_run\_dotty\_decorator::step\_set [private]

### 7.165.3 Constructor & Destructor Documentation

**7.165.3.1** spot::tgba\_run\_dotty\_decorator::tgba\_run\_dotty\_decorator (const tgba\_run \* *run*)

**7.165.3.2** virtual spot::tgba\_run\_dotty\_decorator::~~tgba\_run\_dotty\_decorator () [virtual]

### 7.165.4 Member Function Documentation

**7.165.4.1** static dotty\_decorator\* spot::dotty\_decorator::instance () [static, inherited]



Get the unique instance of the default [dotty\\_decorator](#).

**7.165.4.2** `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](#).

**7.165.4.3** `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](#).

### 7.165.5 Member Data Documentation

**7.165.5.1** `step_map spot::tgba_run_dotty_decorator::map_ [private]`

## 7.165.5.2 const tgba\_run\* spot::tgba\_run\_dotty\_decorator::run\_ [private]

The documentation for this class was generated from the following file:

- tgbaalgos/[rundotdec.hh](#)

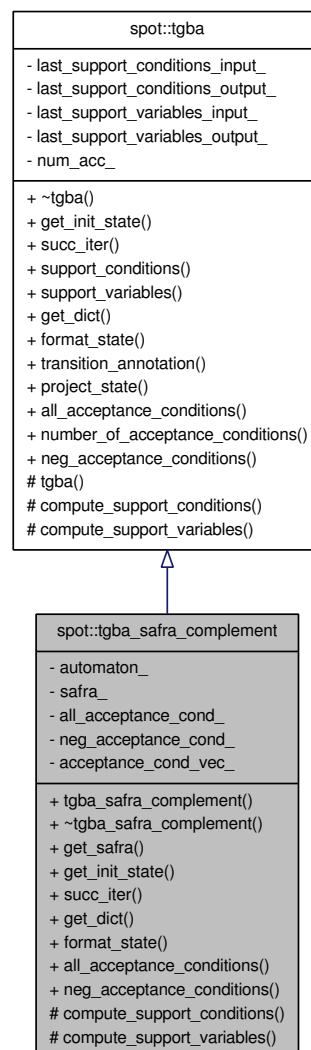
## 7.166 spot::tgba\_safra\_complement Class Reference

Build a complemented automaton.

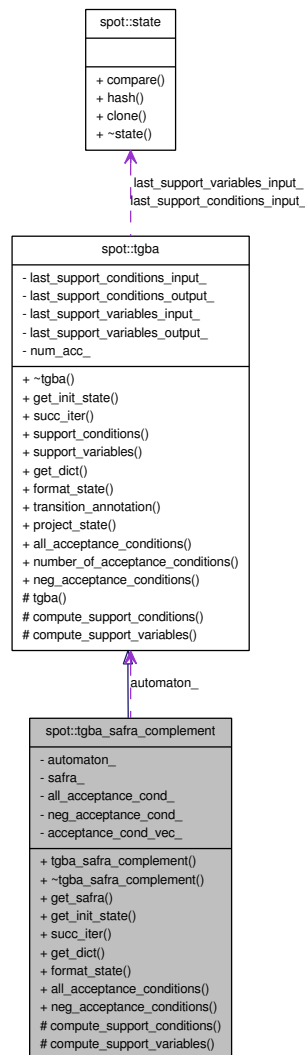
It creates an automaton that recognizes the negated language of *aut*.

```
#include <tgba/tgbasafracomplement.hh>
```

Inheritance diagram for spot::tgba\_safra\_complement:



Collaboration diagram for spot::tgba\_safra\_complement:



## Public Member Functions

- [tgba\\_safra\\_complement](#) (const [tgba](#) \*a)
- virtual [~tgba\\_safra\\_complement](#) ()
- safra\_tree\_automaton \* [get\\_safra](#) () const
- 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 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\(\)](#).*

### Private Attributes

- const [tgba](#) \* [automaton\\_](#)
- [safra\\_tree\\_automaton](#) \* [safra\\_](#)
- bdd [all\\_acceptance\\_cond\\_](#)
- bdd [neg\\_acceptance\\_cond\\_](#)
- std::vector< int > [acceptance\\_cond\\_vec\\_](#)

#### 7.166.1 Detailed Description

Build a complemented automaton.

It creates an automaton that recognizes the negated language of *aut*. 1. First Safra construction algorithm produces a deterministic Rabin automaton. 2. Interpreting this deterministic Rabin automaton as a deterministic Streett will produce a complemented automaton. 3. Then we use a transformation from deterministic Streett automaton to nondeterministic Büchi automaton.

Safra construction is done in *tgba\_complement*, the transformation is done on-the-fly when successors are called.

#### See also

safra\_determinisation, [tgba\\_safra\\_complement::succ\\_iter](#).

### 7.166.2 Constructor & Destructor Documentation

#### 7.166.2.1 spot::tgba\_safra\_complement::tgba\_safra\_complement (const tgba \* *a*)

#### 7.166.2.2 virtual spot::tgba\_safra\_complement::~~tgba\_safra\_complement () [virtual]

### 7.166.3 Member Function Documentation

#### 7.166.3.1 virtual bdd spot::tgba\_safra\_complement::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 7.166.3.2 virtual bdd spot::tgba\_safra\_complement::compute\_support\_conditions (const state \* *state*) const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 7.166.3.3 virtual bdd spot::tgba\_safra\_complement::compute\_support\_variables (const state \* *state*) const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.166.3.4 virtual std::string spot::tgba\_safra\_complement::format\_state (const state \* *state*) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

### 7.166.3.5 `virtual bdd_dict* spot::tgba_safra_complement::get_dict () const [virtual]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

### 7.166.3.6 `virtual state* spot::tgba_safra_complement::get_init_state () const [virtual]`

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

### 7.166.3.7 `safra_tree_automaton* spot::tgba_safra_complement::get_safra () const [inline]`

References `safra_`.

### 7.166.3.8 `virtual bdd spot::tgba_safra_complement::neg_acceptance_conditions () const [virtual]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

### 7.166.3.9 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

### 7.166.3.10 virtual state\* spot::tgba::project\_state (const state \* *s*, const tgba \* *t*) const [virtual, inherited]

Project a state on an automaton.

This converts *s*, into that corresponding [spot::state](#) for *t*. This is useful when you have the state of a product, and want restrict this state to a specific automata occuring in the product.

It goes without saying that *s* and *t* should be compatible (i.e., *s* is a state of *t*).

#### Returns

0 if the projection fails (*s* is unrelated to *t*), or a new state\* (the projected state) that must be deleted by the caller.

Reimplemented in [spot::tgba\\_product](#), [spot::tgba\\_scc](#), [spot::tgba\\_tba\\_proxy](#), and [spot::tgba\\_union](#).

### 7.166.3.11 virtual tgba\_succ\_iterator\* spot::tgba\_safra\_complement::succ\_iter (const state \* *local\_state*, const state \* *global\_state* = 0, const tgba \* *global\_automaton* = 0) const [virtual]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

#### Parameters

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

***global\_automaton*** In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

### 7.166.3.12 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.

#### 7.166.3.13 `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.

#### 7.166.3.14 `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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

### 7.166.4 Member Data Documentation

#### 7.166.4.1 `std::vector<int> spot::tgba_safra_complement::acceptance_cond_vec_ [private]`

#### 7.166.4.2 `bdd spot::tgba_safra_complement::all_acceptance_cond_ [private]`

#### 7.166.4.3 `const tgba* spot::tgba_safra_complement::automaton_ [private]`

#### 7.166.4.4 `bdd spot::tgba_safra_complement::neg_acceptance_cond_ [private]`



## 7.166.4.5 safra\_tree\_automaton\* spot::tgba\_safracomplement::safra\_ [private]

Referenced by get\_safra().

The documentation for this class was generated from the following file:

- [tgba/tgbasafracomplement.hh](#)

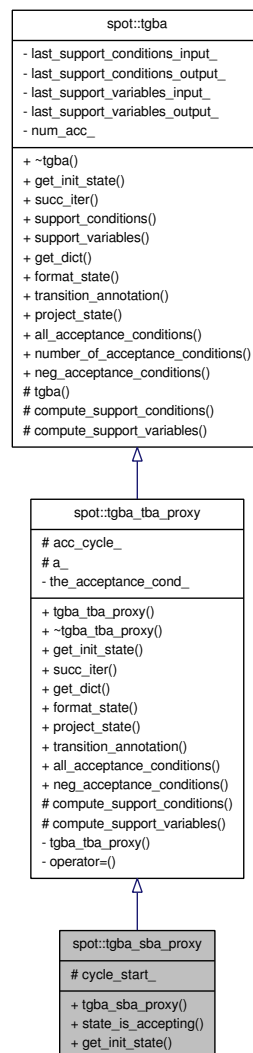
## 7.167 spot::tgba\_sba\_proxy Class Reference

Degeneralize a [spot::tgba](#) on the fly, producing an SBA.

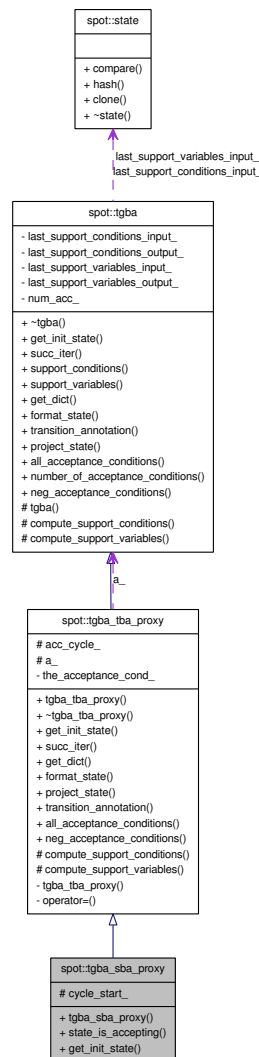
This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly.

```
#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::iterator [cycle\\_start\\_](#)
- [cycle\\_list](#) [acc\\_cycle\\_](#)
- const [tgba](#) \* [a\\_](#)

### 7.167.1 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](#).)

### 7.167.2 Member Typedef Documentation

**7.167.2.1** `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list [inherited]`

### 7.167.3 Constructor & Destructor Documentation

**7.167.3.1** `spot::tgba_sba_proxy::tgba_sba_proxy (const tgba * a)`

### 7.167.4 Member Function Documentation

**7.167.4.1** `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 accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.167.4.2** `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](#).

**7.167.4.3** `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](#).

**7.167.4.4** `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 that owns the state.

Implements [spot::tgba](#).

**7.167.4.5** `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](#).

**7.167.4.6** `virtual state* spot::tgba_sba_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.

Reimplemented from [spot::tgba\\_tba\\_proxy](#).

**7.167.4.7** `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](#).

**7.167.4.8** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

**7.167.4.9** `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`.

**7.167.4.10** `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.

**7.167.4.11** `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`.

**7.167.4.12 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.

**7.167.4.13 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.

**7.167.4.14 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`.

**7.167.5 Member Data Documentation****7.167.5.1 const tgba\* spot::tgba\_tba\_proxy::a\_ [protected, inherited]****7.167.5.2 cycle\_list spot::tgba\_tba\_proxy::acc\_cycle\_ [protected, inherited]**

## 7.167.5.3 cycle\_list::iterator spot::tgba\_sba\_proxy::cycle\_start\_ [protected]

The documentation for this class was generated from the following file:

- [tgba/tgbatba.hh](#)

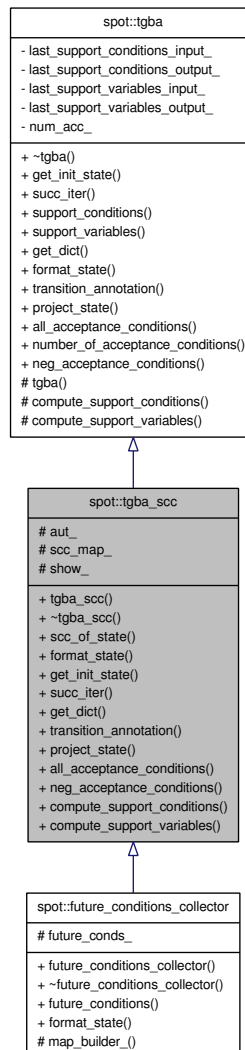
## 7.168 spot::tgba\_scc Class Reference

Wrap a tgba to offer information about strongly connected components.

This class is a [spot::tgba](#) wrapper that simply add a new method [scc\\_of\\_state\(\)](#) to retrieve the number of a SCC a state belongs to.

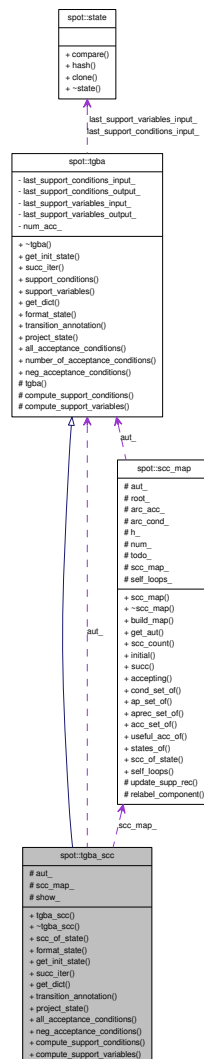
```
#include <tgba/tgbascc.hh>
```

Inheritance diagram for spot::tgba\_scc:





Collaboration diagram for spot::tgba\_scc:



## Public Member Functions

- [tgba\\_scc](#) (const [tgba](#) \*aut, bool show=false)  
Create a [tgba\\_scc](#) wrapper for aut.
- virtual [~tgba\\_scc](#) ()
- unsigned [scc\\_of\\_state](#) (const [spot::state](#) \*s) const  
Returns the number of the SCC s belongs to.
- virtual std::string [format\\_state](#) (const [state](#) \*state) const  
Format a state for output.
- 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 [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.*
- 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().*
- [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 Attributes

- const [tgba](#) \* [aut\\_](#)
- [scc\\_map](#) [scc\\_map\\_](#)
- bool [show\\_](#)

### 7.168.1 Detailed Description

Wrap a tgba to offer information about strongly connected components.

This class is a [spot::tgba](#) wrapper that simply add a new method [scc\\_of\\_state\(\)](#) to retrieve the number of a SCC a state belongs to.

## 7.168.2 Constructor & Destructor Documentation

### 7.168.2.1 spot::tgba\_scc::tgba\_scc (const tgba \* *aut*, bool *show* = false)

Create a [tgba\\_scc](#) wrapper for *aut*.

If *show* is set to true, then the [format\\_state\(\)](#) method will include the SCC number computed for the given state in its output string.

### 7.168.2.2 virtual spot::tgba\_scc::~~tgba\_scc () [virtual]

## 7.168.3 Member Function Documentation

### 7.168.3.1 virtual bdd spot::tgba\_scc::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

### 7.168.3.2 virtual bdd spot::tgba\_scc::compute\_support\_conditions (const state \* *state*) const [virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

### 7.168.3.3 virtual bdd spot::tgba\_scc::compute\_support\_variables (const state \* *state*) const [virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

### 7.168.3.4 virtual std::string spot::tgba\_scc::format\_state (const state \* *state*) const [virtual]

Format a state for output.

If the constructor was called with *show* set to true, then this method will include the SCC number computed for *state* in the output string.

Implements [spot::tgba](#).

Reimplemented in [spot::future\\_conditions\\_collector](#).

### 7.168.3.5 virtual bdd\_dict\* spot::tgba\_scc::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

### 7.168.3.6 virtual state\* spot::tgba\_scc::get\_init\_state () const [virtual]

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

### 7.168.3.7 virtual bdd spot::tgba\_scc::neg\_acceptance\_conditions () const [virtual]

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

### 7.168.3.8 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]

The number of acceptance conditions.

### 7.168.3.9 virtual state\* spot::tgba\_scc::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 from [spot::tgba](#).

#### 7.168.3.10 unsigned spot::tgba\_scc::scc\_of\_state (const spot::state \* s) const

Returns the number of the SCC  $s$  belongs to.

#### 7.168.3.11 virtual tgba\_succ\_iterator\* spot::tgba\_scc::succ\_iter (const state \* local\_state, const state \* global\_state = 0, const tgba \* global\_automaton = 0) const [virtual]

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of [spot::tgba](#) where most values are computed on demand. *global\_automaton* designate the root [spot::tgba](#), and *global\_state* its state. This two objects can be used by [succ\\_iter\(\)](#) to restrict the set of successors to compute.

### Parameters

**local\_state** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

**global\_state** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

**global\_automaton** In a product, the global product automaton. Otherwise, 0.

Implements [spot::tgba](#).

#### 7.168.3.12 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.

**7.168.3.13 `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.

**7.168.3.14 `virtual std::string spot::tgba_scc::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 from `spot::tgba`.

**7.168.4 Member Data Documentation****7.168.4.1 `const tgba* spot::tgba_scc::aut_` [protected]****7.168.4.2 `scc_map spot::tgba_scc::scc_map_` [protected]****7.168.4.3 `bool spot::tgba_scc::show_` [protected]**

The documentation for this class was generated from the following file:

- `tgba/tgbascc.hh`

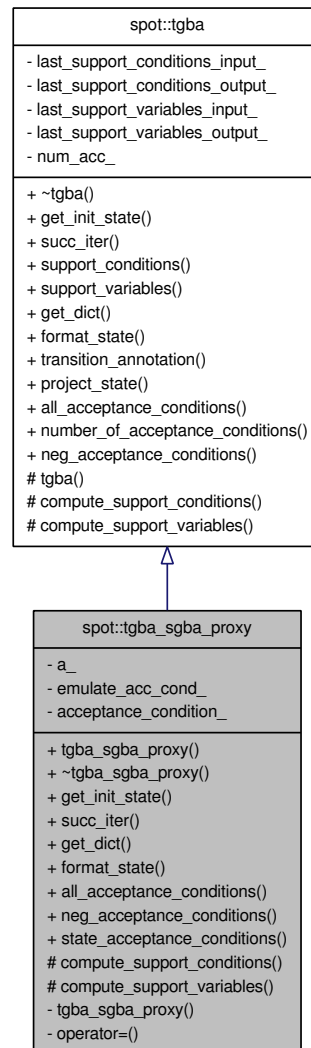
**7.169 `spot::tgba_sgba_proxy` Class Reference**

Change the labeling-mode of `spot::tgba` on the fly, producing a state-based generalized Büchi automaton.

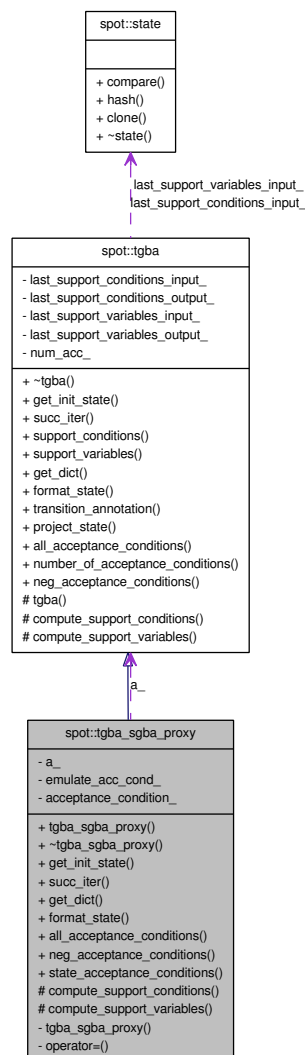
This class acts as a proxy in front of a `spot::tgba`, that should label on states on-the-fly. The result is still a `spot::tgba`, but acceptances conditions are also on states.

```
#include <tgba/tgbasgba.hh>
```

Inheritance diagram for spot::tgba\_sgba\_proxy:



Collaboration diagram for spot::tgba\_sgba\_proxy:



## Public Member Functions

- `tgba_sgba_proxy` (const `tgba` \*a, bool no\_zero\_acc=true)
- virtual `~tgba_sgba_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 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 [state\\_acceptance\\_conditions](#) (const [state](#) \*state) const  
*Retrieve the acceptance condition of a 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 [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\(\)](#).*

### Private Member Functions

- [tgba\\_sgba\\_proxy](#) (const [tgba\\_sgba\\_proxy](#) &)
- [tgba\\_sgba\\_proxy](#) & [operator=](#) (const [tgba\\_sgba\\_proxy](#) &)

### Private Attributes

- const [tgba](#) \* [a\\_](#)
- bool [emulate\\_acc\\_cond\\_](#)
- bdd [acceptance\\_condition\\_](#)

### 7.169.1 Detailed Description

Change the labeling-mode of [spot::tgba](#) on the fly, producing a state-based generalized Büchi automaton.

This class acts as a proxy in front of a [spot::tgba](#), that should label on states on-the-fly. The result is still a [spot::tgba](#), but acceptances conditions are also on states.

### 7.169.2 Constructor & Destructor Documentation

**7.169.2.1** `spot::tgba_sgba_proxy::tgba_sgba_proxy (const tgba * a, bool no_zero_acc = true)`

**7.169.2.2** `virtual spot::tgba_sgba_proxy::~~tgba_sgba_proxy ()` **[virtual]**

**7.169.2.3** `spot::tgba_sgba_proxy::tgba_sgba_proxy (const tgba_sgba_proxy &)` **[private]**

### 7.169.3 Member Function Documentation

**7.169.3.1** `virtual bdd spot::tgba_sgba_proxy::all_acceptance_conditions () const` **[virtual]**

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

**7.169.3.2** `virtual bdd spot::tgba_sgba_proxy::compute_support_conditions (const state * state) const` **[protected, virtual]**

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.169.3.3** `virtual bdd spot::tgba_sgba_proxy::compute_support_variables (const state * state) const` **[protected, virtual]**

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.169.3.4 virtual std::string spot::tgba\_sgba\_proxy::format\_state (const state \* state) const [virtual]**

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

**7.169.3.5 virtual bdd\_dict\* spot::tgba\_sgba\_proxy::get\_dict () const [virtual]**

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.169.3.6 virtual state\* spot::tgba\_sgba\_proxy::get\_init\_state () const [virtual]**

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to delete it when no longer needed.

Implements [spot::tgba](#).

**7.169.3.7 virtual bdd spot::tgba\_sgba\_proxy::neg\_acceptance\_conditions () const [virtual]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.169.3.8 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**7.169.3.9** `tgba_sgba_proxy& spot::tgba_sgba_proxy::operator= (const tgba_sgba_proxy &)`  
`[private]`

**7.169.3.10** `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`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

**7.169.3.11** `bdd spot::tgba_sgba_proxy::state_acceptance_conditions (const state * state) const`

Retrieve the acceptance condition of a state.

**7.169.3.12** `virtual tgba_succ_iterator* spot::tgba_sgba_proxy::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const` `[virtual]`

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

#### Parameters

***local\_state*** The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

***global\_state*** In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

***global\_automaton*** In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**7.169.3.13 bdd spot::tgba::support\_conditions (const state \* *state*) const [inherited]**

Get a formula that must hold whatever successor is taken.

**Returns**

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_conditions()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.169.3.14 bdd spot::tgba::support\_variables (const state \* *state*) const [inherited]**

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

**7.169.3.15 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`, `spot::tgba_scc`, and `spot::tgba_tba_proxy`.

**7.169.4 Member Data Documentation****7.169.4.1 const tgba\* spot::tgba\_sgba\_proxy::a\_ [private]****7.169.4.2 bdd spot::tgba\_sgba\_proxy::acceptance\_condition\_ [private]**

#### 7.169.4.3 `bool spot::tgba_sgba_proxy::emulate_acc_cond_` `[private]`

The documentation for this class was generated from the following file:

- [tgba/tgbasgba.hh](#)

### 7.170 `spot::tgba_statistics` Struct Reference

```
#include <tgbalgorithms/stats.hh>
```

#### Public Member Functions

- `std::ostream & dump` (`std::ostream &out`) `const`

#### Public Attributes

- unsigned [transitions](#)
- unsigned [states](#)

#### 7.170.1 Member Function Documentation

##### 7.170.1.1 `std::ostream& spot::tgba_statistics::dump` (`std::ostream & out`) `const`

#### 7.170.2 Member Data Documentation

##### 7.170.2.1 unsigned `spot::tgba_statistics::states`

##### 7.170.2.2 unsigned `spot::tgba_statistics::transitions`

The documentation for this struct was generated from the following file:

- [tgbalgorithms/stats.hh](#)

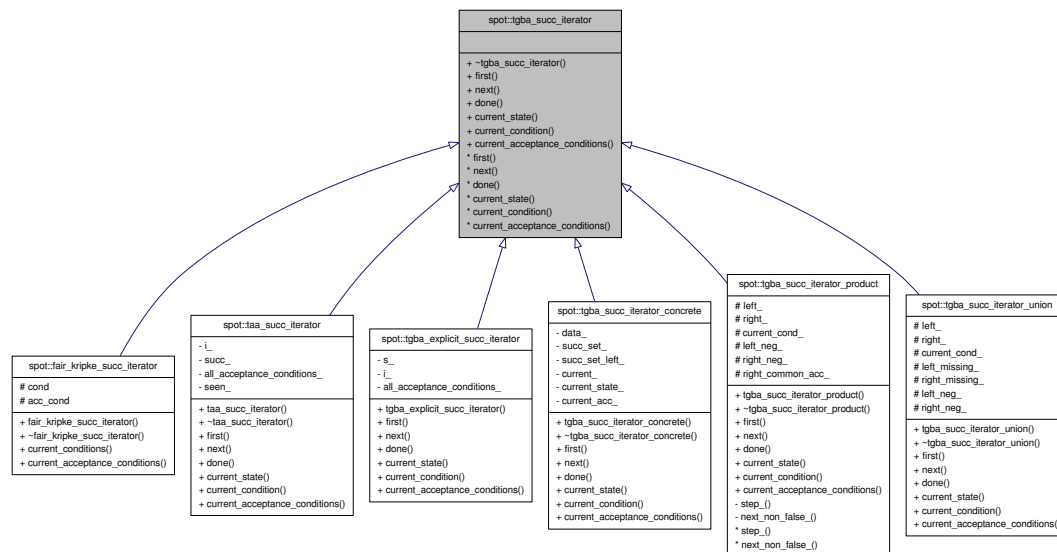
### 7.171 `spot::tgba_succ_iterator` Class Reference

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.

```
#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.*

## 7.171.1 Detailed Description

Iterate over the successors of a state.

This class provides the basic functionalities required to iterate over the successors of a state, as well as querying transition labels. Because transitions are never explicitly encoded, labels (conditions and acceptance conditions) can only be queried while iterating over the successors.

### 7.171.2 Constructor & Destructor Documentation

**7.171.2.1** `virtual spot::tgba_succ_iterator::~tgba_succ_iterator () [inline, virtual]`

### 7.171.3 Member Function Documentation

**7.171.3.1** `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::fair\\_kripke\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_concrete](#), [spot::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.171.3.2** `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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.171.3.3** `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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

**7.171.3.4** `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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

#### 7.171.3.5 `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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

#### 7.171.3.6 `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::taa\\_succ\\_iterator](#), [spot::tgba\\_explicit\\_succ\\_iterator](#), [spot::tgba\\_succ\\_iterator\\_product](#), and [spot::tgba\\_succ\\_iterator\\_union](#).

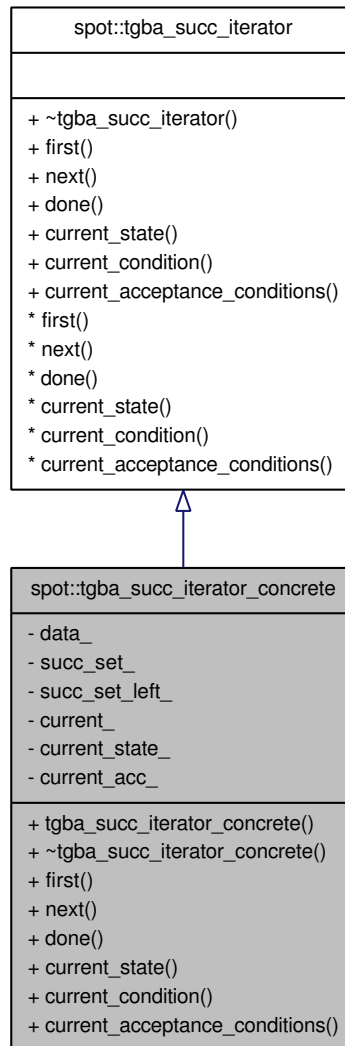
The documentation for this class was generated from the following file:

- [tgba/succiter.hh](#)

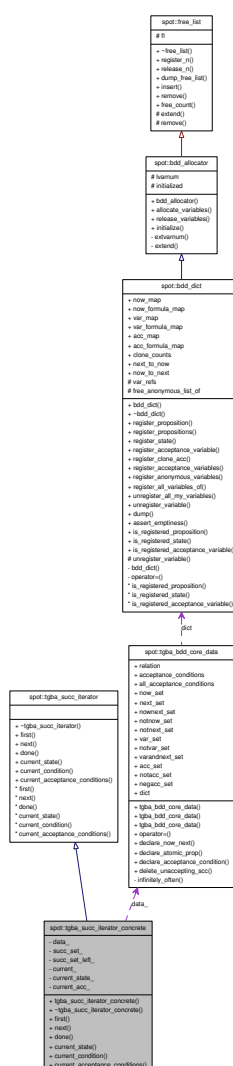
## 7.172 `spot::tgba_succ_iterator_concrete` Class Reference

```
#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.*

### 7.172.1 Detailed Description

A concrete iterator over successors of a TGBA state.

### 7.172.2 Constructor & Destructor Documentation

#### 7.172.2.1 `spot::tgba_succ_iterator_concrete::tgba_succ_iterator_concrete (const tgba_bdd_core_data & d, bdd successors)`

Build a `spot::tgba_succ_iterator_concrete`.

### Parameters

- successors* The set of successors with ingoing conditions and acceptance conditions, represented as a BDD. The job of this iterator will be to enumerate the satisfactions of that BDD and split them into destination states and conditions, and compute acceptance conditions.
- d* The core data of the automata. These contains sets of variables useful to split a BDD, and compute acceptance conditions.

**7.172.2.2** `virtual spot::tgba_succ_iterator_concrete::~~tgba_succ_iterator_concrete ()`  
[**virtual**]

### 7.172.3 Member Function Documentation

**7.172.3.1** `bdd spot::tgba_succ_iterator_concrete::current_acceptance_conditions () const`  
[**virtual**]

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.172.3.2** `bdd spot::tgba_succ_iterator_concrete::current_condition () const` [**virtual**]

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.172.3.3** `state_bdd* spot::tgba_succ_iterator_concrete::current_state () const` [**virtual**]

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

**7.172.3.4** `bool spot::tgba_succ_iterator_concrete::done () const` [**virtual**]

Check whether the iteration is finished.

This function should be called after any call to [first\(\)](#) or [next\(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())  
...
```

Implements [spot::tgba\\_succ\\_iterator](#).

**7.172.3.5** `void spot::tgba_succ_iterator_concrete::first ()` [**virtual**]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

#### Warning

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements `spot::tgba_succ_iterator`.

#### 7.172.3.6 `void spot::tgba_succ_iterator_concrete::next()` [virtual]

Jump to the next successor (if any).

#### Warning

Again, one should always call `done()` to ensure there is a successor.

Implements `spot::tgba_succ_iterator`.

### 7.172.4 Member Data Documentation

#### 7.172.4.1 `bdd spot::tgba_succ_iterator_concrete::current_` [private]

Current successor, as a conjunction of atomic proposition and Next variables.

#### 7.172.4.2 `bdd spot::tgba_succ_iterator_concrete::current_acc_` [private]

Acceptance conditions for the current transition.

#### 7.172.4.3 `bdd spot::tgba_succ_iterator_concrete::current_state_` [private]

Current successor, as a conjunction of Now variables.

#### 7.172.4.4 `const tgba_bdd_core_data& spot::tgba_succ_iterator_concrete::data_` [private]

Core data of the automaton.

#### 7.172.4.5 `bdd spot::tgba_succ_iterator_concrete::succ_set_` [private]

The set of successors.

## 7.172.4.6 bdd spot::tgba\_succ\_iterator\_concrete::succ\_set\_left\_ [private]

Unexplored successors (including current\_).

The documentation for this class was generated from the following file:

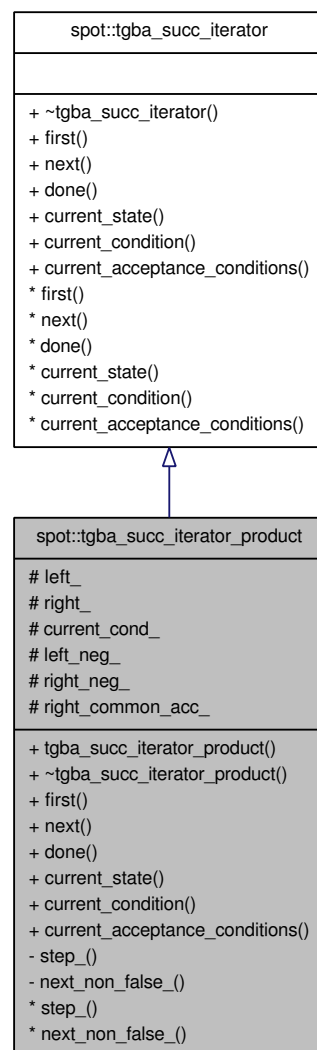
- [tgba/succiterconcrete.hh](#)

## 7.173 spot::tgba\_succ\_iterator\_product Class Reference

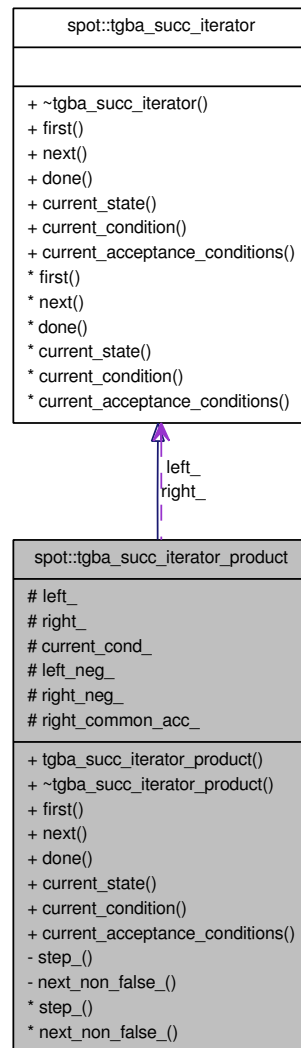
Iterate over the successors of a product computed on the fly.

```
#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, bddPair \*right\_common\_acc)
- 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\\_](#)
- bddPair \* [right\\_common\\_acc\\_](#)

### Private Member Functions

- void [step\\_](#) ()  
*Internal routines to advance to the next successor.*
- void [next\\_non\\_false\\_](#) ()

### Friends

- class [tgba\\_product](#)

#### 7.173.1 Detailed Description

Iterate over the successors of a product computed on the fly.

#### 7.173.2 Constructor & Destructor Documentation

**7.173.2.1** `spot::tgba_succ_iterator_product::tgba_succ_iterator_product (tgba_succ_iterator * left, tgba_succ_iterator * right, bdd left_neg, bdd right_neg, bddPair * right_common_acc)`

**7.173.2.2** `virtual spot::tgba_succ_iterator_product::~~tgba_succ_iterator_product ()`  
`[virtual]`

### 7.173.3 Member Function Documentation

#### 7.173.3.1 `bdd spot::tgba_succ_iterator_product::current_acceptance_conditions () const` **[virtual]**

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.173.3.2 `bdd spot::tgba_succ_iterator_product::current_condition () const` **[virtual]**

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.173.3.3 `state_product* spot::tgba_succ_iterator_product::current_state () const` **[virtual]**

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.173.3.4 `bool spot::tgba_succ_iterator_product::done () const` **[virtual]**

Check whether the iteration is finished.

This function should be called after any call to [first \(\)](#) or [next \(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first(); !s->done(); s->next())  
...
```

Implements [spot::tgba\\_succ\\_iterator](#).

#### 7.173.3.5 `void spot::tgba_succ_iterator_product::first ()` **[virtual]**

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

### Warning

One should always call `done()` to ensure there is a successor, even after `first()`. A common trap is to assume that there is at least one successor: this is wrong.

Implements `spot::tgba_succ_iterator`.

#### 7.173.3.6 void spot::tgba\_succ\_iterator\_product::next() [virtual]

Jump to the next successor (if any).

### Warning

Again, one should always call `done()` to ensure there is a successor.

Implements `spot::tgba_succ_iterator`.

#### 7.173.3.7 void spot::tgba\_succ\_iterator\_product::next\_non\_false() [private]

#### 7.173.3.8 void spot::tgba\_succ\_iterator\_product::step() [private]

Internal routines to advance to the next successor.

### 7.173.4 Friends And Related Function Documentation

#### 7.173.4.1 friend class tgba\_product [friend]

### 7.173.5 Member Data Documentation

#### 7.173.5.1 bdd spot::tgba\_succ\_iterator\_product::current\_cond\_ [protected]

#### 7.173.5.2 tgba\_succ\_iterator\* spot::tgba\_succ\_iterator\_product::left\_ [protected]

#### 7.173.5.3 bdd spot::tgba\_succ\_iterator\_product::left\_neg\_ [protected]

7.173.5.4 `tgba_succ_iterator*` `spot::tgba_succ_iterator_product::right_` `[protected]`

7.173.5.5 `bddPair*` `spot::tgba_succ_iterator_product::right_common_acc_` `[protected]`

7.173.5.6 `bdd` `spot::tgba_succ_iterator_product::right_neg_` `[protected]`

The documentation for this class was generated from the following file:

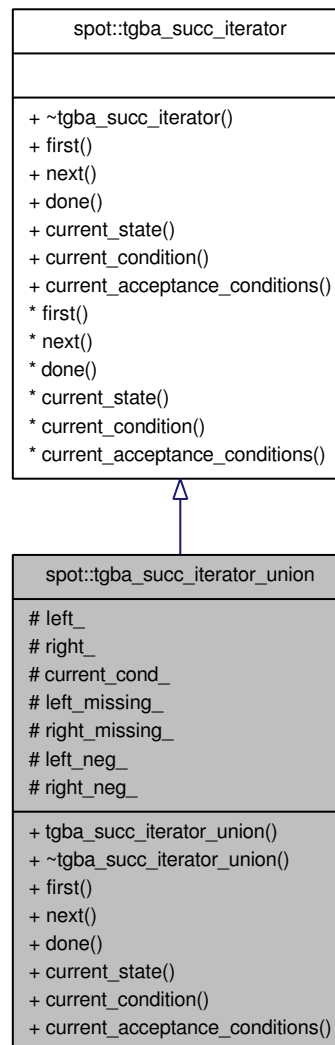
- `tgba/tgbaproduct.hh`

## 7.174 `spot::tgba_succ_iterator_union` Class Reference

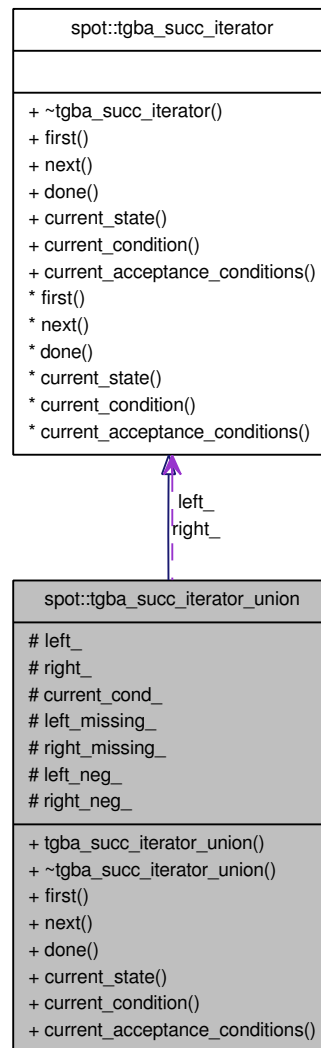
Iterate over the successors of an union computed on the fly.

```
#include <tgba/tgbaunion.hh>
```

Inheritance diagram for spot::tgba\_succ\_iterator\_union:



Collaboration diagram for spot::tgba\_succ\_iterator\_union:



## Public Member Functions

- `tgba_succ_iterator_union` (`tgba_succ_iterator` \*left, `tgba_succ_iterator` \*right, bdd left\_missing, bdd right\_missing, bdd left\_var, bdd right\_var)
- virtual `~tgba_succ_iterator_union` ()
- 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_union` \* `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\\_missing\\_](#)
- bdd [right\\_missing\\_](#)
- bdd [left\\_neg\\_](#)
- bdd [right\\_neg\\_](#)

### Friends

- class [tgba\\_union](#)

#### 7.174.1 Detailed Description

Iterate over the successors of an union computed on the fly.

#### 7.174.2 Constructor & Destructor Documentation

- 7.174.2.1** `spot::tgba_succ_iterator_union::tgba_succ_iterator_union (tgba_succ_iterator * left, tgba_succ_iterator * right, bdd left_missing, bdd right_missing, bdd left_var, bdd right_var)`

- 7.174.2.2** `virtual spot::tgba_succ_iterator_union::~~tgba_succ_iterator_union ()` **[virtual]**

#### 7.174.3 Member Function Documentation

- 7.174.3.1** `bdd spot::tgba_succ_iterator_union::current_acceptance_conditions () const`  
**[virtual]**

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.174.3.2 bdd spot::tgba\_succ\_iterator\_union::current\_condition () const [virtual]

Get the condition on the transition leading to this successor.

This is a boolean function of atomic propositions.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.174.3.3 state\_union\* spot::tgba\_succ\_iterator\_union::current\_state () const [virtual]

Get the state of the current successor.

Note that the same state may occur at different points in the iteration. These actually correspond to the same destination. It just means there were several transitions, with different conditions, leading to the same state.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.174.3.4 bool spot::tgba\_succ\_iterator\_union::done () const [virtual]

Check whether the iteration is finished.

This function should be called after any call to [first \(\)](#) or [next \(\)](#) and before any enquiry about the current state.

The usual way to do this is with a `for` loop.

```
for (s->first (); !s->done (); s->next ())  
...
```

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.174.3.5 void spot::tgba\_succ\_iterator\_union::first () [virtual]

Position the iterator on the first successor (if any).

This method can be called several times to make multiple passes over successors.

#### Warning

One should always call [done \(\)](#) to ensure there is a successor, even after [first \(\)](#). A common trap is to assume that there is at least one successor: this is wrong.

Implements [spot::tgba\\_succ\\_iterator](#).

### 7.174.3.6 void spot::tgba\_succ\_iterator\_union::next () [virtual]

Jump to the next successor (if any).



## Warning

Again, one should always call `done ()` to ensure there is a successor.

Implements [spot::tgba\\_succ\\_iterator](#).

## 7.174.4 Friends And Related Function Documentation

### 7.174.4.1 friend class tgba\_union [friend]

## 7.174.5 Member Data Documentation

### 7.174.5.1 bdd spot::tgba\_succ\_iterator\_union::current\_cond\_ [protected]

### 7.174.5.2 tgba\_succ\_iterator\* spot::tgba\_succ\_iterator\_union::left\_ [protected]

### 7.174.5.3 bdd spot::tgba\_succ\_iterator\_union::left\_missing\_ [protected]

### 7.174.5.4 bdd spot::tgba\_succ\_iterator\_union::left\_neg\_ [protected]

### 7.174.5.5 tgba\_succ\_iterator\* spot::tgba\_succ\_iterator\_union::right\_ [protected]

### 7.174.5.6 bdd spot::tgba\_succ\_iterator\_union::right\_missing\_ [protected]

### 7.174.5.7 bdd spot::tgba\_succ\_iterator\_union::right\_neg\_ [protected]

The documentation for this class was generated from the following file:

- [tgba/tgbaunion.hh](#)

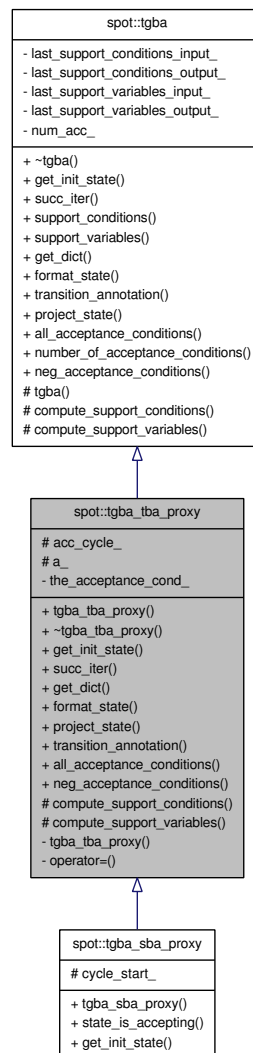
## 7.175 spot::tgba\_tba\_proxy Class Reference

Degeneralize a [spot::tgba](#) on the fly, producing a TBA.

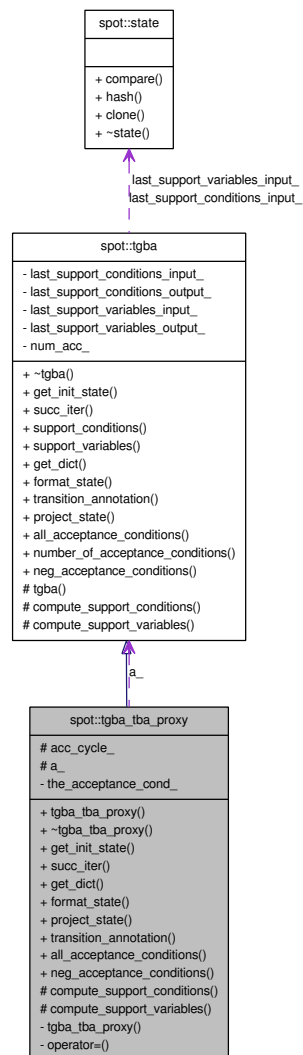
This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly. The result is still a [spot::tgba](#), but it will always have exactly one acceptance condition so it could be called TBA (without the G).

```
#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_`
- `const tgba * a_`

### Private Member Functions

- `tgba_tba_proxy (const tgba_tba_proxy &)`
- `tgba_tba_proxy & operator= (const tgba_tba_proxy &)`

### Private Attributes

- `bdd the_acceptance_cond_`

### 7.175.1 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](#)

### 7.175.2 Member Typedef Documentation

**7.175.2.1** `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list`

### 7.175.3 Constructor & Destructor Documentation

**7.175.3.1** `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba * a)`

**7.175.3.2** `virtual spot::tgba_tba_proxy::~~tgba_tba_proxy ()` **[virtual]**

**7.175.3.3** `spot::tgba_tba_proxy::tgba_tba_proxy (const tgba_tba_proxy &)` **[private]**

### 7.175.4 Member Function Documentation

**7.175.4.1** `virtual bdd spot::tgba_tba_proxy::all_acceptance_conditions () const` **[virtual]**

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements `spot::tgba`.

**7.175.4.2** `virtual bdd spot::tgba_tba_proxy::compute_support_conditions (const state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

**7.175.4.3** `virtual bdd spot::tgba_tba_proxy::compute_support_variables (const state * state) const [protected, virtual]`

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

**7.175.4.4** `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 that owns the state.

Implements [spot::tgba](#).

**7.175.4.5** `virtual bdd_dict* spot::tgba_tba_proxy::get_dict () const [virtual]`

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.175.4.6** `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](#).

Reimplemented in [spot::tgba\\_sba\\_proxy](#).

**7.175.4.7** `virtual bdd spot::tgba_tba_proxy::neg_acceptance_conditions () const [virtual]`

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.175.4.8** `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`  
`[virtual, inherited]`

The number of acceptance conditions.

**7.175.4.9** `tgba_tba_proxy& spot::tgba_tba_proxy::operator= (const tgba_tba_proxy &)`  
`[private]`

**7.175.4.10** `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](#).

**7.175.4.11** `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]`

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.

Implements [spot::tgba](#).

**7.175.4.12 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.

**7.175.4.13 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.

**7.175.4.14 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 is the empty string.

**Parameters**

*t* a non-done [tgba\\_succ\\_iterator](#) for this automata

Reimplemented from [spot::tgba](#).



### 7.175.5 Member Data Documentation

7.175.5.1 `const tgba* spot::tgba_tba_proxy::a_` `[protected]`

7.175.5.2 `cycle_list spot::tgba_tba_proxy::acc_cycle_` `[protected]`

7.175.5.3 `bdd spot::tgba_tba_proxy::the_acceptance_cond_` `[private]`

The documentation for this class was generated from the following file:

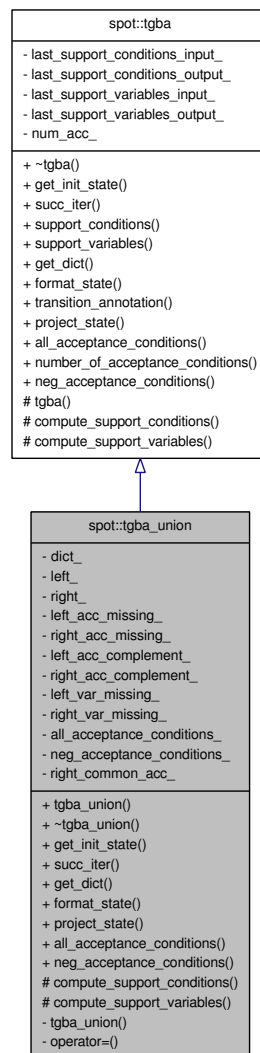
- [tgba/tgbatba.hh](#)

## 7.176 spot::tgba\_union Class Reference

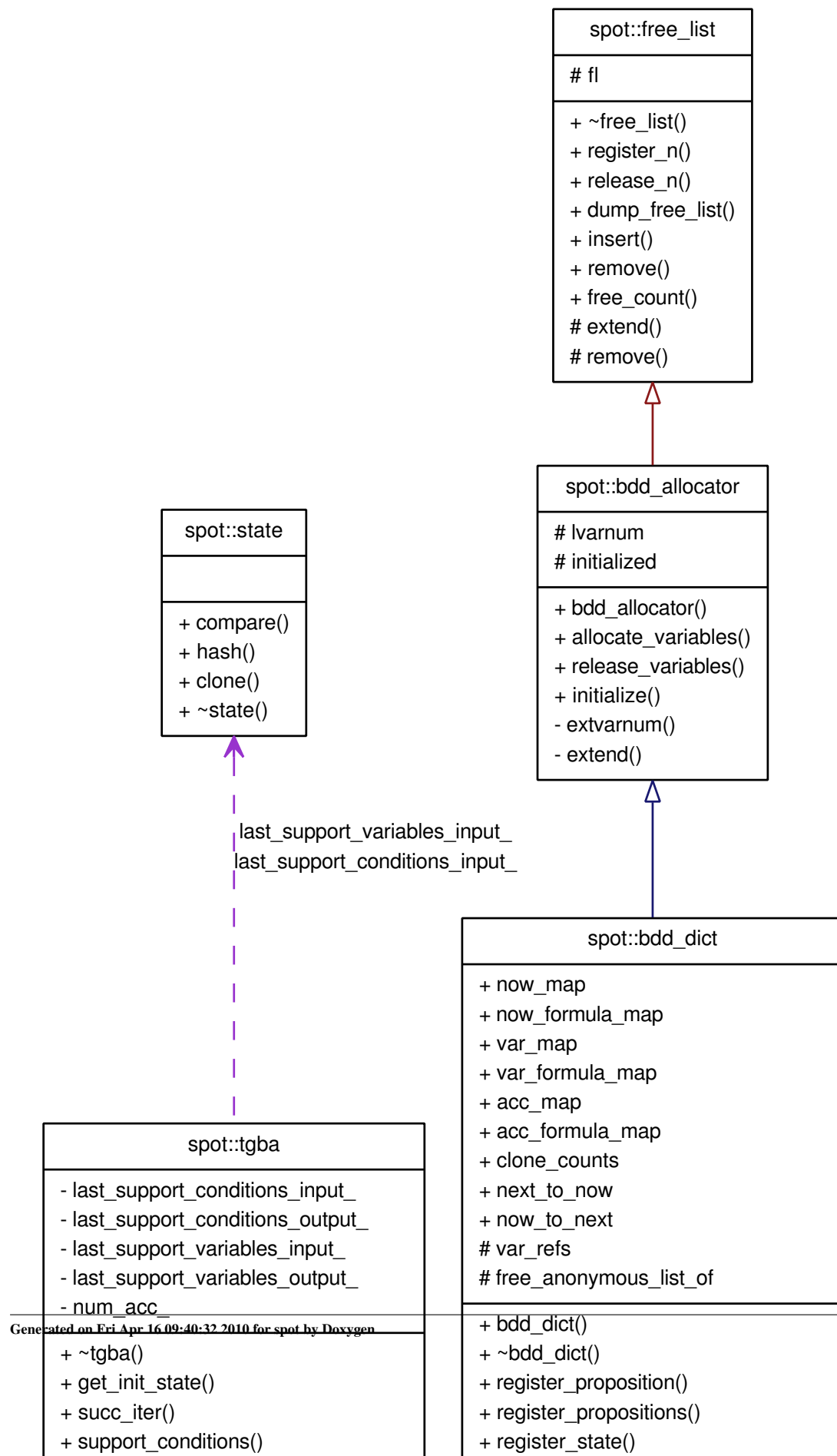
A lazy union. (States are computed on the fly.).

```
#include <tgba/tgbaunion.hh>
```

Inheritance diagram for spot::tgba\_union:



Collaboration diagram for spot::tgba\_union:



## Public Member Functions

- [tgba\\_union](#) (const [tgba](#) \*left, const [tgba](#) \*right)  
*Constructor.*
- virtual [~tgba\\_union](#) ()
- virtual [state](#) \* [get\\_init\\_state](#) () const  
*Get the initial state of the automaton.*
- virtual [tgba\\_succ\\_iterator\\_union](#) \* [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 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 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\\_union](#) (const [tgba\\_union](#) &)
- [tgba\\_union](#) & [operator=](#) (const [tgba\\_union](#) &)

### Private Attributes

- [bdd\\_dict](#) \* [dict\\_](#)
- const [tgba](#) \* [left\\_](#)
- const [tgba](#) \* [right\\_](#)
- [bdd](#) [left\\_acc\\_missing\\_](#)
- [bdd](#) [right\\_acc\\_missing\\_](#)
- [bdd](#) [left\\_acc\\_complement\\_](#)
- [bdd](#) [right\\_acc\\_complement\\_](#)
- [bdd](#) [left\\_var\\_missing\\_](#)
- [bdd](#) [right\\_var\\_missing\\_](#)
- [bdd](#) [all\\_acceptance\\_conditions\\_](#)
- [bdd](#) [neg\\_acceptance\\_conditions\\_](#)
- [bddPair](#) \* [right\\_common\\_acc\\_](#)

#### 7.176.1 Detailed Description

A lazy union. (States are computed on the fly.).

#### 7.176.2 Constructor & Destructor Documentation

##### 7.176.2.1 spot::tgba\_union::tgba\_union (const [tgba](#) \* *left*, const [tgba](#) \* *right*)

Constructor.

#### Parameters

*left* The left automata in the union.

*right* The right automata in the union.

##### 7.176.2.2 virtual spot::tgba\_union::~tgba\_union () [virtual]

##### 7.176.2.3 spot::tgba\_union::tgba\_union (const [tgba\\_union](#) &) [private]

### 7.176.3 Member Function Documentation

#### 7.176.3.1 virtual bdd spot::tgba\_union::all\_acceptance\_conditions () const [virtual]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

#### 7.176.3.2 virtual bdd spot::tgba\_union::compute\_support\_conditions (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_conditions\(\)](#).

Implements [spot::tgba](#).

#### 7.176.3.3 virtual bdd spot::tgba\_union::compute\_support\_variables (const state \* state) const [protected, virtual]

Do the actual computation of [tgba::support\\_variables\(\)](#).

Implements [spot::tgba](#).

#### 7.176.3.4 virtual std::string spot::tgba\_union::format\_state (const state \* state) const [virtual]

Format the state as a string for printing.

This formatting is the responsibility of the automata that owns the state.

Implements [spot::tgba](#).

#### 7.176.3.5 virtual bdd\_dict\* spot::tgba\_union::get\_dict () const [virtual]

Get the dictionary associated to the automaton.

State are represented as BDDs. The dictionary allows to map BDD variables back to formulae, and vice versa. This is useful when dealing with several automata (which may use the same BDD variable for different formula), or simply when printing.

Implements [spot::tgba](#).

**7.176.3.6 virtual state\* spot::tgba\_union::get\_init\_state () const [virtual]**

Get the initial state of the automaton.

The state has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

Implements [spot::tgba](#).

**7.176.3.7 virtual bdd spot::tgba\_union::neg\_acceptance\_conditions () const [virtual]**

Return the conjunction of all negated acceptance variables.

For instance if the automaton uses variables `Acc[a]`, `Acc[b]` and `Acc[c]` to describe acceptance sets, this function should return `!Acc[a] & !Acc[b] & !Acc[c]`.

This is useful when making products: each operand's condition set should be augmented with the [neg\\_acceptance\\_conditions\(\)](#) of the other operand.

Implements [spot::tgba](#).

**7.176.3.8 virtual unsigned int spot::tgba::number\_of\_acceptance\_conditions () const [virtual, inherited]**

The number of acceptance conditions.

**7.176.3.9 tgba\_union& spot::tgba\_union::operator= (const tgba\_union &) [private]****7.176.3.10 virtual state\* spot::tgba\_union::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](#).

**7.176.3.11** `virtual tgba_succ_iterator_union* spot::tgba_union::succ_iter (const state * local_state,  
const state * global_state = 0, const tgba * global_automaton = 0) const` **[virtual]**

Get an iterator over the successors of *local\_state*.

The iterator has been allocated with `new`. It is the responsibility of the caller to `delete` it when no longer needed.

During synchornized products, additional informations are passed about the entire product and its state. Recall that products can be nested, forming a tree of `spot::tgba` where most values are computed on demand. *global\_automaton* designate the root `spot::tgba`, and *global\_state* its state. This two objects can be used by `succ_iter()` to restrict the set of successors to compute.

#### Parameters

*local\_state* The state whose successors are to be explored. This pointer is not adopted in any way by `succ_iter`, and it is still the caller's responsibility to delete it when appropriate (this can be done during the lifetime of the iterator).

*global\_state* In a product, the state of the global product automaton. Otherwise, 0. Like *locale\_state*, *global\_state* is not adopted by `succ_iter`.

*global\_automaton* In a product, the global product automaton. Otherwise, 0.

Implements `spot::tgba`.

**7.176.3.12** `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.

**7.176.3.13** `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.



**7.176.3.14** `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](#), [spot::tgba\\_scc](#), and [spot::tgba\\_tba\\_proxy](#).

### 7.176.4 Member Data Documentation

**7.176.4.1** `bdd spot::tgba_union::all_acceptance_conditions_` [private]

**7.176.4.2** `bdd_dict* spot::tgba_union::dict_` [private]

**7.176.4.3** `const tgba* spot::tgba_union::left_` [private]

**7.176.4.4** `bdd spot::tgba_union::left_acc_complement_` [private]

**7.176.4.5** `bdd spot::tgba_union::left_acc_missing_` [private]

**7.176.4.6** `bdd spot::tgba_union::left_var_missing_` [private]

**7.176.4.7** `bdd spot::tgba_union::neg_acceptance_conditions_` [private]

**7.176.4.8** `const tgba* spot::tgba_union::right_` [private]

**7.176.4.9** `bdd spot::tgba_union::right_acc_complement_` `[private]`

**7.176.4.10** `bdd spot::tgba_union::right_acc_missing_` `[private]`

**7.176.4.11** `bddPair* spot::tgba_union::right_common_acc_` `[private]`

**7.176.4.12** `bdd spot::tgba_union::right_var_missing_` `[private]`

The documentation for this class was generated from the following file:

- [tgba/tgbaunion.hh](#)

## 7.177 `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](#)

### 7.177.1 Detailed Description

A structure to record elapsed time in clock ticks.

### 7.177.2 Constructor & Destructor Documentation

**7.177.2.1** `spot::time_info::time_info ()` `[inline]`

### 7.177.3 Member Data Documentation

#### 7.177.3.1 `clock_t spot::time_info::stime`

Referenced by `spot::timer::start()`, `spot::timer::stime()`, and `spot::timer::stop()`.

#### 7.177.3.2 `clock_t spot::time_info::utime`

Referenced by `spot::timer::start()`, `spot::timer::stop()`, and `spot::timer::utime()`.

The documentation for this struct was generated from the following file:

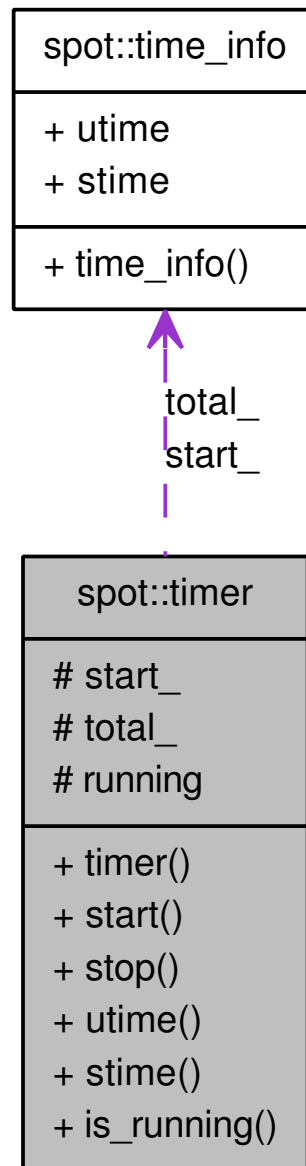
- [misc/timer.hh](#)

## 7.178 `spot::timer` Class Reference

A timekeeper that accumulate interval of time.

```
#include <misc/timer.hh>
```

Collaboration diagram for spot::timer:



### Public Member Functions

- `timer ()`
- `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.*
- `bool is_running () const`  
*Whether the timer is running.*

### Protected Attributes

- `time_info start_`
- `time_info total_`
- `bool running`

#### 7.178.1 Detailed Description

A timekeeper that accumulate interval of time.

#### 7.178.2 Constructor & Destructor Documentation

##### 7.178.2.1 `spot::timer::timer () [inline]`

#### 7.178.3 Member Function Documentation

##### 7.178.3.1 `bool spot::timer::is_running () const [inline]`

Whether the timer is running.

References `running`.

##### 7.178.3.2 `void spot::timer::start () [inline]`

Start a time interval.

References `running`, `start_`, `spot::time_info::stime`, and `spot::time_info::utime`.

##### 7.178.3.3 `clock_t spot::timer::stime () const [inline]`

Return the system time of all accumulated interval.

Any time interval that has been `start()`ed but not `stop()`ed will not be accounted for.

References `spot::time_info::stime`, and `total_`.

### 7.178.3.4 `void spot::timer::stop () [inline]`

Stop a time interval and update the sum of all intervals.

References `running`, `start_`, `spot::time_info::stime`, `total_`, and `spot::time_info::utime`.

### 7.178.3.5 `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.

References `total_`, and `spot::time_info::utime`.

## 7.178.4 Member Data Documentation

### 7.178.4.1 `bool spot::timer::running [protected]`

Referenced by `is_running()`, `start()`, and `stop()`.

### 7.178.4.2 `time_info spot::timer::start_ [protected]`

Referenced by `start()`, and `stop()`.

### 7.178.4.3 `time_info spot::timer::total_ [protected]`

Referenced by `stime()`, `stop()`, and `utime()`.

The documentation for this class was generated from the following file:

- [misc/timer.hh](#)

## 7.179 `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`

#### 7.179.1 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.

#### 7.179.2 Member Typedef Documentation

**7.179.2.1** typedef std::pair<`spot::timer`, int> `spot::timer_map::item_type` `[protected]`

**7.179.2.2** typedef std::map<std::string, `item_type`> `spot::timer_map::tm_type` `[protected]`

#### 7.179.3 Member Function Documentation

**7.179.3.1** 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.

References `tm`.

### 7.179.3.2 `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.

References `tm`.

### 7.179.3.3 `std::ostream& spot::timer_map::print (std::ostream & os) const`

Format information about all timers in a table.

### 7.179.3.4 `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.

References `tm`.

### 7.179.3.5 `void spot::timer_map::stop (const std::string & name) [inline]`

Stop timer *name*.

The timer must have been previously started with `start()`.

References `tm`.

### 7.179.3.6 `spot::timer& spot::timer_map::timer (const std::string & name) [inline]`

Return the timer *name*.

References `tm`.

### 7.179.3.7 `const spot::timer& spot::timer_map::timer (const std::string & name) const [inline]`



Return the timer *name*.

References tm.

#### 7.179.4 Member Data Documentation

##### 7.179.4.1 tm\_type spot::timer\_map::tm [protected]

Referenced by cancel(), empty(), start(), stop(), and timer().

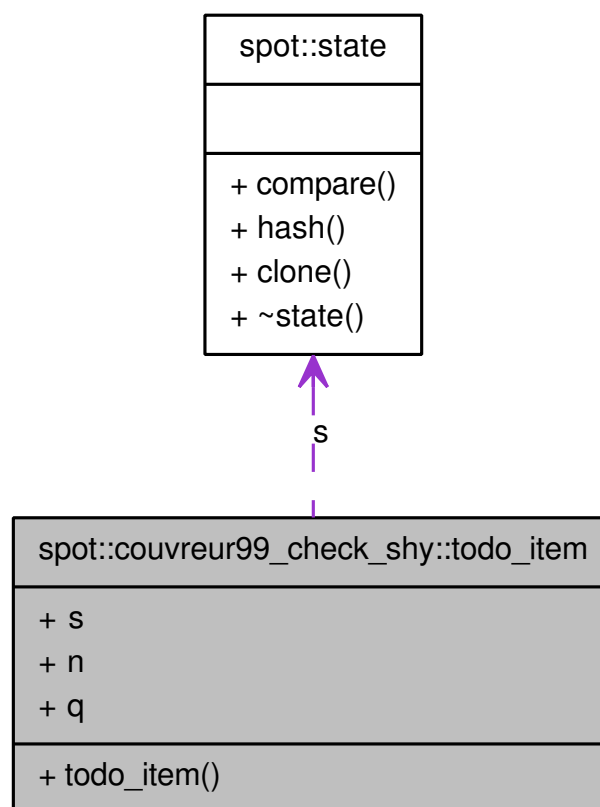
The documentation for this class was generated from the following file:

- [misc/timer.hh](#)

#### 7.180 spot::couvreur99\_check\_shy::todo\_item Struct Reference

```
#include <tgbalgos/gtec/gtec.hh>
```

Collaboration diagram for spot::couvreur99\_check\_shy::todo\_item:



#### Public Member Functions

- `todo_item` (const [state](#) \*s, int n, [couvreur99\\_check\\_shy](#) \*shy)

### Public Attributes

- `const state * s`
- `int n`
- `succ_queue q`

### 7.180.1 Constructor & Destructor Documentation

**7.180.1.1** `spot::couvreur99_check_shy::todo_item::todo_item (const state * s, int n, couvreur99_check_shy * shy)`

### 7.180.2 Member Data Documentation

**7.180.2.1** `int spot::couvreur99_check_shy::todo_item::n`

**7.180.2.2** `succ_queue spot::couvreur99_check_shy::todo_item::q`

**7.180.2.3** `const state* spot::couvreur99_check_shy::todo_item::s`

The documentation for this struct was generated from the following file:

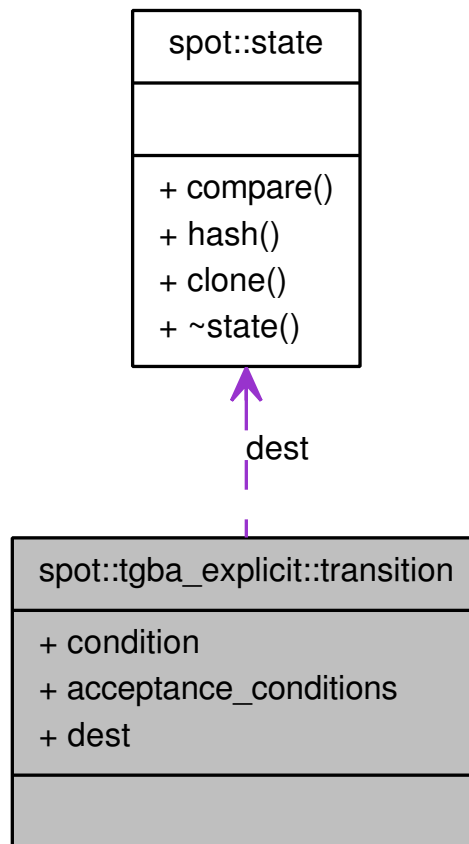
- `tgbaalgos/gtec/gtec.hh`

## 7.181 `spot::tgba_explicit::transition` Struct Reference

Explicit transitions (used by `spot::tgba_explicit`).

```
#include <tgba/tgbaexplicit.hh>
```

Collaboration diagram for spot::tgba\_explicit::transition:



#### Public Attributes

- bdd [condition](#)
- bdd [acceptance\\_conditions](#)
- const [state](#) \* [dest](#)

#### 7.181.1 Detailed Description

Explicit transitions (used by [spot::tgba\\_explicit](#)).

#### 7.181.2 Member Data Documentation

##### 7.181.2.1 bdd spot::tgba\_explicit::transition::acceptance\_conditions

##### 7.181.2.2 bdd spot::tgba\_explicit::transition::condition

### 7.181.2.3 const state\* spot::tgba\_explicit::transition::dest

The documentation for this struct was generated from the following file:

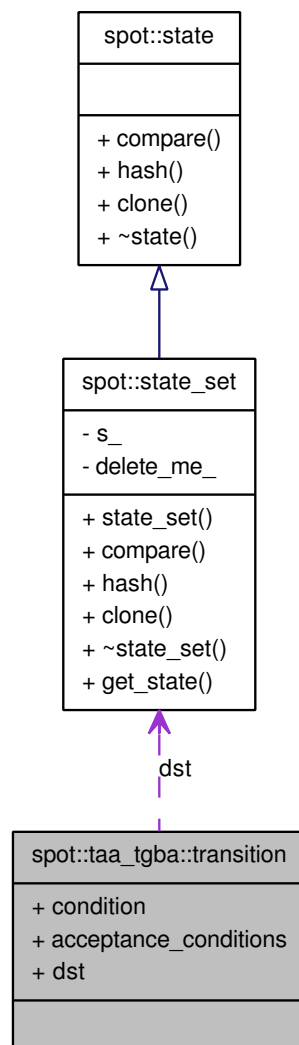
- [tgba/tgbaexplicit.hh](#)

## 7.182 spot::taa\_tgba::transition Struct Reference

Explicit transitions.

```
#include <tgba/taatgba.hh>
```

Collaboration diagram for spot::taa\_tgba::transition:



### Public Attributes

- bdd [condition](#)

- bdd [acceptance\\_conditions](#)
- const [state\\_set](#) \* [dst](#)

### 7.182.1 Detailed Description

Explicit transitions.

### 7.182.2 Member Data Documentation

#### 7.182.2.1 `bdd spot::taa_tgba::transition::acceptance_conditions`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::taa_tgba_labelled< std::string, string_hash >::create_transition()`.

#### 7.182.2.2 `bdd spot::taa_tgba::transition::condition`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::create_transition()`.

#### 7.182.2.3 `const state_set* spot::taa_tgba::transition::dst`

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::create_transition()`.

The documentation for this struct was generated from the following file:

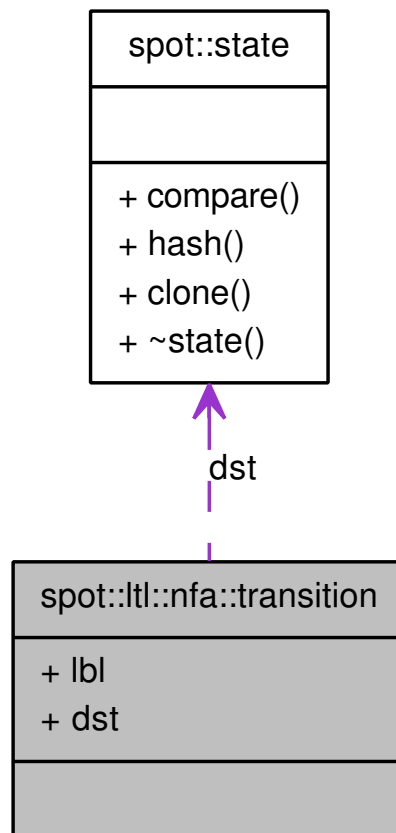
- [tgba/taatgba.hh](#)

## 7.183 `spot::ltl::nfa::transition` Struct Reference

Explicit transitions.

```
#include <ltlast/nfa.hh>
```

Collaboration diagram for spot::ltl::nfa::transition:



#### Public Attributes

- [label lbl](#)
- `const state * dst`

#### 7.183.1 Detailed Description

Explicit transitions.

#### 7.183.2 Member Data Documentation

##### 7.183.2.1 `const state* spot::ltl::nfa::transition::dst`

##### 7.183.2.2 `label spot::ltl::nfa::transition::lbl`

The documentation for this struct was generated from the following file:

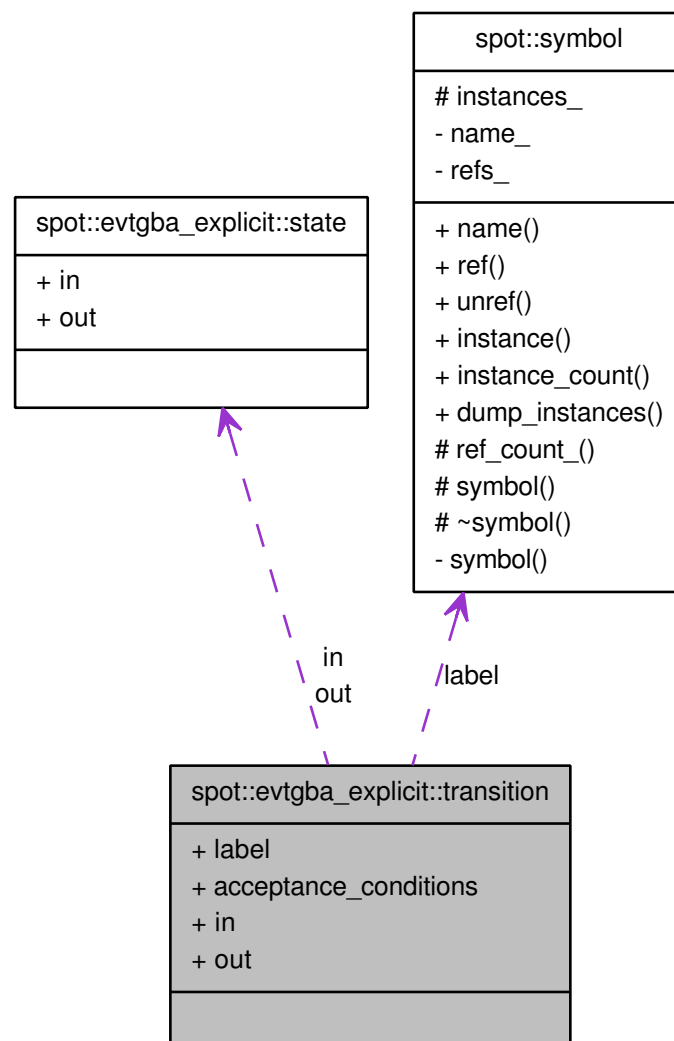
- [ltlast/nfa.hh](#)

## 7.184 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`

### 7.184.1 Detailed Description

Explicit transitions (used by [spot::evtgba\\_explicit](#)).

### 7.184.2 Member Data Documentation

**7.184.2.1** `symbol_set spot::evtgba_explicit::transition::acceptance_conditions`

**7.184.2.2** `state* spot::evtgba_explicit::transition::in`

**7.184.2.3** `const symbol* spot::evtgba_explicit::transition::label`

**7.184.2.4** `state* spot::evtgba_explicit::transition::out`

The documentation for this struct was generated from the following file:

- [evtgba/explicit.hh](#)

## 7.185 `spot::ltl::automatop::tripletcmp` Struct Reference

Comparison functor used internally by [ltl::automatop](#).

```
#include <ltlast/automatop.hh>
```

### Public Member Functions

- `bool operator() (const triplet &p1, const triplet &p2) const`

### 7.185.1 Detailed Description

Comparison functor used internally by [ltl::automatop](#).

### 7.185.2 Member Function Documentation

**7.185.2.1** `bool spot::ltl::automatop::tripletcmp::operator() (const triplet &p1, const triplet &p2) const [inline]`

The documentation for this struct was generated from the following file:

- [ltlast/automatop.hh](#)



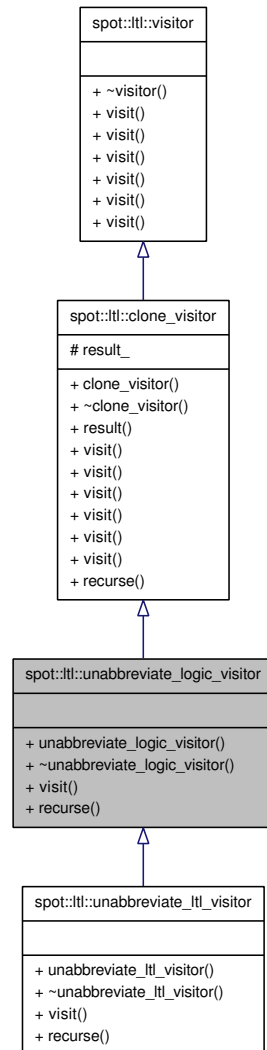
## 7.186 spot::ltl::unabbreviate\_logic\_visitor Class Reference

Clone and rewrite a formula to remove most of the abbreviated logical operators.

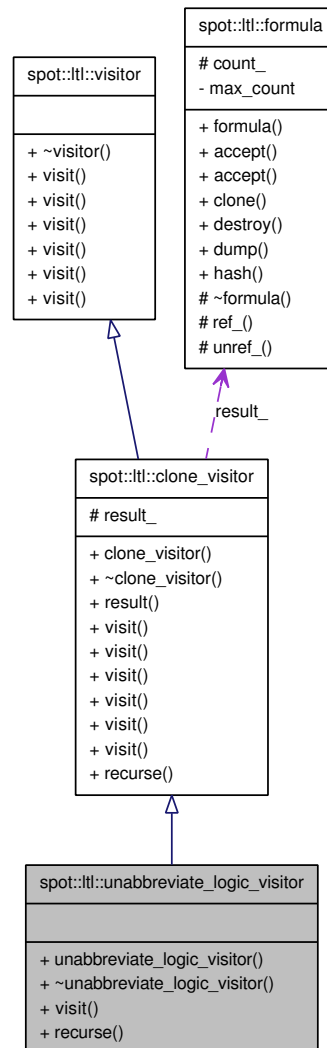
This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#).

```
#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\(automatop \\*mo\)](#)
- [void visit\(multop \\*mo\)](#)
- [void visit\(constant \\*c\)](#)

### Protected Attributes

- [formula](#) \* [result\\_](#)

### Private Types

- typedef [clone\\_visitor](#) [super](#)

#### 7.186.1 Detailed Description

Clone and rewrite a formula to remove most of the abbreviated logical operators.

This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#). This visitor is public, because it's convenient to derive from it and override some of its methods. But if you just want the functionality, consider using [spot::ltl::unabbreviate\\_logic](#) instead.

#### 7.186.2 Member Typedef Documentation

**7.186.2.1** typedef [clone\\_visitor](#) [spot::ltl::unabbreviate\\_logic\\_visitor::super](#) **[private]**

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

#### 7.186.3 Constructor & Destructor Documentation

**7.186.3.1** [spot::ltl::unabbreviate\\_logic\\_visitor::unabbreviate\\_logic\\_visitor \(\)](#)

**7.186.3.2** virtual [spot::ltl::unabbreviate\\_logic\\_visitor::~~unabbreviate\\_logic\\_visitor \(\)](#)  
**[virtual]**

#### 7.186.4 Member Function Documentation

**7.186.4.1** virtual [formula](#)\* [spot::ltl::unabbreviate\\_logic\\_visitor::recurse \(formula \\* f\)](#)  
**[virtual]**

Reimplemented from [spot::ltl::clone\\_visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**7.186.4.2** [formula](#)\* [spot::ltl::clone\\_visitor::result \(\) const](#) **[inherited]**

**7.186.4.3** `void spot::ltl::clone_visitor::visit (constant * c)` `[virtual, inherited]`

Implements [spot::ltl::visitor](#).

**7.186.4.4** `void spot::ltl::clone_visitor::visit (multop * mo)` `[virtual, inherited]`

Implements [spot::ltl::visitor](#).

**7.186.4.5** `void spot::ltl::clone_visitor::visit (automatop * mo)` `[virtual, inherited]`

Implements [spot::ltl::visitor](#).

**7.186.4.6** `void spot::ltl::clone_visitor::visit (unop * uo)` `[virtual, inherited]`

Implements [spot::ltl::visitor](#).

Reimplemented in [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**7.186.4.7** `void spot::ltl::clone_visitor::visit (atomic_prop * ap)` `[virtual, inherited]`

Implements [spot::ltl::visitor](#).

**7.186.4.8** `void spot::ltl::unabbreviate_logic_visitor::visit (binop * bo)` `[virtual]`

Reimplemented from [spot::ltl::clone\\_visitor](#).

## 7.186.5 Member Data Documentation

**7.186.5.1** `formula* spot::ltl::clone_visitor::result_` `[protected, inherited]`

The documentation for this class was generated from the following file:

- [ltlvisit/lunabbrev.hh](#)

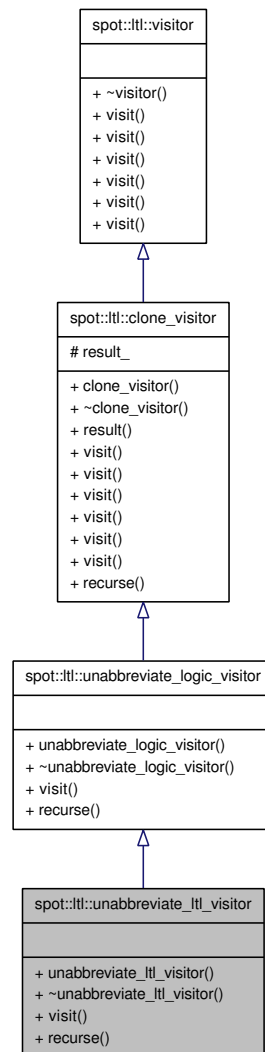
## 7.187 `spot::ltl::unabbreviate_ltl_visitor` Class Reference

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

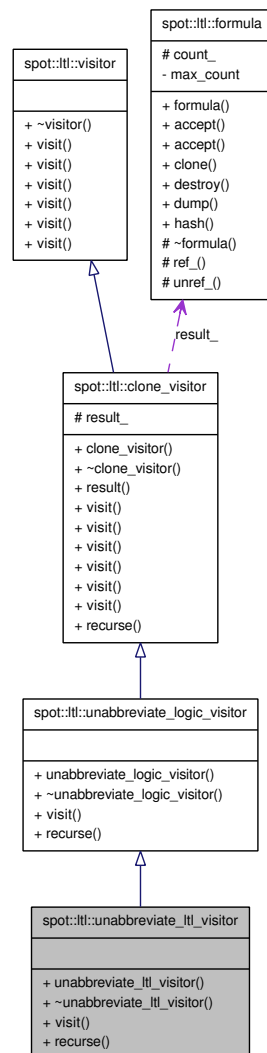
The rewriting performed on logical operator is the same as the one done by [spot::ltl::unabbreviate\\_logic\\_visitor](#).

```
#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 (automatop *mo)`
- `void visit (multop *mo)`
- `void visit (constant *c)`
- `formula * result () const`

### Protected Attributes

- `formula * result_`

### Private Types

- typedef `unabbreviate_logic_visitor` `super`

#### 7.187.1 Detailed Description

Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic_visitor`. This will also rewrite unary operators such as `unop::F`, and `unop::G`, using only `binop::U`, and `binop::R`.

This visitor is public, because it's convenient to derive from it and override some of its methods. But if you just want the functionality, consider using `spot::ltl::unabbreviate_ltl` instead.

#### 7.187.2 Member Typedef Documentation

**7.187.2.1** `typedef unabbreviate_logic_visitor spot::ltl::unabbreviate_ltl_visitor::super`  
[`private`]

Reimplemented from `spot::ltl::unabbreviate_logic_visitor`.

#### 7.187.3 Constructor & Destructor Documentation

**7.187.3.1** `spot::ltl::unabbreviate_ltl_visitor::unabbreviate_ltl_visitor ()`

**7.187.3.2** `virtual spot::ltl::unabbreviate_ltl_visitor::~~unabbreviate_ltl_visitor ()` [`virtual`]

#### 7.187.4 Member Function Documentation

**7.187.4.1** `formula* spot::ltl::unabbreviate_ltl_visitor::recurse (formula *f)` [`virtual`]

Reimplemented from `spot::ltl::unabbreviate_logic_visitor`.

**7.187.4.2** `formula* spot::ltl::clone_visitor::result () const` [`inherited`]

**7.187.4.3** void spot::ltl::clone\_visitor::visit (constant \* *c*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**7.187.4.4** void spot::ltl::clone\_visitor::visit (multop \* *mo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**7.187.4.5** void spot::ltl::clone\_visitor::visit (automatop \* *mo*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**7.187.4.6** void spot::ltl::clone\_visitor::visit (atomic\_prop \* *ap*) [virtual, inherited]

Implements [spot::ltl::visitor](#).

**7.187.4.7** void spot::ltl::unabbreviate\_logic\_visitor::visit (binop \* *bo*) [virtual, inherited]

Reimplemented from [spot::ltl::clone\\_visitor](#).

**7.187.4.8** void spot::ltl::unabbreviate\_ltl\_visitor::visit (unop \* *uo*) [virtual]

Reimplemented from [spot::ltl::clone\\_visitor](#).

## 7.187.5 Member Data Documentation

**7.187.5.1** formula\* spot::ltl::clone\_visitor::result\_ [protected, inherited]

The documentation for this class was generated from the following file:

- ltlvisit/[tunabbrev.hh](#)

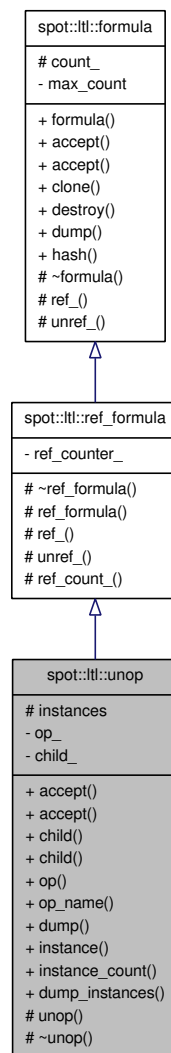
## 7.188 spot::ltl::unop Class Reference

Unary operators.

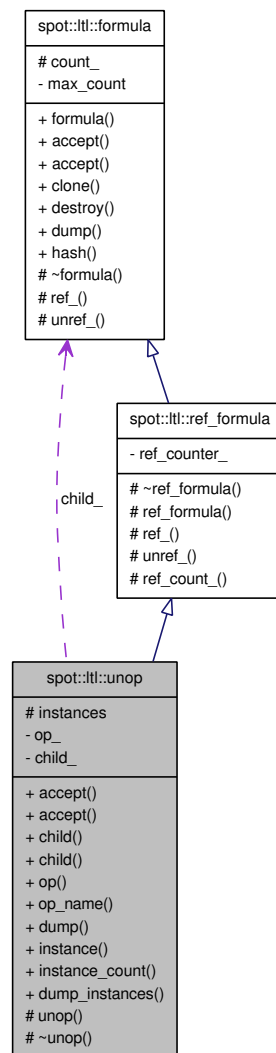
```
#include <ltlast/unop.hh>
```



Inheritance diagram for spot::ltl::unop:



Collaboration diagram for spot::ltl::unop:



## Public Types

- enum `type` {  
`Not`, `X`, `F`, `G`,  
`Finish` }

## 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.*
- virtual std::string dump () const  
*Return a canonic representation of the atomic proposition.*
- formula \* clone () const  
*clone this node*
- void destroy () const  
*release this node*
- 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 std::ostream & dump\_instances (std::ostream &os)  
*Dump all instances. For debugging.*

### Protected Types

- typedef std::pair< type, formula \* > pair
- typedef std::map< pair, unop \* > 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.*

### Protected Attributes

- size\_t [count\\_](#)  
*The hash key of this formula.*

### Static Protected Attributes

- static [map instances](#)

### Private Attributes

- type [op\\_](#)
- formula \* [child\\_](#)

## 7.188.1 Detailed Description

Unary operators.

## 7.188.2 Member Typedef Documentation

**7.188.2.1** `typedef std::map<pair, unop*> spot::ltl::unop::map [protected]`

**7.188.2.2** `typedef std::pair<type, formula*> spot::ltl::unop::pair [protected]`

## 7.188.3 Member Enumeration Documentation

**7.188.3.1** `enum spot::ltl::unop::type`

### Enumerator:

*Not*

*X*

*F*

*G*

*Finish*

### 7.188.4 Constructor & Destructor Documentation

**7.188.4.1** spot::ltl::unop::unop (type *op*, formula \* *child*) [protected]

**7.188.4.2** virtual spot::ltl::unop::~~unop () [protected, virtual]

### 7.188.5 Member Function Documentation

**7.188.5.1** virtual void spot::ltl::unop::accept (const\_visitor & *v*) const [virtual]

Entry point for vspot::ltl::const\_visitor instances.

Implements [spot::ltl::formula](#).

**7.188.5.2** virtual void spot::ltl::unop::accept (visitor & *v*) [virtual]

Entry point for vspot::ltl::visitor instances.

Implements [spot::ltl::formula](#).

**7.188.5.3** formula\* spot::ltl::unop::child ()

Get the sole operand of this operator.

**7.188.5.4** const formula\* spot::ltl::unop::child () const

Get the sole operand of this operator.

**7.188.5.5** formula\* spot::ltl::formula::clone () const [inherited]

clone this node

This increments the reference counter of this node (if one is used).

**7.188.5.6** void spot::ltl::formula::destroy () const [inherited]

release this node

This decrements the reference counter of this node (if one is used) and can free the object.

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition()`.

#### 7.188.5.7 virtual std::string spot::ltl::unop::dump () const [virtual]

Return a canonic representation of the atomic proposition.

Implements [spot::ltl::formula](#).

#### 7.188.5.8 static std::ostream& spot::ltl::unop::dump\_instances (std::ostream & os) [static]

Dump all instances. For debugging.

#### 7.188.5.9 size\_t spot::ltl::formula::hash () const [inline, inherited]

Return a hash key for the formula.

References `spot::ltl::formula::count_`.

#### 7.188.5.10 static unop\* spot::ltl::unop::instance (type op, formula \* child) [static]

Build an unary operator with operation *op* and child *child*.

#### 7.188.5.11 static unsigned spot::ltl::unop::instance\_count () [static]

Number of instantiated unary operators. For debugging.

#### 7.188.5.12 type spot::ltl::unop::op () const

Get the type of this operator.

#### 7.188.5.13 const char\* spot::ltl::unop::op\_name () const

Get the type of this operator, as a string.

#### 7.188.5.14 void spot::ltl::ref\_formula::ref\_ () [protected, virtual, inherited]

increment reference counter if any

Reimplemented from [spot::ltl::formula](#).

#### 7.188.5.15 `unsigned spot::ltl::ref_formula::ref_count_()` `[protected, inherited]`

Number of references to this formula.

#### 7.188.5.16 `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](#).

### 7.188.6 Member Data Documentation

#### 7.188.6.1 `formula* spot::ltl::unop::child_` `[private]`

#### 7.188.6.2 `size_t spot::ltl::formula::count_` `[protected, inherited]`

The hash key of this formula.

Referenced by `spot::ltl::formula::hash()`.

#### 7.188.6.3 `map spot::ltl::unop::instances` `[static, protected]`

#### 7.188.6.4 `type spot::ltl::unop::op_` `[private]`

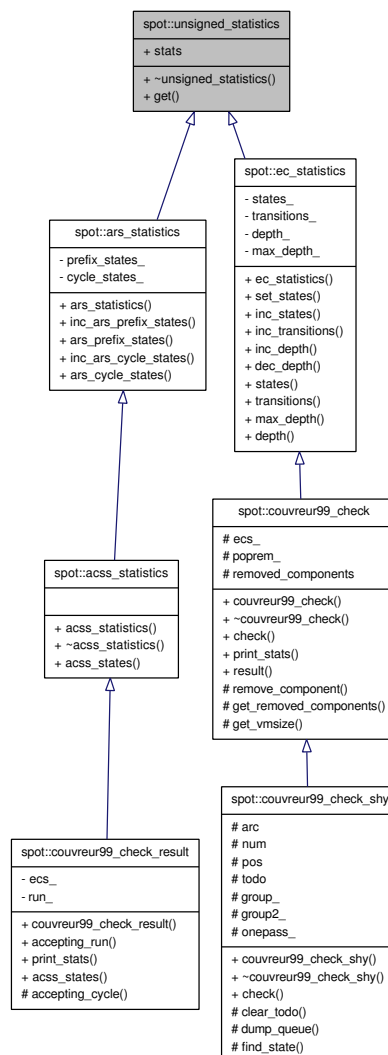
The documentation for this class was generated from the following file:

- [ltlast/unop.hh](#)

### 7.189 `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`



### 7.189.1 Member Typedef Documentation

**7.189.1.1** `typedef std::map<const char*, unsigned_fun, char_ptr_less_than>  
spot::unsigned_statistics::stats_map`

**7.189.1.2** `typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const`

### 7.189.2 Constructor & Destructor Documentation

**7.189.2.1** `virtual spot::unsigned_statistics::~~unsigned_statistics() [inline, virtual]`

### 7.189.3 Member Function Documentation

**7.189.3.1** `unsigned spot::unsigned_statistics::get(const char * str) const [inline]`

References stats.

### 7.189.4 Member Data Documentation

**7.189.4.1** `stats_map spot::unsigned_statistics::stats`

Referenced by `spot::acss_statistics::acss_statistics()`, `spot::ars_statistics::ars_statistics()`, `spot::ec_statistics::ec_statistics()`, `get()`, and `spot::unsigned_statistics_copy::seteq()`.

The documentation for this struct was generated from the following file:

- [tgbaalgos/emptiness\\_stats.hh](#)

## 7.190 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](#)

### 7.190.1 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.)

### 7.190.2 Member Typedef Documentation

**7.190.2.1** `typedef std::map<const char*, unsigned, char_ptr_less_than>  
spot::unsigned_statistics_copy::stats_map`

### 7.190.3 Constructor & Destructor Documentation

**7.190.3.1** `spot::unsigned_statistics_copy::unsigned_statistics_copy () [inline]`

**7.190.3.2** `spot::unsigned_statistics_copy::unsigned_statistics_copy (const unsigned_statistics &o)  
[inline]`

References [seteq\(\)](#).

### 7.190.4 Member Function Documentation

**7.190.4.1** `bool spot::unsigned_statistics_copy::operator!= (const unsigned_statistics_copy &o)  
const [inline]`

**7.190.4.2** `bool spot::unsigned_statistics_copy::operator==(const unsigned_statistics_copy & o)  
const [inline]`

References stats.

**7.190.4.3** `bool spot::unsigned_statistics_copy::seteq(const unsigned_statistics & o) [inline]`

References stats, and spot::unsigned\_statistics::stats.

Referenced by unsigned\_statistics\_copy().

## 7.190.5 Member Data Documentation

**7.190.5.1** `bool spot::unsigned_statistics_copy::set`

**7.190.5.2** `stats_map spot::unsigned_statistics_copy::stats`

Referenced by operator==(), and seteq().

The documentation for this class was generated from the following file:

- [tgbaalgos/emptiness\\_stats.hh](#)

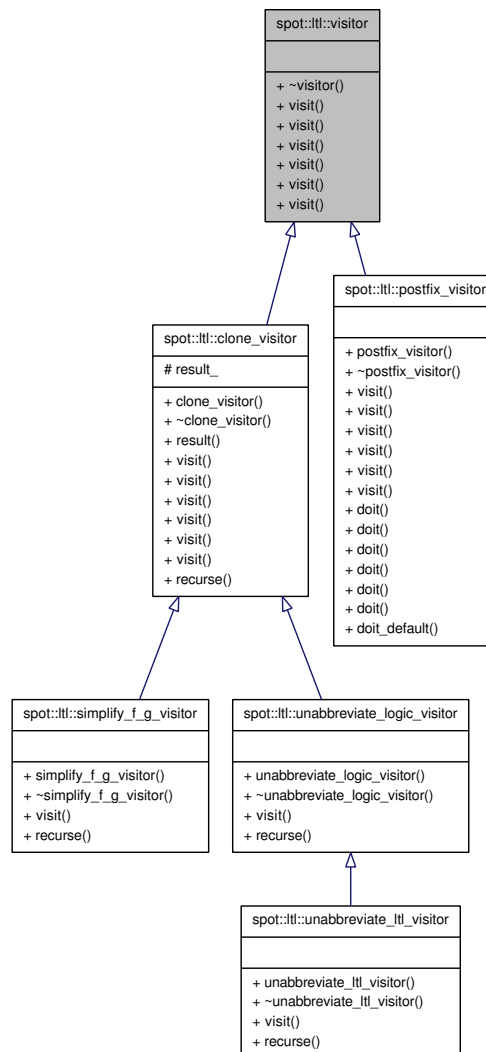
## 7.191 spot::ltl::visitor Struct Reference

Formula visitor that can modify the formula.

Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.

```
#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`
- virtual void `visit(automatop *node)=0`

### 7.191.1 Detailed Description

Formula visitor that can modify the formula.

Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast. If you do not need to modify the visited formula, inherit from `spot::ltl::const_visitor` instead.

### 7.191.2 Constructor & Destructor Documentation

**7.191.2.1** `virtual spot::ltl::visitor::~~visitor () [inline, virtual]`

### 7.191.3 Member Function Documentation

**7.191.3.1** `virtual void spot::ltl::visitor::visit (automatop * node) [pure virtual]`

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

**7.191.3.2** `virtual void spot::ltl::visitor::visit (multop * node) [pure virtual]`

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

**7.191.3.3** `virtual void spot::ltl::visitor::visit (unop * node) [pure virtual]`

Implemented in [spot::ltl::clone\\_visitor](#), [spot::ltl::postfix\\_visitor](#), and [spot::ltl::unabbreviate\\_ltl\\_visitor](#).

**7.191.3.4** `virtual void spot::ltl::visitor::visit (binop * node) [pure virtual]`

Implemented in [spot::ltl::clone\\_visitor](#), [spot::ltl::unabbreviate\\_logic\\_visitor](#), [spot::ltl::postfix\\_visitor](#), and [spot::ltl::simplify\\_f\\_g\\_visitor](#).

**7.191.3.5** `virtual void spot::ltl::visitor::visit (constant * node) [pure virtual]`

Implemented in [spot::ltl::clone\\_visitor](#), and [spot::ltl::postfix\\_visitor](#).

**7.191.3.6** `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](#)

## 7.192 `spot::weight` Class Reference

Manage for a given automaton a vector of counter indexed by its acceptance condition.

```
#include <tgbalgos/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)

#### 7.192.1 Detailed Description

Manage for a given automaton a vector of counter indexed by its acceptance condition.

#### 7.192.2 Member Typedef Documentation

##### 7.192.2.1 typedef std::map<int, int> spot::weight::weight\_vector [private]

### 7.192.3 Constructor & Destructor Documentation

#### 7.192.3.1 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\(\)](#)).

### 7.192.4 Member Function Documentation

#### 7.192.4.1 static void spot::weight::dec\_weight\_handler (char \* *varset*, int *size*) [**static**, **private**]

#### 7.192.4.2 static void spot::weight::inc\_weight\_handler (char \* *varset*, int *size*) [**static**, **private**]

#### 7.192.4.3 weight& spot::weight::operator+= (const bdd & *acc*)

Increment by one the counters of each acceptance condition in *acc*.

#### 7.192.4.4 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*.

#### 7.192.4.5 weight& spot::weight::operator-= (const bdd & *acc*)

Decrement by one the counters of each acceptance condition in *acc*.

### 7.192.5 Friends And Related Function Documentation

#### 7.192.5.1 std::ostream& operator<< (std::ostream & *os*, const weight & *w*) [**friend**]

### 7.192.6 Member Data Documentation

7.192.6.1 `weight_vector spot::weight::m` [`private`]

7.192.6.2 `bdd spot::weight::neg_all_acc` [`private`]

7.192.6.3 `weight_vector* spot::weight::pm` [`static`, `private`]

The documentation for this class was generated from the following file:

- [tgbaalgos/weight.hh](#)

## 8 File Documentation

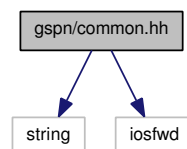
### 8.1 mainpage.dox File Reference

### 8.2 gspn/common.hh File Reference

```
#include <string>
```

```
#include <iosfwd>
```

Include dependency graph for common.hh:



#### Classes

- class [spot::gspn\\_exception](#)  
*An exception used to forward GSPN errors.*

#### Namespaces

- namespace [spot](#)

#### Functions

- `std::ostream & spot::operator<< (std::ostream &os, const gspn_exception &e)`

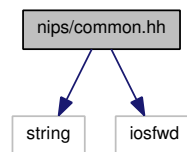


### 8.3 nips/common.hh File Reference

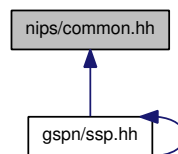
```
#include <string>
```

```
#include <iosfwd>
```

Include dependency graph for common.hh:



This graph shows which files directly or indirectly include this file:



#### Classes

- class [spot::nips\\_exception](#)  
*An exception used to forward NIPS errors.*

#### Namespaces

- namespace [spot](#)

#### Functions

- `std::ostream & spot::operator<< (std::ostream &os, const nips_exception &e)`

### 8.4 gspn/gspn.hh File Reference

```
#include <string>
```

```
#include "tgba/tgba.hh"
```

```
#include "common.hh"
```

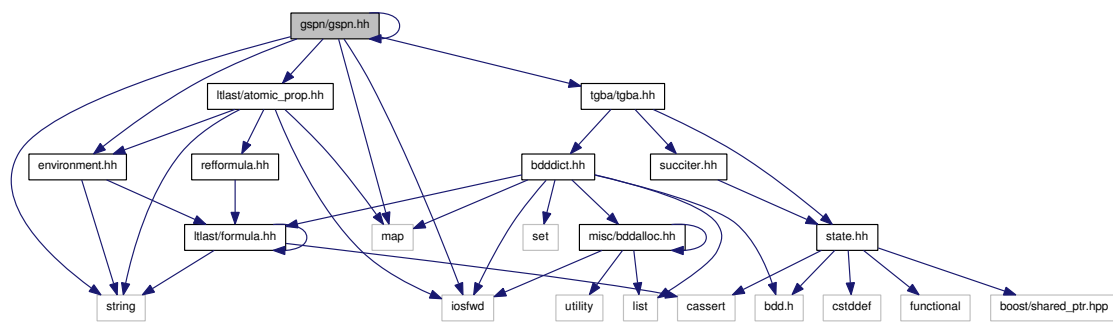
```
#include <iosfwd>
```

```
#include "environment.hh"
```

```
#include <map>
```

```
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for gspn.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::gspn\\_interface](#)

## Namespaces

- namespace [spot](#)

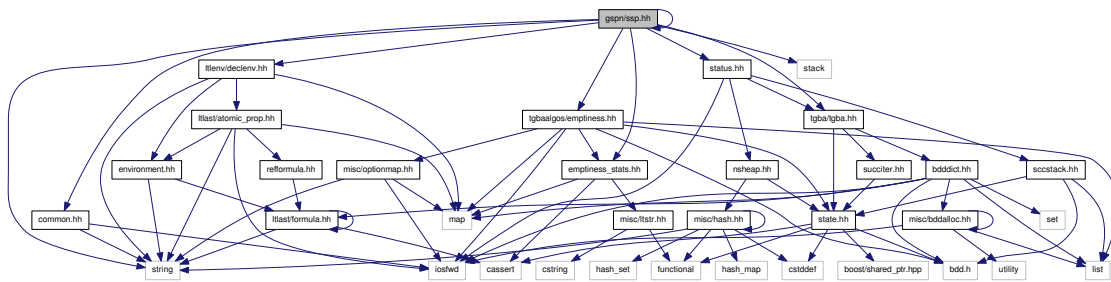
## 8.5 gspn/ssp.hh File Reference

```

#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "tgbaalgos/gtec/gtec.hh"
#include <stack>
#include "status.hh"
#include "tgbaalgos/emptiness.hh"
#include "tgbaalgos/emptiness_stats.hh"
#include "ltlenv/declenv.hh"

```

Include dependency graph for ssp.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::gspn\\_ssp\\_interface](#)

## Namespaces

- namespace [spot](#)

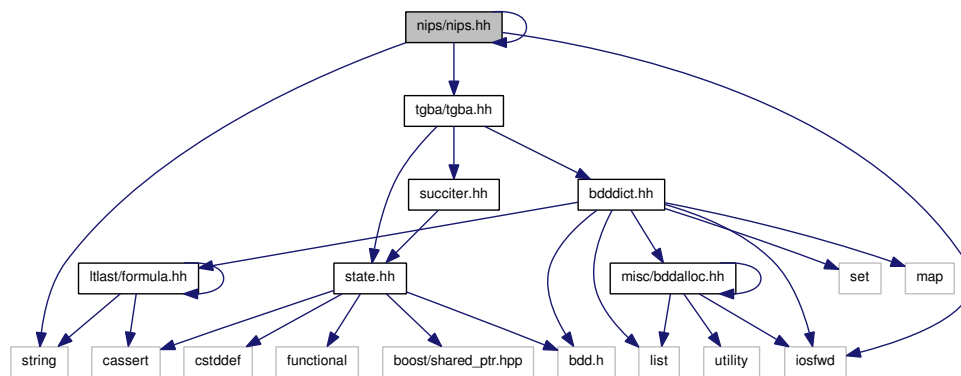
## Functions

- `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, bool stack_inclusion=true, bool double_inclusion=false, bool reversed_double_inclusion=false, bool no_decomp=false)`

## 8.6 nips/nips.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include <iosfwd>
```

Include dependency graph for nips.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::nips\\_interface](#)  
*An interface to provide a PROMELA front-end.*

## Namespaces

- namespace [spot](#)

## Typedefs

- typedef struct [nipsvm\\_t](#) [nipsvm\\_t](#)
- typedef struct t\_bytecode [nipsvm\\_bytecode\\_t](#)

### 8.6.1 Typedef Documentation

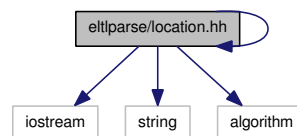
#### 8.6.1.1 typedef struct t\_bytecode nipsvm\_bytecode\_t

#### 8.6.1.2 typedef struct nipsvm\_t nipsvm\_t

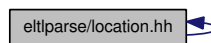
## 8.7 eltlparse/location.hh File Reference

```
#include <iostream>
#include <string>
#include "position.hh"
#include <algorithm>
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [eltlyy::location](#)  
*Abstract a location.*

### Namespaces

- namespace [eltlyy](#)

### Functions

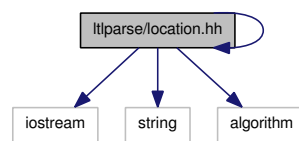
- const location [eltlyy::operator+](#) (const location &begin, const location &end)  
*Join two location objects to create a location.*
- const location [eltlyy::operator+](#) (const location &begin, unsigned int width)  
*Add two location objects.*
- location & [eltlyy::operator+=](#) (location &res, unsigned int width)  
*Add and assign a location.*
- bool [eltlyy::operator==](#) (const location &loc1, const location &loc2)  
*Compare two location objects.*
- bool [eltlyy::operator!=](#) (const location &loc1, const location &loc2)  
*Compare two location objects.*
- std::ostream & [eltlyy::operator<<](#) (std::ostream &ostr, const location &loc)

*Intercept output stream redirection.*

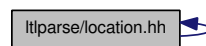
## 8.8 Itlparse/location.hh File Reference

```
#include <iostream>
#include <string>
#include "position.hh"
#include <algorithm>
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `ltlty::location`  
*Abstract a location.*

### Namespaces

- namespace `ltlty`

### Functions

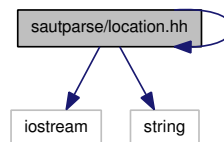
- const location `ltlty::operator+` (const location &begin, const location &end)  
*Join two location objects to create a location.*
- const location `ltlty::operator+` (const location &begin, unsigned int width)  
*Add two location objects.*
- location & `ltlty::operator+=` (location &res, unsigned int width)  
*Add and assign a location.*
- bool `ltlty::operator==` (const location &loc1, const location &loc2)  
*Compare two location objects.*

- bool `ltlyy::operator!=` (const location &loc1, const location &loc2)  
*Compare two location objects.*
- `std::ostream & ltlyy::operator<<` (std::ostream &ostr, const location &loc)  
*Intercept output stream redirection.*

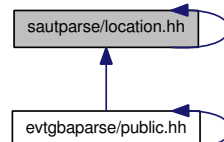
## 8.9 sautparse/location.hh File Reference

```
#include <iostream>
#include <string>
#include "position.hh"
```

Include dependency graph for location.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `sauty::location`  
*Abstract a location.*

### Namespaces

- namespace `sauty`

### Functions

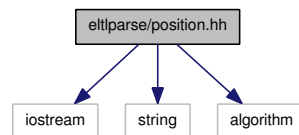
- const location `sauty::operator+` (const location &begin, const location &end)  
*Join two location objects to create a location.*
- const location `sauty::operator+` (const location &begin, unsigned int width)  
*Add two location objects.*

- location & [sautyy::operator+=](#) (location &res, unsigned int width)  
*Add and assign a location.*
- std::ostream & [sautyy::operator<<](#) (std::ostream &ostr, const location &loc)  
*Intercept output stream redirection.*

## 8.10 eltlparse/position.hh File Reference

```
#include <iostream>
#include <string>
#include <algorithm>
```

Include dependency graph for position.hh:



### Classes

- class [eltlyy::position](#)  
*Abstract a position.*

### Namespaces

- namespace [eltlyy](#)

### Functions

- const position & [eltlyy::operator+=](#) (position &res, const int width)  
*Add and assign a position.*
- const position [eltlyy::operator+](#) (const position &begin, const int width)  
*Add two position objects.*
- const position & [eltlyy::operator-=](#) (position &res, const int width)  
*Add and assign a position.*
- const position [eltlyy::operator-](#) (const position &begin, const int width)  
*Add two position objects.*
- bool [eltlyy::operator==](#) (const position &pos1, const position &pos2)  
*Compare two position objects.*

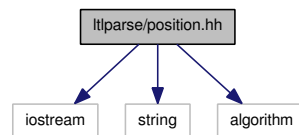


- bool `eltly::operator!=` (const position &pos1, const position &pos2)  
*Compare two position objects.*
- `std::ostream & eltly::operator<<` (std::ostream &ostr, const position &pos)  
*Intercept output stream redirection.*

## 8.11 Itlparse/position.hh File Reference

```
#include <iostream>
#include <string>
#include <algorithm>
```

Include dependency graph for position.hh:



### Classes

- class `ltly::position`  
*Abstract a position.*

### Namespaces

- namespace `ltly`

### Functions

- const position & `ltly::operator+=` (position &res, const int width)  
*Add and assign a position.*
- const position `ltly::operator+` (const position &begin, const int width)  
*Add two position objects.*
- const position & `ltly::operator-=` (position &res, const int width)  
*Add and assign a position.*
- const position `ltly::operator-` (const position &begin, const int width)  
*Add two position objects.*
- bool `ltly::operator==` (const position &pos1, const position &pos2)  
*Compare two position objects.*

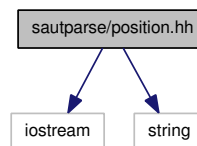
- bool `ltlyy::operator!=` (const position &pos1, const position &pos2)  
*Compare two position objects.*
- `std::ostream & ltlyy::operator<<` (std::ostream &ostr, const position &pos)  
*Intercept output stream redirection.*

## 8.12 sautparse/position.hh File Reference

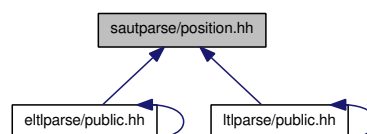
```
#include <iostream>
```

```
#include <string>
```

Include dependency graph for position.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `sauty::position`  
*Abstract a position.*

### Namespaces

- namespace `sauty`

### Functions

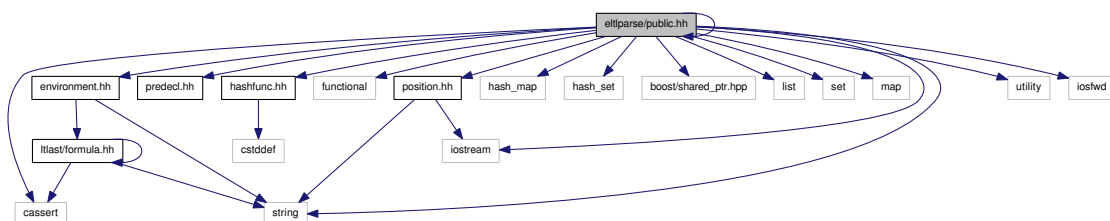
- const position & `sauty::operator+=` (position &res, const int width)  
*Add and assign a position.*
- const position `sauty::operator+` (const position &begin, const int width)  
*Add two position objects.*
- const position & `sauty::operator-=` (position &res, const int width)  
*Add and assign a position.*

- const position [sauty::operator-](#) (const position &begin, const int width)  
*Add two position objects.*
- std::ostream & [sauty::operator<<](#) (std::ostream &ostr, const position &pos)  
*Intercept output stream redirection.*

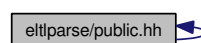
### 8.13 eltlparse/public.hh File Reference

```
#include "ltlast/formula.hh"
#include <string>
#include <cassert>
#include "predecl.hh"
#include "environment.hh"
#include <functional>
#include "hashfunc.hh"
#include <hash_map>
#include <hash_set>
#include <boost/shared_ptr.hpp>
#include <list>
#include <set>
#include <map>
#include <iostream>
#include "position.hh"
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::eltl](#)

## Typedefs

- typedef std::pair< std::string, std::string > [spot::eltl::spair](#)
- typedef std::pair< [eltly::location](#), spair > [spot::eltl::parse\\_error](#)  
*A parse diagnostic <location, <file, message>>.*
- typedef std::list< parse\_error > [spot::eltl::parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*

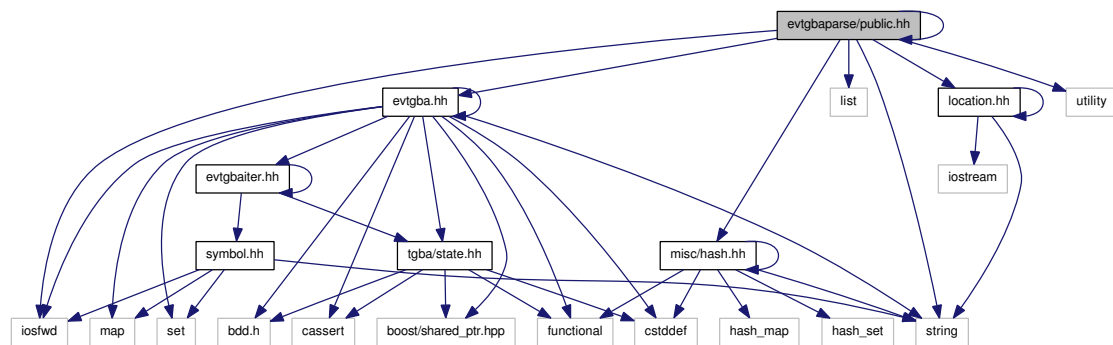
## Functions

- formula \* [spot::eltl::parse\\_file](#) (const std::string &filename, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
*Build a formula from a text file.*
- formula \* [spot::eltl::parse\\_string](#) (const std::string &eltl\_string, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)  
*Build a formula from an ETL string.*
- bool [spot::eltl::format\\_parse\\_errors](#) (std::ostream &os, parse\_error\_list &error\_list)  
*Format diagnostics produced by spot::eltl::parse.*

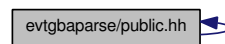
## 8.14 evtgbaparse/public.hh File Reference

```
#include "evtgba/explicit.hh"
#include "evtgba.hh"
#include <list>
#include "misc/hash.hh"
#include "location.hh"
#include <string>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Typedefs

- typedef std::pair< [evtgba::location](#), std::string > [spot::evtgba\\_parse\\_error](#)  
A parse diagnostic with its location.
- typedef std::list< [evtgba\\_parse\\_error](#) > [spot::evtgba\\_parse\\_error\\_list](#)  
A list of parser diagnostics, as filled by parse.

## Functions

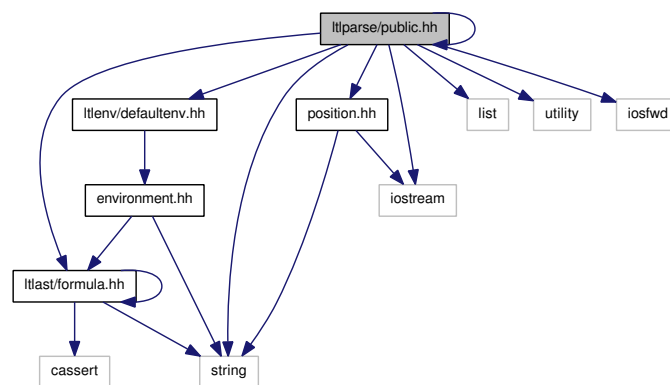
- [evtgba\\_explicit](#) \* [spot::evtgba\\_parse](#) (const std::string &filename, [evtgba\\_parse\\_error\\_list](#) &error\_list, bool debug=false)  
Build a [spot::evtgba\\_explicit](#) from a text file.
- bool [spot::format\\_evtgba\\_parse\\_errors](#) (std::ostream &os, const std::string &filename, [evtgba\\_parse\\_error\\_list](#) &error\_list)  
Format diagnostics produced by [spot::evtgba\\_parse](#).

## 8.15 Itlparse/public.hh File Reference

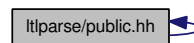
```
#include "ltlast/formula.hh"
#include "ltlparse/location.hh"
```

```
#include <iostream>
#include <string>
#include "position.hh"
#include "ltlenv/defaultenv.hh"
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Typedefs

- typedef std::pair< [ltl::location](#), std::string > [spot::ltl::parse\\_error](#)  
*A parse diagnostic with its location.*
- typedef std::list< parse\_error > [spot::ltl::parse\\_error\\_list](#)  
*A list of parser diagnostics, as filled by parse.*

## Functions

- formula \* [spot::ltl::parse](#) (const std::string &ltl\_string, parse\_error\_list &error\_list, environment &env=default\_environment::instance(), bool debug=false)

*Build a formula from an LTL string.*

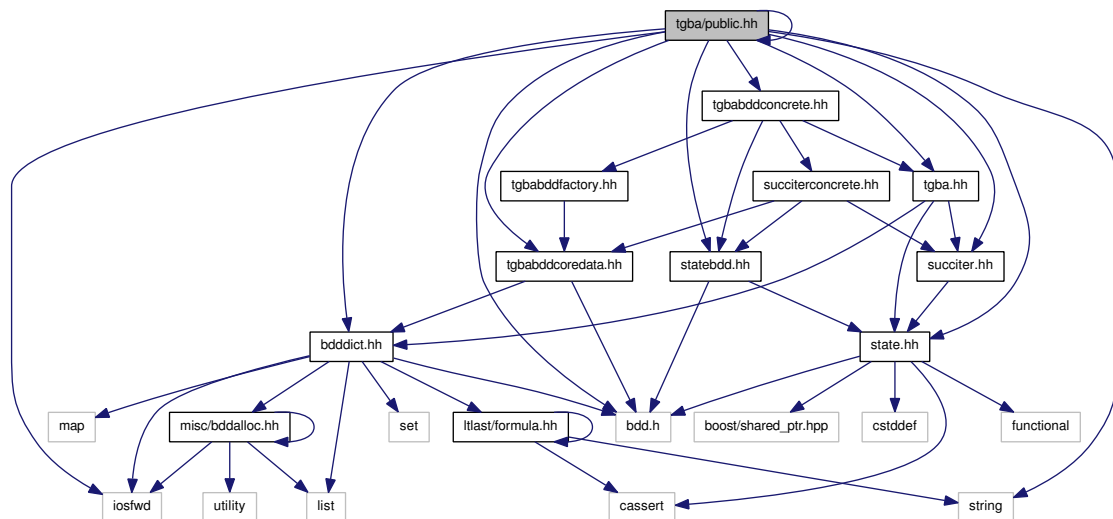
- bool `spot::ltl::format_parse_errors` (std::ostream &os, const std::string &ltl\_string, parse\_error\_list &error\_list)

*Format diagnostics produced by `spot::ltl::parse`.*

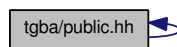
## 8.16 tgba/public.hh File Reference

```
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include <bdd.h>
#include "state.hh"
#include "bdddict.hh"
#include "statebdd.hh"
#include "succiter.hh"
#include "tgbabddcoredata.hh"
#include "tgbabddconcrete.hh"
#include <string>
#include <iosfwd>
```

Include dependency graph for public.hh:



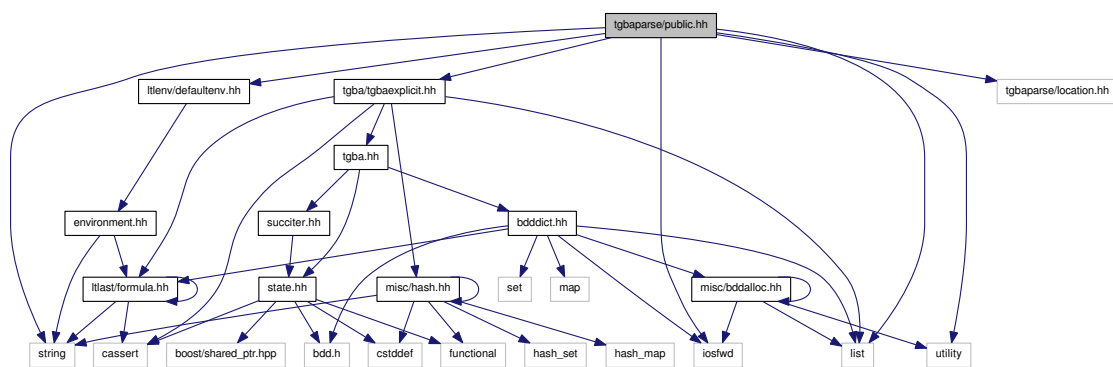
This graph shows which files directly or indirectly include this file:



## 8.17 tgbaparse/public.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
#include "tgbaparse/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< tgbayy::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

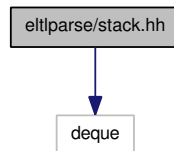
- tgba\_explicit\_string \* [spot::tgba\\_parse](#) (const std::string &filename, tgba\_parse\_error\_list &error\_list, bdd\_dict \*dict, ltl::environment &env=ltl::default\_environment::instance(), ltl::environment &envacc=ltl::default\_environment::instance(), bool debug=false)  
*Build a [spot::tgba\\_explicit](#) from a text file.*
- bool [spot::format\\_tgba\\_parse\\_errors](#) (std::ostream &os, const std::string &filename, tgba\_parse\_error\_list &error\_list)  
*Format diagnostics produced by [spot::tgba\\_parse](#).*



## 8.18 `eltlparse/stack.hh` File Reference

```
#include <deque>
```

Include dependency graph for `stack.hh`:



### Classes

- class `eltlyy::stack< T, S >`
- class `eltlyy::slice< T, S >`

*Present a slice of the top of a stack.*

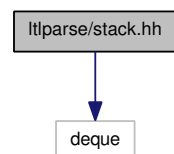
### Namespaces

- namespace `eltlyy`

## 8.19 `ltlparse/stack.hh` File Reference

```
#include <deque>
```

Include dependency graph for `stack.hh`:



### Classes

- class `ltlyy::stack< T, S >`
- class `ltlyy::slice< T, S >`

*Present a slice of the top of a stack.*

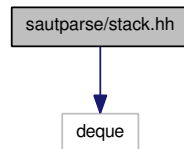
### Namespaces

- namespace `ltlyy`

## 8.20 sautparse/stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



### Classes

- class `sauty::stack< T, S >`
- class `sauty::slice< T, S >`  
*Present a slice of the top of a stack.*

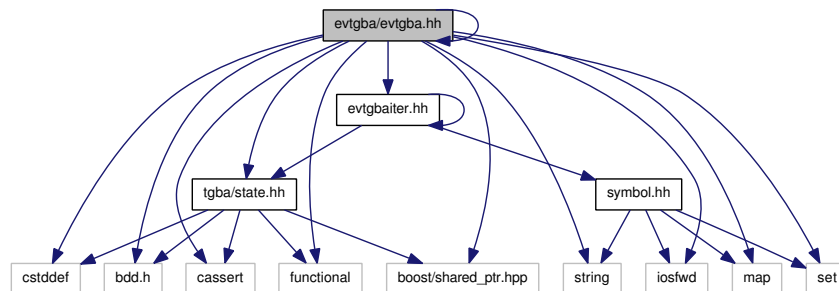
### Namespaces

- namespace `sauty`

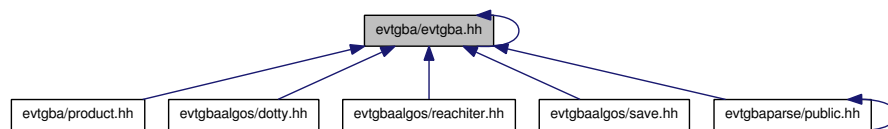
## 8.21 evtgba/evtgba.hh File Reference

```
#include "tgba/state.hh"
#include <cstdint>
#include <bdd.h>
#include <cassert>
#include <functional>
#include <boost/shared_ptr.hpp>
#include "tgba/state.hh"
#include <string>
#include <iosfwd>
#include <map>
#include <set>
#include "evtgbaiter.hh"
```

Include dependency graph for evtgba.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::evtgba](#)

## Namespaces

- namespace [spot](#)

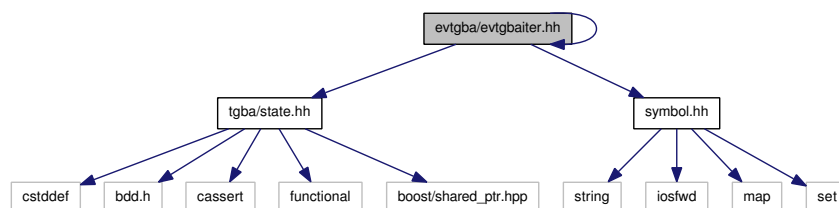
## 8.22 evtgba/evtgbaiter.hh File Reference

```
#include "tgba/state.hh"
```

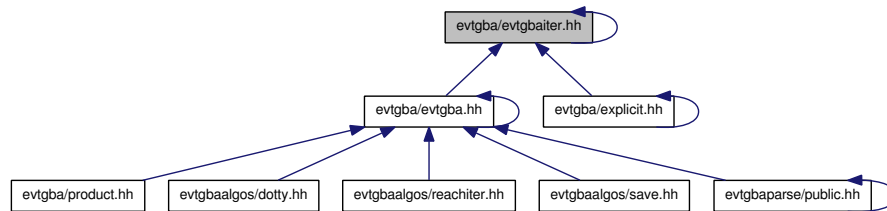
```
#include "symbol.hh"
```

```
#include "evtgbaiter.hh"
```

Include dependency graph for evtgbaiter.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::evtgba\\_iterator](#)

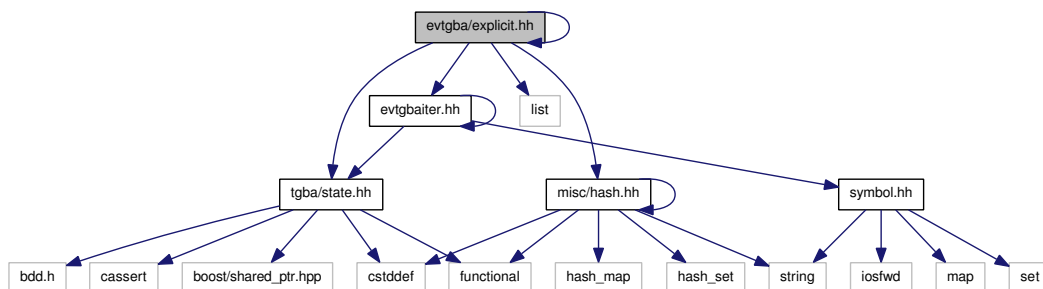
## Namespaces

- namespace [spot](#)

## 8.23 evtgba/explicit.hh File Reference

```
#include "evtgba.hh"
#include "tgba/state.hh"
#include "evgtbaiter.hh"
#include <list>
#include "misc/hash.hh"
```

Include dependency graph for explicit.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::evtgba\\_explicit](#)

- struct [spot::evtgba\\_explicit::state](#)
- struct [spot::evtgba\\_explicit::transition](#)  
*Explicit transitions (used by [spot::evtgba\\_explicit](#)).*
- class [spot::state\\_evtgba\\_explicit](#)  
*States used by [spot::tgba\\_evtgba\\_explicit](#).*

## Namespaces

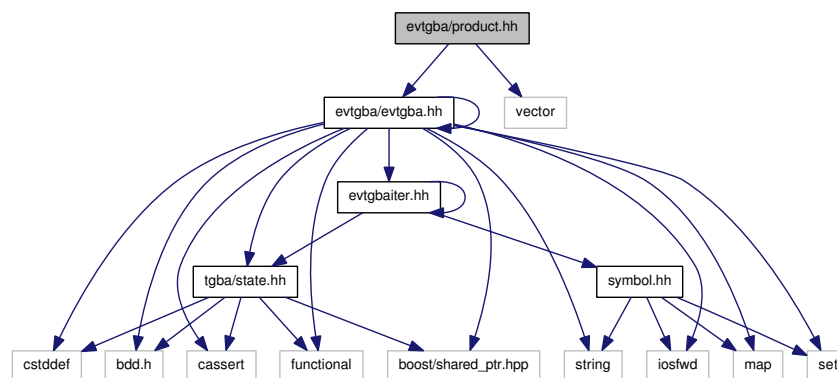
- namespace [spot](#)

## 8.24 evtgba/product.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <vector>
```

Include dependency graph for product.hh:



## Classes

- class [spot::evtgba\\_product](#)

## Namespaces

- namespace [spot](#)

## 8.25 evtgba/symbol.hh File Reference

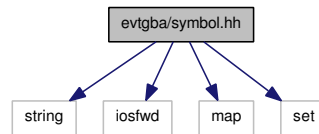
```
#include <string>
```

```
#include <iosfwd>
```

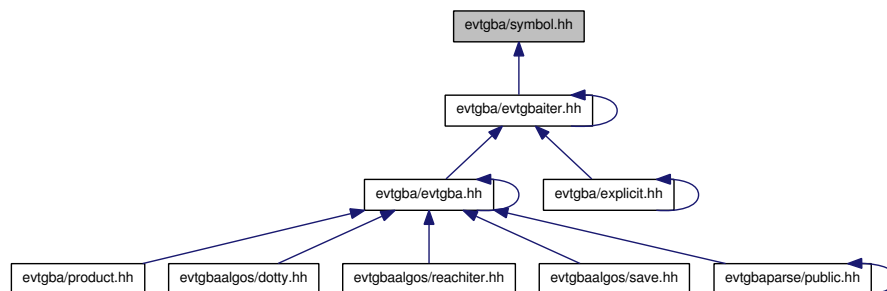
```
#include <map>
```

```
#include <set>
```

Include dependency graph for symbol.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::symbol](#)
- class [spot::rsymbol](#)

## Namespaces

- namespace [spot](#)

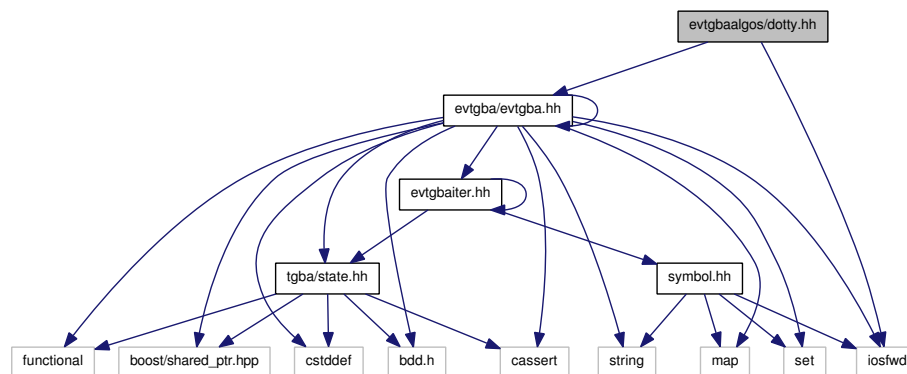
## Typedefs

- typedef `std::set< const symbol * >` [spot::symbol\\_set](#)
- typedef `std::set< rsymbol >` [spot::rsymbol\\_set](#)

## 8.26 evtgbaalgos/dotty.hh File Reference

```
#include "evtgba/evtgba.hh"
#include <iosfwd>
```

Include dependency graph for dotted.hh:



## Namespaces

- namespace **spot**

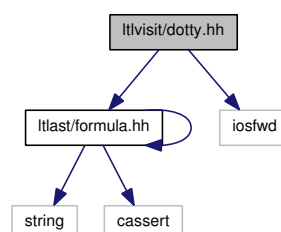
## Functions

- `std::ostream & spot::dotty_reachable (std::ostream &os, const evtgba *g)`  
*Print reachable states in dot format.*

## 8.27 ltlvisit/dotty.hh File Reference

```
#include <ltlast/formula.hh>
#include <iosfwd>
```

Include dependency graph for dotty.hh:



## Namespaces

- namespace `spot`
- namespace `spot::ltl`

## Functions

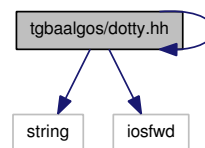
- `std::ostream & spot::ltl::dotty (std::ostream &os, const formula *f)`

*Write a formula tree using dot's syntax.*

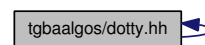
## 8.28 tgbaalgos/dotty.hh File Reference

```
#include "dottydec.hh"
#include <string>
#include <iosfwd>
```

Include dependency graph for dotty.hh:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [spot](#)

### Functions

- `std::ostream & spot::dotty\_reachable (std::ostream &os, const tgba *g, dotty_decorator *dd=dotty_decorator::instance())`

*Print reachable states in dot format.*

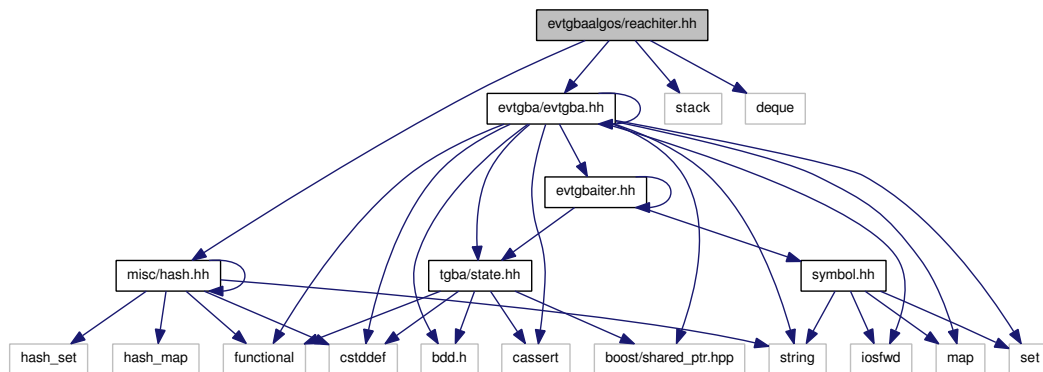
*The dd argument allows to customize the output in various ways. See [this page](#) for a list of available decorators.*

## 8.29 evtgbaalgos/reachiter.hh File Reference

```
#include "misc/hash.hh"
#include "evtgba/evtgba.hh"
#include <stack>
#include <deque>
```



Include dependency graph for reachiter.hh:



## Classes

- class [spot::evtgba\\_reachable\\_iterator](#)  
*Iterate over all reachable states of a [spot::evtgba](#).*
- class [spot::evtgba\\_reachable\\_iterator\\_depth\\_first](#)  
*An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states depth first.*
- class [spot::evtgba\\_reachable\\_iterator\\_breadth\\_first](#)  
*An implementation of [spot::evtgba\\_reachable\\_iterator](#) that browses states breadth first.*

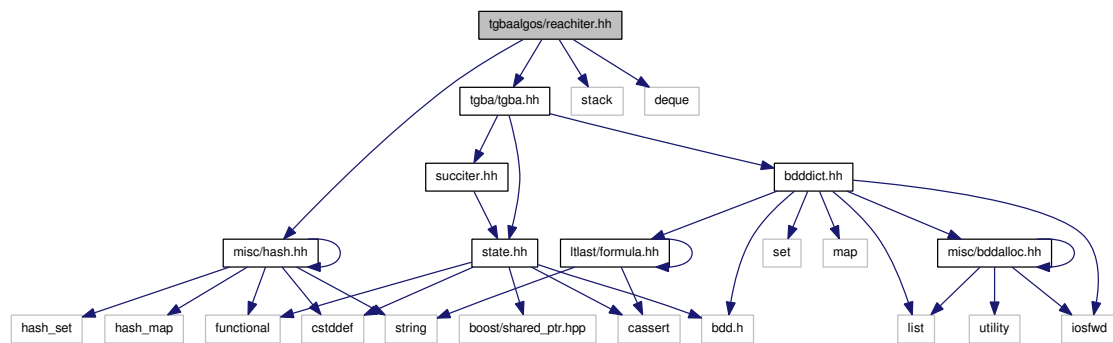
## Namespaces

- namespace [spot](#)

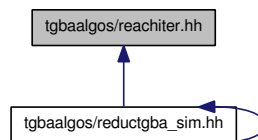
## 8.30 tgbaalgos/reachiter.hh File Reference

```
#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:



## 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.*

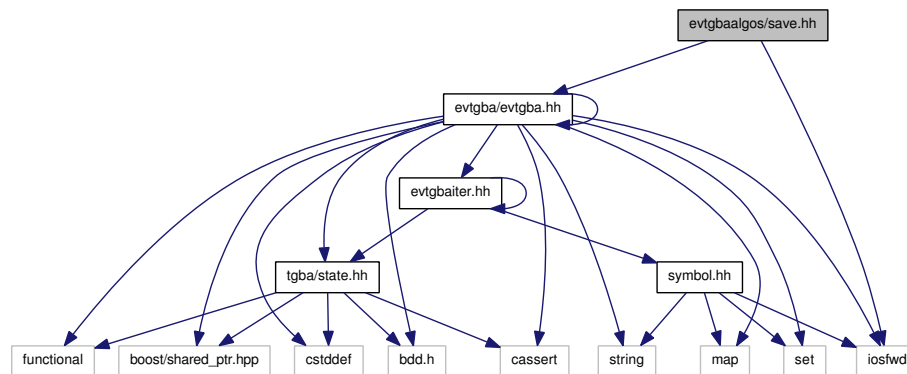
## Namespaces

- namespace [spot](#)

## 8.31 evtgbaalgos/save.hh File Reference

```
#include "evtgba/evtgba.hh"
#include <iosfwd>
```

Include dependency graph for save.hh:



## Namespaces

- namespace [spot](#)

## Functions

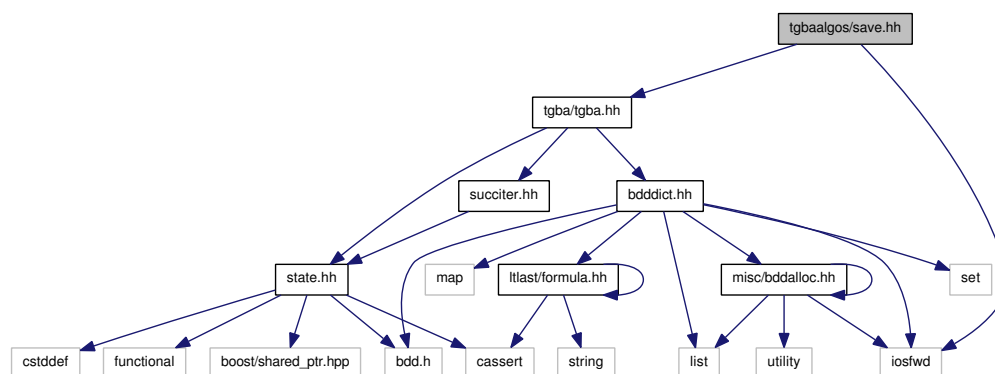
- `std::ostream & spot::evtgbal\_save\_reachable (std::ostream &os, const evtgbal *g)`  
*Save reachable states in text format.*

## 8.32 tgbaalgos/save.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



## Namespaces

- namespace [spot](#)

**Functions**

- `std::ostream & spot::tgba_save_reachable` (`std::ostream &os, const tgba *g`)  
*Save reachable states in text format.*

**8.33 evtgbaalgos/tgba2evtgba.hh File Reference****Namespaces**

- namespace `spot`

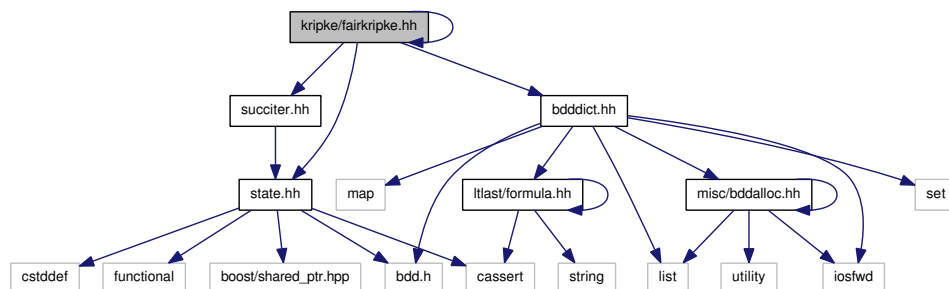
**Functions**

- `evtgba_explicit * spot::tgba_to_evtgba` (`const tgba *a`)  
*Convert a tgba into an evtgba.*

**8.34 kripke/fairkripke.hh File Reference**

```
#include "tgba/tgba.hh"
#include "state.hh"
#include "succiter.hh"
#include "bdddict.hh"
```

Include dependency graph for fairkripke.hh:



This graph shows which files directly or indirectly include this file:

**Classes**

- class `spot::fair_kripke_succ_iterator`
- class `spot::fair_kripke`

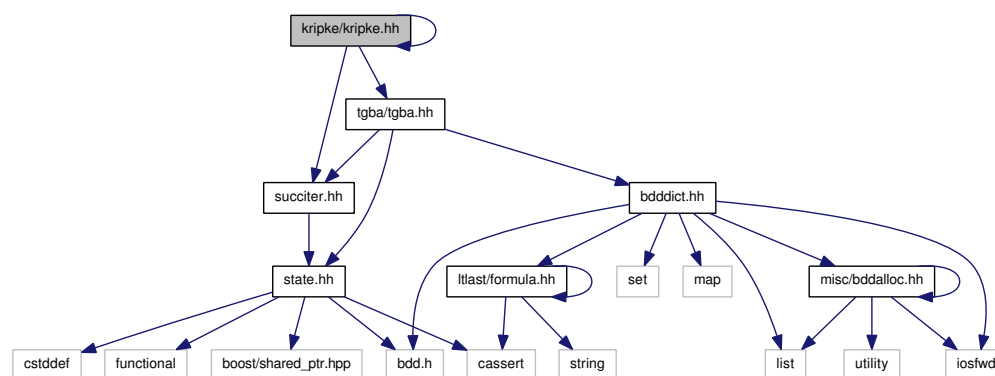
### Namespaces

- namespace [spot](#)

## 8.35 kripke/kripke.hh File Reference

```
#include "fairkripke.hh"
#include "tgba/tgba.hh"
#include "tgba/succiter.hh"
```

Include dependency graph for kripke.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::kripke](#)

### Namespaces

- namespace [spot](#)

### Typedefs

- typedef fair\_kripke\_succ\_iterator [spot::kripke\\_succ\\_iterator](#)

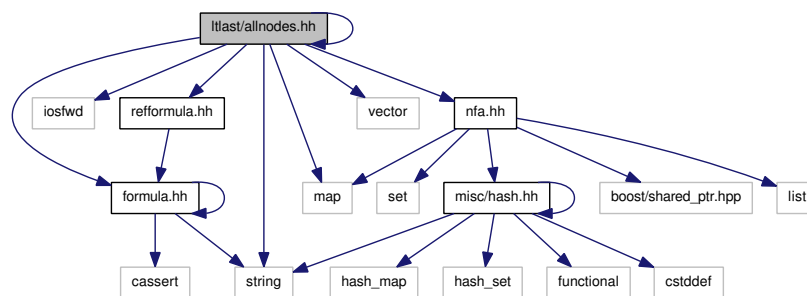
## 8.36 ltlast/allnodes.hh File Reference

Define all LTL node types.

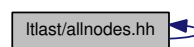
```
#include "binop.hh"
#include <map>
```

```
#include <iosfwd>
#include "formula.hh"
#include "reformula.hh"
#include <vector>
#include <string>
#include "nfa.hh"
```

Include dependency graph for allnodes.hh:



This graph shows which files directly or indirectly include this file:



### 8.36.1 Detailed Description

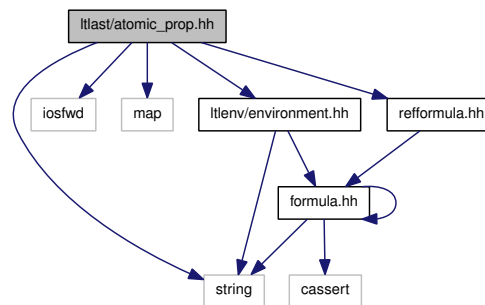
Define all LTL node types. This file is usually needed when **defining** a visitor. Prefer [ltlast/predecl.hh](#) when only **declaring** methods and functions over LTL nodes.

## 8.37 ltlast/atomic\_prop.hh File Reference

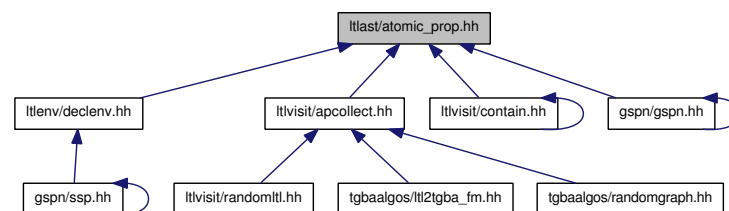
LTL atomic propositions.

```
#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:



## Classes

- class `spot::ltl::atomic_prop`  
*Atomic propositions.*

## Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 8.37.1 Detailed Description

LTL atomic propositions.

## 8.38 Itlast/automatop.hh File Reference

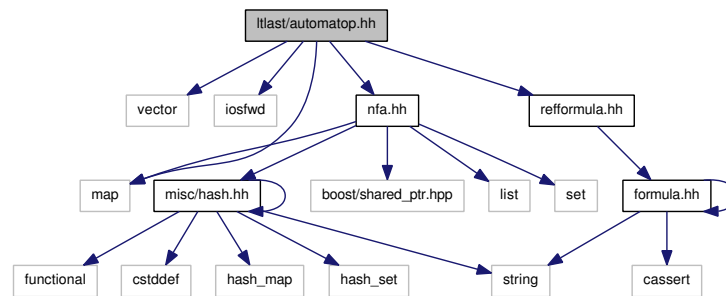
ELTL automaton operators.

```

#include <vector>
#include <iosfwd>
#include <map>
#include "nfa.hh"
#include "refformula.hh"

```

Include dependency graph for automaton.hh:



## Classes

- class [spot::ltl::automaton](#)  
*Automaton operators.*
- struct [spot::ltl::automaton::tripletcmp](#)  
*Comparison functor used internally by [ltl::automaton](#).*

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 8.38.1 Detailed Description

ELTL automaton operators.

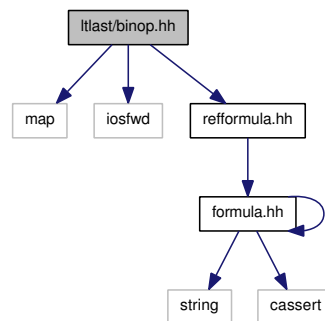
## 8.39 ltlast/binop.hh File Reference

LTL binary operators.

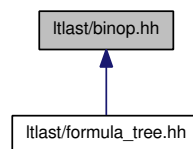
```
#include <map>
#include <iosfwd>
#include "refformula.hh"
```



Include dependency graph for binop.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class `spot::ltl::binop`  
*Binary operator.*

## Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 8.39.1 Detailed Description

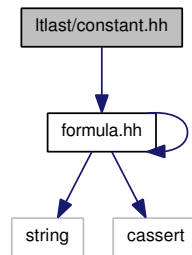
LTL binary operators. This does not include AND and OR operators. These are considered to be multi-operand operators (see `spot::ltl::multop`).

## 8.40 Itlast/constant.hh File Reference

LTL constants.

```
#include "formula.hh"
```

Include dependency graph for constant.hh:



## Classes

- class `spot::ltl::constant`  
A constant (True or False).

## Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 8.40.1 Detailed Description

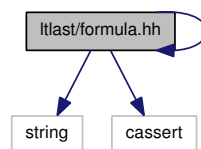
LTL constants.

## 8.41 Itlast/formula.hh File Reference

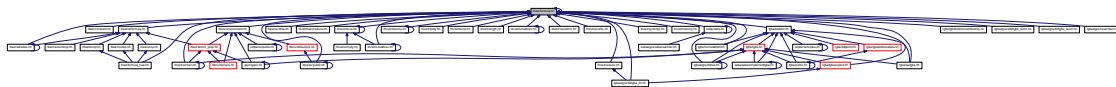
LTL formula interface.

```
#include <string>
#include <cassert>
#include "predecl.hh"
```

Include dependency graph for formula.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::formula](#)

*An LTL formula.*

*The only way you can work with a formula is to build a [spot::ltl::visitor](#) or [spot::ltl::const\\_visitor](#).*

- struct [spot::ltl::formula\\_ptr\\_less\\_than](#)

*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*`.*

- struct [spot::ltl::formula\\_ptr\\_hash](#)

*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*`.*

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 8.41.1 Detailed Description

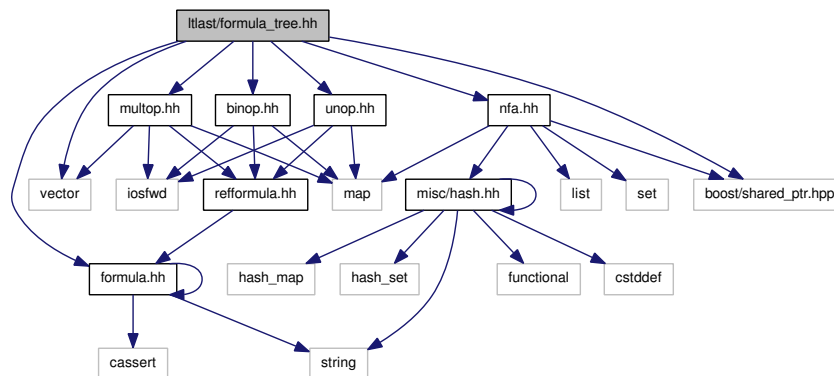
LTL formula interface.

## 8.42 Itlast/formula\_tree.hh File Reference

Trees representing formulae where atomic propositions are unknown.

```
#include <vector>
#include <boost/shared_ptr.hpp>
#include "formula.hh"
#include "multop.hh"
#include "binop.hh"
#include "unop.hh"
#include "nfa.hh"
```

Include dependency graph for formula\_tree.hh:



## Classes

- struct [spot::ltl::formula\\_tree::node](#)
- struct [spot::ltl::formula\\_tree::node\\_unop](#)
- struct [spot::ltl::formula\\_tree::node\\_binop](#)
- struct [spot::ltl::formula\\_tree::node\\_multop](#)
- struct [spot::ltl::formula\\_tree::node\\_nfa](#)
- struct [spot::ltl::formula\\_tree::node\\_atomic](#)

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)
- namespace [spot::ltl::formula\\_tree](#)

*Trees representing formulae where atomic propositions are unknown.*

## Typedefs

- typedef `boost::shared_ptr< node >` [spot::ltl::formula\\_tree::node\\_ptr](#)

*We use boost::shared\_ptr to easily handle deletion.*

## Enumerations

- enum { [spot::ltl::formula\\_tree::True](#) = -1, [spot::ltl::formula\\_tree::False](#) = -2 }

*Integer values for True and False used in node\_atomic.*

## Functions

- formula \* [spot::ltl::formula\\_tree::instantiate](#) (const node\_ptr np, const std::vector< formula \* > &v)
- size\_t [spot::ltl::formula\\_tree::arity](#) (const node\_ptr np)

*Get the arity.*

### 8.42.1 Detailed Description

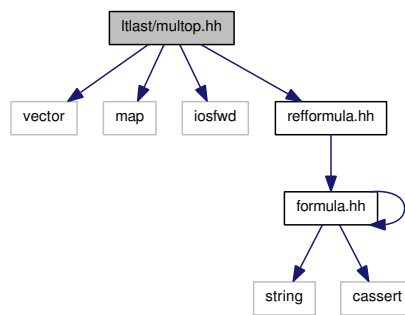
Trees representing formulae where atomic propositions are unknown.

## 8.43 Itlast/multop.hh File Reference

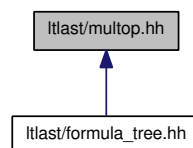
LTL multi-operand operators.

```
#include <vector>
#include <map>
#include <iosfwd>
#include "reformula.hh"
```

Include dependency graph for multop.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `spot::ltl::multop`  
*Multi-operand operators.  
 These operators are considered commutative and associative.*
- struct `spot::ltl::multop::paircmp`  
*Comparison functor used internally by `ltl::multop`.*

### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 8.43.1 Detailed Description

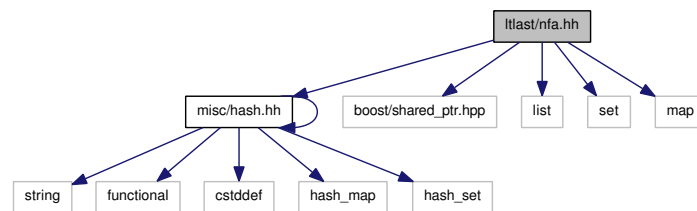
LTL multi-operand operators.

## 8.44 Itlast/nfa.hh File Reference

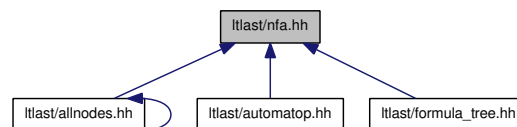
NFA interface used by automatop.

```
#include "misc/hash.hh"
#include <boost/shared_ptr.hpp>
#include <list>
#include <set>
#include <map>
```

Include dependency graph for nfa.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::ltl::nfa](#)  
*Nondeterministic Finite Automata used by automata operators.*
- struct [spot::ltl::nfa::transition](#)  
*Explicit transitions.*
- class [spot::ltl::succ\\_iterator](#)

### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)
- namespace [spot::ltl::formula\\_tree](#)  
*Trees representing formulae where atomic propositions are unknown.*

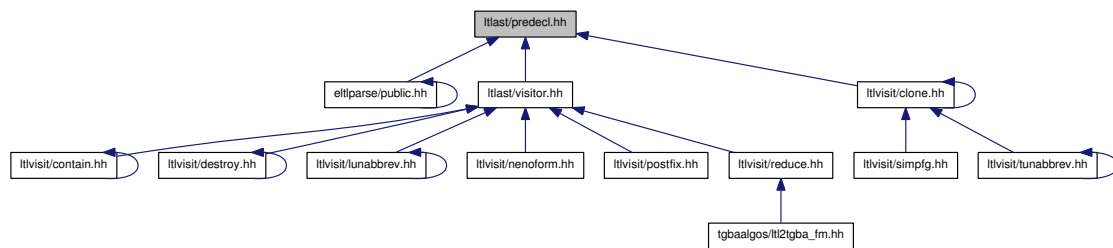
### 8.44.1 Detailed Description

NFA interface used by automatop.

## 8.45 ltlast/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](#)

### 8.45.1 Detailed Description

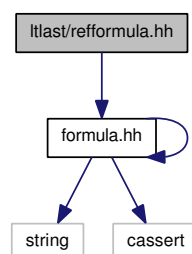
Predeclare all LTL node types. This file is usually used when **declaring** methods and functions over LTL nodes. Use [ltlast/allnodes.hh](#) or an individual header when the definition of the node is actually needed.

## 8.46 ltlast/reformula.hh File Reference

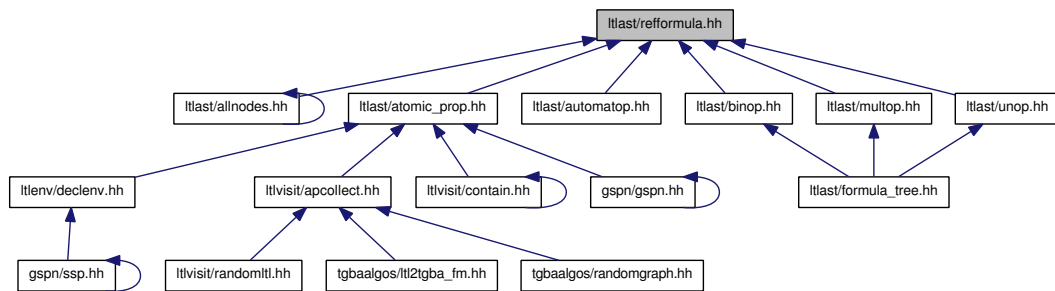
Reference-counted LTL formulae.

```
#include "formula.hh"
```

Include dependency graph for reformula.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::ref\\_formula](#)  
A reference-counted LTL formula.

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

### 8.46.1 Detailed Description

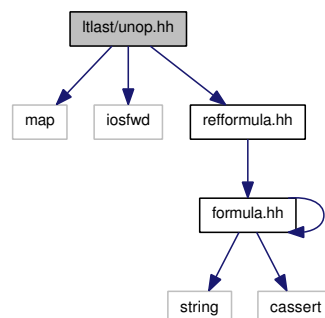
Reference-counted LTL formulae.

## 8.47 Itlast/unop.hh File Reference

LTL unary operators.

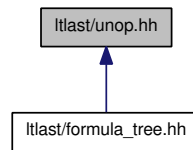
```
#include <map>
#include <iosfwd>
#include "refformula.hh"
```

Include dependency graph for unop.hh:





This graph shows which files directly or indirectly include this file:



## Classes

- class `spot::ltl::unop`  
*Unary operators.*

## Namespaces

- namespace `spot`
- namespace `spot::ltl`

### 8.47.1 Detailed Description

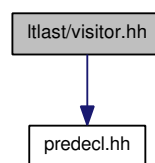
LTL unary operators.

## 8.48 Itlast/visitor.hh File Reference

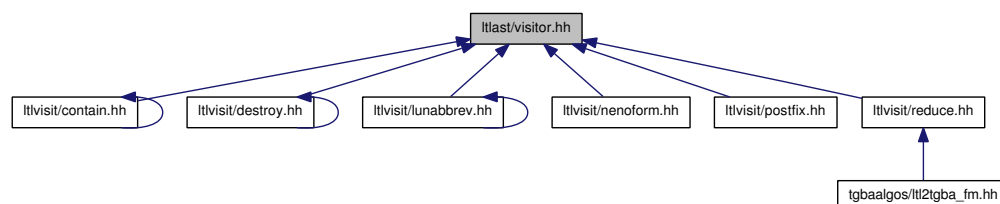
LTL visitor interface.

```
#include "predecl.hh"
```

Include dependency graph for visitor.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [spot::ltl::visitor](#)  
*Formula visitor that can modify the formula.*  
*Writing visitors is the preferred way to traverse a formula, since it doesn't involve any cast.*
- struct [spot::ltl::const\\_visitor](#)  
*Formula visitor that cannot modify the formula.*

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

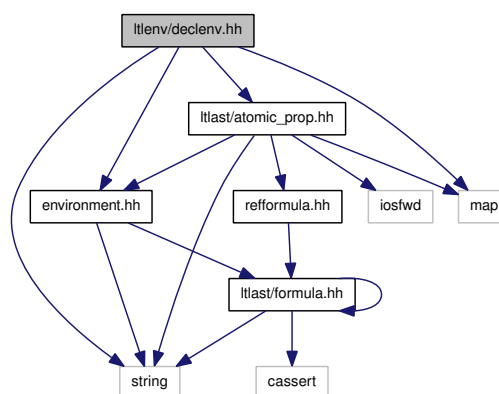
### 8.48.1 Detailed Description

LTL visitor interface.

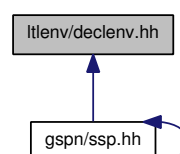
## 8.49 Itlenv/declenv.hh File Reference

```
#include "environment.hh"
#include <string>
#include <map>
#include "ltlast/atomic_prop.hh"
```

Include dependency graph for declenv.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::declarative\\_environment](#)

*A declarative environment.*

*This environment recognizes all atomic propositions that have been previously declared. It will reject other.*

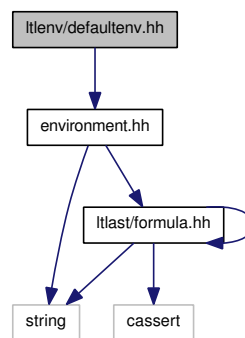
## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

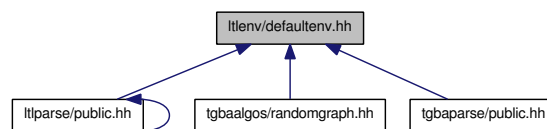
## 8.50 Itlenv/defaultenv.hh File Reference

```
#include "environment.hh"
```

Include dependency graph for defaultenv.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::default\\_environment](#)

*A laxist environment.*

*This environment recognizes all atomic propositions.*

## Namespaces

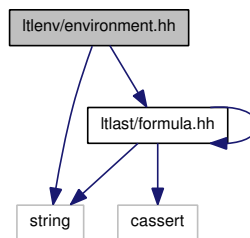
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.51 Itlenv/environment.hh File Reference

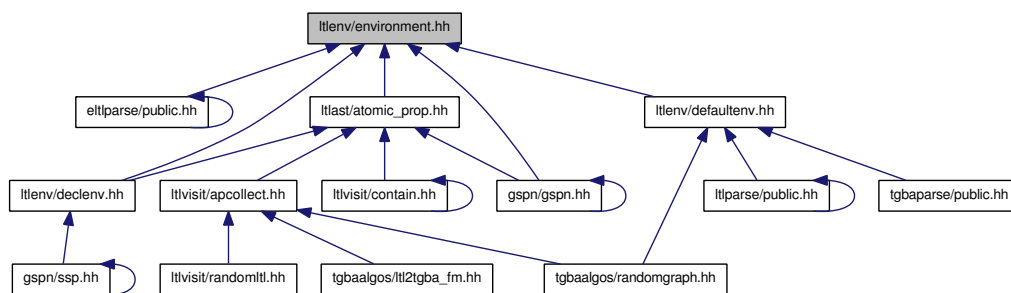
```
#include "ltlast/formula.hh"
```

```
#include <string>
```

Include dependency graph for environment.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::ltl::environment](#)  
*An environment that describes atomic propositions.*

### Namespaces

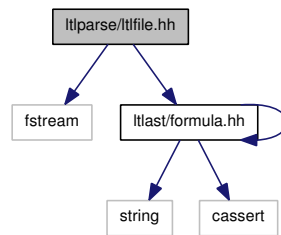
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.52 Itlparse/ltlfile.hh File Reference

```
#include <fstream>
```

```
#include "ltlast/formula.hh"
```

Include dependency graph for ltlfile.hh:



## Classes

- class [spot::ltl::ltl\\_file](#)  
*Read LTL formulae from a file, one by one.*

## Namespaces

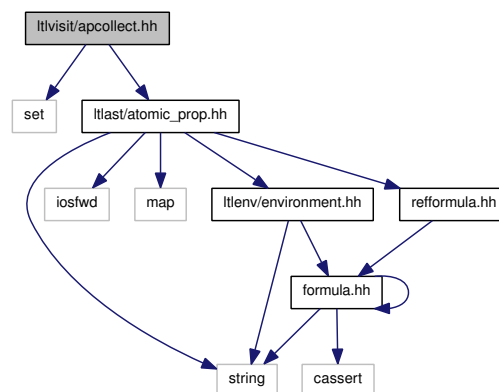
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.53 ltlvisit/apcollect.hh File Reference

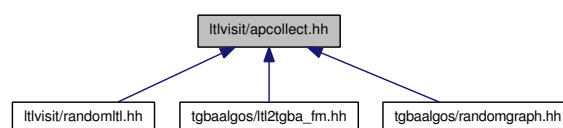
```
#include <set>
```

```
#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 >` [spot::ltl::atomic\\_prop\\_set](#)  
*Set of atomic propositions.*

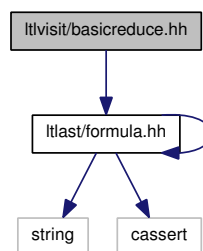
## Functions

- `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.*

## 8.54 ltlvisit/basicreduce.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for basicreduce.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

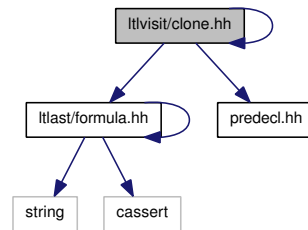
## Functions

- `formula *` [spot::ltl::basic\\_reduce](#) (`const formula *f`)  
*Basic rewritings.*
- `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.*

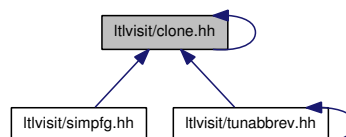
## 8.55 Itlvisit/clone.hh File Reference

```
#include "ltlast/formula.hh"
#include "ltlast/visitor.hh"
#include "predecl.hh"
```

Include dependency graph for clone.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `spot::ltl::clone_visitor`

*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::formula::clone` instead, it is way faster.*

### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### Functions

- `formula * spot::ltl::clone (const formula *f)`

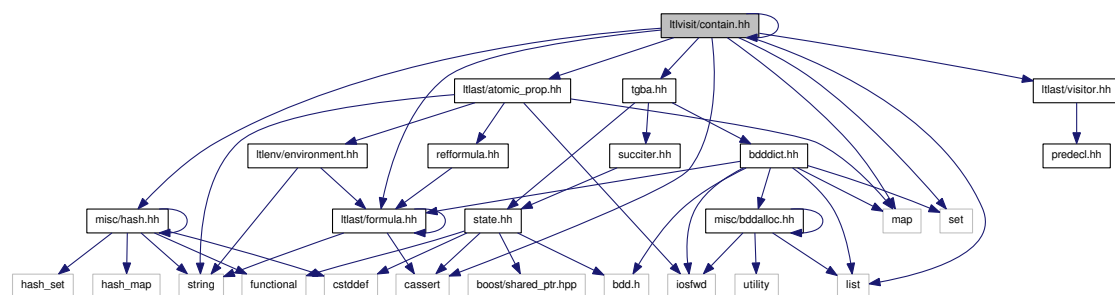
*Clone a formula.*

## 8.56 Itlvisit/contain.hh File Reference

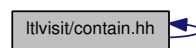
```
#include "ltlast/formula.hh"
#include "tgbaalgos/ltl2tgba_fm.hh"
```

```
#include "misc/hash.hh"
#include <list>
#include "tgba.hh"
#include <cassert>
#include <set>
#include "ltlast/atomic_prop.hh"
#include "ltlast/visitor.hh"
#include <map>
```

Include dependency graph for contain.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::language\\_containment\\_checker](#)
- struct [spot::ltl::language\\_containment\\_checker::record\\_](#)

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

- formula \* [spot::ltl::reduce\\_tau03](#) (const formula \*f, bool stronger=true)  
*Reduce a formula using language containment relationships.*

## 8.57 Itlvisit/destroy.hh File Reference

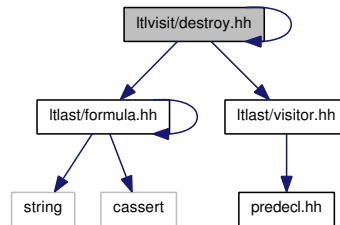
```
#include "ltlvisit/postfix.hh"
```



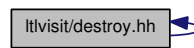
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.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

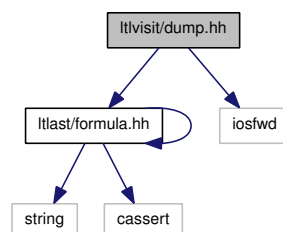
- void `spot::ltl::destroy` (const formula \*f)  
*Destroys a formula.*

## 8.58 Itlvisit/dump.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dump.hh:



## Namespaces

- namespace `spot`

- namespace `spot::ltl`

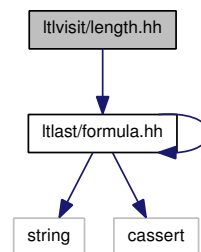
### Functions

- `std::ostream & spot::ltl::dump` (`std::ostream &os`, `const formula *f`)  
*Dump a formula tree.*

## 8.59 Itlvisit/length.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for length.hh:



### Namespaces

- namespace `spot`
- namespace `spot::ltl`

### Functions

- `int spot::ltl::length` (`const formula *f`)  
*Compute the length of a formula.*  
*The length of a formula is the number of atomic properties, constants, and operators (logical and temporal) occurring in the formula. `spot::ltl::multops` count only for 1, even if they have more than two operands (e.g. `a | b | c` has length 4, because `|` is represented once internally).*

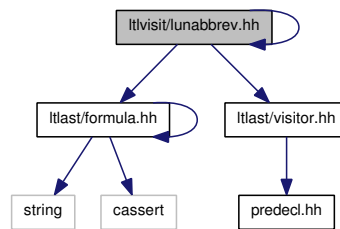
## 8.60 Itlvisit/lunabbrev.hh File Reference

```
#include "clone.hh"
```

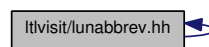
```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for lunabbrev.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::ltl::unabbreviate\\_logic\\_visitor](#)

*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](#).*

## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

- formula \* [spot::ltl::unabbreviate\\_logic](#) (const formula \*f)

*Clone and rewrite a formula to remove most of the abbreviated logical operators.*

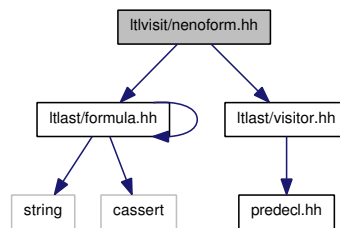
*This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#).*

## 8.61 Itlvisit/nenoform.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for nenoform.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

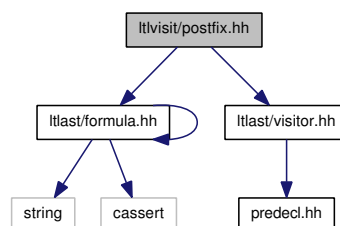
- formula \* [spot::ltl::negative\\_normal\\_form](#) (const formula \*f, bool negated=false)  
*Build the negative normal form of f.*  
*All negations of the formula are pushed in front of the atomic propositions.*

## 8.62 Itlvisit/postfix.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "ltlast/visitor.hh"
```

Include dependency graph for postfix.hh:



## Classes

- class [spot::ltl::postfix\\_visitor](#)  
*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.*

## Namespaces

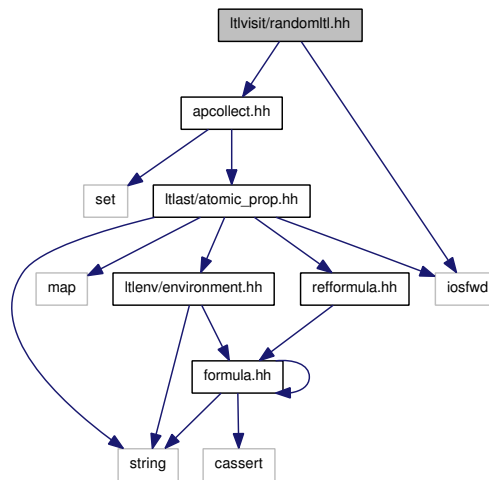
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.63 Itlvisit/randomltl.hh File Reference

```
#include "apcollect.hh"
```

```
#include <iosfwd>
```

Include dependency graph for randomltl.hh:



### Classes

- class [spot::ltl::random\\_ltl](#)

*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.*

- struct [spot::ltl::random\\_ltl::op\\_proba](#)

### Namespaces

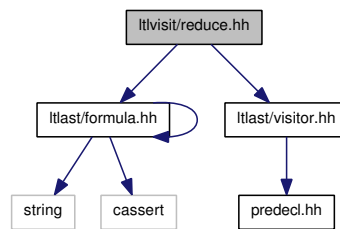
- namespace [spot](#)
- namespace [spot::ltl](#)

## 8.64 Itlvisit/reduce.hh File Reference

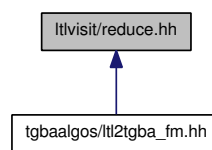
```
#include "ltlast/formula.hh"
```

```
#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 `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_Containment_Checks` = 8, `spot::ltl::Reduce_Containment_Checks_Stronger` = 16,  
`spot::ltl::Reduce_All` = -1U }  
*Options for `spot::ltl::reduce`.*

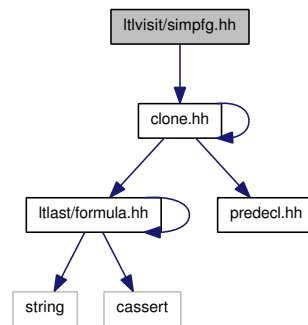
## Functions

- formula \* `spot::ltl::reduce` (const formula \*f, int opt=Reduce\_All)  
*Reduce a formula f.*
- bool `spot::ltl::is_eventual` (const formula \*f)  
*Check whether a formula is a pure eventuality.  
 Pure eventuality formulae are defined in.*
- bool `spot::ltl::is_universal` (const formula \*f)  
*Check whether a formula is purely universal.  
 Purely universal formulae are defined in.*

## 8.65 Itlvisit/simpfg.hh File Reference

```
#include "clone.hh"
```

Include dependency graph for simpfg.hh:



### Classes

- class [spot::ltl::simplify\\_f\\_g\\_visitor](#)  
*Replace  $true \ U \ f$  and  $false \ R \ g$  by  $F \ f$  and  $G \ g$ .*

### Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

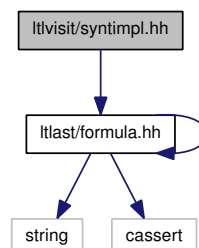
### Functions

- formula \* [spot::ltl::simplify\\_f\\_g](#) (const formula \*f)  
*Replace  $true \ U \ f$  and  $false \ R \ g$  by  $F \ f$  and  $G \ g$ .*

## 8.66 Itlvisit/syntimpl.hh File Reference

```
#include "ltlast/formula.hh"
```

Include dependency graph for syntimpl.hh:



## Namespaces

- namespace [spot](#)
- namespace [spot::ltl](#)

## Functions

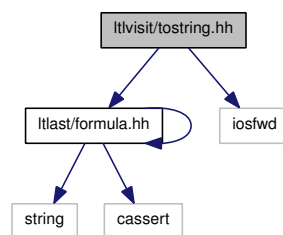
- bool [spot::ltl::syntactic\\_implication](#) (const formula \*f1, const formula \*f2)  
*Syntactic implication.  
This comes from.*
- bool [spot::ltl::syntactic\\_implication\\_neg](#) (const formula \*f1, const formula \*f2, bool right)  
*Syntactic implication.  
If right==false, true if !f1 < f2, false otherwise. If right==true, true if f1 < !f2, false otherwise.*

## 8.67 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 & [spot::ltl::to\\_string](#) (const formula \*f, std::ostream &os, bool full\_parent=false)  
*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ECTL formulae).*
- std::string [spot::ltl::to\\_string](#) (const formula \*f, bool full\_parent=false)  
*Output a formula as a string which is parsable unless the formula contains automaton operators (used in ECTL formulae).*
- std::ostream & [spot::ltl::to\\_spin\\_string](#) (const formula \*f, std::ostream &os)  
*Output a formula as a (parsable by Spin) string.*

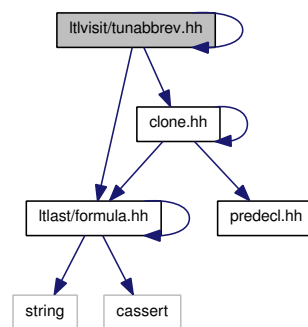


- `std::string spot::ltl::to_spin_string` (const formula \*f)  
Convert a formula into a (parsable by Spin) string.

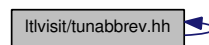
## 8.68 Itlvisit/tunabbrev.hh File Reference

```
#include "ltlast/formula.hh"
#include "ltlvisit/ltunabbrev.hh"
#include "clone.hh"
```

Include dependency graph for tunabbrev.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class `spot::ltl::unabbreviate_ltl_visitor`  
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`.

### Namespaces

- namespace `spot`
- namespace `spot::ltl`

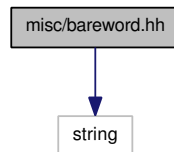
### Functions

- formula \* `spot::ltl::unabbreviate_ltl` (const formula \*f)  
Clone and rewrite a formula to remove most of the abbreviated LTL and logical operators.

## 8.69 misc/bareword.hh File Reference

```
#include <string>
```

Include dependency graph for bareword.hh:



### Namespaces

- namespace [spot](#)

### 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.*

## 8.70 misc/bddalloc.hh File Reference

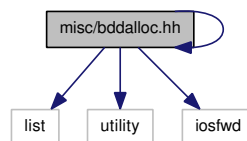
```
#include "freelist.hh"
```

```
#include <list>
```

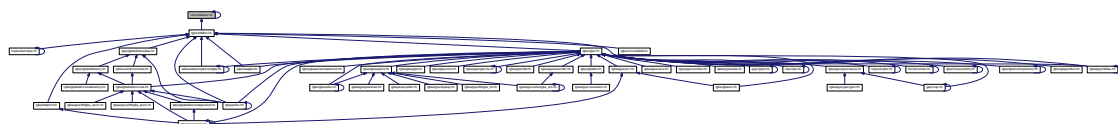
```
#include <utility>
```

```
#include <iosfwd>
```

Include dependency graph for bddalloc.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::bdd\\_allocator](#)

*Manage ranges of variables.*

## Namespaces

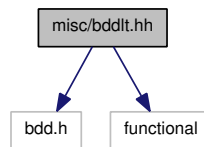
- namespace [spot](#)

## 8.71 misc/bddlt.hh File Reference

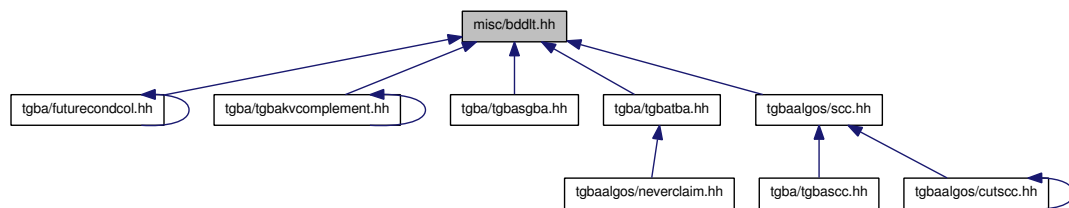
```
#include <bdd.h>
```

```
#include <functional>
```

Include dependency graph for bddlt.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [spot::bdd\\_less\\_than](#)  
*Comparison functor for BDDs.*

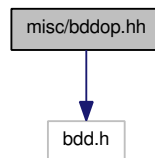
## Namespaces

- namespace [spot](#)

## 8.72 misc/bddop.hh File Reference

```
#include "bdd.h"
```

Include dependency graph for bddop.hh:



### Namespaces

- namespace [spot](#)

### Functions

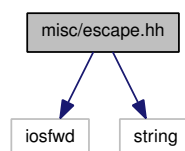
- bdd [spot::compute\\_all\\_acceptance\\_conditions](#) (bdd neg\_acceptance\_conditions)  
*Compute all acceptance conditions from all neg acceptance conditions.*
- bdd [spot::compute\\_neg\\_acceptance\\_conditions](#) (bdd all\_acceptance\_conditions)  
*Compute neg acceptance conditions from all acceptance conditions.*

## 8.73 misc/escape.hh File Reference

```
#include <iosfwd>
```

```
#include <string>
```

Include dependency graph for escape.hh:



### Namespaces

- namespace [spot](#)

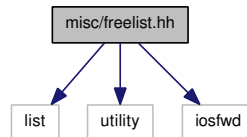
### Functions

- 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.*

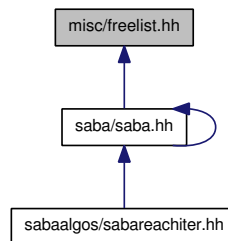
## 8.74 misc/freelist.hh File Reference

```
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for freelist.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::free\\_list](#)  
*Manage list of free integers.*

### Namespaces

- namespace [spot](#)

## 8.75 misc/hash.hh File Reference

```
#include <string>
#include <functional>
#include "hashfunc.hh"
#include <cstdint>
#include <hash_map>
#include <hash_set>
```



## Namespaces

- namespace [spot](#)

## Functions

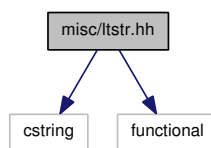
- `size_t spot::wang32_hash (size_t key)`  
*Thomas Wang's 32 bit hash function.*
- `size_t spot::knuth32_hash (size_t key)`  
*Knuth's Multiplicative hash function.*

## 8.77 misc/ltstr.hh File Reference

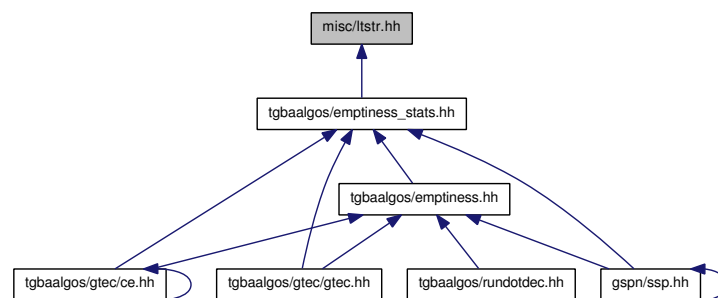
```
#include <cstring>
```

```
#include <functional>
```

Include dependency graph for ltstr.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [spot::char\\_ptr\\_less\\_than](#)  
*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\*.*

## Namespaces

- namespace [spot](#)

## 8.78 misc/memusage.hh File Reference

### Namespaces

- namespace [spot](#)

### Functions

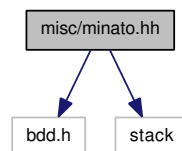
- int [spot::memusage](#) ()  
*Total number of pages in use by the program.*

## 8.79 misc/minato.hh File Reference

```
#include <bdd.h>
```

```
#include <stack>
```

Include dependency graph for minato.hh:



### Classes

- class [spot::minato\\_isop](#)  
*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.*
- struct [spot::minato\\_isop::local\\_vars](#)  
*Internal variables for [minato\\_isop](#).*

### Namespaces

- namespace [spot](#)

## 8.80 misc/modgray.hh File Reference

### Classes

- class [spot::loopless\\_modular\\_mixed\\_radix\\_gray\\_code](#)  
*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.*



## Namespaces

- namespace [spot](#)

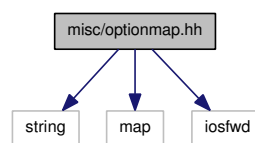
## 8.81 misc/optionmap.hh File Reference

```
#include <string>
```

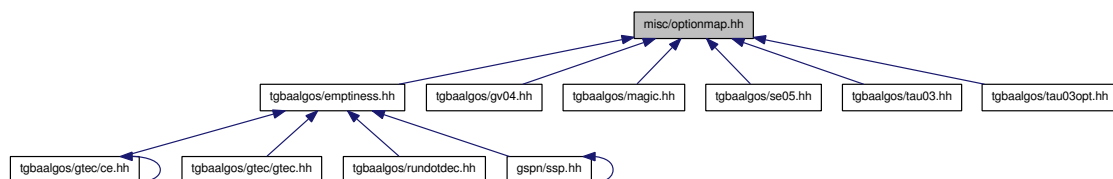
```
#include <map>
```

```
#include <iosfwd>
```

Include dependency graph for optionmap.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::option\\_map](#)  
*Manage a map of options.*  
*Each option is defined by a string and is associated to an integer value.*

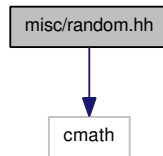
## Namespaces

- namespace [spot](#)

## 8.82 misc/random.hh File Reference

```
#include <cmath>
```

Include dependency graph for random.hh:



## Classes

- class `spot::barand< gen >`  
*Compute pseudo-random integer value between 0 and n included, following a binomial distribution for probability p.*

## Namespaces

- namespace `spot`

## Functions

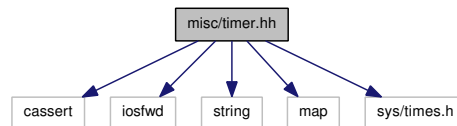
- void `spot::srand` (unsigned int seed)  
*Reset the seed of the pseudo-random number generator.*
- int `spot::rrand` (int min, int max)  
*Compute a pseudo-random integer value between min and max included.*
- int `spot::mrand` (int max)  
*Compute a pseudo-random integer value between 0 and max-1 included.*
- double `spot::drand` ()  
*Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).*
- double `spot::nrand` ()  
*Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).*
- double `spot::bmrnd` ()  
*Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).*
- int `spot::prand` (double p)  
*Return a pseudo-random positive integer value following a Poisson distribution with parameter p.*

## 8.83 misc/timer.hh File Reference

```
#include <cassert>
#include <iosfwd>
```

```
#include <string>
#include <map>
#include <sys/times.h>
```

Include dependency graph for timer.hh:



## Classes

- 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.*

## Namespaces

- namespace [spot](#)

## 8.84 misc/version.hh File Reference

### Namespaces

- namespace [spot](#)

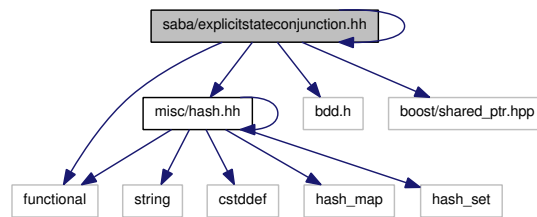
### Functions

- const char \* [spot::version](#) ()  
*Return Spot's version.*

## 8.85 saba/explicitstateconjunction.hh File Reference

```
#include <misc/hash.hh>
#include "sabasucciter.hh"
#include <bdd.h>
#include <functional>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for explicitstateconjunction.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class `spot::explicit_state_conjunction`  
*Basic implementation of `saba_state_conjunction`.  
 This class provides a basic implementation to iterate over a conjunction of states of a saba.*

## Namespaces

- namespace `spot`

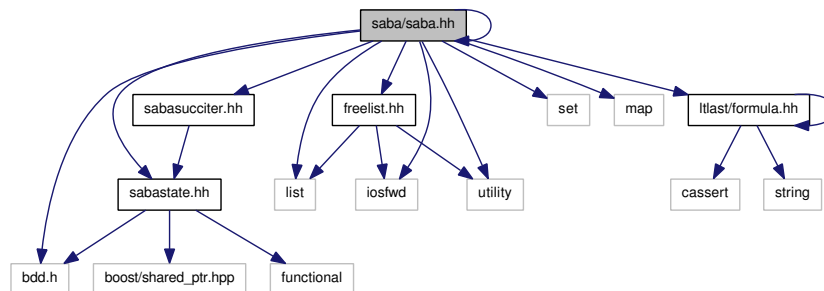
## 8.86 saba/saba.hh File Reference

```

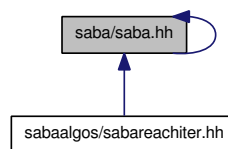
#include "sabastate.hh"
#include "sabasucciter.hh"
#include <tgba/bdddict.hh>
#include <list>
#include <set>
#include <map>
#include <iosfwd>
#include <bdd.h>
#include "ltlast/formula.hh"
#include "freelist.hh"
#include <utility>

```

Include dependency graph for saba.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::saba](#)

*A State-based Alternating (Generalized) Büchi Automaton.*

*Browsing such automaton can be achieved using two functions: `get_init_state`, and `succ_iter`.*

*The former returns the initial state while the latter lists the successor states of any state.*

## Namespaces

- namespace [spot](#)

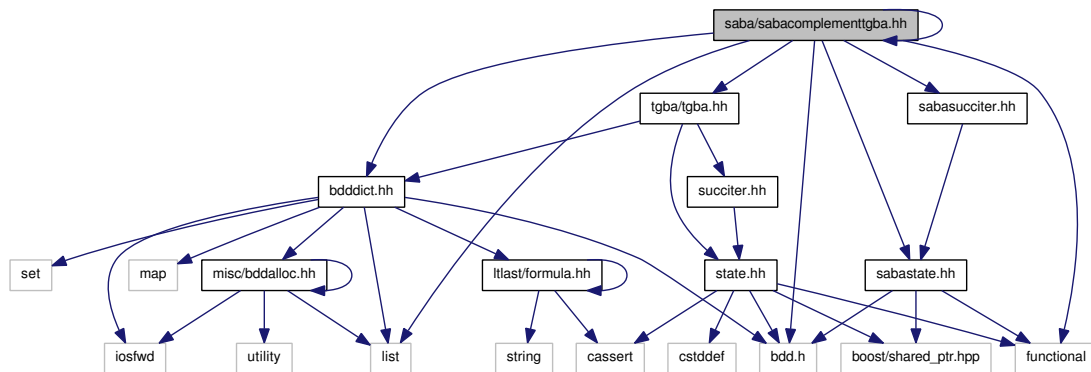
## 8.87 saba/sabacomplementtgba.hh File Reference

```

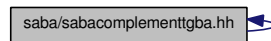
#include <tgba/tgba.hh>
#include <tgba/tgbatba.hh>
#include <list>
#include <bdd.h>
#include <functional>
#include "sabastate.hh"
#include "sabasucciter.hh"
#include <tgba/bdddict.hh>

```

Include dependency graph for sabacomplementtgba.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::saba\\_complement\\_tgba](#)  
*Complement a TGBA and produce a SABA.  
 The original TGBA is transformed into a States-based Büchi Automaton.*

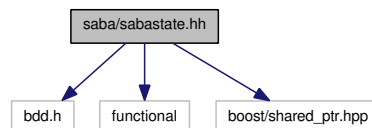
## Namespaces

- namespace [spot](#)

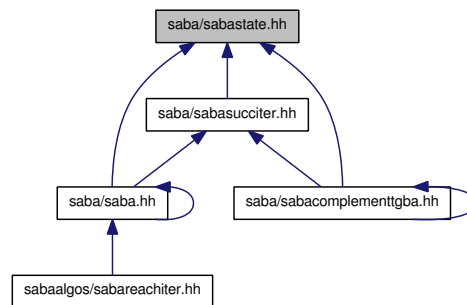
## 8.88 saba/sabastate.hh File Reference

```
#include <bdd.h>
#include <functional>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for sabastate.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::saba\\_state](#)  
*Abstract class for saba states.*
- struct [spot::saba\\_state\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for saba\_state\*.  
This is meant to be used as a comparison functor for STL map whose key are of type saba\_state\*.*
- struct [spot::saba\\_state\\_ptr\\_equal](#)  
*An Equivalence Relation for saba\_state\*.  
This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type saba\_state\*.*
- struct [spot::saba\\_state\\_ptr\\_hash](#)  
*Hash Function for saba\_state\*.  
This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type saba\_state\*.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for shared\_saba\_state (shared\_ptr<const saba\_state\*>).  
This is meant to be used as a comparison functor for STL map whose key are of type shared\_saba\_state.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_equal](#)  
*An Equivalence Relation for shared\_saba\_state (shared\_ptr<const saba\_state\*>).  
This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type shared\_saba\_state.*
- struct [spot::saba\\_state\\_shared\\_ptr\\_hash](#)  
*Hash Function for shared\_saba\_state (shared\_ptr<const saba\_state\*>).  
This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type shared\_saba\_state.*

## Namespaces

- namespace [spot](#)

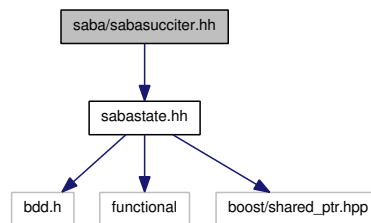
### Typedefs

- typedef boost::shared\_ptr< const saba\_state > [spot::shared\\_saba\\_state](#)

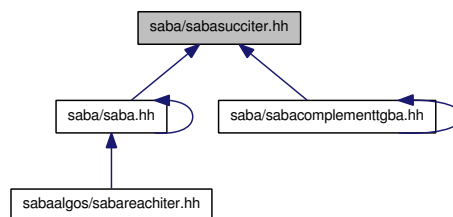
## 8.89 saba/sabasucciter.hh File Reference

```
#include "sabastate.hh"
```

Include dependency graph for sabasucciter.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::saba\\_state\\_conjunction](#)  
*Iterate over a conjunction of [saba\\_state](#).  
 This class provides the basic functionalities required to iterate over a conjunction of states of a saba.*
- class [spot::saba\\_succ\\_iterator](#)  
*Iterate over the successors of a [saba\\_state](#).  
 This class provides the basic functionalities required to iterate over the successors of a state of a saba. Since transitions of an alternating automaton are defined as a boolean function with conjunctions (universal) and disjunctions (non-deterministic),.*

### Namespaces

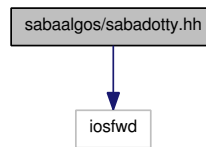
- namespace [spot](#)

## 8.90 sabaalgos/sabadotty.hh File Reference

```
#include <iosfwd>
```



Include dependency graph for sabadotty.hh:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & spot::saba\_dotty\_reachable (std::ostream &os, const saba *g)`  
*Print reachable states in dot format.*

## 8.91 sabaalgorithms/sabareachiter.hh File Reference

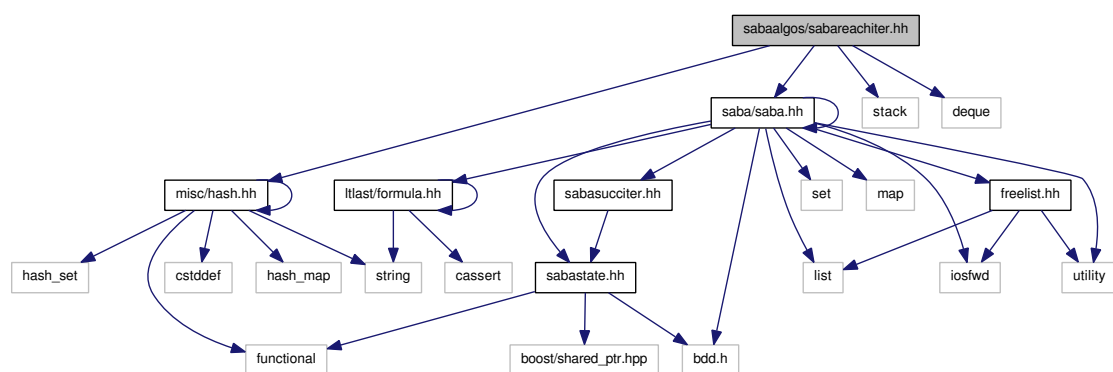
```
#include "misc/hash.hh"
```

```
#include "saba/saba.hh"
```

```
#include <stack>
```

```
#include <deque>
```

Include dependency graph for sabareachiter.hh:



## Classes

- class [spot::saba\\_reachable\\_iterator](#)  
*Iterate over all reachable states of a [spot::saba](#).*
- class [spot::saba\\_reachable\\_iterator\\_depth\\_first](#)  
*An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states depth first.*

- class [spot::saba\\_reachable\\_iterator\\_breadth\\_first](#)

*An implementation of [spot::saba\\_reachable\\_iterator](#) that browses states breadth first.*

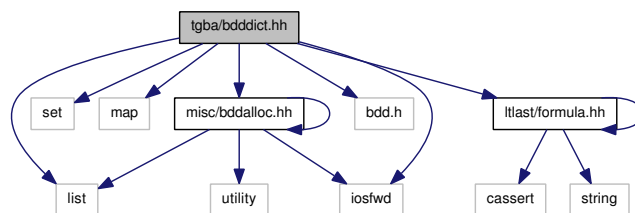
## Namespaces

- namespace [spot](#)

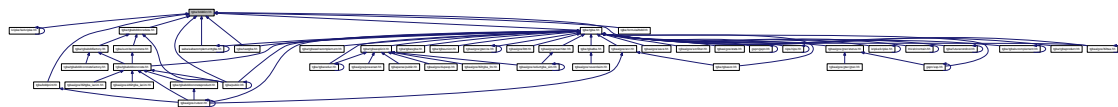
## 8.92 tgba/bdddict.hh File Reference

```
#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:



## Classes

- class [spot::bdd\\_dict](#)
- class [spot::bdd\\_dict::anon\\_free\\_list](#)

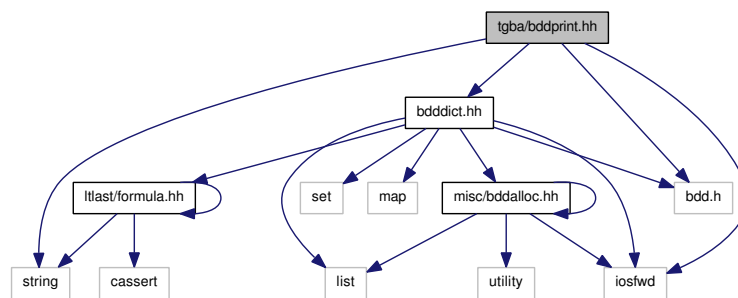
## Namespaces

- namespace [spot](#)

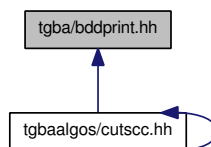
## 8.93 tgba/bddprint.hh File Reference

```
#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 & spot::bdd\_print\_sat (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a list of literals.*
- `std::string spot::bdd\_format\_sat (const bdd_dict *dict, bdd b)`  
*Format a BDD as a list of literals.*
- `std::ostream & spot::bdd\_print\_acc (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a list of acceptance conditions.*
- `std::ostream & spot::bdd\_print\_accset (std::ostream &os, const bdd_dict *dict, bdd b)`  
*Print a BDD as a set of acceptance conditions.*
- `std::string spot::bdd\_format\_accset (const bdd_dict *dict, bdd b)`  
*Format a BDD as a set of acceptance conditions.*

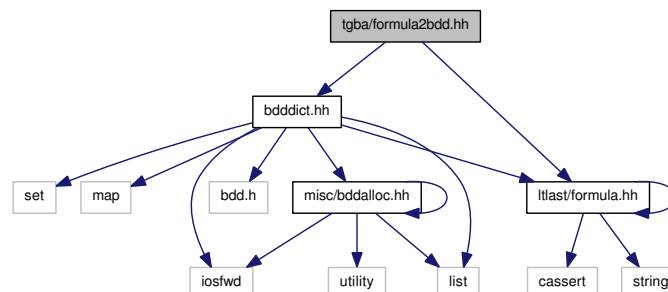
- `std::ostream & spot::bdd_print_set` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a set.*
- `std::string spot::bdd_format_set` (`const bdd_dict *dict`, `bdd b`)  
*Format a BDD as a set.*
- `std::ostream & spot::bdd_print_formula` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a formula.*
- `std::string spot::bdd_format_formula` (`const bdd_dict *dict`, `bdd b`)  
*Format a BDD as a formula.*
- `std::ostream & spot::bdd_print_dot` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a diagram in doty format.*
- `std::ostream & spot::bdd_print_table` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)  
*Print a BDD as a table.*

## 8.94 tgba/formula2bdd.hh File Reference

```
#include "bdddict.hh"
```

```
#include "ltnlast/formula.hh"
```

Include dependency graph for formula2bdd.hh:



### Namespaces

- namespace `spot`

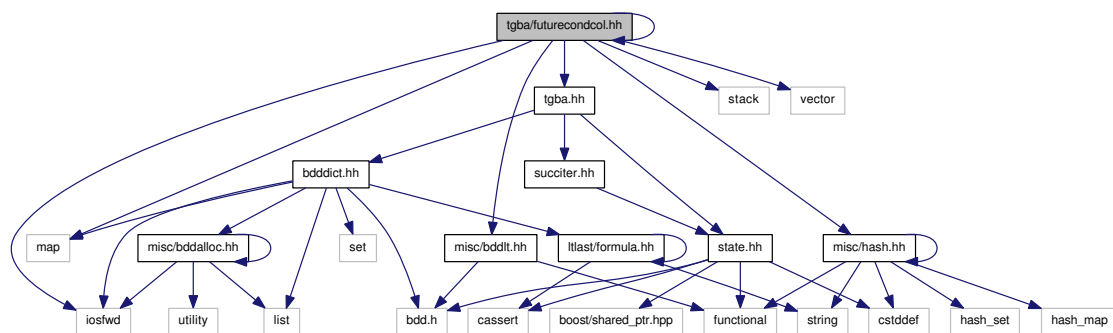
### Functions

- `bdd spot::formula_to_bdd` (`const ltl::formula *f`, `bdd_dict *d`, `void *for_me`)
- `const ltl::formula * spot::bdd_to_formula` (`bdd f`, `const bdd_dict *d`)

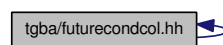
## 8.95 tgba/futurecondcol.hh File Reference

```
#include "tgbascc.hh"
#include "tgba.hh"
#include <map>
#include <stack>
#include <vector>
#include <iosfwd>
#include "misc/hash.hh"
#include "misc/bddlt.hh"
```

Include dependency graph for futurecondcol.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::future\\_conditions\\_collector](#)

Wrap a tgba to offer information about upcoming conditions.

This class is a [spot::tgba](#) wrapper that simply add a new method, [future\\_conditions\(\)](#), to any [spot::tgba](#).

### Namespaces

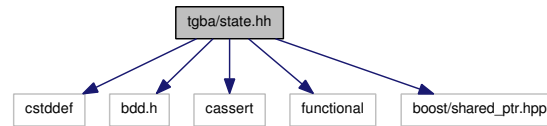
- namespace [spot](#)

## 8.96 tgba/state.hh File Reference

```
#include <cstdint>
#include <bdd.h>
#include <cassert>
```

```
#include <functional>
#include <boost/shared_ptr.hpp>
```

Include dependency graph for state.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::state](#)  
*Abstract class for states.*
- struct [spot::state\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for state\*.  
This is meant to be used as a comparison functor for STL map whose key are of type state\*.*
- struct [spot::state\\_ptr\\_equal](#)  
*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\*.*
- struct [spot::state\\_ptr\\_hash](#)  
*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\*.*
- struct [spot::state\\_shared\\_ptr\\_less\\_than](#)  
*Strict Weak Ordering for shared\_state (shared\_ptr<const state\*>).  
This is meant to be used as a comparison functor for STL map whose key are of type shared\_state.*
- struct [spot::state\\_shared\\_ptr\\_equal](#)  
*An Equivalence Relation for shared\_state (shared\_ptr<const state\*>).  
This is meant to be used as a comparison functor for Sgi hash\_map whose key are of type shared\_state.*
- struct [spot::state\\_shared\\_ptr\\_hash](#)  
*Hash Function for shared\_state (shared\_ptr<const state\*>).  
This is meant to be used as a hash functor for Sgi's hash\_map whose key are of type shared\_state.*

## Namespaces

- namespace [spot](#)

**Typedefs**

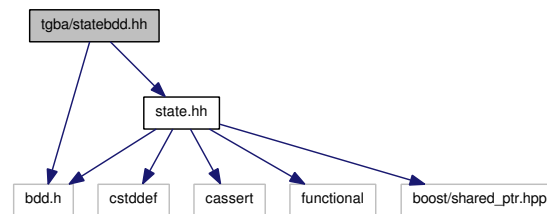
- typedef boost::shared\_ptr< const state > [spot::shared\\_state](#)

**8.97 tgba/statebdd.hh File Reference**

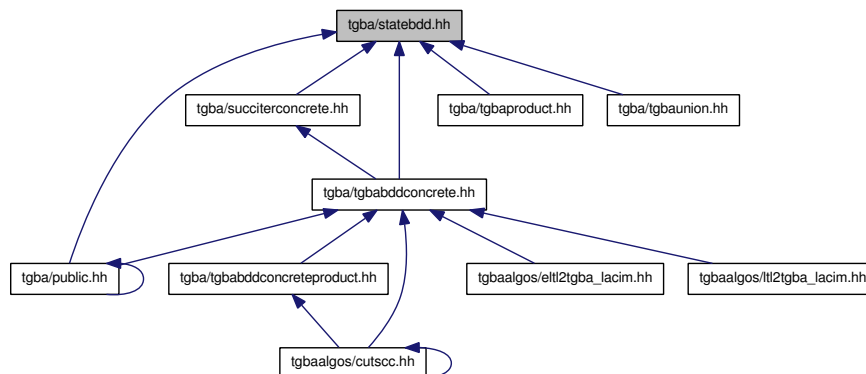
```
#include <bdd.h>
```

```
#include "state.hh"
```

Include dependency graph for statebdd.hh:



This graph shows which files directly or indirectly include this file:

**Classes**

- class [spot::state\\_bdd](#)

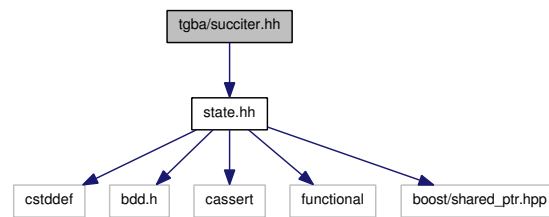
**Namespaces**

- namespace [spot](#)

**8.98 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:



## Classes

- class [spot::tgba\\_succ\\_iterator](#)

*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.*

## Namespaces

- namespace [spot](#)

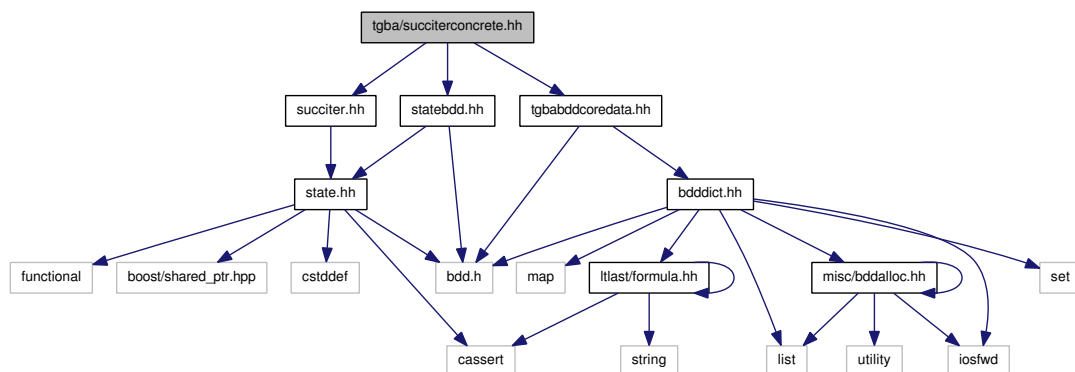
## 8.99 tgba/succiterconcrete.hh File Reference

```
#include "statebdd.hh"
```

```
#include "succiter.hh"
```

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for succiterconcrete.hh:







A self-loop Transition-based Alternating Automaton (TAA) which is seen as a TGBA (abstract class, see below).

- struct `spot::taa_tgba::transition`  
*Explicit transitions.*
- class `spot::state_set`  
*Set of states deriving from `spot::state`.*
- class `spot::taa_succ_iterator`
- struct `spot::taa_succ_iterator::distance_sort`
- class `spot::taa_tgba_labelled< label, label_hash >`
- class `spot::taa_tgba_string`
- class `spot::taa_tgba_formula`

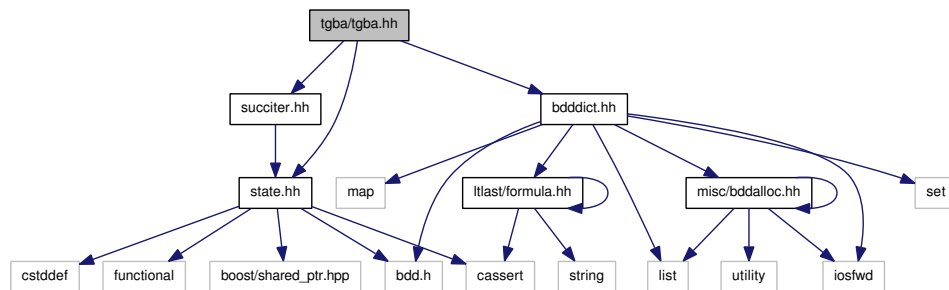
## Namespaces

- namespace `spot`

## 8.101 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:



## Classes

- class `spot::tgba`

### A Transition-based Generalized Büchi Automaton.

The acronym TGBA (Transition-based Generalized Büchi Automaton) was coined by Dimitra Giannakopoulou and Flavio Lerda in "From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata". (FORTE'02).

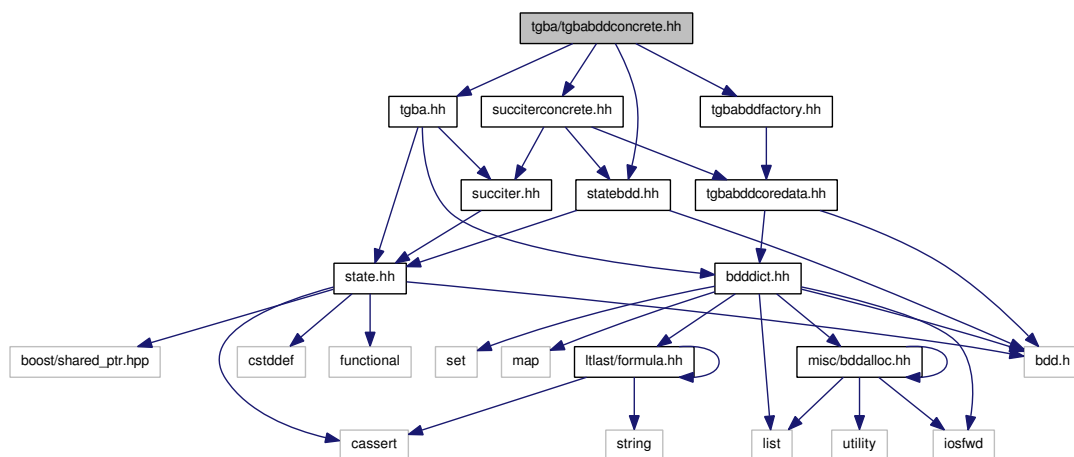
## Namespaces

- namespace **spot**

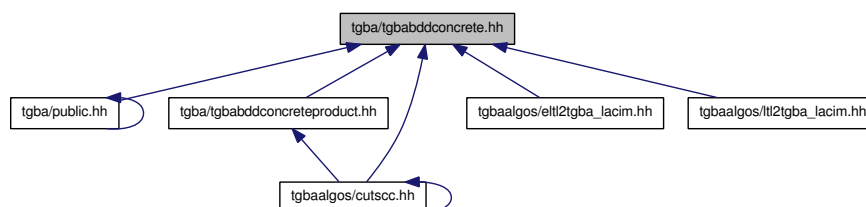
### 8.102 tgba/tgbabddconcrete.hh File Reference

```
#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:



## Classes

- class `spot::tgba_bdd_concrete`  
*A concrete `spot::tgba` implemented using BDDs.*

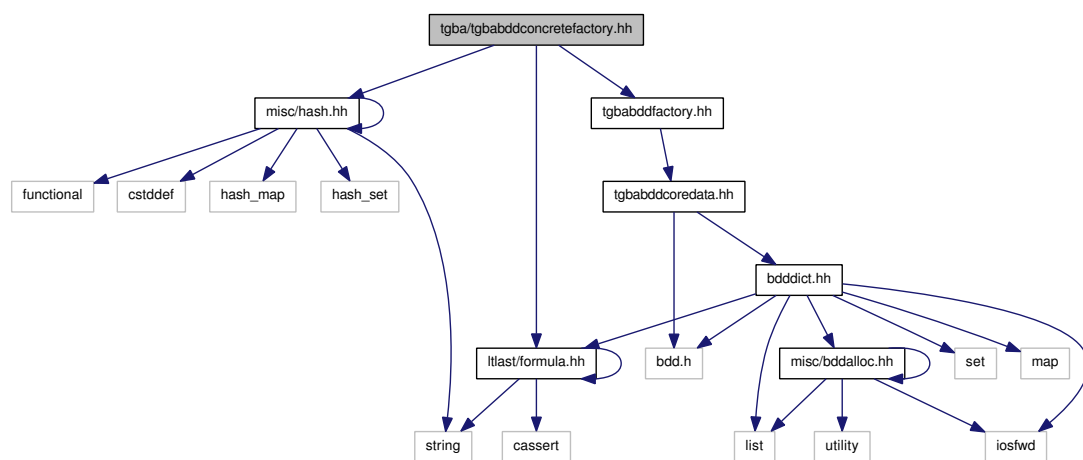
## Namespaces

- namespace **spot**

### 8.103 tgba/tgbabddconcretfactory.hh File Reference

```
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "tgbabddfactory.hh"
```

Include dependency graph for tgbabddconcretfactory.hh:



## Classes

- class `spot::tgba_bdd_concrete_factory`  
*Helper class to build a `spot::tgba_bdd_concrete` object.*

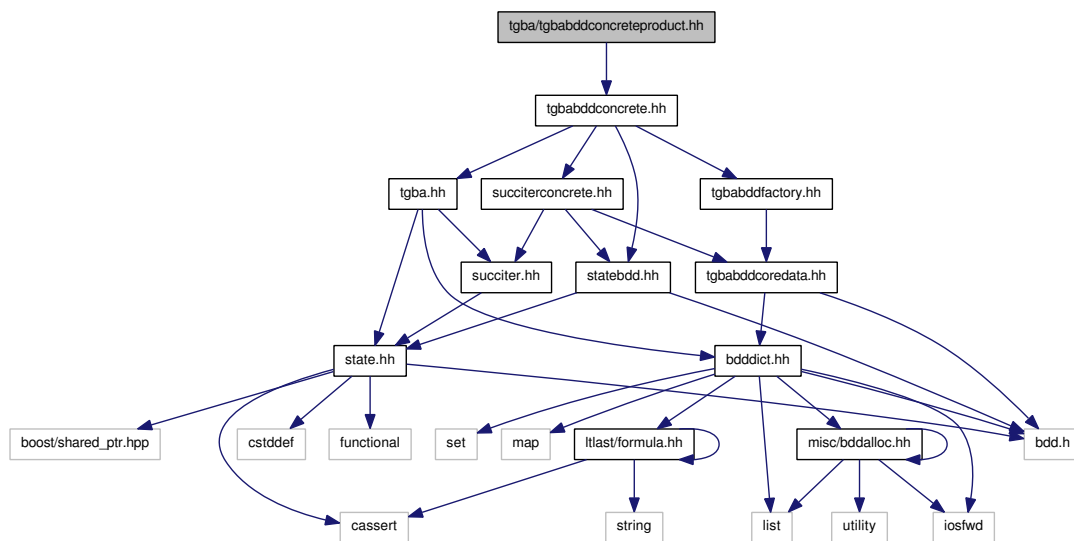
## Namespaces

- namespace **spot**

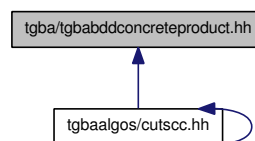
### 8.104 tgba/tgbabddconcreteproduct.hh File Reference

```
#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 * spot::product (const tgba_bdd_concrete *left, const tgba_bdd_concrete *right)`

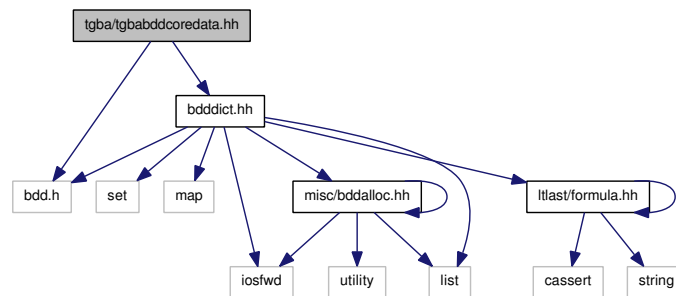
*Multiplies two [spot::tgba\\_bdd\\_concrete](#) automata.*

*This function builds the resulting product as another [spot::tgba\\_bdd\\_concrete](#) automaton.*

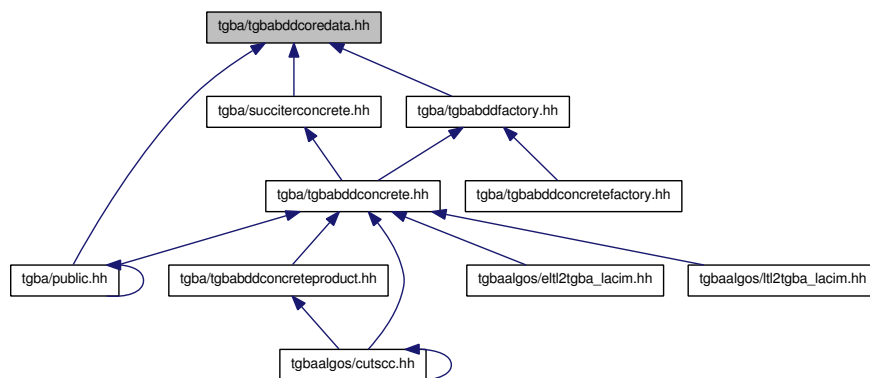
## 8.105 tgba/tgbabddcoredata.hh File Reference

```
#include <bdd.h>
#include "bdddict.hh"
```

Include dependency graph for tgbabddcoredata.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct `spot::tgba_bdd_core_data`  
*Core data for a TGBA encoded using BDDs.*

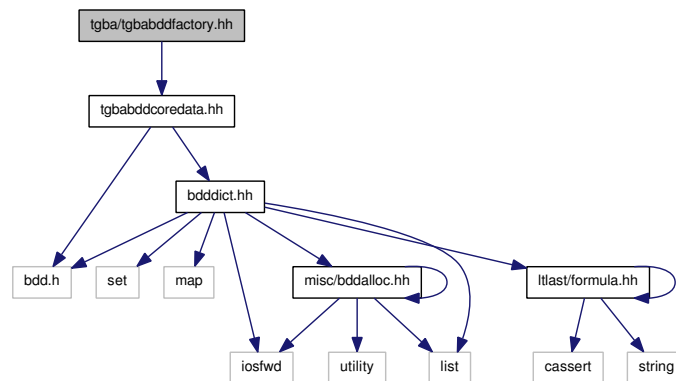
## Namespaces

- namespace **spot**

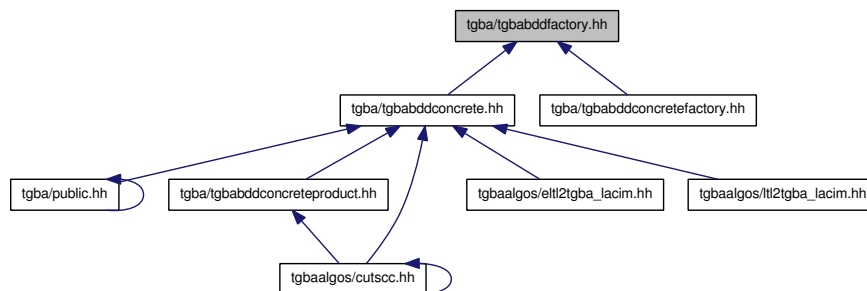
### 8.106 tgba/tgbabddfactory.hh File Reference

```
#include "tgbabddcoredata.hh"
```

Include dependency graph for `tgbabddfactory.hh`:



This graph shows which files directly or indirectly include this file:



## Classes

- class `spot::tgba_bdd_factory`  
*Abstract class for `spot::tgba_bdd_concrete` factories.*

## Namespaces

- namespace `spot`

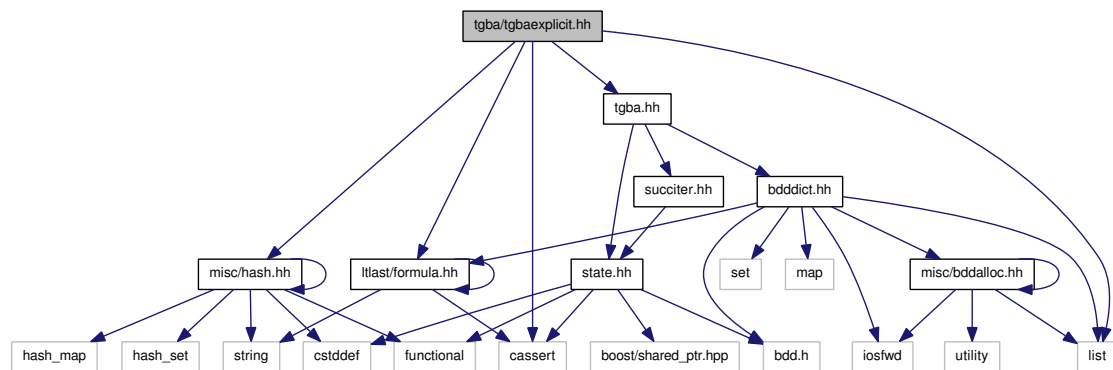
## 8.107 tgba/tgbaexplicit.hh File Reference

```

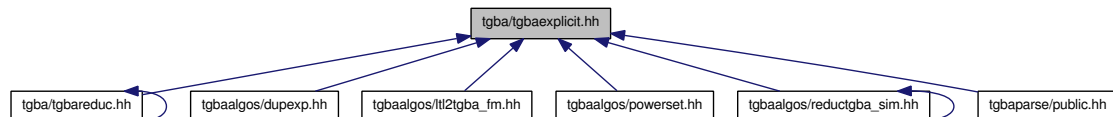
#include "misc/hash.hh"
#include <list>
#include "tgba.hh"
#include "ltlast/formula.hh"
#include <cassert>

```

Include dependency graph for tgbaexplicit.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::tgba\\_explicit](#)
- struct [spot::tgba\\_explicit::transition](#)  
Explicit transitions (used by [spot::tgba\\_explicit](#)).
- class [spot::state\\_explicit](#)
- class [spot::tgba\\_explicit\\_succ\\_iterator](#)
- class [spot::tgba\\_explicit\\_labelled< label, label\\_hash >](#)  
A [tgba\\_explicit](#) instance with states labeled by a given type.
- class [spot::tgba\\_explicit\\_string](#)
- class [spot::tgba\\_explicit\\_formula](#)

## Namespaces

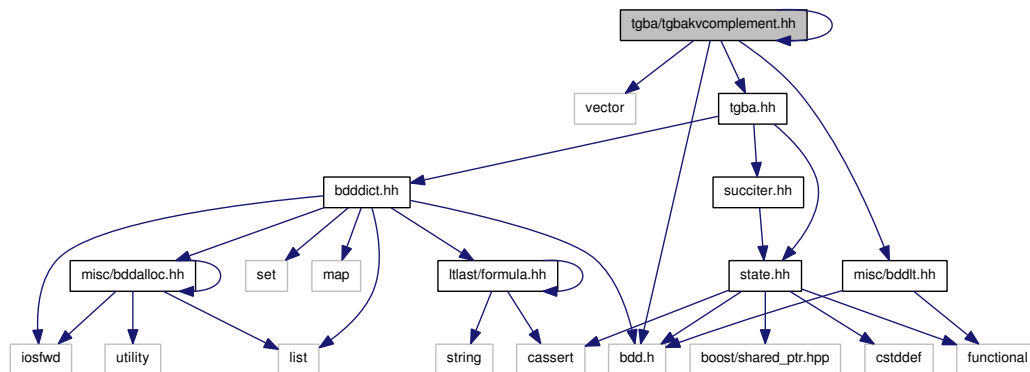
- namespace [spot](#)

## 8.108 tgba/tgbakvcomplement.hh File Reference

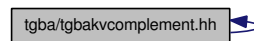
```
#include <vector>
#include "bdd.h"
#include "tgba.hh"
#include "tgba/tgbasgba.hh"
#include "misc/bddlt.hh"
```



Include dependency graph for tgbakvcomplement.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::bdd\\_ordered](#)
- class [spot::tgba\\_kv\\_complement](#)  
*Build a complemented automaton.  
 The construction comes from:*

## Namespaces

- namespace [spot](#)

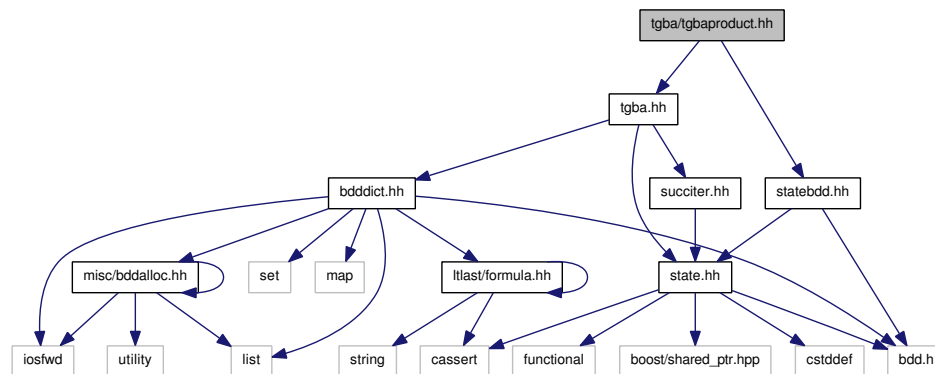
## Typedefs

- typedef `std::vector< bdd_ordered >` [spot::acc\\_list\\_t](#)

## 8.109 tgba/tgbaproduct.hh File Reference

```
#include "tgba.hh"
#include "statebdd.hh"
```

Include dependency graph for `tgbaproduct.hh`:



## Classes

- class `spot::state_product`  
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.
- class `spot::tgba_succ_iterator_product`  
Iterate over the successors of a product computed on the fly.
- class `spot::tgba_product`  
A lazy product. (States are computed on the fly.).

## Namespaces

- namespace `spot`

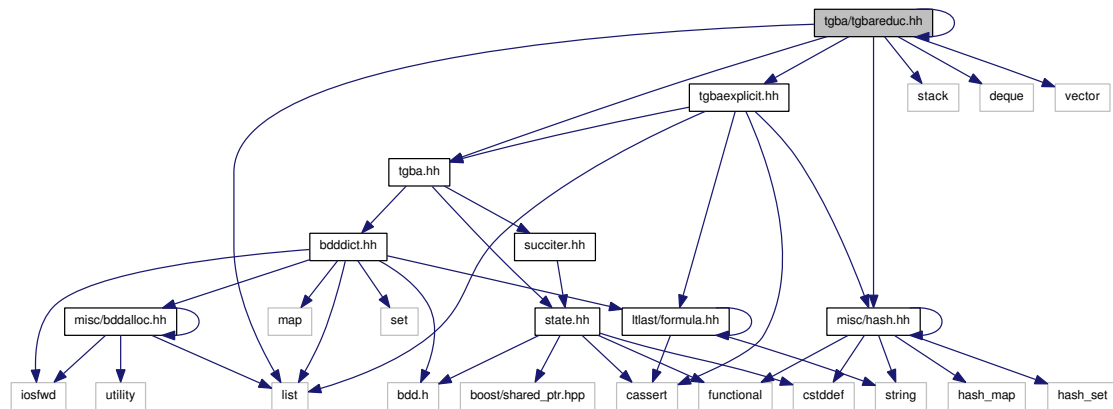
## 8.110 tgba/tgbareduc.hh File Reference

```

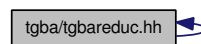
#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#include "misc/hash.hh"
#include "tgba/tgba.hh"
#include <stack>
#include <deque>
#include <list>
#include <vector>

```

Include dependency graph for `tgba/tgbareduc.hh`:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::direct\\_simulation\\_relation](#)
- class [spot::delayed\\_simulation\\_relation](#)
- class [spot::tgba\\_reduc](#)

## Namespaces

- namespace [spot](#)

## Typedefs

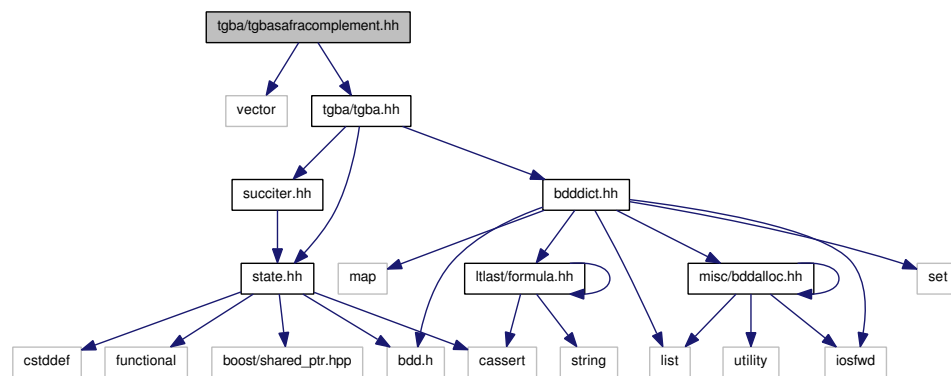
- typedef `std::pair< const spot::state *, const spot::state *>` [spot::state\\_couple](#)
- typedef `std::vector< state\_couple *>` [spot::simulation\\_relation](#)

## 8.111 tgba/tgbasafracomplement.hh File Reference

```
#include <vector>

#include "tgba/tgba.hh"
```

Include dependency graph for `tgba/tgbascc.hh`:



## Classes

- class `spot::tgba_safracomplement`  
*Build a complemented automaton.  
 It creates an automaton that recognizes the negated language of aut.*

## Namespaces

- namespace `spot`

## Defines

- `#define TRANSFORM_TO_TGBA (!TRANSFORM_TO_TBA)`

## Functions

- void `spot::display_safracomplement` (const `tgba_safracomplement` \*a)  
*Produce a dot output of the Safra automaton associated to a.*

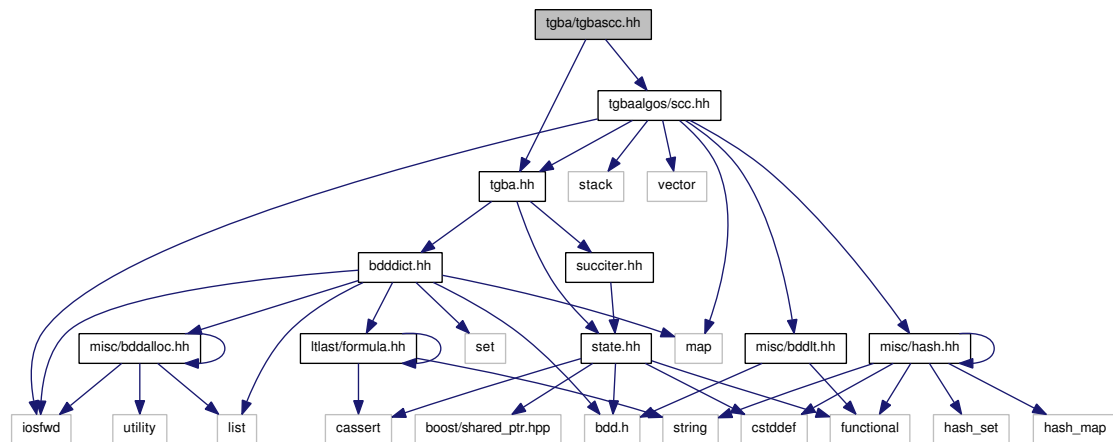
### 8.111.1 Define Documentation

#### 8.111.1.1 `#define TRANSFORM_TO_TGBA (!TRANSFORM_TO_TBA)`

## 8.112 tgba/tgbascc.hh File Reference

```
#include "tgba.hh"
#include "tgbaalgos/scc.hh"
```

Include dependency graph for tgbascc.hh:



## Classes

- class [spot::tgba\\_scc](#)

Wrap a *tgba* to offer information about strongly connected components.

This class is a [spot::tgba](#) wrapper that simply add a new method [scc\\_of\\_state\(\)](#) to retrieve the number of a SCC a state belongs to.

## Namespaces

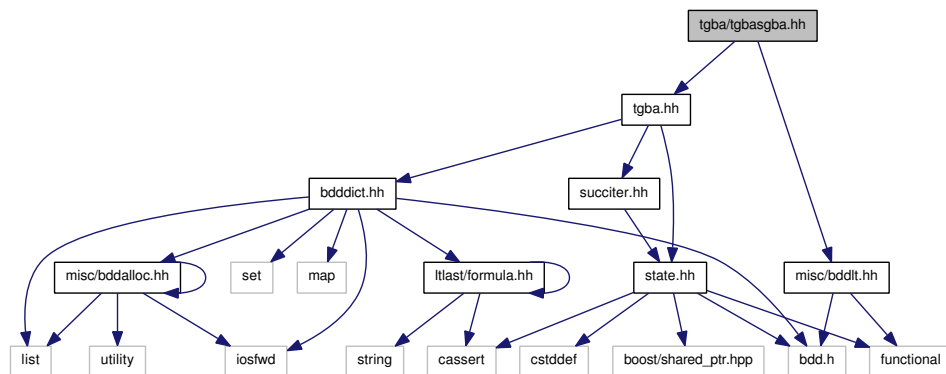
- namespace [spot](#)

## 8.113 tgba/tgbasgba.hh File Reference

```
#include "tgba.hh"
```

```
#include "misc/bddlt.hh"
```

Include dependency graph for tgbasgba.hh:



## Classes

- class [spot::tgba\\_sgba\\_proxy](#)

Change the labeling-mode of [spot::tgba](#) on the fly, producing a state-based generalized Büchi automaton. This class acts as a proxy in front of a [spot::tgba](#), that should label on states on-the-fly. The result is still a [spot::tgba](#), but acceptances conditions are also on states.

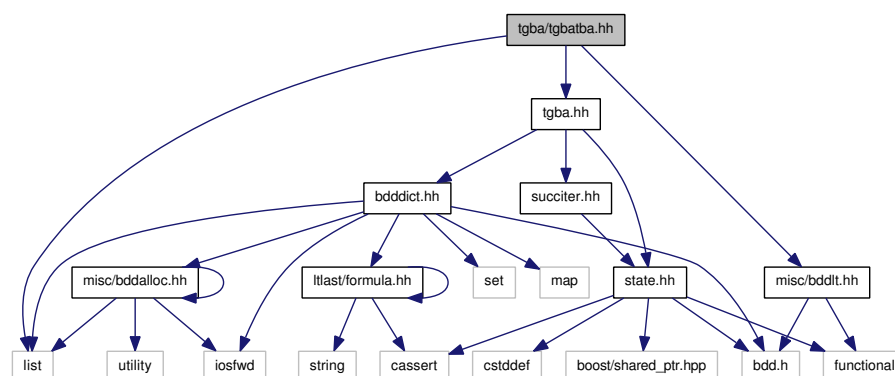
## Namespaces

- namespace [spot](#)

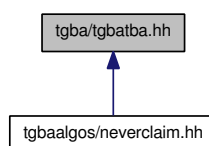
## 8.114 tgba/tgbatba.hh File Reference

```
#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:



## Classes

- class [spot::tgba\\_tba\\_proxy](#)

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).

- class `spot::tgba_sba_proxy`

*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.*

## Namespaces

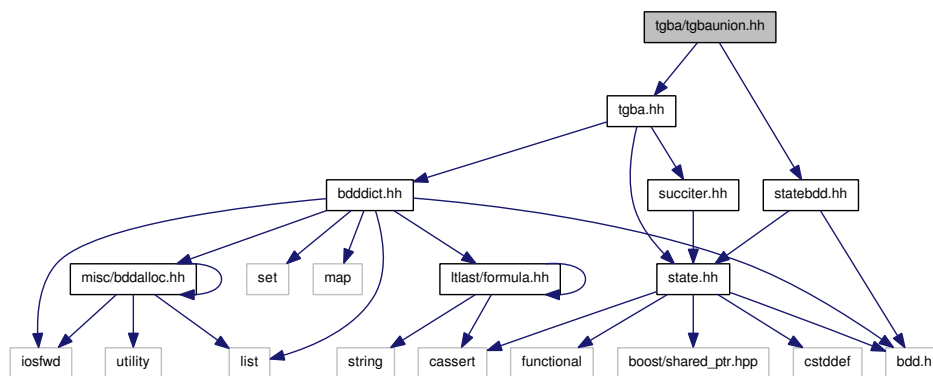
- namespace `spot`

## 8.115 tgba/tgbaunion.hh File Reference

```
#include "tgba.hh"
```

```
#include "statebdd.hh"
```

Include dependency graph for tgbaunion.hh:



## Classes

- class `spot::state_union`

*A state for `spot::tgba_union`.*

*This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.*

- class `spot::tgba_succ_iterator_union`

*Iterate over the successors of an union computed on the fly.*

- class `spot::tgba_union`

*A lazy union. (States are computed on the fly.).*

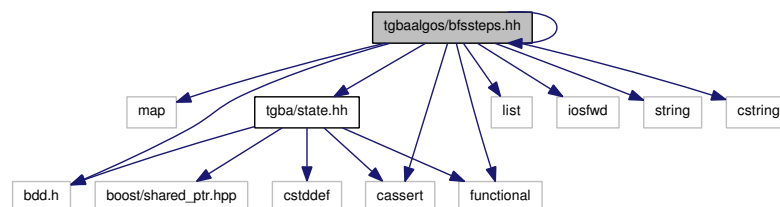
## Namespaces

- namespace `spot`

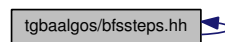
## 8.116 tgbaalgos/bfssteps.hh File Reference

```
#include <map>
#include "tgba/state.hh"
#include "emptiness.hh"
#include <list>
#include <iosfwd>
#include <bdd.h>
#include <string>
#include <cassert>
#include <cstring>
#include <functional>
```

Include dependency graph for bfssteps.hh:



This graph shows which files directly or indirectly include this file:



### Classes

- class [spot::bfs\\_steps](#)

*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.*

### Namespaces

- namespace [spot](#)

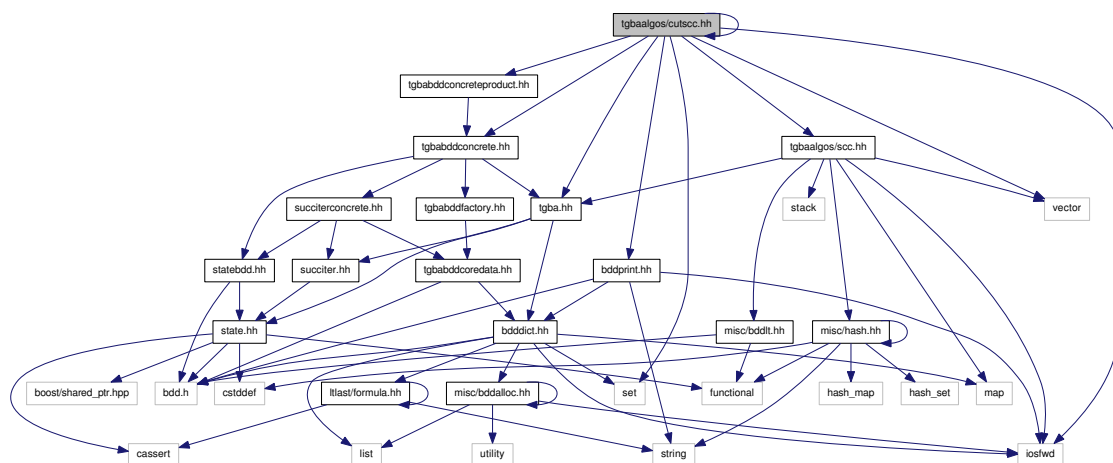
## 8.117 tgbaalgos/cutsc.h File Reference

```
#include <iosfwd>
#include <set>
#include <vector>
```

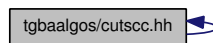


```
#include "tgba/public.hh"
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include "tgbabddconcreteproduct.hh"
#include "bddprint.hh"
#include "tgbaalgos/scc.hh"
```

Include dependency graph for cutscc.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [spot::sccs\\_set](#)

## Namespaces

- namespace [spot](#)

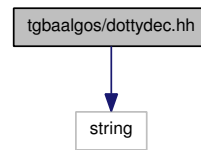
## Functions

- `std::vector< std::vector< sccs_set * > > * spot::find\_paths (tgba *a, const scc_map &m)`
- `unsigned spot::max\_spanning\_paths (std::vector< sccs_set * > *paths, scc_map &m)`
- `std::list< tgba * > spot::split\_tgba (tgba *a, const scc_map &m, unsigned split_number)`

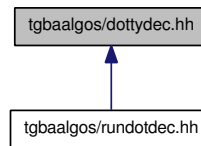
## 8.118 tgbaalgos/dottydec.hh File Reference

```
#include <string>
```

Include dependency graph for dottydec.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::dotty\\_decorator](#)  
Choose state and link styles for [spot::dotty\\_reachable](#).

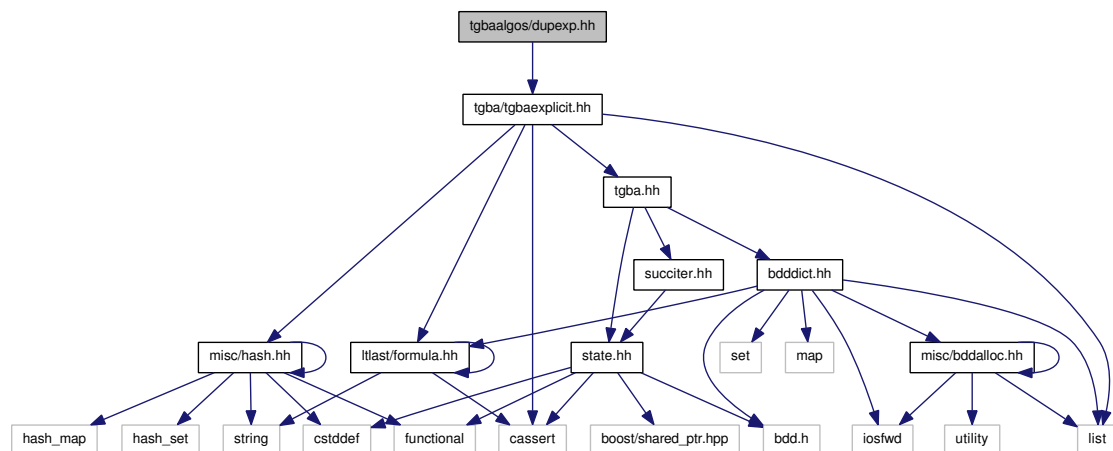
## Namespaces

- namespace [spot](#)

## 8.119 tgbaalgos/dupeexp.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for dupeexp.hh:



## Namespaces

- namespace [spot](#)

## Functions

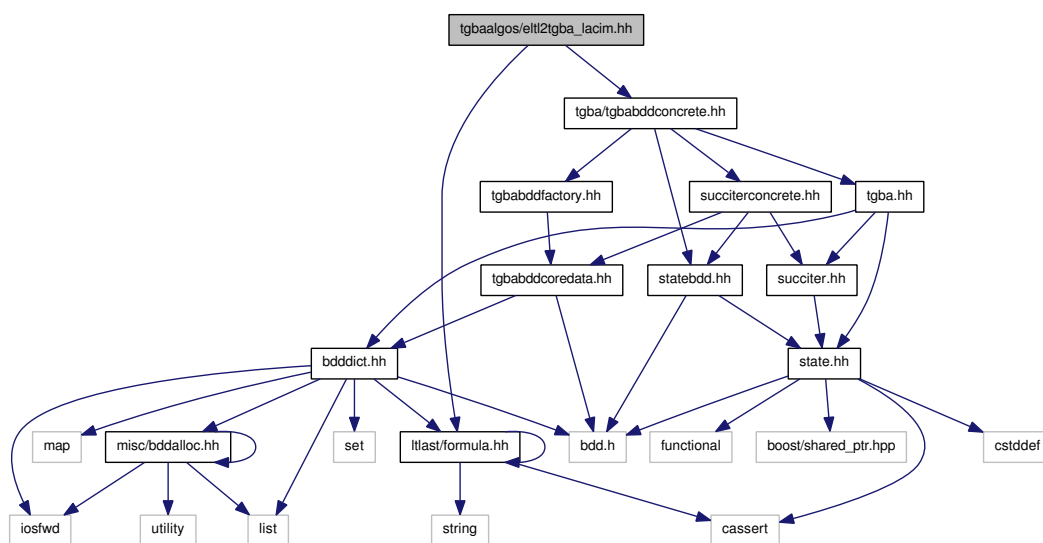
- tgba\_explicit \* [spot::tgba\\_dupexp\\_bfs](#) (const tgba \*aut)  
*Build an explicit automata from all states of aut, numbering states in bread first order as they are processed.*
- tgba\_explicit \* [spot::tgba\\_dupexp\\_dfs](#) (const tgba \*aut)  
*Build an explicit automata from all states of aut, numbering states in depth first order as they are processed.*

## 8.120 tgbaalgos/eltl2tgba\_lacim.hh File Reference

```
#include "ltlast/formula.hh"
```

```
#include "tgba/tgbabddconcrete.hh"
```

Include dependency graph for eltl2tgba\_lacim.hh:



## Namespaces

- namespace [spot](#)

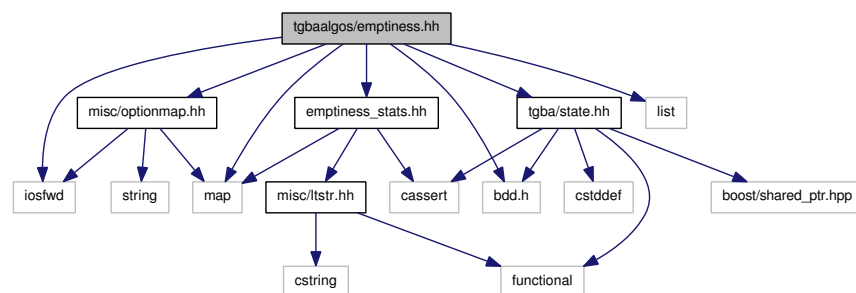
## Functions

- tgba\_bdd\_concrete \* [spot::eltl\\_to\\_tgba\\_lacim](#) (const ltl::formula \*f, bdd\_dict \*dict)  
*Build a [spot::tgba\\_bdd\\_concrete](#) from an ETL formula.  
This is based on the following paper.*

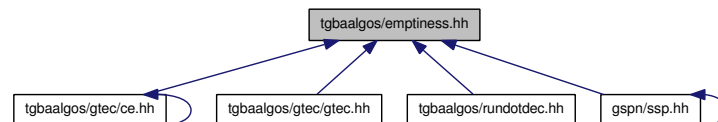
## 8.121 tgbaalgos/emptiness.hh File Reference

```
#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:



### 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](#)
- struct [spot::tgba\\_run](#)  
*An accepted run, for a tgba.*
- struct [spot::tgba\\_run::step](#)

### Namespaces

- namespace [spot](#)

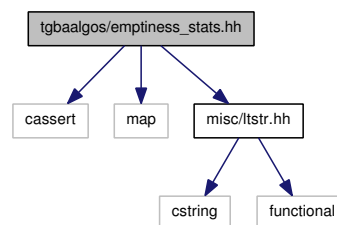
## Functions

- `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).*

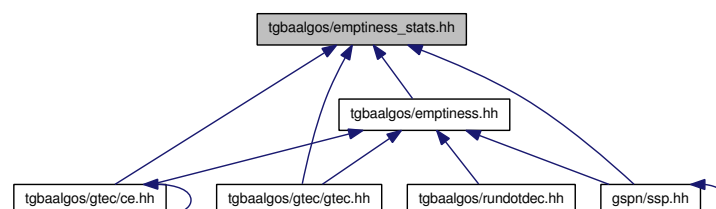
## 8.122 tgbaalgos/emptiness\_stats.hh File Reference

```
#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:



## 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.*

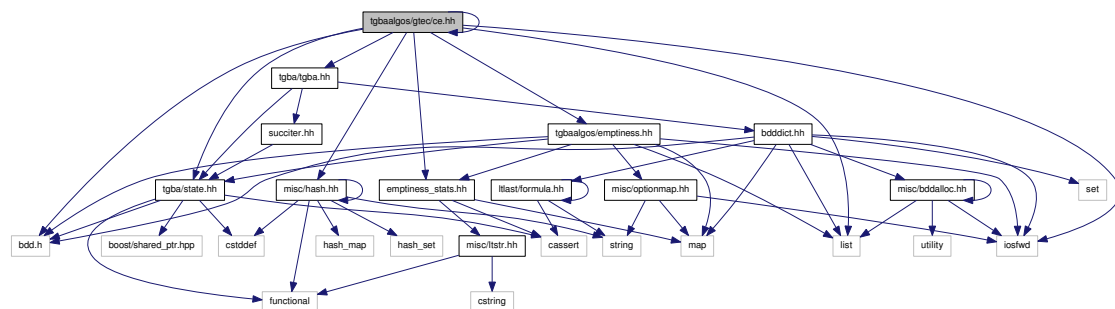
## Namespaces

- namespace [spot](#)

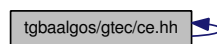
## 8.123 tgbaalgos/gtec/ce.hh File Reference

```
#include "status.hh"
#include <bdd.h>
#include <list>
#include <tgba/state.hh>
#include "misc/hash.hh"
#include "tgba/tgba.hh"
#include <iosfwd>
#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:



## Classes

- class [spot::couvreur99\\_check\\_result](#)  
Compute a counter example from a [spot::couvreur99\\_check\\_status](#).

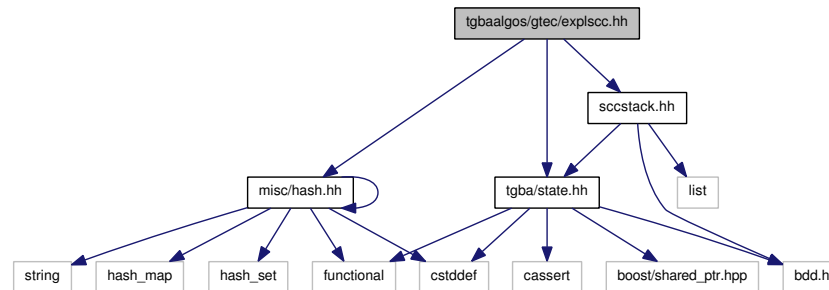
## Namespaces

- namespace [spot](#)

## 8.124 tgbaalgos/gtec/explsc.h File Reference

```
#include "misc/hash.hh"
#include "tgba/state.hh"
#include "sccstack.hh"
```

Include dependency graph for explsc.h:



### Classes

- class [spot::explicit\\_connected\\_component](#)  
*An SCC storing all its states explicitly.*
- class [spot::connected\\_component\\_hash\\_set](#)
- class [spot::explicit\\_connected\\_component\\_factory](#)  
*Abstract factory for [explicit\\_connected\\_component](#).*
- class [spot::connected\\_component\\_hash\\_set\\_factory](#)  
*Factory for [connected\\_component\\_hash\\_set](#).*

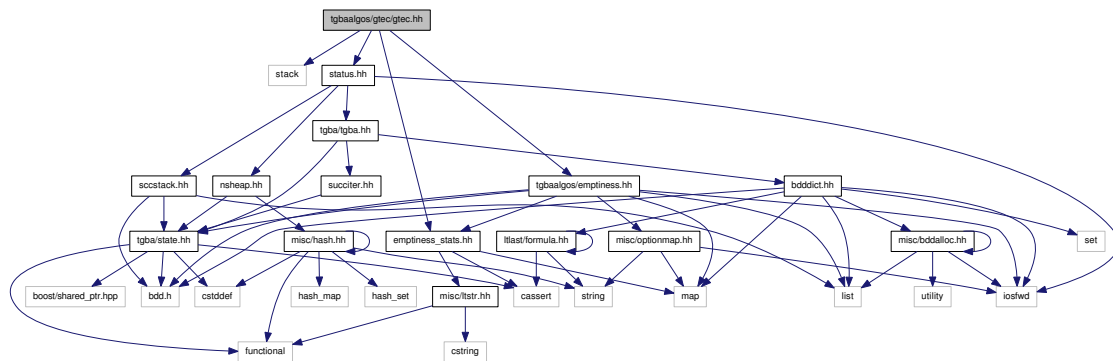
### Namespaces

- namespace [spot](#)

## 8.125 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.hh:



## 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.*
- struct [spot::couvreur99\\_check\\_shy::successor](#)
- struct [spot::couvreur99\\_check\\_shy::todo\\_item](#)

## Namespaces

- namespace [spot](#)

## Functions

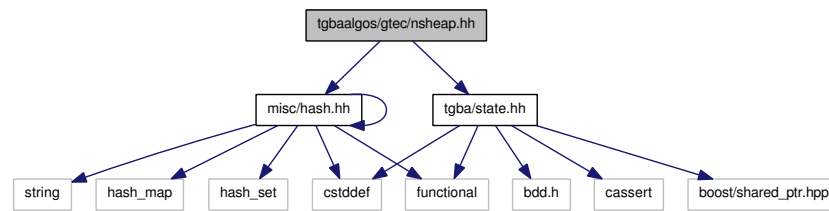
- `emptiness_check * spot::couvreur99 (const tgba *a, option_map options=option_map(), const numbered_state_heap_factory *nshf=numbered_state_heap_hash_map_factory::instance())`  
*Check whether the language of an automate is empty.*

## 8.126 tgbaalgos/gtec/nsheap.hh File Reference

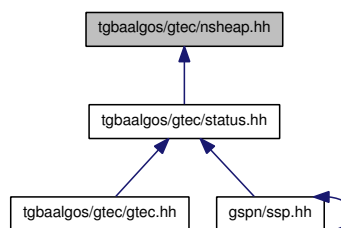
```
#include "tgba/state.hh"
#include "misc/hash.hh"
```



Include dependency graph for nsheap.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::numbered\\_state\\_heap\\_const\\_iterator](#)  
*Iterator on [numbered\\_state\\_heap](#) objects.*
- class [spot::numbered\\_state\\_heap](#)  
*Keep track of a large quantity of indexed states.*
- class [spot::numbered\\_state\\_heap\\_factory](#)  
*Abstract factory for [numbered\\_state\\_heap](#).*
- class [spot::numbered\\_state\\_heap\\_hash\\_map](#)  
*A straightforward implementation of [numbered\\_state\\_heap](#) with a hash map.*
- class [spot::numbered\\_state\\_heap\\_hash\\_map\\_factory](#)  
*Factory for [numbered\\_state\\_heap\\_hash\\_map](#).*

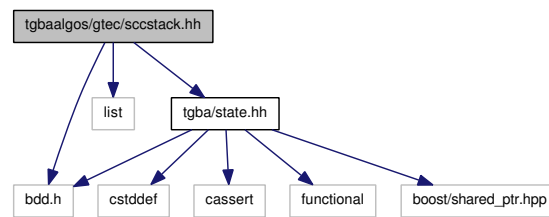
## Namespaces

- namespace [spot](#)

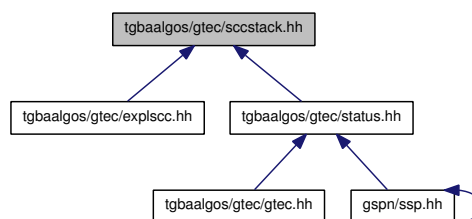
## 8.127 tgbaalgos/gtec/sccstack.hh File Reference

```
#include <bdd.h>
#include <list>
#include <tgba/state.hh>
```

Include dependency graph for sccstack.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- class [spot::scc\\_stack](#)
- struct [spot::scc\\_stack::connected\\_component](#)

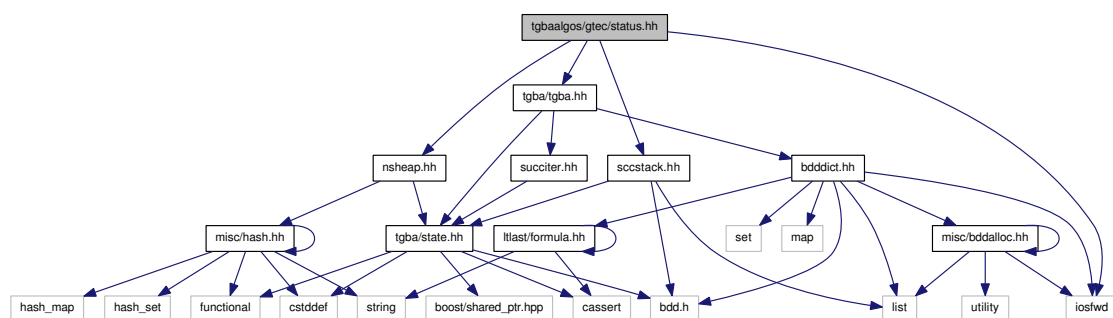
## Namespaces

- namespace [spot](#)

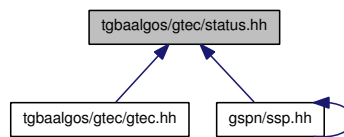
## 8.128 tgbaalgos/gtec/status.hh File Reference

```
#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:



## Classes

- class [spot::couvreur99\\_check\\_status](#)  
*The status of the emptiness-check on success.*

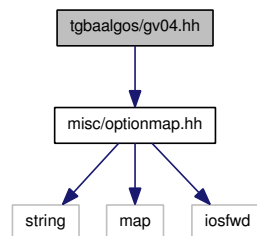
## Namespaces

- namespace [spot](#)

## 8.129 tgbaalgos/gv04.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for gv04.hh:



## Namespaces

- namespace [spot](#)

## Functions

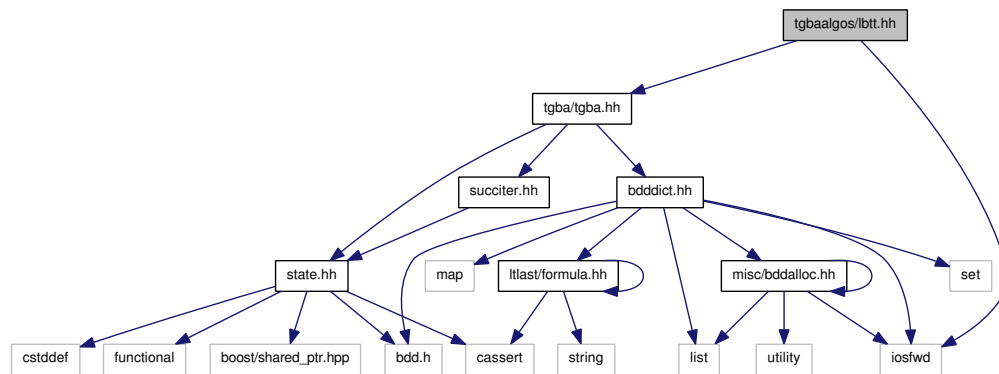
- emptiness\_check \* [spot::explicit\\_gv04\\_check](#) (const tgba \*a, option\_map o=option\_map())  
*Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.*

## 8.130 tgbaalgos/lbtt.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for lbt.h:



## Namespaces

- namespace [spot](#)

## Functions

- `std::ostream & spot::lbt\_reachable (std::ostream &os, const tgba *g)`  
*Print reachable states in LBT format.*

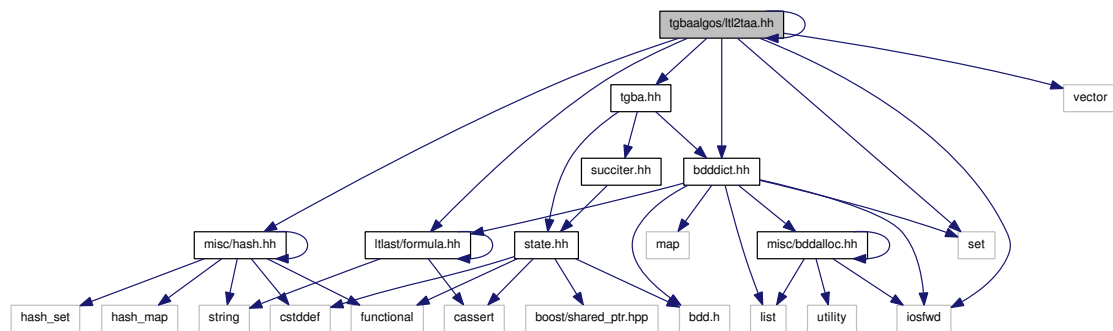
## 8.131 tgbaalgos/ltl2taa.hh File Reference

```

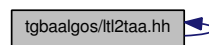
#include "ltlast/formula.hh"
#include "tgba/taatgba.hh"
#include <set>
#include <iosfwd>
#include <vector>
#include "misc/hash.hh"
#include "bdddict.hh"
#include "tgba.hh"

```

Include dependency graph for ltl2taa.hh:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [spot](#)

## Functions

- taa\_tgba \* [spot::ltl\\_to\\_taa](#) (const ltl::formula \*f, bdd\_dict \*dict, bool refined\_rules=false)  
*Build a spot::taa\* from an LTL formula.*  
*This is based on the following.*

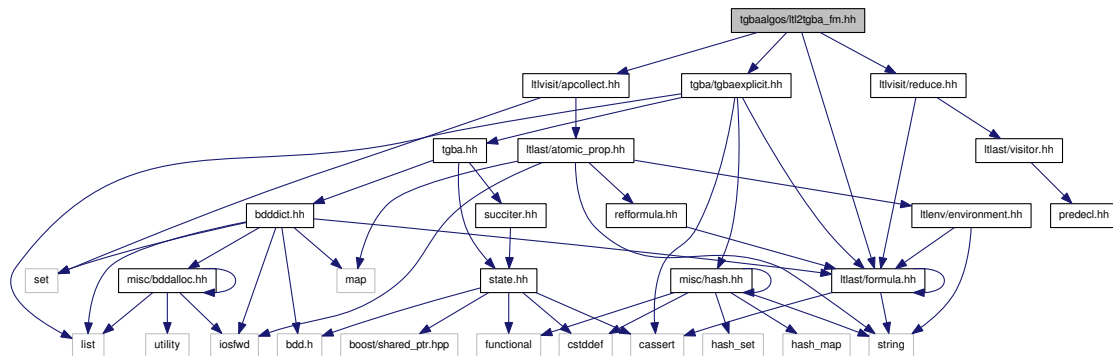
## 8.132 tgbaalgos/ltl2tgba\_fm.hh File Reference

```

#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 * 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.*

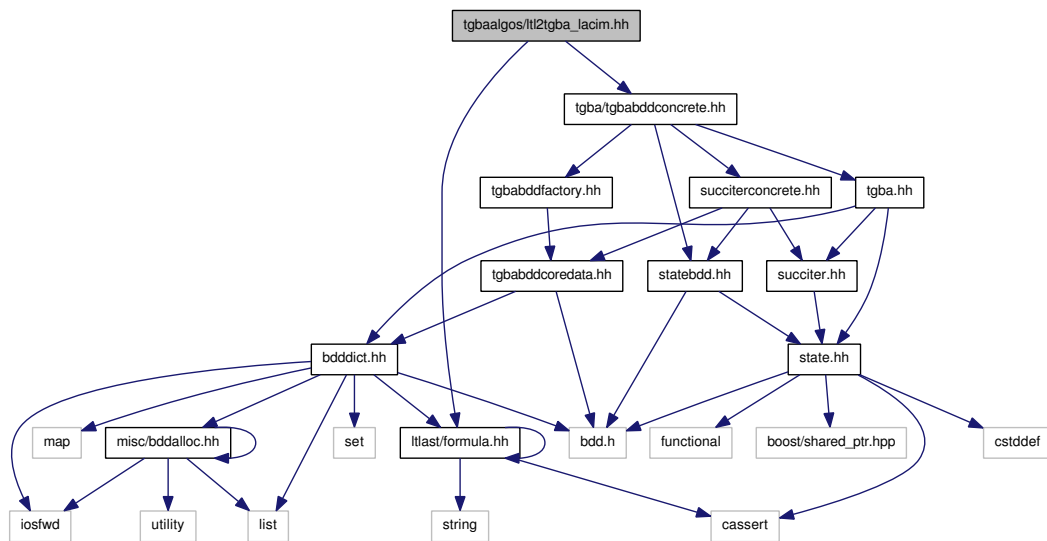
*This is based on the following paper.*

## 8.133 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

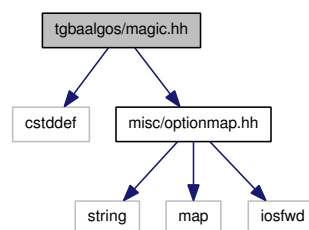
- `tgba_bdd_concrete * spot::ltl_to_tgba_lacim (const ltl::formula *f, bdd_dict *dict)`  
*Build a `spot::tgba_bdd_concrete` from an LTL formula.*  
*This is based on the following paper.*

## 8.134 tgbaalgos/magic.hh File Reference

```
#include <cstdddef>
```

```
#include "misc/optionmap.hh"
```

Include dependency graph for `magic.hh`:



## Namespaces

- namespace [spot](#)

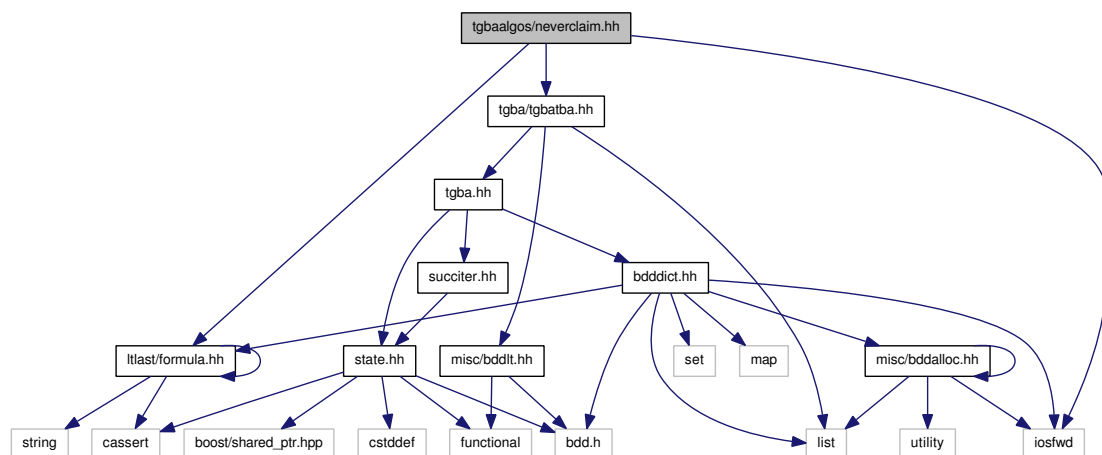
## Functions

- 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.*

## 8.135 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

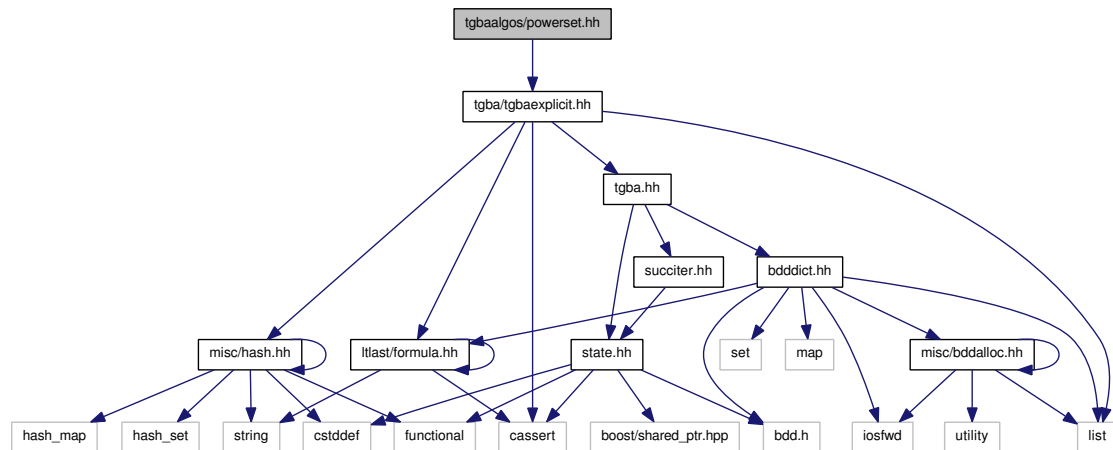
- std::ostream & [spot::never\\_claim\\_reachable](#) (std::ostream &os, const tgba\_sba\_proxy \*g, const ltlast::formula \*f=0, bool comments=false)  
*Print reachable states in Spin never claim format.*



## 8.136 tgbaalgos/powerset.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for powerset.hh:



### Namespaces

- namespace [spot](#)

### Functions

- tgba\_explicit \* [spot::tgba\\_powerset](#) (const tgba \*aut)

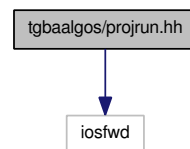
*Build a deterministic automaton, ignoring acceptance conditions.*

*This create a deterministic automaton that recognize the same language as aut would if its acceptance conditions were ignored. This is the classical powerset algorithm.*

## 8.137 tgbaalgos/projrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for projrun.hh:



### Namespaces

- namespace [spot](#)

**Functions**

- `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`.*

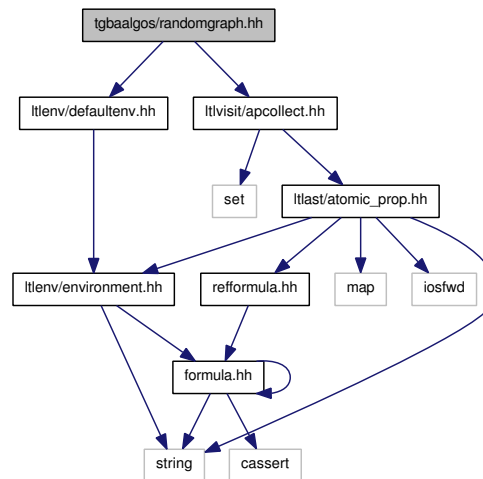
*If a `tgba_run` has been generated on a product, or any other on-the-fly algorithm with `tgba` operands,.*

**8.138 tgbaalgos/randomgraph.hh File Reference**

```
#include "ltlvisit/apcollect.hh"
```

```
#include "ltlenv/defaultenv.hh"
```

Include dependency graph for randomgraph.hh:

**Namespaces**

- namespace `spot`

**Functions**

- `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.*

**8.139 tgbaalgos/reducerun.hh File Reference****Namespaces**

- namespace `spot`

## Functions

- `tgba_run * spot::reduce_run (const tgba *a, const tgba_run *org)`

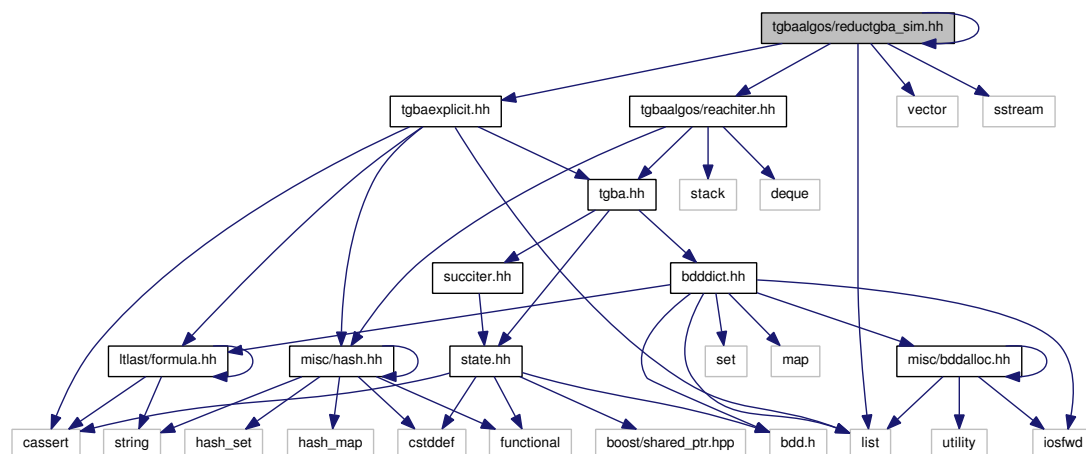
*Reduce an accepting run.*

*Return a run which is accepting for and that is no longer that org.*

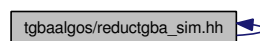
## 8.140 tgbaalgos/reductgba\_sim.hh File Reference

```
#include "tgba/tgbareduc.hh"
#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#include <list>
#include <vector>
#include <sstream>
```

Include dependency graph for reductgba\_sim.hh:



This graph shows which files directly or indirectly include this file:



## 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](#)

## Namespaces

- namespace [spot](#)

## 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.*

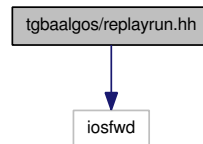
## Functions

- const 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.*

## 8.141 tgbaalgos/replayrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for replayrun.hh:



### Namespaces

- namespace [spot](#)

### Functions

- 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](#).

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.

## 8.142 tgbaalgos/rundotdec.hh File Reference

```
#include <map>
```

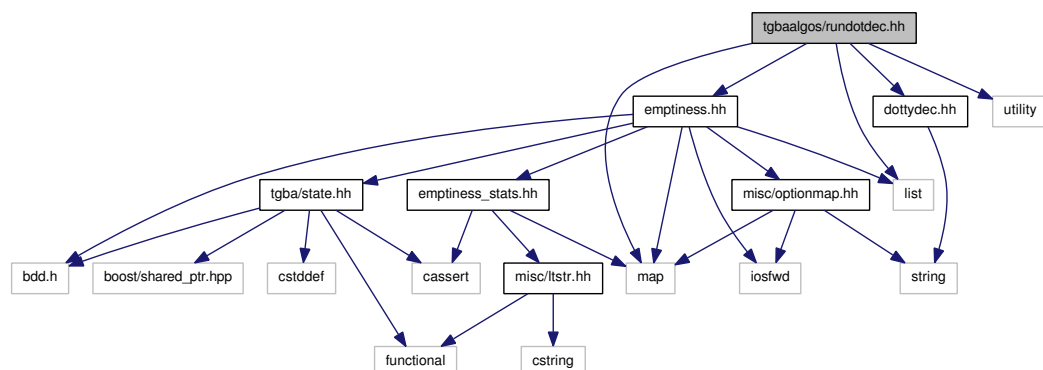
```
#include <utility>
```

```
#include <list>
```

```
#include "dottydec.hh"
```

```
#include "emptiness.hh"
```

Include dependency graph for rundotdec.hh:



## Classes

- class [spot::tgba\\_run\\_dotty\\_decorator](#)

*Highlight a [spot::tgba\\_run](#) on a [spot::tgba](#).*

*An instance of this class can be passed to [spot::dotty\\_reachable](#).*

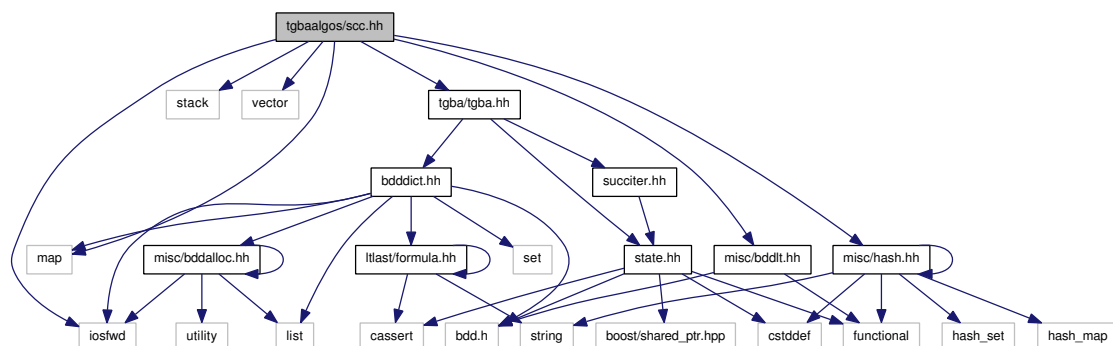
## Namespaces

- namespace [spot](#)

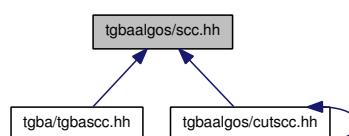
## 8.143 tgbaalgos/scc.hh File Reference

```
#include <map>
#include <stack>
#include <vector>
#include "tgba/tgba.hh"
#include <iosfwd>
#include "misc/hash.hh"
#include "misc/bddlt.hh"
```

Include dependency graph for scc.hh:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [spot::scc\\_stats](#)
- class [spot::scc\\_map](#)

*Build a map of Strongly Connected components in in a TGBA.*

- struct `spot::scc_map::scc`

## Namespaces

- namespace `spot`

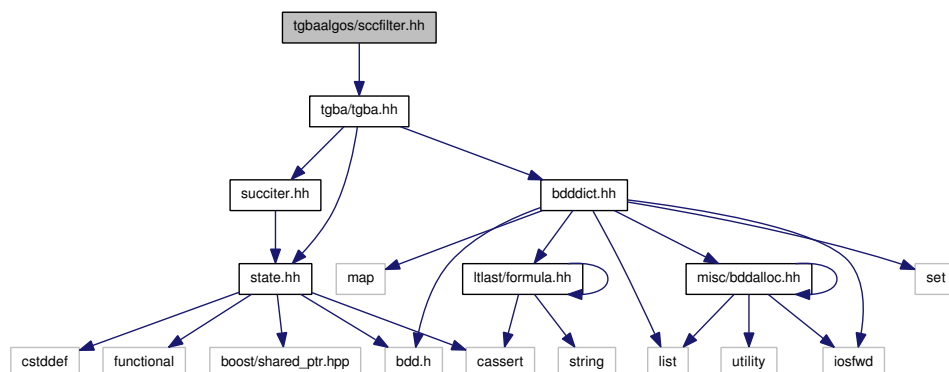
## Functions

- `scc_stats` `spot::build_scc_stats` (const tgba \*a)
- `scc_stats` `spot::build_scc_stats` (const scc\_map &m)
- `std::ostream &` `spot::dump_scc_dot` (const tgba \*a, `std::ostream &`out, bool verbose=false)
- `std::ostream &` `spot::dump_scc_dot` (const scc\_map &m, `std::ostream &`out, bool verbose=false)

## 8.144 tgbaalgorithms/sccfilter.hh File Reference

```
#include "tgba/tgba.hh"
```

Include dependency graph for sccfilter.hh:



## Namespaces

- namespace `spot`

## Functions

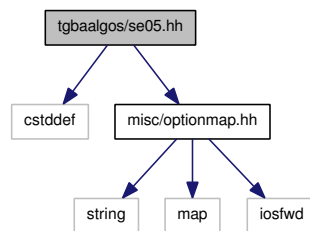
- `tgba *` `spot::scc_filter` (const tgba \*aut, bool remove\_all\_useless=false)  
*Prune unaccepting SCCs and remove superfluous acceptance conditions.*

## 8.145 tgbaalgorithms/se05.hh File Reference

```
#include <cstdint>
```

```
#include "misc/optionmap.hh"
```

Include dependency graph for se05.hh:



## Namespaces

- namespace [spot](#)

## Functions

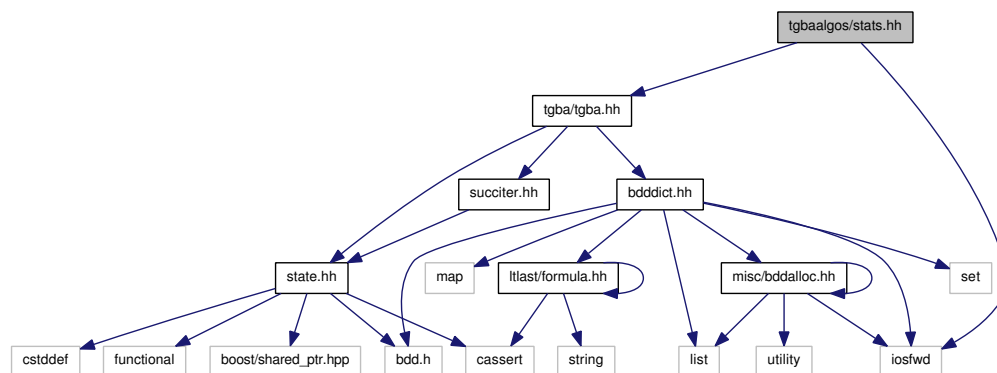
- emptiness\_check \* [spot::explicit\\_se05\\_search](#) (const tgba \*a, option\_map o=option\_map())  
*Returns an emptiness check on the [spot::tgba](#) automaton a.*
- emptiness\_check \* [spot::bit\\_state\\_hashing\\_se05\\_search](#) (const tgba \*a, size\_t size, option\_map o=option\_map())  
*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.*

## 8.146 tgbaalgos/stats.hh File Reference

```
#include "tgba/tgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for stats.hh:





## Classes

- struct [spot::tgba\\_statistics](#)

## Namespaces

- namespace [spot](#)

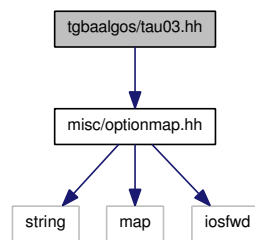
## Functions

- tgba\_statistics [spot::stats\\_reachable](#) (const tgba \*g)  
*Compute statistics for an automaton.*

## 8.147 tgbaalgos/tau03.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for tau03.hh:



## Namespaces

- namespace [spot](#)

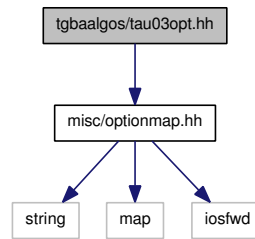
## Functions

- emptiness\_check \* [spot::explicit\\_tau03\\_search](#) (const tgba \*a, option\_map o=option\_map())  
*Returns an emptiness checker on the [spot::tgba](#) automaton a.*

## 8.148 tgbaalgos/tau03opt.hh File Reference

```
#include "misc/optionmap.hh"
```

Include dependency graph for tau03opt.hh:



## Namespaces

- namespace [spot](#)

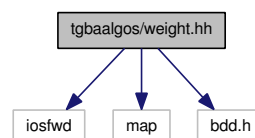
## Functions

- `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.*

## 8.149 tgbaalgos/weight.hh File Reference

```
#include <iosfwd>
#include <map>
#include <bdd.h>
```

Include dependency graph for weight.hh:



## Classes

- class [spot::weight](#)  
*Manage for a given automaton a vector of counter indexed by its acceptance condition.*

## Namespaces

- namespace [spot](#)

## Index

- ~acss\_statistics
  - spot::acss\_statistics, [101](#)
- ~atomic\_prop
  - spot::ltl::atomic\_prop, [115](#)
- ~automatop
  - spot::ltl::automatop, [122](#)
- ~bdd\_dict
  - spot::bdd\_dict, [138](#)
- ~bfs\_steps
  - spot::bfs\_steps, [146](#)
- ~binop
  - spot::ltl::binop, [152](#)
- ~clone\_visitor
  - spot::ltl::clone\_visitor, [158](#)
- ~connected\_component\_hash\_set
  - spot::connected\_component\_hash\_set, [164](#)
- ~connected\_component\_hash\_set\_factory
  - spot::connected\_component\_hash\_set\_factory, [167](#)
- ~const\_visitor
  - spot::ltl::const\_visitor, [168](#)
- ~constant
  - spot::ltl::constant, [173](#)
- ~couvreur99\_check
  - spot::couvreur99\_check, [179](#)
- ~couvreur99\_check\_shy
  - spot::couvreur99\_check\_shy, [194](#)
- ~couvreur99\_check\_status
  - spot::couvreur99\_check\_status, [201](#)
- ~declarative\_environment
  - spot::ltl::declarative\_environment, [205](#)
- ~default\_environment
  - spot::ltl::default\_environment, [209](#)
- ~dotty\_decorator
  - spot::dotty\_decorator, [212](#)
- ~duplicator\_node
  - spot::duplicator\_node, [216](#)
- ~duplicator\_node\_delayed
  - spot::duplicator\_node\_delayed, [221](#)
- ~emptiness\_check
  - spot::emptiness\_check, [232](#)
- ~emptiness\_check\_result
  - spot::emptiness\_check\_result, [240](#)
- ~environment
  - spot::ltl::environment, [243](#)
- ~evtgba
  - spot::evtgba, [245](#)
- ~evtgba\_explicit
  - spot::evtgba\_explicit, [250](#)
- ~evtgba\_iterator
  - spot::evtgba\_iterator, [252](#)
- ~evtgba\_product
  - spot::evtgba\_product, [256](#)
- ~evtgba\_reachable\_iterator
  - spot::evtgba\_reachable\_iterator, [261](#)
- ~explicit\_connected\_component
  - spot::explicit\_connected\_component, [276](#)
- ~explicit\_connected\_component\_factory
  - spot::explicit\_connected\_component\_factory, [277](#)
- ~explicit\_state\_conjunction
  - spot::explicit\_state\_conjunction, [281](#)
- ~fair\_kripke\_succ\_iterator
  - spot::fair\_kripke\_succ\_iterator, [293](#)
- ~formula
  - spot::ltl::formula, [297](#)
- ~free\_list
  - spot::free\_list, [303](#)
- ~future\_conditions\_collector
  - spot::future\_conditions\_collector, [308](#)
- ~gspn\_interface
  - spot::gspn\_interface, [316](#)
- ~gspn\_ssp\_interface
  - spot::gspn\_ssp\_interface, [319](#)
- ~kripke
  - spot::kripke, [322](#)
- ~language\_containment\_checker
  - spot::ltl::language\_containment\_checker, [328](#)
- ~loopless\_modular\_mixed\_radix\_gray\_code
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, [343](#)
- ~multop
  - spot::ltl::multop, [353](#)
- ~nfa
  - spot::ltl::nfa, [359](#)
- ~nips\_interface
  - spot::nips\_interface, [365](#)
- ~node
  - spot::ltl::formula\_tree::node, [366](#)
- ~numbered\_state\_heap
  - spot::numbered\_state\_heap, [376](#)
- ~numbered\_state\_heap\_const\_iterator
  - spot::numbered\_state\_heap\_const\_iterator, [378](#)
- ~numbered\_state\_heap\_factory
  - spot::numbered\_state\_heap\_factory, [380](#)
- ~numbered\_state\_heap\_hash\_map
  - spot::numbered\_state\_heap\_hash\_map, [383](#)
- ~numbered\_state\_heap\_hash\_map\_factory
  - spot::numbered\_state\_heap\_hash\_map\_factory, [383](#)

- factory, 387
- ~parity\_game\_graph
  - spot::parity\_game\_graph, 396
- ~parity\_game\_graph\_delayed
  - spot::parity\_game\_graph\_delayed, 403
- ~parity\_game\_graph\_direct
  - spot::parity\_game\_graph\_direct, 411
- ~postfix\_visitor
  - spot::ltl::postfix\_visitor, 423
- ~random\_ltl
  - spot::ltl::random\_ltl, 427
- ~ref\_formula
  - spot::ltl::ref\_formula, 434
- ~rsymbol
  - spot::rsymbol, 438
- ~saba
  - spot::saba, 441
- ~saba\_complement\_tgba
  - spot::saba\_complement\_tgba, 446
- ~saba\_reachable\_iterator
  - spot::saba\_reachable\_iterator, 450
- ~saba\_state
  - spot::saba\_state, 463
- ~saba\_state\_conjunction
  - spot::saba\_state\_conjunction, 466
- ~saba\_succ\_iterator
  - spot::saba\_succ\_iterator, 472
- ~scc\_map
  - spot::scc\_map, 479
- ~simplify\_f\_g\_visitor
  - spot::ltl::simplify\_f\_g\_visitor, 490
- ~spoiler\_node
  - spot::spoiler\_node, 497
- ~spoiler\_node\_delayed
  - spot::spoiler\_node\_delayed, 502
- ~state
  - spot::state, 513
- ~state\_evtgba\_explicit
  - spot::state\_evtgba\_explicit, 521
- ~state\_explicit
  - spot::state\_explicit, 525
- ~state\_product
  - spot::state\_product, 529
- ~state\_set
  - spot::state\_set, 536
- ~state\_union
  - spot::state\_union, 542
- ~symbol
  - spot::symbol, 549
- ~taa\_succ\_iterator
  - spot::taa\_succ\_iterator, 554
- ~taa\_tgba
  - spot::taa\_tgba, 561
- ~taa\_tgba\_formula
  - spot::taa\_tgba\_formula, 570
- ~taa\_tgba\_string
  - spot::taa\_tgba\_string, 592
- ~tgba
  - spot::tgba, 602
- ~tgba\_bdd\_concrete
  - spot::tgba\_bdd\_concrete, 610
- ~tgba\_bdd\_concrete\_factory
  - spot::tgba\_bdd\_concrete\_factory, 618
- ~tgba\_bdd\_factory
  - spot::tgba\_bdd\_factory, 628
- ~tgba\_explicit
  - spot::tgba\_explicit, 633
- ~tgba\_explicit\_formula
  - spot::tgba\_explicit\_formula, 643
- ~tgba\_explicit\_labelled
  - spot::tgba\_explicit\_labelled, 655
- ~tgba\_explicit\_string
  - spot::tgba\_explicit\_string, 667
- ~tgba\_kv\_complement
  - spot::tgba\_kv\_complement, 683
- ~tgba\_product
  - spot::tgba\_product, 691
- ~tgba\_reachable\_iterator
  - spot::tgba\_reachable\_iterator, 698
- ~tgba\_reduc
  - spot::tgba\_reduc, 717
- ~tgba\_run
  - spot::tgba\_run, 729
- ~tgba\_run\_dotty\_decorator
  - spot::tgba\_run\_dotty\_decorator, 732
- ~tgba\_safra\_complement
  - spot::tgba\_safra\_complement, 737
- ~tgba\_scc
  - spot::tgba\_scc, 751
- ~tgba\_sgba\_proxy
  - spot::tgba\_sgba\_proxy, 758
- ~tgba\_succ\_iterator
  - spot::tgba\_succ\_iterator, 764
- ~tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 768
- ~tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 773
- ~tgba\_succ\_iterator\_union
  - spot::tgba\_succ\_iterator\_union, 779
- ~tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 785
- ~tgba\_union
  - spot::tgba\_union, 793
- ~unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 815
- ~unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 819
- ~unop

- spot::ltl::unop, 825
- ~unsigned\_statistics
  - spot::unsigned\_statistics, 829
- ~visitor
  - spot::ltl::visitor, 833
- a\_
  - spot::bfs\_steps, 147
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 198
  - spot::emptiness\_check, 234
  - spot::emptiness\_check\_result, 241
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
  - spot::tgba\_sba\_proxy, 747
  - spot::tgba\_sgba\_proxy, 761
  - spot::tgba\_tba\_proxy, 789
- a\_first
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 343
- a\_last
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 343
- a\_next
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 343
- acc
  - spot::couvreur99\_check\_shy::successor, 548
  - spot::scc\_map::scc, 474
  - spot::tgba\_run::step, 545
- acc\_
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::tgba\_bdd\_concrete\_factory, 620
- acc\_cond
  - spot::fair\_kripke\_succ\_iterator, 295
- acc\_cycle\_
  - spot::tgba\_sba\_proxy, 747
  - spot::tgba\_tba\_proxy, 789
- acc\_formula\_map
  - spot::bdd\_dict, 141
- acc\_list\_
  - spot::tgba\_kv\_complement, 686
- acc\_list\_t
  - spot, 84
- acc\_map
  - spot::bdd\_dict, 141
- acc\_map\_
  - spot::tgba\_bdd\_concrete\_factory, 618
- acc\_paths
  - spot::scc\_stats, 485
- acc\_scc
  - spot::scc\_stats, 485
- acc\_set
  - spot::tgba\_bdd\_core\_data, 624
- acc\_set\_
  - spot::evtgba\_explicit, 252
- acc\_set\_of
  - spot::scc\_map, 479
- accept
  - spot::ltl::atomic\_prop, 115
  - spot::ltl::automatop, 122
  - spot::ltl::binop, 152
  - spot::ltl::constant, 173
  - spot::ltl::formula, 298
  - spot::ltl::multop, 353
  - spot::ltl::ref\_formula, 435
  - spot::ltl::unop, 825
- acceptance\_cond\_vec\_
  - spot::tgba\_safra\_complement, 740
- acceptance\_condition\_
  - spot::tgba\_sgba\_proxy, 761
- acceptance\_condition\_visited\_
  - spot::spoiler\_node\_delayed, 504
- acceptance\_conditions
  - spot::evtgba\_explicit::transition, 812
  - spot::saba\_state, 463
  - spot::taa\_tgba::transition, 809
  - spot::tgba\_bdd\_core\_data, 624
  - spot::tgba\_explicit::transition, 807
- acceptance\_conditions\_of\_state
  - spot::fair\_kripke, 286
  - spot::kripke, 322
- accepting
  - spot::scc\_map, 479
- accepting\_cycle
  - spot::couvreur99\_check\_result, 187
- accepting\_run
  - spot::couvreur99\_check\_result, 187
  - spot::emptiness\_check\_result, 240
- acss\_states
  - spot::acss\_statistics, 101
  - spot::couvreur99\_check\_result, 187
- acss\_statistics
  - spot::acss\_statistics, 101
- add
  - spot::explicit\_state\_conjunction, 282
- add\_acceptance\_condition
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 592
  - spot::tgba\_explicit, 633
  - spot::tgba\_explicit\_formula, 643
  - spot::tgba\_explicit\_labelled, 655
  - spot::tgba\_explicit\_string, 667
  - spot::tgba\_reduc, 717
- add\_acceptance\_conditions

- spot::tgba\_explicit, 633
- spot::tgba\_explicit\_formula, 643
- spot::tgba\_explicit\_labelled, 655
- spot::tgba\_explicit\_string, 667
- spot::tgba\_reduc, 717
- add\_condition
  - spot::taa\_tgba, 561
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 592
  - spot::tgba\_explicit, 633
  - spot::tgba\_explicit\_formula, 643
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 667
  - spot::tgba\_reduc, 717
- add\_conditions
  - spot::tgba\_explicit, 633
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_reduc, 717
- add\_default\_init
  - spot::tgba\_explicit, 633
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_reduc, 717
- add\_duplicator\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 404
- add\_pred
  - spot::duplicator\_node, 216
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 497
  - spot::spoiler\_node\_delayed, 502
- add\_spoiler\_node\_delayed
  - spot::parity\_game\_graph\_delayed, 404
- add\_state
  - spot::evtgba\_reachable\_iterator, 262
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 266
  - spot::evtgba\_reachable\_iterator\_depth\_first, 271
  - spot::ltl::nfa, 360
  - spot::parity\_game\_graph, 396
  - spot::parity\_game\_graph\_delayed, 404
  - spot::parity\_game\_graph\_direct, 411
  - spot::saba\_reachable\_iterator, 451
  - spot::saba\_reachable\_iterator\_breadth\_first, 455
  - spot::saba\_reachable\_iterator\_depth\_first, 460
  - spot::taa\_tgba\_labelled, 582
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_reachable\_iterator, 699
  - spot::tgba\_reachable\_iterator\_breadth\_first, 703
  - spot::tgba\_reachable\_iterator\_depth\_first, 708
  - spot::tgba\_reduc, 717
- add\_state\_set
  - spot::taa\_tgba\_labelled, 582
- add\_succ
  - spot::duplicator\_node, 216
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 497
  - spot::spoiler\_node\_delayed, 502
- add\_transition
  - spot::evtgba\_explicit, 250
  - spot::ltl::nfa, 360
- Algorithm patterns, 33
- Algorithms for LTL formulae, 12
- all\_acc\_
  - spot::evtgba\_product, 258
- all\_acceptance\_cond\_
  - spot::tgba\_safra\_complement, 740
- all\_acceptance\_conditions
  - spot::evtgba, 245
  - spot::evtgba\_explicit, 250
  - spot::evtgba\_product, 256
  - spot::fair\_kripke, 286
  - spot::future\_conditions\_collector, 308
  - spot::kripke, 323
  - spot::saba, 441
  - spot::saba\_complement\_tgba, 446
  - spot::taa\_tgba, 561
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 582
  - spot::taa\_tgba\_string, 592
  - spot::tgba, 602
  - spot::tgba\_bdd\_concrete, 611
  - spot::tgba\_bdd\_core\_data, 625
  - spot::tgba\_explicit, 633
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_kv\_complement, 683
  - spot::tgba\_product, 691
  - spot::tgba\_reduc, 718
  - spot::tgba\_safra\_complement, 737
  - spot::tgba\_sba\_proxy, 744
  - spot::tgba\_scc, 751
  - spot::tgba\_sgba\_proxy, 758
  - spot::tgba\_tba\_proxy, 785
  - spot::tgba\_union, 794
- all\_acceptance\_conditions\_
  - spot::taa\_succ\_iterator, 555
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 575

- spot::taa\_tgba\_labelled, 587
- spot::taa\_tgba\_string, 597
- spot::tgba\_explicit, 637
- spot::tgba\_explicit\_formula, 649
- spot::tgba\_explicit\_labelled, 661
- spot::tgba\_explicit\_string, 673
- spot::tgba\_explicit\_succ\_iterator, 678
- spot::tgba\_product, 695
- spot::tgba\_reduc, 726
- spot::tgba\_union, 797
- all\_acceptance\_conditions\_computed\_
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_explicit, 637
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 661
  - spot::tgba\_explicit\_string, 673
  - spot::tgba\_reduc, 726
- allocate\_variables
  - spot::bdd\_allocator, 130
  - spot::bdd\_dict, 138
- alphabet
  - spot::evtgba, 245
  - spot::evtgba\_explicit, 250
  - spot::evtgba\_product, 256
- alphabet\_
  - spot::evtgba\_explicit, 252
  - spot::evtgba\_product, 258
- And
  - spot::ltl::multop, 353
- anon\_free\_list
  - spot::bdd\_dict::anon\_free\_list, 105
- ap
  - spot::ltl::random\_ltl, 427
- ap\_
  - spot::ltl::random\_ltl, 429
- ap\_set\_of
  - spot::scc\_map, 479
- aprec\_set\_of
  - spot::scc\_map, 479
- arc
  - spot::couvereur99\_check\_shy, 198
- arc\_acc\_
  - spot::scc\_map, 482
- arc\_cond\_
  - spot::scc\_map, 482
- arity
  - spot::ltl::formula\_tree, 98
  - spot::ltl::nfa, 360
- arity\_
  - spot::ltl::nfa, 361
- ars\_cycle\_states
  - spot::acss\_statistics, 101
  - spot::ars\_statistics, 110
  - spot::couvereur99\_check\_result, 187
- ars\_prefix\_states
  - spot::acss\_statistics, 102
  - spot::ars\_statistics, 110
  - spot::couvereur99\_check\_result, 188
- ars\_statistics
  - spot::ars\_statistics, 110
- as\_bdd
  - spot::state\_bdd, 517
- assert\_emptiness
  - spot::bdd\_dict, 138
- atomic\_prop
  - spot::ltl::atomic\_prop, 115
- atomic\_prop\_collect
  - ltl\_misc, 16
- atomic\_prop\_set
  - ltl\_misc, 15
- aut
  - spot::couvereur99\_check\_status, 201
- aut\_
  - spot::future\_conditions\_collector, 312
  - spot::scc\_map, 482
  - spot::tgba\_scc, 754
- automata\_
  - spot::evtgba\_reachable\_iterator, 263
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 268
  - spot::evtgba\_reachable\_iterator\_depth\_first, 273
  - spot::parity\_game\_graph, 398
  - spot::parity\_game\_graph\_delayed, 406
  - spot::parity\_game\_graph\_direct, 413
  - spot::saba\_reachable\_iterator, 452
  - spot::saba\_reachable\_iterator\_breadth\_first, 457
  - spot::saba\_reachable\_iterator\_depth\_first, 462
  - spot::tgba\_reachable\_iterator, 700
  - spot::tgba\_reachable\_iterator\_breadth\_first, 705
  - spot::tgba\_reachable\_iterator\_depth\_first, 710
  - spot::tgba\_reduc, 727
- automaton
  - spot::couvereur99\_check, 180
  - spot::couvereur99\_check\_result, 188
  - spot::couvereur99\_check\_shy, 194
  - spot::emptiness\_check, 232
  - spot::emptiness\_check\_result, 241
  - spot::gspn\_interface, 316
  - spot::gspn\_ssp\_interface, 319
  - spot::nips\_interface, 365
- automaton\_
  - spot::saba\_complement\_tgba, 447

- spot::tgba\_kv\_complement, 686
- spot::tgba\_safracomplement, 740
- automatop
  - spot::ltl::automatop, 122
- barand
  - spot::barand, 125
- basic\_reduce
  - ltl\_rewriting, 13
- bdd\_
  - spot::bdd\_ordered, 144
- bdd\_allocator
  - spot::bdd\_allocator, 130
- bdd\_dict
  - spot::bdd\_dict, 138
- bdd\_format\_accset
  - spot, 85
- bdd\_format\_formula
  - spot, 85
- bdd\_format\_sat
  - spot, 85
- bdd\_format\_set
  - spot, 85
- bdd\_ordered
  - spot::bdd\_ordered, 144
- bdd\_print\_acc
  - spot, 86
- bdd\_print\_accset
  - spot, 86
- bdd\_print\_dot
  - spot, 86
- bdd\_print\_formula
  - spot, 87
- bdd\_print\_sat
  - spot, 87
- bdd\_print\_set
  - spot, 87
- bdd\_print\_table
  - spot, 87
- bdd\_to\_formula
  - spot, 88
- bdd\_v
  - spot::parity\_game\_graph\_delayed, 403
- begin
  - eltly::location, 335
  - eltly::stack, 511
  - ltly::location, 341
  - ltly::stack, 508
  - sauty::location, 338
  - sauty::stack, 506
  - spot::ltl::nfa, 360
- bfs\_steps
  - spot::bfs\_steps, 146
- binop
  - spot::ltl::binop, 152
- bit\_state\_hashing\_magic\_search
  - emptiness\_check\_algorithms, 40
- bit\_state\_hashing\_se05\_search
  - emptiness\_check\_algorithms, 40
- bmrand
  - random, 22
- bounds\_t
  - spot::taa\_succ\_iterator, 553
- branching\_postponement\_
  - spot::ltl::language\_containment\_checker, 329
- build
  - spot::connected\_component\_hash\_set\_factory, 167
  - spot::explicit\_connected\_component\_factory, 278
  - spot::ltl::random\_ltl::op\_proba, 389
  - spot::numbered\_state\_heap\_factory, 380
  - spot::numbered\_state\_heap\_hash\_map\_factory, 387
- build\_graph
  - spot::parity\_game\_graph, 396
  - spot::parity\_game\_graph\_delayed, 404
  - spot::parity\_game\_graph\_direct, 411
- build\_link
  - spot::parity\_game\_graph\_direct, 411
- build\_map
  - spot::scc\_map, 480
- build\_recurse\_successor\_duplicator
  - spot::parity\_game\_graph\_delayed, 404
- build\_recurse\_successor\_spoiler
  - spot::parity\_game\_graph\_delayed, 404
- build\_scc\_stats
  - spot, 88
- builder
  - spot::ltl::random\_ltl::op\_proba, 389
- bytecode\_
  - spot::nips\_interface, 365
- cancel
  - spot::timer\_map, 803
- cc\_map
  - spot::bdd\_dict, 137
- check
  - spot::couvreur99\_check, 180
  - spot::couvreur99\_check\_shy, 194
  - spot::emptiness\_check, 232
- child
  - spot::ltl::formula\_tree::node\_unop, 374
  - spot::ltl::unop, 825
- child\_
  - spot::ltl::unop, 827
- children
  - spot::ltl::formula\_tree::node\_nfa, 372



- children\_
  - spot::ltl::automatop, 124
  - spot::ltl::multop, 356
- clear\_rem
  - spot::scc\_stack, 484
- clear\_todo
  - spot::couvreur99\_check\_shy, 195
- clone
  - ltl\_essential, 10
  - spot::explicit\_state\_conjunction, 282
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 122
  - spot::ltl::binop, 152
  - spot::ltl::constant, 173
  - spot::ltl::formula, 298
  - spot::ltl::multop, 353
  - spot::ltl::ref\_formula, 435
  - spot::ltl::unop, 825
  - spot::saba\_state, 463
  - spot::saba\_state\_conjunction, 466
  - spot::state, 513
  - spot::state\_bdd, 517
  - spot::state\_evtgba\_explicit, 521
  - spot::state\_explicit, 525
  - spot::state\_product, 530
  - spot::state\_set, 536
  - spot::state\_union, 543
- clone\_counts
  - spot::bdd\_dict, 141
- clone\_if
  - spot::taa\_tgba\_formula, 571
  - spot::taa\_tgba\_labelled, 582
  - spot::taa\_tgba\_string, 593
- clone\_visitor
  - spot::ltl::clone\_visitor, 158
- column
  - eltlyy::position, 417
  - ltlyy::position, 419
  - sautyy::position, 415
- columns
  - eltlyy::location, 334
  - eltlyy::position, 417
  - ltlyy::location, 340
  - ltlyy::position, 419
  - sautyy::location, 337
  - sautyy::position, 415
- common\_symbol\_table
  - spot::evtgba\_product, 256
- common\_symbols\_
  - spot::evtgba\_product, 258
- compare
  - spot::duplicator\_node, 216
  - spot::duplicator\_node\_delayed, 222
  - spot::saba\_state, 464
  - spot::spoiler\_node, 497
  - spot::spoiler\_node\_delayed, 503
  - spot::state, 514
  - spot::state\_bdd, 517
  - spot::state\_evtgba\_explicit, 521
  - spot::state\_explicit, 525
  - spot::state\_product, 530
  - spot::state\_set, 536
  - spot::state\_union, 543
- complement\_all\_acceptance\_conditions
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 656
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_reduc, 718
- compute\_all\_acceptance\_conditions
  - spot, 88
- compute\_neg\_acceptance\_conditions
  - spot, 88
- compute\_support\_conditions
  - spot::fair\_kripke, 287
  - spot::future\_conditions\_collector, 309
  - spot::kripke, 323
  - spot::taa\_tgba, 561
  - spot::taa\_tgba\_formula, 571
  - spot::taa\_tgba\_labelled, 582
  - spot::taa\_tgba\_string, 593
  - spot::tgba, 602
  - spot::tgba\_bdd\_concrete, 611
  - spot::tgba\_explicit, 634
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 657
  - spot::tgba\_explicit\_string, 668
  - spot::tgba\_kv\_complement, 683
  - spot::tgba\_product, 692
  - spot::tgba\_reduc, 718
  - spot::tgba\_safra\_complement, 737
  - spot::tgba\_sba\_proxy, 744
  - spot::tgba\_scc, 751
  - spot::tgba\_sgba\_proxy, 758
  - spot::tgba\_tba\_proxy, 785
  - spot::tgba\_union, 794
- compute\_support\_variables
  - spot::fair\_kripke, 287
  - spot::future\_conditions\_collector, 309
  - spot::kripke, 323
  - spot::taa\_tgba, 561
  - spot::taa\_tgba\_formula, 571
  - spot::taa\_tgba\_labelled, 583
  - spot::taa\_tgba\_string, 593
  - spot::tgba, 602
  - spot::tgba\_bdd\_concrete, 611
  - spot::tgba\_explicit, 634
  - spot::tgba\_explicit\_formula, 644
  - spot::tgba\_explicit\_labelled, 657

- spot::tgba\_explicit\_string, 669
- spot::tgba\_kv\_complement, 684
- spot::tgba\_product, 692
- spot::tgba\_reduc, 718
- spot::tgba\_safra\_complement, 737
- spot::tgba\_sba\_proxy, 744
- spot::tgba\_scc, 751
- spot::tgba\_sgba\_proxy, 758
- spot::tgba\_tba\_proxy, 786
- spot::tgba\_union, 794
- cond
  - spot::fair\_kripke\_succ\_iterator, 295
- cond\_set
  - spot::future\_conditions\_collector, 308
  - spot::scc\_map, 478
- cond\_set\_of
  - spot::scc\_map, 480
- condition
  - spot::connected\_component\_hash\_set, 165
  - spot::explicit\_connected\_component, 276
  - spot::scc\_stack::connected\_component, 161
  - spot::taa\_tgba::transition, 809
  - spot::tgba\_explicit::transition, 807
- conditions\_of\_state
  - spot::fair\_kripke, 287
  - spot::kripke, 323
- conds
  - spot::scc\_map::scc, 474
- connected\_component
  - spot::scc\_stack::connected\_component, 161
- connected\_component\_hash\_set\_factory
  - spot::connected\_component\_hash\_set\_factory, 167
- const\_iterator
  - eltly::stack, 510
  - ltly::stack, 508
  - sauty::stack, 506
- constant
  - spot::ltl::constant, 173
- constrain\_relation
  - spot::tgba\_bdd\_concrete\_factory, 618
- construct
  - spot::emptiness\_check\_instantiator, 236
- contained
  - spot::ltl::language\_containment\_checker, 329
- contained\_neg
  - spot::ltl::language\_containment\_checker, 329
- copy\_acceptance\_conditions\_of
  - spot::tgba\_explicit, 634
  - spot::tgba\_explicit\_formula, 645
  - spot::tgba\_explicit\_labelled, 657
  - spot::tgba\_explicit\_string, 669
  - spot::tgba\_reduc, 718
- count\_
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::automatop, 124
  - spot::ltl::binop, 154
  - spot::ltl::constant, 175
  - spot::ltl::formula, 299
  - spot::ltl::multop, 356
  - spot::ltl::ref\_formula, 436
  - spot::ltl::unop, 827
- couvreur99
  - emptiness\_check\_algorithms, 41
- couvreur99\_check
  - spot::couvreur99\_check, 179
- couvreur99\_check\_result
  - spot::couvreur99\_check\_result, 187
- couvreur99\_check\_shy
  - spot::couvreur99\_check\_shy, 194
- couvreur99\_check\_ssp\_semi
  - emptiness\_check\_ssp, 4
- couvreur99\_check\_ssp\_shy
  - emptiness\_check\_ssp, 4
- couvreur99\_check\_ssp\_shy\_semi
  - emptiness\_check\_ssp, 4
- couvreur99\_check\_status
  - spot::couvreur99\_check\_status, 201
- create\_anonymous\_state
  - spot::tgba\_bdd\_concrete\_factory, 618
- create\_atomic\_prop
  - spot::tgba\_bdd\_concrete\_factory, 619
- create\_state
  - spot::tgba\_bdd\_concrete\_factory, 619
- create\_transition
  - spot::taa\_tgba\_formula, 571
  - spot::taa\_tgba\_labelled, 583
  - spot::taa\_tgba\_string, 593
  - spot::tgba\_explicit, 634
  - spot::tgba\_explicit\_formula, 645
  - spot::tgba\_explicit\_labelled, 657
  - spot::tgba\_explicit\_string, 669
  - spot::tgba\_reduc, 719
- cube\_
  - spot::minato\_isop, 348
- current\_
  - spot::tgba\_succ\_iterator\_concrete, 770
- current\_acc\_
  - spot::tgba\_succ\_iterator\_concrete, 770
- current\_acceptance\_conditions
  - spot::evtgba\_iterator, 253
  - spot::fair\_kripke\_succ\_iterator, 293
  - spot::taa\_succ\_iterator, 554
  - spot::tgba\_explicit\_succ\_iterator, 677
  - spot::tgba\_succ\_iterator, 764
  - spot::tgba\_succ\_iterator\_concrete, 769
  - spot::tgba\_succ\_iterator\_product, 774
  - spot::tgba\_succ\_iterator\_union, 779

- current\_cond\_
  - spot::tgba\_succ\_iterator\_product, 775
  - spot::tgba\_succ\_iterator\_union, 781
- current\_condition
  - spot::fair\_kripke\_succ\_iterator, 293
  - spot::saba\_succ\_iterator, 472
  - spot::taa\_succ\_iterator, 554
  - spot::tgba\_explicit\_succ\_iterator, 677
  - spot::tgba\_succ\_iterator, 764
  - spot::tgba\_succ\_iterator\_concrete, 769
  - spot::tgba\_succ\_iterator\_product, 774
  - spot::tgba\_succ\_iterator\_union, 779
- current\_conditions
  - spot::fair\_kripke\_succ\_iterator, 293
- current\_conjunction
  - spot::saba\_succ\_iterator, 472
- current\_label
  - spot::evtgba\_iterator, 253
- current\_state
  - spot::evtgba\_iterator, 253
  - spot::explicit\_state\_conjunction, 282
  - spot::fair\_kripke\_succ\_iterator, 294
  - spot::saba\_state\_conjunction, 466
  - spot::taa\_succ\_iterator, 554
  - spot::tgba\_explicit\_succ\_iterator, 677
  - spot::tgba\_succ\_iterator, 764
  - spot::tgba\_succ\_iterator\_concrete, 769
  - spot::tgba\_succ\_iterator\_product, 774
  - spot::tgba\_succ\_iterator\_union, 780
- current\_state\_
  - spot::tgba\_succ\_iterator\_concrete, 770
- cycle
  - spot::tgba\_run, 729
- cycle\_list
  - spot::tgba\_sba\_proxy, 744
  - spot::tgba\_tba\_proxy, 785
- cycle\_seed
  - spot::couvreur99\_check\_status, 201
- cycle\_start\_
  - spot::tgba\_sba\_proxy, 747
- cycle\_states\_
  - spot::ars\_statistics, 111
- data\_
  - spot::tgba\_bdd\_concrete, 615
  - spot::tgba\_bdd\_concrete\_factory, 620
  - spot::tgba\_succ\_iterator\_concrete, 770
- dead\_
  - spot::gspn\_interface, 316
- dead\_paths
  - spot::scc\_stats, 486
- dead\_scc
  - spot::scc\_stats, 486
- dec\_depth
  - spot::couvreur99\_check, 180
  - spot::couvreur99\_check\_shy, 195
  - spot::ec\_statistics, 228
- dec\_weight\_handler
  - spot::weight, 835
- declarative\_environment
  - spot::ltl::declarative\_environment, 205
- declare
  - spot::ltl::declarative\_environment, 205
- declare\_acceptance\_condition
  - spot::evtgba\_explicit, 250
  - spot::tgba\_bdd\_concrete\_factory, 619
  - spot::tgba\_bdd\_core\_data, 623
  - spot::tgba\_explicit\_formula, 645
  - spot::tgba\_explicit\_labelled, 657
  - spot::tgba\_explicit\_string, 669
  - spot::tgba\_reduc, 719
- declare\_atomic\_prop
  - spot::tgba\_bdd\_core\_data, 623
- declare\_now\_next
  - spot::tgba\_bdd\_core\_data, 624
- declare\_state
  - spot::evtgba\_explicit, 250
- Decorating the dot output, 38
- default\_environment
  - spot::ltl::default\_environment, 209
- del\_pred
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 497
  - spot::spoiler\_node\_delayed, 503
- del\_succ
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- delete\_me\_
  - spot::state\_set, 537
- delete\_scc
  - spot::tgba\_reduc, 719
- delete\_transitions
  - spot::tgba\_reduc, 719
- delete\_unaccepting\_scc
  - spot::tgba\_bdd\_concrete, 611
  - spot::tgba\_bdd\_core\_data, 624
- depth
  - spot::couvreur99\_check, 180
  - spot::couvreur99\_check\_shy, 195
  - spot::ec\_statistics, 228
- depth\_
  - spot::ec\_statistics, 229
- Derivable visitors, 12
- dest
  - spot::tgba\_explicit::transition, 807

- destroy
  - ltl\_essential, 10
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 122
  - spot::ltl::binop, 152
  - spot::ltl::constant, 173
  - spot::ltl::formula, 298
  - spot::ltl::multop, 353
  - spot::ltl::ref\_formula, 435
  - spot::ltl::unop, 825
- dict
  - spot::tgba\_bdd\_core\_data, 625
- dict\_
  - spot::bdd\_dict::anon\_free\_list, 107
  - spot::gspn\_interface, 316
  - spot::gspn\_ssp\_interface, 319
  - spot::ltl::language\_containment\_checker, 329
  - spot::nips\_interface, 365
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_explicit, 637
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 661
  - spot::tgba\_explicit\_string, 673
  - spot::tgba\_product, 695
  - spot::tgba\_reduc, 727
  - spot::tgba\_union, 797
- display\_rel\_sim
  - spot::tgba\_reduc, 719
- display\_safra
  - spot, 88
- display\_scc
  - spot::tgba\_reduc, 719
- dn\_v
  - tgba\_reduction, 34
- doit
  - spot::ltl::postfix\_visitor, 423
- doit\_default
  - spot::ltl::postfix\_visitor, 423
- done
  - spot::evtgba\_iterator, 253
  - spot::explicit\_state\_conjunction, 282
  - spot::fair\_kripke\_succ\_iterator, 294
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 343
  - spot::numbered\_state\_heap\_const\_iterator, 378
  - spot::saba\_state\_conjunction, 466
  - spot::saba\_succ\_iterator, 472
  - spot::taa\_succ\_iterator, 554
  - spot::tgba\_explicit\_succ\_iterator, 678
  - spot::tgba\_succ\_iterator, 764
  - spot::tgba\_succ\_iterator\_concrete, 769
  - spot::tgba\_succ\_iterator\_product, 774
  - spot::tgba\_succ\_iterator\_union, 780
- done\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
- dotty
  - ltl\_io, 6
- dotty\_decorator
  - spot::dotty\_decorator, 212
- dotty\_reachable
  - spot, 88
  - tgba\_io, 28
- drand
  - random, 22
- dst
  - spot::ltl::nfa::transition, 810
  - spot::taa\_tgba::transition, 809
- dump
  - ltl\_io, 7
  - spot::bdd\_dict, 138
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 122
  - spot::ltl::binop, 153
  - spot::ltl::constant, 174
  - spot::ltl::formula, 298
  - spot::ltl::multop, 354
  - spot::ltl::ref\_formula, 435
  - spot::ltl::unop, 826
  - spot::scc\_stats, 485
  - spot::tgba\_statistics, 762
- dump\_free\_list
  - spot::bdd\_allocator, 130
  - spot::bdd\_dict::anon\_free\_list, 106
  - spot::free\_list, 303
- dump\_instances
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 123
  - spot::ltl::binop, 153
  - spot::ltl::multop, 354
  - spot::ltl::unop, 826
  - spot::symbol, 549
- dump\_priorities
  - spot::ltl::random\_ltl, 428
- dump\_queue
  - spot::couvreur99\_check\_shy, 195
- dump\_scc\_dot
  - spot, 88, 89
- duplicator\_node
  - spot::duplicator\_node, 216
- duplicator\_node\_delayed
  - spot::duplicator\_node\_delayed, 221
- duplicator\_vertice\_
  - spot::parity\_game\_graph, 398

- spot::parity\_game\_graph\_delayed, 406
- spot::parity\_game\_graph\_direct, 413
- ec\_statistics
  - spot::ec\_statistics, 227
- ecs\_
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 198
- eltl\_to\_tgba\_lacim
  - tgba\_ltl, 30
- eltlparse/ Directory Reference, 51
- eltlparse/location.hh, 841
- eltlparse/position.hh, 844
- eltlparse/public.hh, 847
- eltlparse/stack.hh, 853
- eltly, 61
  - operator<<, 63, 64
  - operator+, 63
  - operator+=, 63
  - operator-, 63
  - operator=, 63
  - operator==, 64
- eltly::location, 332
  - begin, 335
  - columns, 334
  - end, 335
  - initialize, 334
  - lines, 334
  - location, 334
  - step, 334
- eltly::position, 416
  - column, 417
  - columns, 417
  - filename, 417
  - initialize, 417
  - line, 418
  - lines, 417
  - position, 417
- eltly::slice, 491
  - range\_, 492
  - slice, 492
  - stack\_, 492
- eltly::stack, 510
  - begin, 511
  - const\_iterator, 510
  - end, 511
  - height, 511
  - iterator, 510
  - pop, 511
  - push, 511
  - seq\_, 512
  - stack, 510
- Emptiness-check algorithms, 39
- Emptiness-check algorithms for SSP, 4
- Emptiness-check statistics, 50
- Emptiness-checks, 38
- emptiness\_check
  - spot::emptiness\_check, 232
- emptiness\_check\_algorithms
  - bit\_state\_hashing\_magic\_search, 40
  - bit\_state\_hashing\_se05\_search, 40
  - couvreur99, 41
  - explicit\_gv04\_check, 43
  - explicit\_magic\_search, 43
  - explicit\_se05\_search, 44
  - explicit\_tau03\_opt\_search, 45
  - explicit\_tau03\_search, 46
  - magic\_search, 48
  - se05, 48
- emptiness\_check\_instantiator
  - spot::emptiness\_check\_instantiator, 236
- emptiness\_check\_result
  - spot::emptiness\_check\_result, 240
- emptiness\_check\_ssp
  - couvreur99\_check\_ssp\_semi, 4
  - couvreur99\_check\_ssp\_shy, 4
  - couvreur99\_check\_ssp\_shy\_semi, 4
- empty
  - spot::scc\_stack, 484
  - spot::timer\_map, 804
- emulate\_acc\_cond\_
  - spot::tgba\_sgba\_proxy, 761
- end
  - eltly::location, 335
  - eltly::stack, 511
  - ltly::location, 341
  - ltly::stack, 508
  - sauty::location, 338
  - sauty::stack, 506
  - spot::evtgba\_reachable\_iterator, 262
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 266
  - spot::evtgba\_reachable\_iterator\_depth\_first, 271
  - spot::ltl::nfa, 360
  - spot::parity\_game\_graph, 396
  - spot::parity\_game\_graph\_delayed, 404
  - spot::parity\_game\_graph\_direct, 411
  - spot::saba\_reachable\_iterator, 451
  - spot::saba\_reachable\_iterator\_breadth\_first, 456
  - spot::saba\_reachable\_iterator\_depth\_first, 461
  - spot::tgba\_reachable\_iterator, 699
  - spot::tgba\_reachable\_iterator\_breadth\_first, 703
  - spot::tgba\_reachable\_iterator\_depth\_first, 708
  - spot::tgba\_reduc, 720

- env
  - spot::ltl::atomic\_prop, 116
- env\_
  - spot::gspn\_interface, 316
  - spot::gspn\_ssp\_interface, 319
  - spot::ltl::atomic\_prop, 117
- equal
  - spot::ltl::language\_containment\_checker, 329
- Equiv
  - spot::ltl::binop, 152
- err\_
  - spot::gspn\_exception, 313
  - spot::nips\_exception, 363
- err\_defined\_
  - spot::nips\_exception, 363
- escape\_str
  - misc\_tools, 19
- Essential LTL types, 10
- Essential SABA types, 23
- Essential TGBA types, 24
- evtgba
  - spot::evtgba, 245
- evtgba/ Directory Reference, 51
- evtgba/evtgba.hh, 854
- evtgba/evtgbaiter.hh, 855
- evtgba/explicit.hh, 856
- evtgba/product.hh, 857
- evtgba/symbol.hh, 857
- evtgba\_explicit
  - spot::evtgba\_explicit, 250
- evtgba\_parse
  - spot, 89
- evtgba\_parse\_error
  - spot, 84
- evtgba\_parse\_error\_list
  - spot, 84
- evtgba\_product
  - spot::evtgba\_product, 256
- evtgba\_product\_operands
  - spot::evtgba\_product, 256
- evtgba\_reachable\_iterator
  - spot::evtgba\_reachable\_iterator, 261
- evtgba\_reachable\_iterator\_breadth\_first
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 266
- evtgba\_reachable\_iterator\_depth\_first
  - spot::evtgba\_reachable\_iterator\_depth\_first, 271
- evtgba\_save\_reachable
  - spot, 89
- evtgbalogs/ Directory Reference, 52
- evtgbalogs/dotty.hh, 858
- evtgbalogs/reachiter.hh, 860
- evtgbalogs/save.hh, 862
- evtgbalogs/tgba2evtgba.hh, 864
- evtgbaparse/ Directory Reference, 52
- evtgbaparse/public.hh, 848
- explicit\_gv04\_check
  - emptiness\_check\_algorithms, 43
- explicit\_magic\_search
  - emptiness\_check\_algorithms, 43
- explicit\_se05\_search
  - emptiness\_check\_algorithms, 44
- explicit\_state\_conjunction
  - spot::explicit\_state\_conjunction, 281
- explicit\_tau03\_opt\_search
  - emptiness\_check\_algorithms, 45
- explicit\_tau03\_search
  - emptiness\_check\_algorithms, 46
- exprop\_
  - spot::ltl::language\_containment\_checker, 329
- extend
  - spot::bdd\_allocator, 130
  - spot::bdd\_dict::anon\_free\_list, 106
  - spot::free\_list, 303
- extvarnum
  - spot::bdd\_allocator, 131
- F
  - spot::ltl::unop, 824
- f0\_max
  - spot::minato\_isop::local\_vars, 331
- f0\_min
  - spot::minato\_isop::local\_vars, 331
- f1\_max
  - spot::minato\_isop::local\_vars, 331
- f1\_min
  - spot::minato\_isop::local\_vars, 331
- f\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
- f\_max
  - spot::minato\_isop::local\_vars, 331
- f\_min
  - spot::minato\_isop::local\_vars, 331
- fair\_kripke\_succ\_iterator
  - spot::fair\_kripke\_succ\_iterator, 293
- fair\_loop\_approx\_
  - spot::ltl::language\_containment\_checker, 330
- False
  - spot::ltl::constant, 173
  - spot::ltl::formula\_tree, 98
- false\_instance
  - spot::ltl::constant, 174
- fc\_map
  - spot::future\_conditions\_collector, 308
- filename
  - eltlyy::position, 417

- ltlyy::position, [419](#)
- sautyy::position, [415](#)
- filter
  - spot::bfs\_steps, [146](#)
- finalize
  - spot::bfs\_steps, [146](#)
- finals\_
  - spot::ltl::nfa, [361](#)
- find
  - spot::numbered\_state\_heap, [376](#)
  - spot::numbered\_state\_heap\_hash\_map, [384](#)
- find\_paths
  - spot, [89](#)
- find\_state
  - spot::couvreur99\_check\_shy, [195](#)
- Finish
  - spot::ltl::unop, [824](#)
- finish
  - spot::tgba\_bdd\_concrete\_factory, [619](#)
- first
  - spot::evtgba\_iterator, [253](#)
  - spot::explicit\_state\_conjunction, [282](#)
  - spot::fair\_kripke\_succ\_iterator, [294](#)
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, [344](#)
  - spot::ltl::binop, [153](#)
  - spot::numbered\_state\_heap\_const\_iterator, [378](#)
  - spot::saba\_state\_conjunction, [467](#)
  - spot::saba\_succ\_iterator, [473](#)
  - spot::taa\_succ\_iterator, [555](#)
  - spot::tgba\_explicit\_succ\_iterator, [678](#)
  - spot::tgba\_succ\_iterator, [765](#)
  - spot::tgba\_succ\_iterator\_concrete, [769](#)
  - spot::tgba\_succ\_iterator\_product, [774](#)
  - spot::tgba\_succ\_iterator\_union, [780](#)
- first\_
  - spot::ltl::binop, [155](#)
- FirstStep
  - spot::minato\_isop::local\_vars, [331](#)
- fl
  - spot::bdd\_allocator, [132](#)
  - spot::bdd\_dict::anon\_free\_list, [107](#)
  - spot::free\_list, [304](#)
- format\_acceptance\_condition
  - spot::evtgba, [245](#)
  - spot::evtgba\_explicit, [250](#)
  - spot::evtgba\_product, [257](#)
- format\_acceptance\_conditions
  - spot::evtgba, [245](#)
  - spot::evtgba\_explicit, [251](#)
  - spot::evtgba\_product, [257](#)
- format\_evtgba\_parse\_errors
  - spot, [89](#)
- format\_label
  - spot::evtgba, [245](#)
  - spot::evtgba\_explicit, [251](#)
  - spot::evtgba\_product, [257](#)
- format\_parse\_errors
  - ltl\_io, [7](#)
- format\_state
  - spot::evtgba, [245](#)
  - spot::evtgba\_explicit, [251](#)
  - spot::evtgba\_product, [257](#)
  - spot::fair\_kripke, [287](#)
  - spot::future\_conditions\_collector, [309](#)
  - spot::kripke, [323](#)
  - spot::ltl::nfa, [360](#)
  - spot::saba, [441](#)
  - spot::saba\_complement\_tgba, [446](#)
  - spot::taa\_tgba, [561](#)
  - spot::taa\_tgba\_formula, [571](#)
  - spot::taa\_tgba\_labelled, [583](#)
  - spot::taa\_tgba\_string, [593](#)
  - spot::tgba, [602](#)
  - spot::tgba\_bdd\_concrete, [611](#)
  - spot::tgba\_explicit, [634](#)
  - spot::tgba\_explicit\_formula, [645](#)
  - spot::tgba\_explicit\_labelled, [657](#)
  - spot::tgba\_explicit\_string, [669](#)
  - spot::tgba\_kv\_complement, [684](#)
  - spot::tgba\_product, [692](#)
  - spot::tgba\_reduc, [720](#)
  - spot::tgba\_safra\_complement, [737](#)
  - spot::tgba\_sba\_proxy, [745](#)
  - spot::tgba\_scc, [751](#)
  - spot::tgba\_sgba\_proxy, [758](#)
  - spot::tgba\_tba\_proxy, [786](#)
  - spot::tgba\_union, [794](#)
- format\_state\_set
  - spot::taa\_tgba\_labelled, [583](#)
- format\_tgba\_parse\_errors
  - tgba\_io, [28](#)
- formula
  - spot::ltl::formula, [297](#)
- formula\_to\_bdd
  - spot, [90](#)
- FourthStep
  - spot::minato\_isop::local\_vars, [331](#)
- free\_anonymous\_list\_of
  - spot::bdd\_dict, [142](#)
- free\_anonymous\_list\_of\_type
  - spot::bdd\_dict, [137](#)
- free\_count
  - spot::bdd\_allocator, [131](#)
  - spot::bdd\_dict::anon\_free\_list, [106](#)
  - spot::free\_list, [303](#)
- free\_list\_type



- spot::bdd\_allocator, 130
- spot::bdd\_dict::anon\_free\_list, 105
- spot::free\_list, 302
- free\_relation\_simulation
  - tgba\_reduction, 35
- future\_conditions
  - spot::future\_conditions\_collector, 309
- future\_conditions\_collector
  - spot::future\_conditions\_collector, 308
- future\_conds\_
  - spot::future\_conditions\_collector, 312
- fv\_map
  - spot::bdd\_dict, 137
- G
  - spot::ltl::unop, 824
- g0
  - spot::minato\_isop::local\_vars, 331
- g1
  - spot::minato\_isop::local\_vars, 331
- generate
  - spot::ltl::random\_ltl, 428
- get
  - spot::acss\_statistics, 102
  - spot::ars\_statistics, 110
  - spot::couvreur99\_check, 180
  - spot::couvreur99\_check\_result, 188
  - spot::couvreur99\_check\_shy, 195
  - spot::ec\_statistics, 228
  - spot::option\_map, 390
  - spot::unsigned\_statistics, 829
- get\_acc
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
- get\_acc\_list
  - spot::tgba\_kv\_complement, 684
- get\_acceptance\_condition
  - spot::tgba\_explicit, 634
  - spot::tgba\_explicit\_formula, 645
  - spot::tgba\_explicit\_labelled, 658
  - spot::tgba\_explicit\_string, 670
  - spot::tgba\_reduc, 720
- get\_acceptance\_condition\_visited
  - spot::spoiler\_node\_delayed, 503
- get\_aut
  - spot::scc\_map, 480
- get\_bdd
  - spot::bdd\_ordered, 144
- get\_core\_data
  - spot::tgba\_bdd\_concrete, 612
  - spot::tgba\_bdd\_concrete\_factory, 620
  - spot::tgba\_bdd\_factory, 628
- get\_delayed\_relation\_simulation
  - tgba\_reduction, 35
- get\_dict
  - spot::fair\_kripke, 287
  - spot::future\_conditions\_collector, 309
  - spot::kripke, 323
  - spot::saba, 442
  - spot::saba\_complement\_tgba, 446
  - spot::taa\_tgba, 562
  - spot::taa\_tgba\_formula, 572
  - spot::taa\_tgba\_labelled, 583
  - spot::taa\_tgba\_string, 594
  - spot::tgba, 603
  - spot::tgba\_bdd\_concrete, 612
  - spot::tgba\_bdd\_concrete\_factory, 620
  - spot::tgba\_explicit, 635
  - spot::tgba\_explicit\_formula, 645
  - spot::tgba\_explicit\_labelled, 658
  - spot::tgba\_explicit\_string, 670
  - spot::tgba\_kv\_complement, 684
  - spot::tgba\_product, 692
  - spot::tgba\_reduc, 720
  - spot::tgba\_safra\_complement, 738
  - spot::tgba\_sba\_proxy, 745
  - spot::tgba\_scc, 752
  - spot::tgba\_sgba\_proxy, 759
  - spot::tgba\_tba\_proxy, 786
  - spot::tgba\_union, 794
- get\_direct\_relation\_simulation
  - tgba\_reduction, 35
- get\_duplicator\_node
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- get\_err
  - spot::gspn\_exception, 313
  - spot::nips\_exception, 362
- get\_err\_defined
  - spot::nips\_exception, 362
- get\_index
  - spot::numbered\_state\_heap\_const\_iterator, 378
- get\_init\_bdd
  - spot::tgba\_bdd\_concrete, 612
- get\_init\_state
  - spot::fair\_kripke, 288
  - spot::future\_conditions\_collector, 309
  - spot::kripke, 324
  - spot::ltl::nfa, 360
  - spot::saba, 442
  - spot::saba\_complement\_tgba, 446
  - spot::taa\_tgba, 562
  - spot::taa\_tgba\_formula, 572
  - spot::taa\_tgba\_labelled, 584
  - spot::taa\_tgba\_string, 594



- spot::tgba, 603
- spot::tgba\_bdd\_concrete, 612
- spot::tgba\_explicit, 635
- spot::tgba\_explicit\_formula, 646
- spot::tgba\_explicit\_labelled, 658
- spot::tgba\_explicit\_string, 670
- spot::tgba\_kv\_complement, 684
- spot::tgba\_product, 692
- spot::tgba\_reduc, 720
- spot::tgba\_safra\_complement, 738
- spot::tgba\_sba\_proxy, 745
- spot::tgba\_scc, 752
- spot::tgba\_sgba\_proxy, 759
- spot::tgba\_tba\_proxy, 786
- spot::tgba\_union, 794
- get\_label
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::tgba\_explicit\_formula, 646
  - spot::tgba\_explicit\_labelled, 658
  - spot::tgba\_explicit\_string, 670
  - spot::tgba\_reduc, 720, 721
- get\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node\_delayed, 503
- get\_name
  - spot::ltl::nfa, 360
- get\_nb\_succ
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- get\_nfa
  - spot::ltl::automatop, 123
- get\_pair
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 222
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- get\_progress\_measure
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node\_delayed, 503
- get\_prop\_map
  - spot::ltl::declarative\_environment, 205
- get\_relation
  - spot::parity\_game\_graph, 396
  - spot::parity\_game\_graph\_delayed, 404
  - spot::parity\_game\_graph\_direct, 411
- get\_removed\_components
  - spot::couvereur99\_check, 180
  - spot::couvereur99\_check\_shy, 195
- get\_safra
  - spot::tgba\_safra\_complement, 738
- get\_spoiler\_node
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- get\_state
  - spot::numbered\_state\_heap\_const\_iterator, 378
  - spot::state\_evtgba\_explicit, 521
  - spot::state\_explicit, 525
  - spot::state\_set, 536
- get\_vmsize
  - spot::couvereur99\_check, 180
  - spot::couvereur99\_check\_shy, 196
- get\_where
  - spot::gspn\_exception, 313
  - spot::nips\_exception, 363
- group2\_
  - spot::couvereur99\_check\_shy, 198
- group\_
  - spot::couvereur99\_check\_shy, 198
- gspn/ Directory Reference, 53
- gspn/common.hh, 836
- gspn/gspn.hh, 837
- gspn/ssp.hh, 838
- gspn\_exception
  - spot::gspn\_exception, 313
- gspn\_interface
  - spot::gspn\_interface, 316
- gspn\_ssp\_interface
  - spot::gspn\_ssp\_interface, 319
- h
  - spot::couvereur99\_check\_status, 202
  - spot::numbered\_state\_heap\_hash\_map, 385
- h\_
  - spot::scc\_map, 482
- has\_acceptance\_condition
  - spot::tgba\_explicit, 635
  - spot::tgba\_explicit\_formula, 646
  - spot::tgba\_explicit\_labelled, 658
  - spot::tgba\_explicit\_string, 670
  - spot::tgba\_reduc, 721
- has\_monitor
  - spot::nips\_interface, 365
- has\_state
  - spot::connected\_component\_hash\_set, 164
  - spot::explicit\_connected\_component, 276
  - spot::tgba\_explicit\_formula, 646
  - spot::tgba\_explicit\_labelled, 659
  - spot::tgba\_explicit\_string, 671
  - spot::tgba\_reduc, 721
- hash
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 123

- spot::ltl::binop, 153
- spot::ltl::constant, 174
- spot::ltl::formula, 298
- spot::ltl::multop, 354
- spot::ltl::ref\_formula, 435
- spot::ltl::unop, 826
- spot::saba\_state, 464
- spot::state, 514
- spot::state\_bdd, 517
- spot::state\_evtgba\_explicit, 521
- spot::state\_explicit, 525
- spot::state\_product, 530
- spot::state\_set, 537
- spot::state\_union, 543
- hash\_funcs
  - knuth32\_hash, 21
  - wang32\_hash, 21
- hash\_type
  - spot::numbered\_state\_heap\_hash\_map, 383
  - spot::scc\_map, 478
- Hashing functions, 20
- height
  - eltlyy::stack, 511
  - ltlyy::stack, 509
  - sautyy::stack, 506
- i
  - spot::ltl::formula\_tree::node\_atomic, 368
- i\_
  - spot::ltl::succ\_iterator, 546
  - spot::taa\_succ\_iterator, 555
  - spot::tgba\_explicit\_succ\_iterator, 678
- Implies
  - spot::ltl::binop, 152
- implies
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 223
- implies\_acc
  - spot::duplicator\_node\_delayed, 223
- implies\_label
  - spot::duplicator\_node\_delayed, 223
- in
  - spot::evtgba\_explicit::state, 512
  - spot::evtgba\_explicit::transition, 812
  - spot::ltl::ltl\_file, 346
- inc\_ars\_cycle\_states
  - spot::acss\_statistics, 102
  - spot::ars\_statistics, 111
  - spot::couvereur99\_check\_result, 188
- inc\_ars\_prefix\_states
  - spot::acss\_statistics, 102
  - spot::ars\_statistics, 111
  - spot::couvereur99\_check\_result, 188
- inc\_depth
  - spot::couvereur99\_check, 180
  - spot::couvereur99\_check\_shy, 196
  - spot::ec\_statistics, 228
- inc\_states
  - spot::couvereur99\_check, 181
  - spot::couvereur99\_check\_shy, 196
  - spot::ec\_statistics, 228
- inc\_transitions
  - spot::couvereur99\_check, 181
  - spot::couvereur99\_check\_shy, 196
  - spot::ec\_statistics, 228
- inc\_weight\_handler
  - spot::weight, 835
- incomp\_map
  - spot::ltl::language\_containment\_checker::record\_, 431
- incompatible
  - spot::ltl::language\_containment\_checker::record\_, 431
- incompatible\_
  - spot::ltl::language\_containment\_checker, 329
- index
  - spot::connected\_component\_hash\_set, 165
  - spot::explicit\_connected\_component, 276
  - spot::numbered\_state\_heap, 377
  - spot::numbered\_state\_heap\_hash\_map, 384
  - spot::scc\_map::scc, 474
  - spot::scc\_stack::connected\_component, 161
- infinitely\_ofTEN
  - spot::tgba\_bdd\_core\_data, 624
- info\_
  - spot::emptiness\_check\_instantiator, 237
- init\_
  - spot::ltl::nfa, 361
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_bdd\_concrete, 615
  - spot::tgba\_explicit, 638
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 661
  - spot::tgba\_explicit\_string, 673
  - spot::tgba\_reduc, 727
- init\_iter
  - spot::evtgba, 246
  - spot::evtgba\_explicit, 251
  - spot::evtgba\_product, 257
- init\_states\_
  - spot::evtgba\_explicit, 252
- initial
  - spot::scc\_map, 480
- initialize
  - eltlyy::location, 334

- eltlyy::position, 417
- ltlyy::location, 340
- ltlyy::position, 419
- sautyy::location, 337
- sautyy::position, 415
- spot::bdd\_allocator, 131
- spot::bdd\_dict, 138
- initialized
  - spot::bdd\_allocator, 132
  - spot::bdd\_dict, 142
- Input/Output of LTL formulae, 4
- Input/Output of TGBA, 26
- insert
  - spot::bdd\_allocator, 131
  - spot::bdd\_dict::anon\_free\_list, 106
  - spot::connected\_component\_hash\_set, 164
  - spot::explicit\_connected\_component, 276
  - spot::free\_list, 303
  - spot::numbered\_state\_heap, 377
  - spot::numbered\_state\_heap\_hash\_map, 384
- instance
  - spot::connected\_component\_hash\_set\_factory, 167
  - spot::dotty\_decorator, 212
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 123
  - spot::ltl::binop, 153
  - spot::ltl::default\_environment, 209
  - spot::ltl::multop, 354
  - spot::ltl::unop, 826
  - spot::numbered\_state\_heap\_hash\_map\_factory, 387
  - spot::symbol, 549
  - spot::tgba\_run\_dotty\_decorator, 732
- instance\_count
  - spot::ltl::atomic\_prop, 116
  - spot::ltl::automatop, 123
  - spot::ltl::binop, 153
  - spot::ltl::multop, 355
  - spot::ltl::unop, 826
  - spot::symbol, 549
- instances
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::automatop, 124
  - spot::ltl::binop, 155
  - spot::ltl::multop, 356
  - spot::ltl::unop, 827
- instances\_
  - spot::symbol, 550
- instantiate
  - spot::ltl::formula\_tree, 98
- instantiate
  - spot::emptiness\_check\_instantiator, 236
- is\_
  - spot::ltl::nfa, 361
- is\_bare\_word
  - misc\_tools, 19
- is\_eventual
  - ltl\_misc, 16
- is\_FG
  - ltl\_misc, 16
- is\_final
  - spot::ltl::nfa, 360
- is\_GF
  - ltl\_misc, 16
- is\_loop
  - spot::ltl::nfa, 360
- is\_map
  - spot::ltl::nfa, 359
- is\_negated
  - spot::ltl::automatop, 123
- is\_not\_accepting
  - spot::tgba\_reduc, 721
- is\_registered\_acceptance\_variable
  - spot::bdd\_dict, 139
- is\_registered\_proposition
  - spot::bdd\_dict, 139
- is\_registered\_state
  - spot::bdd\_dict, 139
- is\_running
  - spot::timer, 801
- is\_terminal
  - spot::tgba\_reduc, 721
- is\_universal
  - ltl\_misc, 16
- it\_
  - spot::explicit\_state\_conjunction, 283
- item\_type
  - spot::timer\_map, 803
- iterator
  - eltlyy::stack, 510
  - ltlyy::stack, 508
  - sautyy::stack, 506
  - spot::ltl::nfa, 359
  - spot::numbered\_state\_heap, 377
  - spot::numbered\_state\_heap\_hash\_map, 384
  - spot::taa\_succ\_iterator, 553
- iterator\_pair
  - spot::taa\_succ\_iterator, 553
- knuth32\_hash
  - hash\_funcs, 21
- kripke/ Directory Reference, 54
- kripke/fairkripke.hh, 864
- kripke/kripke.hh, 865
- kripke\_succ\_iter
  - spot::fair\_kripke, 288
  - spot::kripke, 324

- kripke\_succ\_iterator
  - spot, 84
- label
  - spot::evtgba\_explicit::transition, 812
  - spot::ltl::nfa, 359
  - spot::tgba\_run::step, 545
- label\_
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
- label\_t
  - spot::taa\_tgba\_formula, 569
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 591
  - spot::tgba\_explicit\_formula, 643
  - spot::tgba\_explicit\_labelled, 655
  - spot::tgba\_explicit\_string, 667
  - spot::tgba\_reduc, 716
- label\_to\_string
  - spot::taa\_tgba\_formula, 572
  - spot::taa\_tgba\_labelled, 584
  - spot::taa\_tgba\_string, 594
- language\_containment\_checker
  - spot::ltl::language\_containment\_checker, 328
- last
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
- last\_support\_conditions\_input\_
  - spot::tgba, 605
- last\_support\_conditions\_output\_
  - spot::tgba, 605
- last\_support\_variables\_input\_
  - spot::tgba, 605
- last\_support\_variables\_output\_
  - spot::tgba, 606
- lbl
  - spot::ltl::nfa::transition, 810
- lbtt\_reachable
  - tgba\_io, 28
- lead\_2\_acc\_all\_
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node\_delayed, 504
- left
  - spot::state\_product, 530
  - spot::state\_union, 543
- left\_
  - spot::state\_product, 531
  - spot::state\_union, 544
  - spot::tgba\_product, 695
  - spot::tgba\_succ\_iterator\_product, 775
  - spot::tgba\_succ\_iterator\_union, 781
  - spot::tgba\_union, 797
- left\_acc\_complement\_
  - spot::tgba\_product, 695
  - spot::tgba\_union, 797
- left\_acc\_missing\_
  - spot::tgba\_union, 797
- left\_missing\_
  - spot::tgba\_succ\_iterator\_union, 781
- left\_neg\_
  - spot::tgba\_succ\_iterator\_product, 775
  - spot::tgba\_succ\_iterator\_union, 781
- left\_var\_missing\_
  - spot::tgba\_union, 797
- length
  - ltl\_misc, 17
- lhs
  - spot::ltl::formula\_tree::node\_binop, 369
  - spot::ltl::formula\_tree::node\_multop, 371
- lift
  - spot::parity\_game\_graph, 396
  - spot::parity\_game\_graph\_delayed, 405
  - spot::parity\_game\_graph\_direct, 412
- line
  - eltlyy::position, 418
  - ltlyy::position, 420
  - sautyy::position, 416
- lines
  - eltlyy::location, 334
  - eltlyy::position, 417
  - ltlyy::location, 340
  - ltlyy::position, 419
  - sautyy::location, 337
  - sautyy::position, 415
- link\_decl
  - spot::dotty\_decorator, 212
  - spot::tgba\_run\_dotty\_decorator, 733
- lnode\_pred
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node, 499
  - spot::spoiler\_node\_delayed, 504
- lnode\_succ
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node, 499
  - spot::spoiler\_node\_delayed, 504
- local\_vars
  - spot::minato\_isop::local\_vars, 331
- location
  - eltlyy::location, 334
  - ltlyy::location, 340
  - sautyy::location, 337
- loopless\_modular\_mixed\_radix\_gray\_code
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 343
- LTL Abstract Syntax Tree, 11
- LTL environments, 11

- LTL formulae, 3
- ltl\_rewriting
  - Reduce\_All, 13
  - Reduce\_Basics, 13
  - Reduce\_Containment\_Checks, 13
  - Reduce\_Containment\_Checks\_Stronger, 13
  - Reduce\_Eventuality\_And\_Universality, 13
  - Reduce\_None, 13
  - Reduce\_Syntactic\_Implications, 13
- ltl\_essential
  - clone, 10
  - destroy, 10
- ltl\_file
  - spot::ltl::ltl\_file, 345
- ltl\_io
  - dotty, 6
  - dump, 7
  - format\_parse\_errors, 7
  - parse, 7
  - parse\_error, 6
  - parse\_error\_list, 6
  - parse\_file, 8
  - parse\_string, 8
  - spair, 6
  - to\_spin\_string, 9
  - to\_string, 9
- ltl\_misc
  - atomic\_prop\_collect, 16
  - atomic\_prop\_set, 15
  - is\_eventual, 16
  - is\_FG, 16
  - is\_GF, 16
  - is\_universal, 16
  - length, 17
  - syntactic\_implication, 17
  - syntactic\_implication\_neg, 17
- ltl\_rewriting
  - basic\_reduce, 13
  - negative\_normal\_form, 13
  - reduce, 14
  - reduce\_options, 13
  - simplify\_f\_g, 14
  - unabbreviate\_logic, 14
- ltl\_to\_taa
  - tgba\_ltl, 30
- ltl\_to\_tgba\_fm
  - tgba\_ltl, 31
- ltl\_to\_tgba\_lacim
  - tgba\_ltl, 32
- ltlast/ Directory Reference, 54
- ltlast/allnodes.hh, 865
- ltlast/atomic\_prop.hh, 866
- ltlast/automatop.hh, 867
- ltlast/binop.hh, 868
- ltlast/constant.hh, 869
- ltlast/formula.hh, 870
- ltlast/formula\_tree.hh, 871
- ltlast/multop.hh, 873
- ltlast/nfa.hh, 874
- ltlast/predecl.hh, 875
- ltlast/refformula.hh, 875
- ltlast/unop.hh, 876
- ltlast/visitor.hh, 877
- ltlenv/ Directory Reference, 55
- ltlenv/declenv.hh, 878
- ltlenv/defaultenv.hh, 879
- ltlenv/environment.hh, 880
- ltlparse/ Directory Reference, 56
- ltlparse/location.hh, 842
- ltlparse/ltlfile.hh, 880
- ltlparse/position.hh, 845
- ltlparse/public.hh, 849
- ltlparse/stack.hh, 853
- ltlvisit/ Directory Reference, 56
- ltlvisit/apcollect.hh, 881
- ltlvisit/basicreduce.hh, 882
- ltlvisit/clone.hh, 883
- ltlvisit/contain.hh, 883
- ltlvisit/destroy.hh, 884
- ltlvisit/dotty.hh, 859
- ltlvisit/dump.hh, 885
- ltlvisit/length.hh, 886
- ltlvisit/lunabbrev.hh, 886
- ltlvisit/nenoform.hh, 887
- ltlvisit/postfix.hh, 888
- ltlvisit/randomltl.hh, 889
- ltlvisit/reduce.hh, 889
- ltlvisit/simpfg.hh, 891
- ltlvisit/syntimpl.hh, 891
- ltlvisit/tostring.hh, 892
- ltlvisit/tunabbrev.hh, 893
- ltlly, 64
  - operator<<, 67
  - operator+, 66
  - operator+==, 66
  - operator-, 66
  - operator==, 67
  - operator==, 67
- ltlly::location, 338
  - begin, 341
  - columns, 340
  - end, 341
  - initialize, 340
  - lines, 340
  - location, 340
  - step, 340
- ltlly::position, 418
  - column, 419

- columns, 419
- filename, 419
- initialize, 419
- line, 420
- lines, 419
- position, 419
- ltlyy::slice, 493
  - range\_, 493
  - slice, 493
  - stack\_, 493
- ltlyy::stack, 507
  - begin, 508
  - const\_iterator, 508
  - end, 508
  - height, 509
  - iterator, 508
  - pop, 509
  - push, 509
  - seq\_, 509
  - stack, 508
- lvarnum
  - spot::bdd\_allocator, 132
  - spot::bdd\_dict, 142
- M
  - spot::ltl::binop, 152
- m
  - spot::weight, 836
- m\_
  - spot::barand, 126
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
- magic\_search
  - emptiness\_check\_algorithms, 48
- mainpage.dox, 836
- map
  - spot::ltl::atomic\_prop, 115
  - spot::ltl::automatop, 121
  - spot::ltl::binop, 151
  - spot::ltl::multop, 352
  - spot::ltl::unop, 824
  - spot::symbol, 549
- map\_
  - spot::tgba\_run\_dotty\_decorator, 733
- map\_builder\_
  - spot::future\_conditions\_collector, 310
- match
  - spot::bfs\_steps, 147
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 223
- max\_acceptance\_conditions
  - spot::emptiness\_check\_instantiator, 236
- max\_count
  - spot::ltl::formula, 299
- max\_depth
  - spot::couvreur99\_check, 181
  - spot::couvreur99\_check\_shy, 196
  - spot::ec\_statistics, 228
- max\_depth\_
  - spot::ec\_statistics, 229
- max\_spanning\_paths
  - spot, 90
- memusage
  - spot, 90
- merge\_state
  - spot::tgba\_reduc, 721
- merge\_state\_delayed
  - spot::tgba\_reduc, 721
- merge\_transitions
  - spot::tgba\_explicit\_formula, 646
  - spot::tgba\_explicit\_labelled, 659
  - spot::tgba\_explicit\_string, 671
  - spot::tgba\_reduc, 722
- min\_acceptance\_conditions
  - spot::emptiness\_check\_instantiator, 236
- min\_n
  - spot::ltl::random\_ltl::op\_proba, 389
- minato\_isop
  - spot::minato\_isop, 347
- misc/ Directory Reference, 57
- misc/bareword.hh, 894
- misc/bddalloc.hh, 894
- misc/bddlt.hh, 895
- misc/bddop.hh, 895
- misc/escape.hh, 896
- misc/freelist.hh, 897
- misc/hash.hh, 897
- misc/hashfunc.hh, 898
- misc/ltstr.hh, 899
- misc/memusage.hh, 900
- misc/minato.hh, 900
- misc/modgray.hh, 900
- misc/optionmap.hh, 901
- misc/random.hh, 901
- misc/timer.hh, 902
- misc/version.hh, 903
- misc\_tools
  - escape\_str, 19
  - is\_bare\_word, 19
  - quote\_unless\_bare\_word, 19
  - version, 20
- Miscellaneous algorithms for LTL formulae, 15
- Miscellaneous algorithms on TGBA, 36
- Miscellaneous helper algorithms, 18
- mrnd
  - random, 22
- multop
  - spot::ltl::multop, 353

- n
  - spot::couvreur99\_check\_shy::todo\_item, 806
- n\_
  - spot::barand, 126
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 345
- name
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::declarative\_environment, 205
  - spot::ltl::default\_environment, 209
  - spot::ltl::environment, 243
  - spot::ltl::random\_ltl::op\_proba, 389
  - spot::symbol, 549
- name\_
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::nfa, 361
  - spot::symbol, 550
- name\_state\_map\_
  - spot::evtgba\_explicit, 252
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 661
  - spot::tgba\_explicit\_string, 674
  - spot::tgba\_reduc, 727
- nb\_node\_parity\_game
  - spot::parity\_game\_graph, 398
  - spot::parity\_game\_graph\_delayed, 406
  - spot::parity\_game\_graph\_direct, 413
- nb\_set\_acc\_cond
  - spot::parity\_game\_graph\_delayed, 405
  - spot::tgba\_reduc, 722
- nb\_states\_
  - spot::saba\_complement\_tgba, 447
  - spot::tgba\_kv\_complement, 687
- neg\_acceptance\_cond\_
  - spot::tgba\_safra\_complement, 740
- neg\_acceptance\_conditions
  - spot::fair\_kripke, 288
  - spot::future\_conditions\_collector, 310
  - spot::kripke, 324
  - spot::taa\_tgba, 562
  - spot::taa\_tgba\_formula, 572
  - spot::taa\_tgba\_labelled, 584
  - spot::taa\_tgba\_string, 594
  - spot::tgba, 603
  - spot::tgba\_bdd\_concrete, 612
  - spot::tgba\_explicit, 635
  - spot::tgba\_explicit\_formula, 646
  - spot::tgba\_explicit\_labelled, 659
  - spot::tgba\_explicit\_string, 671
  - spot::tgba\_kv\_complement, 684
  - spot::tgba\_product, 692
  - spot::tgba\_reduc, 722
  - spot::tgba\_safra\_complement, 738
  - spot::tgba\_sba\_proxy, 745
  - spot::tgba\_scc, 752
  - spot::tgba\_sgba\_proxy, 759
  - spot::tgba\_tba\_proxy, 786
  - spot::tgba\_union, 795
- neg\_acceptance\_conditions\_
  - spot::taa\_tgba, 565
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_explicit, 638
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 662
  - spot::tgba\_explicit\_string, 674
  - spot::tgba\_product, 695
  - spot::tgba\_reduc, 727
  - spot::tgba\_union, 797
- neg\_all\_acc
  - spot::weight, 836
- neg\_contained
  - spot::ltl::language\_containment\_checker, 329
- negacc\_set
  - spot::tgba\_bdd\_core\_data, 625
- negated\_
  - spot::ltl::automatop, 125
- negative\_normal\_form
  - ltl\_rewriting, 13
- never\_claim\_reachable
  - tgba\_io, 28
- next
  - spot::evtgba\_iterator, 253
  - spot::explicit\_state\_conjunction, 282
  - spot::fair\_kripke\_succ\_iterator, 294
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 344
  - spot::ltl::ltl\_file, 346
  - spot::minato\_isop, 347
  - spot::numbered\_state\_heap\_const\_iterator, 379
  - spot::saba\_state\_conjunction, 467
  - spot::saba\_succ\_iterator, 473
  - spot::taa\_succ\_iterator, 555
  - spot::tgba\_explicit\_succ\_iterator, 678
  - spot::tgba\_succ\_iterator, 765
  - spot::tgba\_succ\_iterator\_concrete, 770
  - spot::tgba\_succ\_iterator\_product, 775
  - spot::tgba\_succ\_iterator\_union, 780
- next\_non\_false\_
  - spot::tgba\_succ\_iterator\_product, 775
- next\_set
  - spot::tgba\_bdd\_core\_data, 625
- next\_state



- spot::evtgba\_reachable\_iterator, 262
- spot::evtgba\_reachable\_iterator\_breadth\_first, 267
- spot::evtgba\_reachable\_iterator\_depth\_first, 272
- spot::parity\_game\_graph, 397
- spot::parity\_game\_graph\_delayed, 405
- spot::parity\_game\_graph\_direct, 412
- spot::saba\_reachable\_iterator, 451
- spot::saba\_reachable\_iterator\_breadth\_first, 456
- spot::saba\_reachable\_iterator\_depth\_first, 461
- spot::tgba\_reachable\_iterator, 699
- spot::tgba\_reachable\_iterator\_breadth\_first, 704
- spot::tgba\_reachable\_iterator\_depth\_first, 709
- spot::tgba\_reduc, 722
- next\_to\_now
  - spot::bdd\_dict, 142
- nfa
  - spot::ltl::formula\_tree::node\_nfa, 372
  - spot::ltl::nfa, 359
- nfa\_
  - spot::ltl::automatop, 125
- nips.hh
  - nipsvm\_bytecode\_t, 840
  - nipsvm\_t, 840
- nips/ Directory Reference, 57
- nips/common.hh, 837
- nips/nips.hh, 839
- nips\_exception
  - spot::nips\_exception, 362
- nips\_interface
  - spot::nips\_interface, 365
- nipsvm\_
  - spot::nips\_interface, 365
- nipsvm\_bytecode\_t
  - nips.hh, 840
- nipsvm\_t
  - nips.hh, 840
- node\_ptr
  - spot::ltl::formula\_tree, 97
- non\_one\_radixes\_
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 345
- Not
  - spot::ltl::unop, 824
- not\_win
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node, 499
  - spot::spoiler\_node\_delayed, 504
- notacc\_set
  - spot::tgba\_bdd\_core\_data, 625
- notnext\_set
  - spot::tgba\_bdd\_core\_data, 625
- notnow\_set
  - spot::tgba\_bdd\_core\_data, 625
- notvar\_set
  - spot::tgba\_bdd\_core\_data, 626
- now\_formula\_map
  - spot::bdd\_dict, 142
- now\_map
  - spot::bdd\_dict, 142
- now\_set
  - spot::tgba\_bdd\_core\_data, 626
- now\_to\_next
  - spot::bdd\_dict, 142
- nownext\_set
  - spot::tgba\_bdd\_core\_data, 626
- nrand
  - random, 22
- ns\_map
  - spot::evtgba\_explicit, 249
  - spot::taa\_tgba\_formula, 569
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 591
  - spot::tgba\_explicit\_formula, 643
  - spot::tgba\_explicit\_labelled, 655
  - spot::tgba\_explicit\_string, 667
  - spot::tgba\_reduc, 716
- nth
  - spot::ltl::automatop, 123, 124
  - spot::ltl::multop, 355
- num
  - spot::couvereur99\_check\_shy, 198
- num\_
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::scc\_map, 482
  - spot::spoiler\_node, 499
  - spot::spoiler\_node\_delayed, 504
- num\_acc\_
  - spot::saba, 443
  - spot::tgba, 606
- number\_of\_acceptance\_conditions
  - spot::fair\_kripke, 288
  - spot::future\_conditions\_collector, 310
  - spot::kripke, 324
  - spot::saba, 442
  - spot::saba\_complement\_tgba, 447
  - spot::taa\_tgba, 562
  - spot::taa\_tgba\_formula, 573
  - spot::taa\_tgba\_labelled, 584
  - spot::taa\_tgba\_string, 595
  - spot::tgba, 603
  - spot::tgba\_bdd\_concrete, 613
  - spot::tgba\_explicit, 635



- spot::tgba\_explicit\_formula, 647
- spot::tgba\_explicit\_labelled, 659
- spot::tgba\_explicit\_string, 671
- spot::tgba\_kv\_complement, 685
- spot::tgba\_product, 693
- spot::tgba\_reduc, 722
- spot::tgba\_safra\_complement, 738
- spot::tgba\_sba\_proxy, 745
- spot::tgba\_scc, 752
- spot::tgba\_sgba\_proxy, 759
- spot::tgba\_tba\_proxy, 787
- spot::tgba\_union, 795
- numbered\_state\_heap\_hash\_map\_factory
  - spot::numbered\_state\_heap\_hash\_map\_-factory, 387
- o\_
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 198
  - spot::emptiness\_check, 234
  - spot::emptiness\_check\_instantiator, 237
  - spot::emptiness\_check\_result, 241
- onepass\_
  - spot::couvreur99\_check\_shy, 199
- op
  - spot::ltl::binop, 153
  - spot::ltl::formula\_tree::node\_binop, 369
  - spot::ltl::formula\_tree::node\_multop, 371
  - spot::ltl::formula\_tree::node\_unop, 374
  - spot::ltl::multop, 355
  - spot::ltl::unop, 826
- op\_
  - spot::evtgba\_product, 258
  - spot::ltl::binop, 155
  - spot::ltl::multop, 356
  - spot::ltl::unop, 827
- op\_name
  - spot::ltl::binop, 154
  - spot::ltl::multop, 355
  - spot::ltl::unop, 826
- operator const symbol \*
  - spot::rsymbol, 438
- operator<
  - spot::rsymbol, 438
- operator<<
  - eltlyy, 63, 64
  - ltlyy, 67
  - sautyy, 69, 70
  - spot, 90
  - spot::option\_map, 392
  - spot::weight, 835
- operator\*
  - spot::ltl::succ\_iterator, 546
- operator()
  - spot::bdd\_less\_than, 143
  - spot::char\_ptr\_less\_than, 155
  - spot::ltl::automatop::tripletcmp, 812
  - spot::ltl::formula\_ptr\_hash, 300
  - spot::ltl::formula\_ptr\_less\_than, 300
  - spot::ltl::multop::paircmp, 392
  - spot::ptr\_hash, 425
  - spot::saba\_state\_ptr\_equal, 468
  - spot::saba\_state\_ptr\_hash, 468
  - spot::saba\_state\_ptr\_less\_than, 469
  - spot::saba\_state\_shared\_ptr\_equal, 470
  - spot::saba\_state\_shared\_ptr\_hash, 470
  - spot::saba\_state\_shared\_ptr\_less\_than, 471
  - spot::state\_ptr\_equal, 531
  - spot::state\_ptr\_hash, 532
  - spot::state\_ptr\_less\_than, 533
  - spot::state\_shared\_ptr\_equal, 538
  - spot::state\_shared\_ptr\_hash, 538
  - spot::state\_shared\_ptr\_less\_than, 539
  - spot::string\_hash, 545
  - spot::taa\_succ\_iterator::distance\_sort, 210
- operator+
  - eltlyy, 63
  - ltlyy, 66
  - sautyy, 68, 69
- operator++
  - spot::ltl::succ\_iterator, 546
- operator+=
  - eltlyy, 63
  - ltlyy, 66
  - sautyy, 69
  - spot::weight, 835
- operator-
  - eltlyy, 63
  - ltlyy, 66
  - sautyy, 69
  - spot::weight, 835
- operator-=
  - eltlyy, 63
  - ltlyy, 67
  - sautyy, 69
  - spot::weight, 835
- operator=
  - spot::bdd\_dict, 139
  - spot::explicit\_state\_conjunction, 283
  - spot::ltl::nfa, 361
  - spot::rsymbol, 439
  - spot::taa\_tgba, 562
  - spot::tgba\_bdd\_concrete, 613
  - spot::tgba\_bdd\_core\_data, 624
  - spot::tgba\_explicit, 635
  - spot::tgba\_product, 693
  - spot::tgba\_run, 729

- spot::tgba\_sgba\_proxy, 759
- spot::tgba\_tba\_proxy, 787
- spot::tgba\_union, 795
- operator==
  - eltlyy, 64
  - ltlyy, 67
  - spot::rsymbol, 439
  - spot::unsigned\_statistics\_copy, 830
- options
  - spot::couvreur99\_check, 181
  - spot::couvreur99\_check\_result, 188
  - spot::couvreur99\_check\_shy, 196
  - spot::emptiness\_check, 233
  - spot::emptiness\_check\_instantiator, 237
  - spot::emptiness\_check\_result, 241
- options\_
  - spot::option\_map, 392
- options\_updated
  - spot::couvreur99\_check, 181
  - spot::couvreur99\_check\_result, 188
  - spot::couvreur99\_check\_shy, 196
  - spot::emptiness\_check, 233
  - spot::emptiness\_check\_result, 241
- Or
  - spot::ltl::multop, 353
- order
  - spot::bdd\_ordered, 144
- order\_
  - spot::bdd\_ordered, 144
- out
  - spot::evtgba\_explicit::state, 512
  - spot::evtgba\_explicit::transition, 812
- output
  - spot::taa\_tgba\_formula, 573
  - spot::taa\_tgba\_labelled, 584
  - spot::taa\_tgba\_string, 595
- pair
  - spot::ltl::atomic\_prop, 115
  - spot::ltl::binop, 151
  - spot::ltl::multop, 352
  - spot::ltl::unop, 824
- pair\_state\_iter
  - spot::scc\_map, 478
- pairf
  - spot::ltl::binop, 151
- parity\_game\_graph
  - spot::parity\_game\_graph, 396
- parity\_game\_graph\_delayed
  - spot::parity\_game\_graph\_delayed, 403
- parity\_game\_graph\_direct
  - spot::parity\_game\_graph\_direct, 411
- parse
  - ltl\_io, 7
- parse\_error
  - ltl\_io, 6
- parse\_error\_list
  - ltl\_io, 6
- parse\_file
  - ltl\_io, 8
- parse\_options
  - spot::couvreur99\_check, 181
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 196
  - spot::emptiness\_check, 233
  - spot::emptiness\_check\_result, 241
  - spot::ltl::random\_ltl, 428
  - spot::option\_map, 391
- parse\_string
  - ltl\_io, 8
- pm
  - spot::weight, 836
- pop
  - eltlyy::stack, 511
  - ltlyy::stack, 509
  - sautyy::stack, 507
  - spot::scc\_stack, 484
- poprem\_
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_shy, 199
- pos
  - spot::couvreur99\_check\_shy, 199
- pos\_lenght\_pair
  - spot::bdd\_allocator, 130
  - spot::bdd\_dict::anon\_free\_list, 105
  - spot::free\_list, 302
- position
  - eltlyy::position, 417
  - ltlyy::position, 419
  - sautyy::position, 415
- postfix\_visitor
  - spot::ltl::postfix\_visitor, 423
- prand
  - random, 22
- pred\_iter
  - spot::evtgba, 246
  - spot::evtgba\_explicit, 251
  - spot::evtgba\_product, 257
- prefix
  - spot::tgba\_run, 729
- prefix\_states\_
  - spot::ars\_statistics, 111
- print
  - spot::parity\_game\_graph, 397
  - spot::parity\_game\_graph\_delayed, 405
  - spot::parity\_game\_graph\_direct, 412
  - spot::timer\_map, 804
- print\_stats

- spot::couvreur99\_check, 181
- spot::couvreur99\_check\_result, 189
- spot::couvreur99\_check\_shy, 197
- spot::couvreur99\_check\_status, 201
- spot::emptiness\_check, 233
- print\_tgba\_run
  - tgba\_run, 49
- proba
  - spot::ltl::random\_ltl::op\_proba, 389
- proba\_
  - spot::ltl::random\_ltl, 429
- proba\_2\_
  - spot::ltl::random\_ltl, 429
- process\_link
  - spot::evtgba\_reachable\_iterator, 262
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 267
  - spot::evtgba\_reachable\_iterator\_depth\_first, 272
  - spot::parity\_game\_graph, 397
  - spot::parity\_game\_graph\_delayed, 405
  - spot::parity\_game\_graph\_direct, 412
  - spot::saba\_reachable\_iterator, 451
  - spot::saba\_reachable\_iterator\_breadth\_first, 456
  - spot::saba\_reachable\_iterator\_depth\_first, 461
  - spot::tgba\_reachable\_iterator, 699
  - spot::tgba\_reachable\_iterator\_breadth\_first, 704
  - spot::tgba\_reachable\_iterator\_depth\_first, 709
  - spot::tgba\_reduc, 722, 723
- process\_state
  - spot::evtgba\_reachable\_iterator, 262
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 267
  - spot::evtgba\_reachable\_iterator\_depth\_first, 272
  - spot::parity\_game\_graph, 397
  - spot::parity\_game\_graph\_delayed, 405
  - spot::parity\_game\_graph\_direct, 412
  - spot::saba\_reachable\_iterator, 451
  - spot::saba\_reachable\_iterator\_breadth\_first, 456
  - spot::saba\_reachable\_iterator\_depth\_first, 461
  - spot::tgba\_reachable\_iterator, 699
  - spot::tgba\_reachable\_iterator\_breadth\_first, 704
  - spot::tgba\_reachable\_iterator\_depth\_first, 709
  - spot::tgba\_reduc, 723
- process\_state\_conjunction
  - spot::saba\_reachable\_iterator, 452
  - spot::saba\_reachable\_iterator\_breadth\_first, 456
  - spot::saba\_reachable\_iterator\_depth\_first, 461
- product
  - tgba\_algorithms, 26
- progress\_measure\_
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node\_delayed, 505
- project\_state
  - spot::fair\_kripke, 288
  - spot::future\_conditions\_collector, 310
  - spot::kripke, 324
  - spot::taa\_tgba, 563
  - spot::taa\_tgba\_formula, 573
  - spot::taa\_tgba\_labelled, 585
  - spot::taa\_tgba\_string, 595
  - spot::tgba, 604
  - spot::tgba\_bdd\_concrete, 613
  - spot::tgba\_explicit, 636
  - spot::tgba\_explicit\_formula, 647
  - spot::tgba\_explicit\_labelled, 659
  - spot::tgba\_explicit\_string, 671
  - spot::tgba\_kv\_complement, 685
  - spot::tgba\_product, 693
  - spot::tgba\_reduc, 723
  - spot::tgba\_safra\_complement, 738
  - spot::tgba\_sba\_proxy, 746
  - spot::tgba\_scc, 752
  - spot::tgba\_sgba\_proxy, 760
  - spot::tgba\_tba\_proxy, 787
  - spot::tgba\_union, 795
- project\_tgba\_run
  - tgba\_run, 49
- prop\_map
  - spot::ltl::declarative\_environment, 205
- props\_
  - spot::ltl::declarative\_environment, 206
- prune
  - spot::duplicator\_node, 217
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 503
- ptr
  - spot::ltl::nfa, 359
- push
  - eltlyy::stack, 511
  - ltlyy::stack, 509
  - sautyy::stack, 507
  - spot::scc\_stack, 484
- q
  - spot::couvreur99\_check\_shy::todo\_item, 806
- quote\_unless\_bare\_word
  - misc\_tools, 19
- quotient\_state
  - spot::tgba\_reduc, 723

- R
  - spot::ltl::binop, 152
- rand
  - spot::barand, 126
- random
  - bmrnd, 22
  - drand, 22
  - mrnd, 22
  - nrnd, 22
  - prand, 22
  - rrand, 22
  - srnd, 23
- Random functions, 21
- random\_graph
  - tgba\_misc, 37
- random\_ltl
  - spot::ltl::random\_ltl, 427
- range\_
  - eltlyy::slice, 492
  - ltlyy::slice, 493
  - sautyy::slice, 495
- recurse
  - spot::ltl::clone\_visitor, 158
  - spot::ltl::simplify\_f\_g\_visitor, 490
  - spot::ltl::unabbreviate\_logic\_visitor, 815
  - spot::ltl::unabbreviate\_ltl\_visitor, 819
- redirect\_transition
  - spot::tgba\_reduc, 724
- reduc\_tgba\_sim
  - tgba\_reduction, 35
- reduce
  - ltl\_rewriting, 14
- Reduce\_All
  - ltl\_rewriting, 13
  - tgba\_reduction, 35
- Reduce\_Basics
  - ltl\_rewriting, 13
- Reduce\_Containment\_Checks
  - ltl\_rewriting, 13
- Reduce\_Containment\_Checks\_Stronger
  - ltl\_rewriting, 13
- Reduce\_Eventuality\_And\_Universality
  - ltl\_rewriting, 13
- Reduce\_None
  - ltl\_rewriting, 13
  - tgba\_reduction, 35
- Reduce\_quotient\_Del\_Sim
  - tgba\_reduction, 35
- Reduce\_quotient\_Dir\_Sim
  - tgba\_reduction, 35
- Reduce\_Scc
  - tgba\_reduction, 35
- Reduce\_Syntactic\_Implications
  - ltl\_rewriting, 13
- Reduce\_transition\_Del\_Sim
  - tgba\_reduction, 35
- Reduce\_transition\_Dir\_Sim
  - tgba\_reduction, 35
- reduce\_options
  - ltl\_rewriting, 13
- reduce\_run
  - tgba\_run, 49
- reduce\_tau03
  - spot::ltl, 96
- reduce\_tgba\_options
  - tgba\_reduction, 35
- ref
  - spot::symbol, 550
- ref\_
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::automatop, 124
  - spot::ltl::binop, 154
  - spot::ltl::constant, 174
  - spot::ltl::formula, 299
  - spot::ltl::multop, 355
  - spot::ltl::ref\_formula, 435
  - spot::ltl::unop, 826
- ref\_count\_
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::automatop, 124
  - spot::ltl::binop, 154
  - spot::ltl::multop, 355
  - spot::ltl::ref\_formula, 436
  - spot::ltl::unop, 827
  - spot::symbol, 550
- ref\_counter\_
  - spot::ltl::ref\_formula, 436
- ref\_formula
  - spot::ltl::ref\_formula, 434
- ref\_set
  - spot::bdd\_dict, 137
- refs\_
  - spot::symbol, 550
- register\_acceptance\_variable
  - spot::bdd\_dict, 139
- register\_acceptance\_variables
  - spot::bdd\_dict, 139
- register\_all\_variables\_of
  - spot::bdd\_dict, 139
- register\_anonymous\_variables
  - spot::bdd\_dict, 140
- register\_clone\_acc
  - spot::bdd\_dict, 140
- register\_formula\_
  - spot::ltl::language\_containment\_checker, 329
- register\_n
  - spot::bdd\_allocator, 131
  - spot::bdd\_dict::anon\_free\_list, 106

- spot::free\_list, 303
- register\_proposition
  - spot::bdd\_dict, 140
- register\_propositions
  - spot::bdd\_dict, 140
- register\_state
  - spot::bdd\_dict, 140
- relabel\_component
  - spot::scc\_map, 480
- relation
  - spot::tgba\_bdd\_core\_data, 626
- release\_n
  - spot::bdd\_allocator, 131
  - spot::bdd\_dict::anon\_free\_list, 106
  - spot::free\_list, 303
- release\_variables
  - spot::bdd\_allocator, 131
  - spot::bdd\_dict, 141
- rem
  - spot::connected\_component\_hash\_set, 165
  - spot::explicit\_connected\_component, 276
  - spot::scc\_stack, 484
  - spot::scc\_stack::connected\_component, 161
- remove
  - spot::bdd\_allocator, 131, 132
  - spot::bdd\_dict::anon\_free\_list, 106
  - spot::free\_list, 303, 304
- remove\_acc
  - spot::tgba\_reduc, 724
- remove\_component
  - spot::couvreur99\_check, 181
  - spot::couvreur99\_check\_shy, 197
  - spot::tgba\_reduc, 724
- remove\_predecessor\_state
  - spot::tgba\_reduc, 724
- remove\_scc
  - spot::tgba\_reduc, 724
- remove\_state
  - spot::tgba\_reduc, 724
- removed\_components
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_shy, 199
- replay\_tgba\_run
  - tgba\_run, 49
- require
  - spot::ltl::declarative\_environment, 205
  - spot::ltl::default\_environment, 209
  - spot::ltl::environment, 243
- result
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_shy, 197
  - spot::ltl::clone\_visitor, 158
  - spot::ltl::simplify\_f\_g\_visitor, 490
  - spot::ltl::unabbreviate\_logic\_visitor, 815
  - spot::ltl::unabbreviate\_ltl\_visitor, 819
- result\_
  - spot::ltl::clone\_visitor, 159
  - spot::ltl::simplify\_f\_g\_visitor, 491
  - spot::ltl::unabbreviate\_logic\_visitor, 816
  - spot::ltl::unabbreviate\_ltl\_visitor, 820
- ret\_
  - spot::minato\_isop, 348
- Rewriting LTL formulae, 12
- rhs
  - spot::ltl::formula\_tree::node\_binop, 369
  - spot::ltl::formula\_tree::node\_multop, 371
- right
  - spot::state\_product, 530
  - spot::state\_union, 543
- right\_
  - spot::state\_product, 531
  - spot::state\_union, 544
  - spot::tgba\_product, 695
  - spot::tgba\_succ\_iterator\_product, 775
  - spot::tgba\_succ\_iterator\_union, 781
  - spot::tgba\_union, 797
- right\_acc\_complement\_
  - spot::tgba\_product, 695
  - spot::tgba\_union, 797
- right\_acc\_missing\_
  - spot::tgba\_union, 798
- right\_common\_acc\_
  - spot::tgba\_product, 695
  - spot::tgba\_succ\_iterator\_product, 776
  - spot::tgba\_union, 798
- right\_missing\_
  - spot::tgba\_succ\_iterator\_union, 781
- right\_neg\_
  - spot::tgba\_succ\_iterator\_product, 776
  - spot::tgba\_succ\_iterator\_union, 781
- right\_var\_missing\_
  - spot::tgba\_union, 798
- root
  - spot::couvreur99\_check\_status, 202
- root\_
  - spot::scc\_map, 482
- rrand
  - random, 22
- rsymbol
  - spot::rsymbol, 438
- rsymbol\_set
  - spot, 84
- run
  - spot::evtgba\_reachable\_iterator, 262
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 267
  - spot::evtgba\_reachable\_iterator\_depth\_first, 272

- spot::parity\_game\_graph, 398
- spot::parity\_game\_graph\_delayed, 406
- spot::parity\_game\_graph\_direct, 413
- spot::saba\_reachable\_iterator, 452
- spot::saba\_reachable\_iterator\_breadth\_first, 457
- spot::saba\_reachable\_iterator\_depth\_first, 462
- spot::tgba\_reachable\_iterator, 699
- spot::tgba\_reachable\_iterator\_breadth\_first, 704
- spot::tgba\_reachable\_iterator\_depth\_first, 709
- spot::tgba\_reduc, 724
- run\_
  - spot::couvreur99\_check\_result, 189
  - spot::tgba\_run\_dotty\_decorator, 733
- running
  - spot::timer, 802
- s
  - spot::couvreur99\_check\_shy::successor, 548
  - spot::couvreur99\_check\_shy::todo\_item, 806
  - spot::scc\_stack, 485
  - spot::tgba\_run::step, 545
- s\_
  - spot::barand, 126
  - spot::loopless\_modular\_mixed\_radix\_gray\_code, 345
  - spot::rsymbol, 439
  - spot::state\_set, 537
  - spot::tgba\_explicit\_succ\_iterator, 679
- s\_v
  - tgba\_reduction, 34
- saba
  - spot::saba, 441
- SABA (State-based Alternating Büchi Automata), 3
- saba/ Directory Reference, 58
- saba/explicitstateconjunction.hh, 903
- saba/saba.hh, 904
- saba/sabacomplementtgba.hh, 905
- saba/sabastate.hh, 906
- saba/sabasucciter.hh, 908
- saba\_complement\_tgba
  - spot::saba\_complement\_tgba, 446
- saba\_dotty\_reachable
  - spot, 90
- saba\_reachable\_iterator
  - spot::saba\_reachable\_iterator, 450
- saba\_reachable\_iterator\_breadth\_first
  - spot::saba\_reachable\_iterator\_breadth\_first, 455
- saba\_reachable\_iterator\_depth\_first
  - spot::saba\_reachable\_iterator\_depth\_first, 460
- saba\_state\_set\_t
  - spot::explicit\_state\_conjunction, 281
- sabaalgorithms/ Directory Reference, 58
- sabaalgorithms/sabadotty.hh, 908
- sabaalgorithms/sabareachiter.hh, 909
- safe
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_shy, 197
  - spot::emptiness\_check, 233
- safra\_
  - spot::tgba\_safra\_complement, 740
- sautparse/ Directory Reference, 59
- sautparse/location.hh, 843
- sautparse/position.hh, 846
- sautparse/stack.hh, 854
- sautyy, 68
  - operator<<, 69, 70
  - operator+, 68, 69
  - operator+&, 69
  - operator-, 69
  - operator=, 69
- sautyy::location, 335
  - begin, 338
  - columns, 337
  - end, 338
  - initialize, 337
  - lines, 337
  - location, 337
  - step, 337
- sautyy::position, 414
  - column, 415
  - columns, 415
  - filename, 415
  - initialize, 415
  - line, 416
  - lines, 415
  - position, 415
- sautyy::slice, 494
  - range\_, 495
  - slice, 494
  - stack\_, 495
- sautyy::stack, 505
  - begin, 506
  - const\_iterator, 506
  - end, 506
  - height, 506
  - iterator, 506
  - pop, 507
  - push, 507
  - seq\_, 507
  - stack, 506
- sc\_
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 224
  - spot::spoiler\_node, 499
  - spot::spoiler\_node\_delayed, 505

- scc
  - spot::scc\_map::scc, [474](#)
- scc\_count
  - spot::scc\_map, [480](#)
- scc\_filter
  - spot, [90](#)
- scc\_map
  - spot::scc\_map, [479](#)
- scc\_map\_
  - spot::future\_conditions\_collector, [312](#)
  - spot::scc\_map, [482](#)
  - spot::tgba\_scc, [754](#)
- scc\_map\_type
  - spot::scc\_map, [478](#)
- scc\_of\_state
  - spot::future\_conditions\_collector, [310](#)
  - spot::scc\_map, [481](#)
  - spot::tgba\_scc, [753](#)
- scc\_total
  - spot::scc\_stats, [486](#)
- sccs
  - spot::sccs\_set, [487](#)
- se05
  - emptiness\_check\_algorithms, [48](#)
- search
  - spot::bfs\_steps, [147](#)
- second
  - spot::ltl::binop, [154](#)
- second\_
  - spot::ltl::binop, [155](#)
- SecondStep
  - spot::minato\_isop::local\_vars, [331](#)
- seen
  - spot::evtgba\_reachable\_iterator, [263](#)
  - spot::evtgba\_reachable\_iterator\_breadth\_first, [268](#)
  - spot::evtgba\_reachable\_iterator\_depth\_first, [273](#)
  - spot::parity\_game\_graph, [398](#)
  - spot::parity\_game\_graph\_delayed, [406](#)
  - spot::parity\_game\_graph\_direct, [413](#)
  - spot::saba\_reachable\_iterator, [452](#)
  - spot::saba\_reachable\_iterator\_breadth\_first, [457](#)
  - spot::saba\_reachable\_iterator\_depth\_first, [462](#)
  - spot::tgba\_reachable\_iterator, [700](#)
  - spot::tgba\_reachable\_iterator\_breadth\_first, [705](#)
  - spot::tgba\_reachable\_iterator\_depth\_first, [710](#)
  - spot::tgba\_reduc, [727](#)
- seen\_
  - spot::duplicator\_node\_delayed, [224](#)
  - spot::spoiler\_node\_delayed, [505](#)
  - spot::taa\_succ\_iterator, [555](#)
- seen\_map
  - spot::evtgba\_reachable\_iterator, [261](#)
  - spot::evtgba\_reachable\_iterator\_breadth\_first, [266](#)
  - spot::evtgba\_reachable\_iterator\_depth\_first, [271](#)
  - spot::parity\_game\_graph, [396](#)
  - spot::parity\_game\_graph\_delayed, [403](#)
  - spot::parity\_game\_graph\_direct, [411](#)
  - spot::saba\_reachable\_iterator, [450](#)
  - spot::saba\_reachable\_iterator\_breadth\_first, [455](#)
  - spot::saba\_reachable\_iterator\_depth\_first, [460](#)
  - spot::taa\_succ\_iterator, [553](#)
  - spot::tgba\_reachable\_iterator, [698](#)
  - spot::tgba\_reachable\_iterator\_breadth\_first, [703](#)
  - spot::tgba\_reachable\_iterator\_depth\_first, [708](#)
  - spot::tgba\_reduc, [716](#)
- self\_loops
  - spot::scc\_map, [481](#)
  - spot::scc\_stats, [486](#)
- self\_loops\_
  - spot::scc\_map, [482](#)
- seq\_
  - eltlyy::stack, [512](#)
  - ltlyy::stack, [509](#)
  - sautyy::stack, [507](#)
- set
  - spot::option\_map, [391](#)
  - spot::unsigned\_statistics\_copy, [831](#)
- set\_
  - spot::explicit\_state\_conjunction, [283](#)
- set\_acceptance\_conditions
  - spot::tgba\_explicit, [636](#)
  - spot::tgba\_explicit\_formula, [647](#)
  - spot::tgba\_explicit\_labelled, [659](#)
  - spot::tgba\_explicit\_string, [672](#)
  - spot::tgba\_reduc, [724](#)
- set\_final
  - spot::ltl::nfa, [361](#)
- set\_init\_state
  - spot::evtgba\_explicit, [251](#)
  - spot::ltl::nfa, [361](#)
  - spot::taa\_tgba\_formula, [573](#)
  - spot::taa\_tgba\_labelled, [585](#)
  - spot::taa\_tgba\_string, [595](#)
  - spot::tgba\_bdd\_concrete, [613](#)
  - spot::tgba\_explicit\_formula, [647](#)
  - spot::tgba\_explicit\_labelled, [660](#)
  - spot::tgba\_explicit\_string, [672](#)
  - spot::tgba\_reduc, [725](#)
- set\_lead\_2\_acc\_all
  - spot::duplicator\_node\_delayed, [223](#)



- spot::spoiler\_node\_delayed, 504
- set\_name
  - spot::ltl::nfa, 361
- set\_states
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_shy, 197
  - spot::ec\_statistics, 228
- set\_type
  - spot::connected\_component\_hash\_set, 164
- set\_win
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 504
- seteq
  - spot::unsigned\_statistics\_copy, 831
- setup
  - spot::ltl::random\_ltl::op\_proba, 389
- shared\_saba\_state
  - spot, 84
- shared\_state
  - spot, 84
- show\_
  - spot::future\_conditions\_collector, 312
  - spot::tgba\_scc, 754
- si\_
  - spot::ltl::nfa, 361
- si\_map
  - spot::ltl::nfa, 359
- simplify\_f\_g
  - ltl\_rewriting, 14
- simplify\_f\_g\_visitor
  - spot::ltl::simplify\_f\_g\_visitor, 490
- simulation\_relation
  - spot, 84
- size
  - spot::ltl::automatop, 124
  - spot::ltl::multop, 355
  - spot::numbered\_state\_heap, 377
  - spot::numbered\_state\_heap\_hash\_map, 385
  - spot::scc\_stack, 484
  - spot::sccs\_set, 487
- slice
  - eltlyy::slice, 492
  - ltlyy::slice, 493
  - sautyy::slice, 494
- sn\_map
  - spot::evtgba\_explicit, 249
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 591
  - spot::tgba\_explicit\_formula, 643
  - spot::tgba\_explicit\_labelled, 655
  - spot::tgba\_explicit\_string, 667
  - spot::tgba\_reduc, 716
- sn\_v
  - tgba\_reduction, 34
- sp\_map
  - spot::tgba\_reduc, 716
- spair
  - ltl\_io, 6
- split\_tgba
  - spot, 91
- spoiler\_node
  - spot::spoiler\_node, 497
- spoiler\_node\_delayed
  - spot::spoiler\_node\_delayed, 502
- spoiler\_vertice\_
  - spot::parity\_game\_graph, 398
  - spot::parity\_game\_graph\_delayed, 407
  - spot::parity\_game\_graph\_direct, 413
- spot, 70
  - acc\_list\_t, 84
  - bdd\_format\_accset, 85
  - bdd\_format\_formula, 85
  - bdd\_format\_sat, 85
  - bdd\_format\_set, 85
  - bdd\_print\_acc, 86
  - bdd\_print\_accset, 86
  - bdd\_print\_dot, 86
  - bdd\_print\_formula, 87
  - bdd\_print\_sat, 87
  - bdd\_print\_set, 87
  - bdd\_print\_table, 87
  - bdd\_to\_formula, 88
  - build\_scc\_stats, 88
  - compute\_all\_acceptance\_conditions, 88
  - compute\_neg\_acceptance\_conditions, 88
  - display\_safra, 88
  - dotty\_reachable, 88
  - dump\_scc\_dot, 88, 89
  - evtgba\_parse, 89
  - evtgba\_parse\_error, 84
  - evtgba\_parse\_error\_list, 84
  - evtgba\_save\_reachable, 89
  - find\_paths, 89
  - format\_evtgba\_parse\_errors, 89
  - formula\_to\_bdd, 90
  - kripke\_succ\_iterator, 84
  - max\_spanning\_paths, 90
  - memusage, 90
  - operator<<, 90
  - rsymbol\_set, 84
  - saba\_dotty\_reachable, 90
  - scc\_filter, 90
  - shared\_saba\_state, 84
  - shared\_state, 84
  - simulation\_relation, 84



- split\_tgba, 91
- state\_couple, 84
- symbol\_set, 84
- tgba\_to\_evtgba, 91
- spot::acss\_statistics, 98
  - ~acss\_statistics, 101
  - acss\_states, 101
  - acss\_statistics, 101
  - ars\_cycle\_states, 101
  - ars\_prefix\_states, 102
  - get, 102
  - inc\_ars\_cycle\_states, 102
  - inc\_ars\_prefix\_states, 102
  - stats, 102
  - stats\_map, 101
  - unsigned\_fun, 101
- spot::ars\_statistics, 107
  - ars\_cycle\_states, 110
  - ars\_prefix\_states, 110
  - ars\_statistics, 110
  - cycle\_states\_, 111
  - get, 110
  - inc\_ars\_cycle\_states, 111
  - inc\_ars\_prefix\_states, 111
  - prefix\_states\_, 111
  - stats, 111
  - stats\_map, 110
  - unsigned\_fun, 110
- spot::barand, 125
  - barand, 125
  - m\_, 126
  - n\_, 126
  - rand, 126
  - s\_, 126
- spot::bdd\_allocator, 126
  - allocate\_variables, 130
  - bdd\_allocator, 130
  - dump\_free\_list, 130
  - extend, 130
  - extvarnum, 131
  - fl, 132
  - free\_count, 131
  - free\_list\_type, 130
  - initialize, 131
  - initialized, 132
  - insert, 131
  - lvarnum, 132
  - pos\_lenght\_pair, 130
  - register\_n, 131
  - release\_n, 131
  - release\_variables, 131
  - remove, 131, 132
- spot::bdd\_dict, 132
  - ~bdd\_dict, 138
  - acc\_formula\_map, 141
  - acc\_map, 141
  - allocate\_variables, 138
  - assert\_emptiness, 138
  - bdd\_dict, 138
  - cc\_map, 137
  - clone\_counts, 141
  - dump, 138
  - free\_anonymous\_list\_of, 142
  - free\_anonymous\_list\_of\_type, 137
  - fv\_map, 137
  - initialize, 138
  - initialized, 142
  - is\_registered\_acceptance\_variable, 139
  - is\_registered\_proposition, 139
  - is\_registered\_state, 139
  - lvarnum, 142
  - next\_to\_now, 142
  - now\_formula\_map, 142
  - now\_map, 142
  - now\_to\_next, 142
  - operator=, 139
  - ref\_set, 137
  - register\_acceptance\_variable, 139
  - register\_acceptance\_variables, 139
  - register\_all\_variables\_of, 139
  - register\_anonymous\_variables, 140
  - register\_clone\_acc, 140
  - register\_proposition, 140
  - register\_propositions, 140
  - register\_state, 140
  - release\_variables, 141
  - unregister\_all\_my\_variables, 141
  - unregister\_variable, 141
  - var\_formula\_map, 142
  - var\_map, 142
  - var\_refs, 143
  - vf\_map, 137
  - vr\_map, 138
- spot::bdd\_dict::anon\_free\_list, 102
  - anon\_free\_list, 105
  - dict\_, 107
  - dump\_free\_list, 106
  - extend, 106
  - fl, 107
  - free\_count, 106
  - free\_list\_type, 105
  - insert, 106
  - pos\_lenght\_pair, 105
  - register\_n, 106
  - release\_n, 106
  - remove, 106
- spot::bdd\_less\_than, 143
  - operator(), 143

- spot::bdd\_ordered, 143
  - bdd\_, 144
  - bdd\_ordered, 144
  - get\_bdd, 144
  - order, 144
  - order\_, 144
- spot::bfs\_steps, 145
  - ~bfs\_steps, 146
  - a\_, 147
  - bfs\_steps, 146
  - filter, 146
  - finalize, 146
  - match, 147
  - search, 147
- spot::char\_ptr\_less\_than, 155
  - operator(), 155
- spot::connected\_component\_hash\_set, 161
  - ~connected\_component\_hash\_set, 164
  - condition, 165
  - has\_state, 164
  - index, 165
  - insert, 164
  - rem, 165
  - set\_type, 164
  - states, 165
- spot::connected\_component\_hash\_set\_factory, 165
  - ~connected\_component\_hash\_set\_factory, 167
  - build, 167
  - connected\_component\_hash\_set\_factory, 167
  - instance, 167
- spot::couvreur99\_check, 175
  - ~couvreur99\_check, 179
  - a\_, 183
  - automaton, 180
  - check, 180
  - couvreur99\_check, 179
  - dec\_depth, 180
  - depth, 180
  - ecs\_, 183
  - get, 180
  - get\_removed\_components, 180
  - get\_vmsize, 180
  - inc\_depth, 180
  - inc\_states, 181
  - inc\_transitions, 181
  - max\_depth, 181
  - o\_, 183
  - options, 181
  - options\_updated, 181
  - parse\_options, 181
  - poprem\_, 183
  - print\_stats, 181
  - remove\_component, 181
  - removed\_components, 183
  - result, 182
  - safe, 182
  - set\_states, 182
  - states, 182
  - statistics, 182
  - stats, 183
  - stats\_map, 179
  - transitions, 182
  - unsigned\_fun, 179
- spot::couvreur99\_check\_result, 183
  - a\_, 189
  - accepting\_cycle, 187
  - accepting\_run, 187
  - acss\_states, 187
  - ars\_cycle\_states, 187
  - ars\_prefix\_states, 188
  - automaton, 188
  - couvreur99\_check\_result, 187
  - ecs\_, 189
  - get, 188
  - inc\_ars\_cycle\_states, 188
  - inc\_ars\_prefix\_states, 188
  - o\_, 189
  - options, 188
  - options\_updated, 188
  - parse\_options, 189
  - print\_stats, 189
  - run\_, 189
  - statistics, 189
  - stats, 189
  - stats\_map, 187
  - unsigned\_fun, 187
- spot::couvreur99\_check\_shy, 190
  - ~couvreur99\_check\_shy, 194
  - a\_, 198
  - arc, 198
  - automaton, 194
  - check, 194
  - clear\_todo, 195
  - couvreur99\_check\_shy, 194
  - dec\_depth, 195
  - depth, 195
  - dump\_queue, 195
  - ecs\_, 198
  - find\_state, 195
  - get, 195
  - get\_removed\_components, 195
  - get\_vmsize, 196
  - group2\_, 198
  - group\_, 198
  - inc\_depth, 196
  - inc\_states, 196
  - inc\_transitions, 196

- max\_depth, 196
- num, 198
- o\_, 198
- onepass\_, 199
- options, 196
- options\_updated, 196
- parse\_options, 196
- poprem\_, 199
- pos, 199
- print\_stats, 197
- remove\_component, 197
- removed\_components, 199
- result, 197
- safe, 197
- set\_states, 197
- states, 197
- statistics, 197
- stats, 199
- stats\_map, 194
- succ\_queue, 194
- todo, 199
- todo\_list, 194
- transitions, 198
- unsigned\_fun, 194
- spot::couvreur99\_check\_shy::successor, 547
  - acc, 548
  - s, 548
  - successor, 547
- spot::couvreur99\_check\_shy::todo\_item, 805
  - n, 806
  - q, 806
  - s, 806
  - todo\_item, 806
- spot::couvreur99\_check\_status, 199
  - ~couvreur99\_check\_status, 201
  - aut, 201
  - couvreur99\_check\_status, 201
  - cycle\_seed, 201
  - h, 202
  - print\_stats, 201
  - root, 202
  - states, 201
- spot::delayed\_simulation\_relation, 210
- spot::direct\_simulation\_relation, 210
- spot::dotty\_decorator, 210
  - ~dotty\_decorator, 212
  - dotty\_decorator, 212
  - instance, 212
  - link\_decl, 212
  - state\_decl, 212
- spot::duplicator\_node, 213
  - ~duplicator\_node, 216
  - acc\_, 218
  - add\_pred, 216
  - add\_succ, 216
  - compare, 216
  - del\_pred, 217
  - del\_succ, 217
  - duplicator\_node, 216
  - get\_acc, 217
  - get\_duplicator\_node, 217
  - get\_label, 217
  - get\_nb\_succ, 217
  - get\_pair, 217
  - get\_spoiler\_node, 217
  - implies, 217
  - label\_, 218
  - lnode\_pred, 218
  - lnode\_succ, 218
  - match, 217
  - not\_win, 218
  - num\_, 218
  - prune, 217
  - sc\_, 218
  - set\_win, 218
  - succ\_to\_string, 218
  - to\_string, 218
- spot::duplicator\_node\_delayed, 219
  - ~duplicator\_node\_delayed, 221
  - acc\_, 224
  - add\_pred, 222
  - add\_succ, 222
  - compare, 222
  - del\_pred, 222
  - del\_succ, 222
  - duplicator\_node\_delayed, 221
  - get\_acc, 222
  - get\_duplicator\_node, 222
  - get\_label, 222
  - get\_lead\_2\_acc\_all, 222
  - get\_nb\_succ, 222
  - get\_pair, 222
  - get\_progress\_measure, 223
  - get\_spoiler\_node, 223
  - implies, 223
  - implies\_acc, 223
  - implies\_label, 223
  - label\_, 224
  - lead\_2\_acc\_all\_, 224
  - lnode\_pred, 224
  - lnode\_succ, 224
  - match, 223
  - not\_win, 224
  - num\_, 224
  - progress\_measure\_, 224
  - prune, 223
  - sc\_, 224
  - seen\_, 224

- set\_lead\_2\_acc\_all, 223
  - set\_win, 223
  - succ\_to\_string, 223
  - to\_string, 223
- spot::ec\_statistics, 225
  - dec\_depth, 228
  - depth, 228
  - depth\_, 229
  - ec\_statistics, 227
  - get, 228
  - inc\_depth, 228
  - inc\_states, 228
  - inc\_transitions, 228
  - max\_depth, 228
  - max\_depth\_, 229
  - set\_states, 228
  - states, 228
  - states\_, 229
  - stats, 229
  - stats\_map, 227
  - transitions, 229
  - transitions\_, 229
  - unsigned\_fun, 227
- spot::eltl, 91
- spot::emptiness\_check, 230
  - ~emptiness\_check, 232
  - a\_, 234
  - automaton, 232
  - check, 232
  - emptiness\_check, 232
  - o\_, 234
  - options, 233
  - options\_updated, 233
  - parse\_options, 233
  - print\_stats, 233
  - safe, 233
  - statistics, 233
- spot::emptiness\_check\_instantiator, 234
  - construct, 236
  - emptiness\_check\_instantiator, 236
  - info\_, 237
  - instantiate, 236
  - max\_acceptance\_conditions, 236
  - min\_acceptance\_conditions, 236
  - o\_, 237
  - options, 237
- spot::emptiness\_check\_result, 237
  - ~emptiness\_check\_result, 240
  - a\_, 241
  - accepting\_run, 240
  - automaton, 241
  - emptiness\_check\_result, 240
  - o\_, 241
  - options, 241
- options\_updated, 241
  - parse\_options, 241
  - statistics, 241
- spot::evtgba, 243
  - ~evtgba, 245
  - all\_acceptance\_conditions, 245
  - alphabet, 245
  - evtgba, 245
  - format\_acceptance\_condition, 245
  - format\_acceptance\_conditions, 245
  - format\_label, 245
  - format\_state, 245
  - init\_iter, 246
  - pred\_iter, 246
  - succ\_iter, 246
- spot::evtgba\_explicit, 246
  - ~evtgba\_explicit, 250
  - acc\_set\_, 252
  - add\_transition, 250
  - all\_acceptance\_conditions, 250
  - alphabet, 250
  - alphabet\_, 252
  - declare\_acceptance\_condition, 250
  - declare\_state, 250
  - evtgba\_explicit, 250
  - format\_acceptance\_condition, 250
  - format\_acceptance\_conditions, 251
  - format\_label, 251
  - format\_state, 251
  - init\_iter, 251
  - init\_states\_, 252
  - name\_state\_map\_, 252
  - ns\_map, 249
  - pred\_iter, 251
  - set\_init\_state, 251
  - sn\_map, 249
  - state\_name\_map\_, 252
  - succ\_iter, 251
  - transition\_list, 249
- spot::evtgba\_explicit::state, 512
  - in, 512
  - out, 512
- spot::evtgba\_explicit::transition, 811
  - acceptance\_conditions, 812
  - in, 812
  - label, 812
  - out, 812
- spot::evtgba\_iterator, 252
  - ~evtgba\_iterator, 252
  - current\_acceptance\_conditions, 253
  - current\_label, 253
  - current\_state, 253
  - done, 253
  - first, 253

- next, 253
- spot::evtgba\_product, 253
  - ~evtgba\_product, 256
  - all\_acc\_, 258
  - all\_acceptance\_conditions, 256
  - alphabet, 256
  - alphabet\_, 258
  - common\_symbol\_table, 256
  - common\_symbols\_, 258
  - evtgba\_product, 256
  - evtgba\_product\_operands, 256
  - format\_acceptance\_condition, 257
  - format\_acceptance\_conditions, 257
  - format\_label, 257
  - format\_state, 257
  - init\_iter, 257
  - op\_, 258
  - pred\_iter, 257
  - succ\_iter, 257
- spot::evtgba\_reachable\_iterator, 258
  - ~evtgba\_reachable\_iterator, 261
  - add\_state, 262
  - automata\_, 263
  - end, 262
  - evtgba\_reachable\_iterator, 261
  - next\_state, 262
  - process\_link, 262
  - process\_state, 262
  - run, 262
  - seen, 263
  - seen\_map, 261
  - start, 263
- spot::evtgba\_reachable\_iterator\_breadth\_first, 263
  - add\_state, 266
  - automata\_, 268
  - end, 266
  - evtgba\_reachable\_iterator\_breadth\_first, 266
  - next\_state, 267
  - process\_link, 267
  - process\_state, 267
  - run, 267
  - seen, 268
  - seen\_map, 266
  - start, 267
  - todo, 268
- spot::evtgba\_reachable\_iterator\_depth\_first, 268
  - add\_state, 271
  - automata\_, 273
  - end, 271
  - evtgba\_reachable\_iterator\_depth\_first, 271
  - next\_state, 272
  - process\_link, 272
  - process\_state, 272
  - run, 272
- seen, 273
- seen\_map, 271
- start, 272
- todo, 273
- spot::explicit\_connected\_component, 273
  - ~explicit\_connected\_component, 276
  - condition, 276
  - has\_state, 276
  - index, 276
  - insert, 276
  - rem, 276
- spot::explicit\_connected\_component\_factory, 277
  - ~explicit\_connected\_component\_factory, 277
  - build, 278
- spot::explicit\_state\_conjunction, 278
  - ~explicit\_state\_conjunction, 281
  - add, 282
  - clone, 282
  - current\_state, 282
  - done, 282
  - explicit\_state\_conjunction, 281
  - first, 282
  - it\_, 283
  - next, 282
  - operator=, 283
  - saba\_state\_set\_t, 281
  - set\_, 283
- spot::fair\_kripke, 283
  - acceptance\_conditions\_of\_state, 286
  - all\_acceptance\_conditions, 286
  - compute\_support\_conditions, 287
  - compute\_support\_variables, 287
  - conditions\_of\_state, 287
  - format\_state, 287
  - get\_dict, 287
  - get\_init\_state, 288
  - kripke\_succ\_iter, 288
  - neg\_acceptance\_conditions, 288
  - number\_of\_acceptance\_conditions, 288
  - project\_state, 288
  - succ\_iter, 289
  - support\_conditions, 289
  - support\_variables, 290
  - transition\_annotation, 290
- spot::fair\_kripke\_succ\_iterator, 290
  - ~fair\_kripke\_succ\_iterator, 293
  - acc\_cond, 295
  - cond, 295
  - current\_acceptance\_conditions, 293
  - current\_condition, 293
  - current\_conditions, 293
  - current\_state, 294
  - done, 294
  - fair\_kripke\_succ\_iterator, 293

- first, 294
- next, 294
- spot::free\_list, 301
  - ~free\_list, 303
  - dump\_free\_list, 303
  - extend, 303
  - fl, 304
  - free\_count, 303
  - free\_list\_type, 302
  - insert, 303
  - pos\_lenght\_pair, 302
  - register\_n, 303
  - release\_n, 303
  - remove, 303, 304
- spot::future\_conditions\_collector, 304
  - ~future\_conditions\_collector, 308
  - all\_acceptance\_conditions, 308
  - aut\_, 312
  - compute\_support\_conditions, 309
  - compute\_support\_variables, 309
  - cond\_set, 308
  - fc\_map, 308
  - format\_state, 309
  - future\_conditions, 309
  - future\_conditions\_collector, 308
  - future\_conds\_, 312
  - get\_dict, 309
  - get\_init\_state, 309
  - map\_builder\_, 310
  - neg\_acceptance\_conditions, 310
  - number\_of\_acceptance\_conditions, 310
  - project\_state, 310
  - scc\_map\_, 312
  - scc\_of\_state, 310
  - show\_, 312
  - succ\_iter, 311
  - support\_conditions, 311
  - support\_variables, 311
  - transition\_annotation, 312
- spot::gspn\_exception, 312
  - err\_, 313
  - get\_err, 313
  - get\_where, 313
  - gspn\_exception, 313
  - where\_, 313
- spot::gspn\_interface, 314
  - ~gspn\_interface, 316
  - automaton, 316
  - dead\_, 316
  - dict\_, 316
  - env\_, 316
  - gspn\_interface, 316
- spot::gspn\_ssp\_interface, 317
  - ~gspn\_ssp\_interface, 319
- automaton, 319
- dict\_, 319
- env\_, 319
- gspn\_ssp\_interface, 319
- spot::kripke, 319
  - ~kripke, 322
  - acceptance\_conditions\_of\_state, 322
  - all\_acceptance\_conditions, 323
  - compute\_support\_conditions, 323
  - compute\_support\_variables, 323
  - conditions\_of\_state, 323
  - format\_state, 323
  - get\_dict, 323
  - get\_init\_state, 324
  - kripke\_succ\_iter, 324
  - neg\_acceptance\_conditions, 324
  - number\_of\_acceptance\_conditions, 324
  - project\_state, 324
  - succ\_iter, 325
  - support\_conditions, 325
  - support\_variables, 326
  - transition\_annotation, 326
- spot::loopless\_modular\_mixed\_radix\_gray\_code, 341
  - ~loopless\_modular\_mixed\_radix\_gray\_code, 343
  - a\_, 344
  - a\_first, 343
  - a\_last, 343
  - a\_next, 343
  - done, 343
  - done\_, 344
  - f\_, 344
  - first, 344
  - last, 344
  - loopless\_modular\_mixed\_radix\_gray\_code, 343
  - m\_, 344
  - n\_, 345
  - next, 344
  - non\_one\_radixes\_, 345
  - s\_, 345
- spot::ltl, 92
  - reduce\_tau03, 96
  - unabbreviate\_ltl, 96
- spot::ltl::atomic\_prop, 111
  - ~atomic\_prop, 115
  - accept, 115
  - atomic\_prop, 115
  - clone, 116
  - count\_, 117
  - destroy, 116
  - dump, 116
  - dump\_instances, 116

- env, 116
- env\_, 117
- hash, 116
- instance, 116
- instance\_count, 116
- instances, 117
- map, 115
- name, 117
- name\_, 117
- pair, 115
- ref\_, 117
- ref\_count\_, 117
- unref\_, 117
- spot::ltl::automatop, 118
  - ~automatop, 122
  - accept, 122
  - automatop, 122
  - children\_, 124
  - clone, 122
  - count\_, 124
  - destroy, 122
  - dump, 122
  - dump\_instances, 123
  - get\_nfa, 123
  - hash, 123
  - instance, 123
  - instance\_count, 123
  - instances, 124
  - is\_negated, 123
  - map, 121
  - negated\_, 125
  - nfa\_, 125
  - nth, 123, 124
  - ref\_, 124
  - ref\_count\_, 124
  - size, 124
  - triplet, 121
  - unref\_, 124
  - vec, 121
- spot::ltl::automatop::tripletcmp, 812
  - operator(), 812
- spot::ltl::binop, 148
  - ~binop, 152
  - accept, 152
  - binop, 152
  - clone, 152
  - count\_, 154
  - destroy, 152
  - dump, 153
  - dump\_instances, 153
  - Equiv, 152
  - first, 153
  - first\_, 155
  - hash, 153
  - Implies, 152
  - instance, 153
  - instance\_count, 153
  - instances, 155
  - M, 152
  - map, 151
  - op, 153
  - op\_, 155
  - op\_name, 154
  - pair, 151
  - pairf, 151
  - R, 152
  - ref\_, 154
  - ref\_count\_, 154
  - second, 154
  - second\_, 155
  - type, 152
  - U, 152
  - unref\_, 154
  - W, 152
  - Xor, 152
- spot::ltl::clone\_visitor, 156
  - ~clone\_visitor, 158
  - clone\_visitor, 158
  - recurse, 158
  - result, 158
  - result\_, 159
  - visit, 158, 159
- spot::ltl::const\_visitor, 168
  - ~const\_visitor, 168
  - visit, 168, 169
- spot::ltl::constant, 169
  - ~constant, 173
  - accept, 173
  - clone, 173
  - constant, 173
  - count\_, 175
  - destroy, 173
  - dump, 174
  - False, 173
  - false\_instance, 174
  - hash, 174
  - ref\_, 174
  - True, 173
  - true\_instance, 174
  - type, 173
  - unref\_, 174
  - val, 174
  - val\_, 175
  - val\_name, 175
- spot::ltl::declarative\_environment, 202
  - ~declarative\_environment, 205
  - declarative\_environment, 205
  - declare, 205

- get\_prop\_map, 205
  - name, 205
  - prop\_map, 205
  - props\_, 206
  - require, 205
- spot::ltl::default\_environment, 206
  - ~default\_environment, 209
  - default\_environment, 209
  - instance, 209
  - name, 209
  - require, 209
- spot::ltl::environment, 242
  - ~environment, 243
  - name, 243
  - require, 243
- spot::ltl::formula, 295
  - ~formula, 297
  - accept, 298
  - clone, 298
  - count\_, 299
  - destroy, 298
  - dump, 298
  - formula, 297
  - hash, 298
  - max\_count, 299
  - ref\_, 299
  - unref\_, 299
- spot::ltl::formula\_tree
  - False, 98
  - True, 98
- spot::ltl::formula\_ptr\_hash, 299
  - operator(), 300
- spot::ltl::formula\_ptr\_less\_than, 300
  - operator(), 300
- spot::ltl::formula\_tree, 96
  - arity, 98
  - instantiate, 98
  - node\_ptr, 97
- spot::ltl::formula\_tree::node, 366
  - ~node, 366
- spot::ltl::formula\_tree::node\_atomic, 366
  - i, 368
- spot::ltl::formula\_tree::node\_binop, 368
  - lhs, 369
  - op, 369
  - rhs, 369
- spot::ltl::formula\_tree::node\_multop, 370
  - lhs, 371
  - op, 371
  - rhs, 371
- spot::ltl::formula\_tree::node\_nfa, 371
  - children, 372
  - nfa, 372
- spot::ltl::formula\_tree::node\_unop, 372
  - child, 374
  - op, 374
- spot::ltl::language\_containment\_checker, 326
  - ~language\_containment\_checker, 328
  - branching\_postponement\_, 329
  - contained, 329
  - contained\_neg, 329
  - dict\_, 329
  - equal, 329
  - exprop\_, 329
  - fair\_loop\_approx\_, 330
  - incompatible\_, 329
  - language\_containment\_checker, 328
  - neg\_contained, 329
  - register\_formula\_, 329
  - symb\_merge\_, 330
  - trans\_map, 328
  - translated\_, 330
- spot::ltl::language\_containment\_checker::record\_, 430
  - incomp\_map, 431
  - incompatible, 431
  - translation, 431
- spot::ltl::ltl\_file, 345
  - in, 346
  - ltl\_file, 345
  - next, 346
- spot::ltl::multop, 348
  - ~multop, 353
  - accept, 353
  - And, 353
  - children\_, 356
  - clone, 353
  - count\_, 356
  - destroy, 353
  - dump, 354
  - dump\_instances, 354
  - hash, 354
  - instance, 354
  - instance\_count, 355
  - instances, 356
  - map, 352
  - multop, 353
  - nth, 355
  - op, 355
  - op\_, 356
  - op\_name, 355
  - Or, 353
  - pair, 352
  - ref\_, 355
  - ref\_count\_, 355
  - size, 355
  - type, 353
  - unref\_, 356



- vec, 352
- spot::ltl::multop::paircmp, 392
  - operator(), 392
- spot::ltl::nfa, 356
  - ~nfa, 359
  - add\_state, 360
  - add\_transition, 360
  - arity, 360
  - arity\_, 361
  - begin, 360
  - end, 360
  - finals\_, 361
  - format\_state, 360
  - get\_init\_state, 360
  - get\_name, 360
  - init\_, 361
  - is\_, 361
  - is\_final, 360
  - is\_loop, 360
  - is\_map, 359
  - iterator, 359
  - label, 359
  - name\_, 361
  - nfa, 359
  - operator=, 361
  - ptr, 359
  - set\_final, 361
  - set\_init\_state, 361
  - set\_name, 361
  - si\_, 361
  - si\_map, 359
  - state, 359
- spot::ltl::nfa::transition, 809
  - dst, 810
  - lbl, 810
- spot::ltl::postfix\_visitor, 420
  - ~postfix\_visitor, 423
  - doit, 423
  - doit\_default, 423
  - postfix\_visitor, 423
  - visit, 424
- spot::ltl::random\_ltl, 425
  - ~random\_ltl, 427
  - ap, 427
  - ap\_, 429
  - dump\_priorities, 428
  - generate, 428
  - parse\_options, 428
  - proba\_, 429
  - proba\_2\_, 429
  - random\_ltl, 427
  - total\_1\_, 429
  - total\_2\_, 429
  - total\_2\_and\_more\_, 429
  - update\_sums, 428
- spot::ltl::random\_ltl::op\_proba, 388
  - build, 389
  - builder, 389
  - min\_n, 389
  - name, 389
  - proba, 389
  - setup, 389
- spot::ltl::ref\_formula, 431
  - ~ref\_formula, 434
  - accept, 435
  - clone, 435
  - count\_, 436
  - destroy, 435
  - dump, 435
  - hash, 435
  - ref\_, 435
  - ref\_count\_, 436
  - ref\_counter\_, 436
  - ref\_formula, 434
  - unref\_, 436
- spot::ltl::simplify\_f\_g\_visitor, 487
  - ~simplify\_f\_g\_visitor, 490
  - recurse, 490
  - result, 490
  - result\_, 491
  - simplify\_f\_g\_visitor, 490
  - super, 490
  - visit, 490, 491
- spot::ltl::succ\_iterator, 546
  - i\_, 546
  - operator\*, 546
  - operator++, 546
  - succ\_iterator, 546
- spot::ltl::unabbreviate\_logic\_visitor, 813
  - ~unabbreviate\_logic\_visitor, 815
  - recurse, 815
  - result, 815
  - result\_, 816
  - super, 815
  - unabbreviate\_logic\_visitor, 815
  - visit, 815, 816
- spot::ltl::unabbreviate\_ltl\_visitor, 816
  - ~unabbreviate\_ltl\_visitor, 819
  - recurse, 819
  - result, 819
  - result\_, 820
  - super, 819
  - unabbreviate\_ltl\_visitor, 819
  - visit, 819, 820
- spot::ltl::unop, 820
  - ~unop, 825
  - accept, 825
  - child, 825

- child\_, 827
- clone, 825
- count\_, 827
- destroy, 825
- dump, 826
- dump\_instances, 826
- F, 824
- Finish, 824
- G, 824
- hash, 826
- instance, 826
- instance\_count, 826
- instances, 827
- map, 824
- Not, 824
- op, 826
- op\_, 827
- op\_name, 826
- pair, 824
- ref\_, 826
- ref\_count\_, 827
- type, 824
- unop, 825
- unref\_, 827
- X, 824
- spot::ltl::visitor, 831
  - ~visitor, 833
  - visit, 833
- spot::minato\_isop::local\_vars
  - FirstStep, 331
  - FourthStep, 331
  - SecondStep, 331
  - ThirdStep, 331
- spot::minato\_isop, 346
  - cube\_, 348
  - minato\_isop, 347
  - next, 347
  - ret\_, 348
  - todo\_, 348
- spot::minato\_isop::local\_vars, 330
  - f0\_max, 331
  - f0\_min, 331
  - f1\_max, 331
  - f1\_min, 331
  - f\_max, 331
  - f\_min, 331
  - g0, 331
  - g1, 331
  - local\_vars, 331
  - step, 332
  - v1, 332
  - vars, 332
- spot::nips\_exception, 362
  - err\_, 363
  - err\_defined\_, 363
  - get\_err, 362
  - get\_err\_defined, 362
  - get\_where, 363
  - nips\_exception, 362
  - where\_, 363
- spot::nips\_interface, 363
  - ~nips\_interface, 365
  - automaton, 365
  - bytecode\_, 365
  - dict\_, 365
  - has\_monitor, 365
  - nips\_interface, 365
  - nipsvm\_, 365
- spot::numbered\_state\_heap, 374
  - ~numbered\_state\_heap, 376
  - find, 376
  - index, 377
  - insert, 377
  - iterator, 377
  - size, 377
  - state\_index, 376
  - state\_index\_p, 376
- spot::numbered\_state\_heap\_const\_iterator, 378
  - ~numbered\_state\_heap\_const\_iterator, 378
  - done, 378
  - first, 378
  - get\_index, 378
  - get\_state, 378
  - next, 379
- spot::numbered\_state\_heap\_factory, 379
  - ~numbered\_state\_heap\_factory, 380
  - build, 380
- spot::numbered\_state\_heap\_hash\_map, 380
  - ~numbered\_state\_heap\_hash\_map, 383
  - find, 384
  - h, 385
  - hash\_type, 383
  - index, 384
  - insert, 384
  - iterator, 384
  - size, 385
  - state\_index, 383
  - state\_index\_p, 383
- spot::numbered\_state\_heap\_hash\_map\_factory, 385
  - ~numbered\_state\_heap\_hash\_map\_factory, 387
  - build, 387
  - instance, 387
  - numbered\_state\_heap\_hash\_map\_factory, 387
- spot::option\_map, 389
  - get, 390
  - operator<<, 392

- options\_, 392
- parse\_options, 391
- set, 391
- spot::parity\_game\_graph, 392
  - ~parity\_game\_graph, 396
  - add\_state, 396
  - automata\_, 398
  - build\_graph, 396
  - duplicator\_vertice\_, 398
  - end, 396
  - get\_relation, 396
  - lift, 396
  - nb\_node\_parity\_game, 398
  - next\_state, 397
  - parity\_game\_graph, 396
  - print, 397
  - process\_link, 397
  - process\_state, 397
  - run, 398
  - seen, 398
  - seen\_map, 396
  - spoiler\_vertice\_, 398
  - start, 398
  - tgba\_state\_, 399
  - todo, 399
  - want\_state, 398
- spot::parity\_game\_graph\_delayed, 399
  - ~parity\_game\_graph\_delayed, 403
  - add\_duplicator\_node\_delayed, 404
  - add\_spoiler\_node\_delayed, 404
  - add\_state, 404
  - automata\_, 406
  - bdd\_v, 403
  - build\_graph, 404
  - build\_recurse\_successor\_duplicator, 404
  - build\_recurse\_successor\_spoiler, 404
  - duplicator\_vertice\_, 406
  - end, 404
  - get\_relation, 404
  - lift, 405
  - nb\_node\_parity\_game, 406
  - nb\_set\_acc\_cond, 405
  - next\_state, 405
  - parity\_game\_graph\_delayed, 403
  - print, 405
  - process\_link, 405
  - process\_state, 405
  - run, 406
  - seen, 406
  - seen\_map, 403
  - spoiler\_vertice\_, 407
  - start, 406
  - sub\_set\_acc\_cond\_, 407
  - tgba\_state\_, 407
- todo, 407
- want\_state, 406
- spot::parity\_game\_graph\_direct, 407
  - ~parity\_game\_graph\_direct, 411
  - add\_state, 411
  - automata\_, 413
  - build\_graph, 411
  - build\_link, 411
  - duplicator\_vertice\_, 413
  - end, 411
  - get\_relation, 411
  - lift, 412
  - nb\_node\_parity\_game, 413
  - next\_state, 412
  - parity\_game\_graph\_direct, 411
  - print, 412
  - process\_link, 412
  - process\_state, 412
  - run, 413
  - seen, 413
  - seen\_map, 411
  - spoiler\_vertice\_, 413
  - start, 413
  - tgba\_state\_, 414
  - todo, 414
  - want\_state, 413
- spot::ptr\_hash, 424
  - operator(), 425
- spot::rsymbol, 436
  - ~rsymbol, 438
  - operator const symbol \*, 438
  - operator<, 438
  - operator=, 439
  - operator==, 439
  - rsymbol, 438
  - s\_, 439
- spot::saba, 439
  - ~saba, 441
  - all\_acceptance\_conditions, 441
  - format\_state, 441
  - get\_dict, 442
  - get\_init\_state, 442
  - num\_acc\_, 443
  - number\_of\_acceptance\_conditions, 442
  - saba, 441
  - succ\_iter, 442
- spot::saba\_complement\_tgba, 443
  - ~saba\_complement\_tgba, 446
  - all\_acceptance\_conditions, 446
  - automaton\_, 447
  - format\_state, 446
  - get\_dict, 446
  - get\_init\_state, 446
  - nb\_states\_, 447

- number\_of\_acceptance\_conditions, 447
- saba\_complement\_tgba, 446
- succ\_iter, 447
- the\_acceptance\_cond\_, 447
- spot::saba\_reachable\_iterator, 447
  - ~saba\_reachable\_iterator, 450
  - add\_state, 451
  - automata\_, 452
  - end, 451
  - next\_state, 451
  - process\_link, 451
  - process\_state, 451
  - process\_state\_conjunction, 452
  - run, 452
  - saba\_reachable\_iterator, 450
  - seen, 452
  - seen\_map, 450
  - start, 452
  - want\_state, 452
- spot::saba\_reachable\_iterator\_breadth\_first, 453
  - add\_state, 455
  - automata\_, 457
  - end, 456
  - next\_state, 456
  - process\_link, 456
  - process\_state, 456
  - process\_state\_conjunction, 456
  - run, 457
  - saba\_reachable\_iterator\_breadth\_first, 455
  - seen, 457
  - seen\_map, 455
  - start, 457
  - todo, 457
  - want\_state, 457
- spot::saba\_reachable\_iterator\_depth\_first, 458
  - add\_state, 460
  - automata\_, 462
  - end, 461
  - next\_state, 461
  - process\_link, 461
  - process\_state, 461
  - process\_state\_conjunction, 461
  - run, 462
  - saba\_reachable\_iterator\_depth\_first, 460
  - seen, 462
  - seen\_map, 460
  - start, 462
  - todo, 462
  - want\_state, 462
- spot::saba\_state, 463
  - ~saba\_state, 463
  - acceptance\_conditions, 463
  - clone, 463
  - compare, 464
  - hash, 464
- spot::saba\_state\_conjunction, 464
  - ~saba\_state\_conjunction, 466
  - clone, 466
  - current\_state, 466
  - done, 466
  - first, 467
  - next, 467
- spot::saba\_state\_ptr\_equal, 467
  - operator(), 468
- spot::saba\_state\_ptr\_hash, 468
  - operator(), 468
- spot::saba\_state\_ptr\_less\_than, 469
  - operator(), 469
- spot::saba\_state\_shared\_ptr\_equal, 469
  - operator(), 470
- spot::saba\_state\_shared\_ptr\_hash, 470
  - operator(), 470
- spot::saba\_state\_shared\_ptr\_less\_than, 471
  - operator(), 471
- spot::saba\_succ\_iterator, 471
  - ~saba\_succ\_iterator, 472
  - current\_condition, 472
  - current\_conjunction, 472
  - done, 472
  - first, 473
  - next, 473
- spot::scc\_map, 475
  - ~scc\_map, 479
  - acc\_set\_of, 479
  - accepting, 479
  - ap\_set\_of, 479
  - aprec\_set\_of, 479
  - arc\_acc\_, 482
  - arc\_cond\_, 482
  - aut\_, 482
  - build\_map, 480
  - cond\_set, 478
  - cond\_set\_of, 480
  - get\_aut, 480
  - h\_, 482
  - hash\_type, 478
  - initial, 480
  - num\_, 482
  - pair\_state\_iter, 478
  - relabel\_component, 480
  - root\_, 482
  - scc\_count, 480
  - scc\_map, 479
  - scc\_map\_, 482
  - scc\_map\_type, 478
  - scc\_of\_state, 481
  - self\_loops, 481
  - self\_loops\_, 482

- stack\_type, 478
- states\_of, 481
- succ, 481
- succ\_type, 478
- todo\_, 482
- update\_supp\_rec, 481
- useful\_acc\_of, 481
- spot::scc\_map::scc, 473
  - acc, 474
  - conds, 474
  - index, 474
  - scc, 474
  - states, 474
  - succ, 474
  - supp, 475
  - supp\_rec, 475
  - trivial, 475
  - useful\_acc, 475
- spot::scc\_stack, 483
  - clear\_rem, 484
  - empty, 484
  - pop, 484
  - push, 484
  - rem, 484
  - s, 485
  - size, 484
  - stack\_type, 483
  - top, 484
- spot::scc\_stack::connected\_component, 159
  - condition, 161
  - connected\_component, 161
  - index, 161
  - rem, 161
- spot::scc\_stats, 485
  - acc\_paths, 485
  - acc\_scc, 485
  - dead\_paths, 486
  - dead\_scc, 486
  - dump, 485
  - scc\_total, 486
  - self\_loops, 486
  - useful\_acc, 486
  - useless\_scc\_map, 486
- spot::sccs\_set, 486
  - sccs, 487
  - size, 487
- spot::spoiler\_node, 495
  - ~spoiler\_node, 497
  - add\_pred, 497
  - add\_succ, 497
  - compare, 497
  - del\_pred, 497
  - del\_succ, 498
  - get\_duplicator\_node, 498
  - get\_nb\_succ, 498
  - get\_pair, 498
  - get\_spoiler\_node, 498
  - lnode\_pred, 499
  - lnode\_succ, 499
  - not\_win, 499
  - num\_, 499
  - prune, 498
  - sc\_, 499
  - set\_win, 498
  - spoiler\_node, 497
  - succ\_to\_string, 498
  - to\_string, 498
- spot::spoiler\_node\_delayed, 499
  - ~spoiler\_node\_delayed, 502
  - acceptance\_condition\_visited\_, 504
  - add\_pred, 502
  - add\_succ, 502
  - compare, 503
  - del\_pred, 503
  - del\_succ, 503
  - get\_acceptance\_condition\_visited, 503
  - get\_duplicator\_node, 503
  - get\_lead\_2\_acc\_all, 503
  - get\_nb\_succ, 503
  - get\_pair, 503
  - get\_progress\_measure, 503
  - get\_spoiler\_node, 503
  - lead\_2\_acc\_all\_, 504
  - lnode\_pred, 504
  - lnode\_succ, 504
  - not\_win, 504
  - num\_, 504
  - progress\_measure\_, 505
  - prune, 503
  - sc\_, 505
  - seen\_, 505
  - set\_lead\_2\_acc\_all, 504
  - set\_win, 504
  - spoiler\_node\_delayed, 502
  - succ\_to\_string, 504
  - to\_string, 504
- spot::state, 512
  - ~state, 513
  - clone, 513
  - compare, 514
  - hash, 514
- spot::state\_bdd, 514
  - as\_bdd, 517
  - clone, 517
  - compare, 517
  - hash, 517
  - state\_, 518
  - state\_bdd, 517

- spot::state\_evtgba\_explicit, 518
  - ~state\_evtgba\_explicit, 521
  - clone, 521
  - compare, 521
  - get\_state, 521
  - hash, 521
  - state\_, 522
  - state\_evtgba\_explicit, 521
- spot::state\_explicit, 522
  - ~state\_explicit, 525
  - clone, 525
  - compare, 525
  - get\_state, 525
  - hash, 525
  - state\_, 526
  - state\_explicit, 525
- spot::state\_product, 526
  - ~state\_product, 529
  - clone, 530
  - compare, 530
  - hash, 530
  - left, 530
  - left\_, 531
  - right, 530
  - right\_, 531
  - state\_product, 529
- spot::state\_ptr\_equal, 531
  - operator(), 531
- spot::state\_ptr\_hash, 532
  - operator(), 532
- spot::state\_ptr\_less\_than, 532
  - operator(), 533
- spot::state\_set, 533
  - ~state\_set, 536
  - clone, 536
  - compare, 536
  - delete\_me\_, 537
  - get\_state, 536
  - hash, 537
  - s\_, 537
  - state\_set, 536
- spot::state\_shared\_ptr\_equal, 537
  - operator(), 538
- spot::state\_shared\_ptr\_hash, 538
  - operator(), 538
- spot::state\_shared\_ptr\_less\_than, 539
  - operator(), 539
- spot::state\_union, 539
  - ~state\_union, 542
  - clone, 543
  - compare, 543
  - hash, 543
  - left, 543
  - left\_, 544
  - right, 543
  - right\_, 544
  - state\_union, 542
- spot::string\_hash, 545
  - operator(), 545
- spot::symbol, 548
  - ~symbol, 549
  - dump\_instances, 549
  - instance, 549
  - instance\_count, 549
  - instances\_, 550
  - map, 549
  - name, 549
  - name\_, 550
  - ref, 550
  - ref\_count\_, 550
  - refs\_, 550
  - symbol, 549
  - unref, 550
- spot::taa\_succ\_iterator, 550
  - ~taa\_succ\_iterator, 554
  - all\_acceptance\_conditions\_, 555
  - bounds\_t, 553
  - current\_acceptance\_conditions, 554
  - current\_condition, 554
  - current\_state, 554
  - done, 554
  - first, 555
  - i\_, 555
  - iterator, 553
  - iterator\_pair, 553
  - next, 555
  - seen\_, 555
  - seen\_map, 553
  - succ\_, 555
  - taa\_succ\_iterator, 554
- spot::taa\_succ\_iterator::distance\_sort, 210
  - operator(), 210
- spot::taa\_tgba, 556
  - ~taa\_tgba, 561
  - add\_condition, 561
  - all\_acceptance\_conditions, 561
  - all\_acceptance\_conditions\_, 564
  - all\_acceptance\_conditions\_computed\_, 564
  - compute\_support\_conditions, 561
  - compute\_support\_variables, 561
  - dict\_, 564
  - format\_state, 561
  - get\_dict, 562
  - get\_init\_state, 562
  - init\_, 564
  - neg\_acceptance\_conditions, 562
  - neg\_acceptance\_conditions\_, 565
  - number\_of\_acceptance\_conditions, 562

- operator=, 562
- project\_state, 563
- ss\_vec, 560
- state, 560
- state\_set, 560
- state\_set\_vec\_, 565
- succ\_iter, 563
- support\_conditions, 563
- support\_variables, 564
- taa\_tgba, 561
- transition\_annotation, 564
- spot::taa\_tgba::transition, 808
  - acceptance\_conditions, 809
  - condition, 809
  - dst, 809
- spot::taa\_tgba\_formula, 565
  - ~taa\_tgba\_formula, 570
  - add\_acceptance\_condition, 570
  - add\_condition, 570
  - all\_acceptance\_conditions, 570
  - all\_acceptance\_conditions\_, 575
  - all\_acceptance\_conditions\_computed\_, 575
  - clone\_if, 571
  - compute\_support\_conditions, 571
  - compute\_support\_variables, 571
  - create\_transition, 571
  - dict\_, 575
  - format\_state, 571
  - get\_dict, 572
  - get\_init\_state, 572
  - init\_, 575
  - label\_t, 569
  - label\_to\_string, 572
  - name\_state\_map\_, 575
  - neg\_acceptance\_conditions, 572
  - neg\_acceptance\_conditions\_, 575
  - ns\_map, 569
  - number\_of\_acceptance\_conditions, 573
  - output, 573
  - project\_state, 573
  - set\_init\_state, 573
  - sn\_map, 570
  - ss\_vec, 570
  - state, 570
  - state\_name\_map\_, 575
  - state\_set, 570
  - state\_set\_vec\_, 575
  - succ\_iter, 573
  - support\_conditions, 574
  - support\_variables, 574
  - taa\_tgba\_formula, 570
  - transition\_annotation, 574
- spot::taa\_tgba\_labelled, 576
  - add\_acceptance\_condition, 581
  - add\_condition, 581
  - add\_state, 582
  - add\_state\_set, 582
  - all\_acceptance\_conditions, 582
  - all\_acceptance\_conditions\_, 587
  - all\_acceptance\_conditions\_computed\_, 587
  - clone\_if, 582
  - compute\_support\_conditions, 582
  - compute\_support\_variables, 583
  - create\_transition, 583
  - dict\_, 587
  - format\_state, 583
  - format\_state\_set, 583
  - get\_dict, 583
  - get\_init\_state, 584
  - init\_, 587
  - label\_t, 581
  - label\_to\_string, 584
  - name\_state\_map\_, 587
  - neg\_acceptance\_conditions, 584
  - neg\_acceptance\_conditions\_, 587
  - ns\_map, 581
  - number\_of\_acceptance\_conditions, 584
  - output, 584
  - project\_state, 585
  - set\_init\_state, 585
  - sn\_map, 581
  - ss\_vec, 581
  - state, 581
  - state\_name\_map\_, 587
  - state\_set, 581
  - state\_set\_vec\_, 587
  - succ\_iter, 585
  - support\_conditions, 586
  - support\_variables, 586
  - taa\_tgba\_labelled, 581
  - transition\_annotation, 586
- spot::taa\_tgba\_string, 588
  - ~taa\_tgba\_string, 592
  - add\_acceptance\_condition, 592
  - add\_condition, 592
  - all\_acceptance\_conditions, 592
  - all\_acceptance\_conditions\_, 597
  - all\_acceptance\_conditions\_computed\_, 597
  - clone\_if, 593
  - compute\_support\_conditions, 593
  - compute\_support\_variables, 593
  - create\_transition, 593
  - dict\_, 597
  - format\_state, 593
  - get\_dict, 594
  - get\_init\_state, 594
  - init\_, 597
  - label\_t, 591

- label\_to\_string, 594
- name\_state\_map\_, 597
- neg\_acceptance\_conditions, 594
- neg\_acceptance\_conditions\_, 597
- ns\_map, 591
- number\_of\_acceptance\_conditions, 595
- output, 595
- project\_state, 595
- set\_init\_state, 595
- sn\_map, 591
- ss\_vec, 592
- state, 592
- state\_name\_map\_, 597
- state\_set, 592
- state\_set\_vec\_, 598
- succ\_iter, 595
- support\_conditions, 596
- support\_variables, 596
- taa\_tgba\_string, 592
- transition\_annotation, 596
- spot::tgba, 598
  - ~tgba, 602
  - all\_acceptance\_conditions, 602
  - compute\_support\_conditions, 602
  - compute\_support\_variables, 602
  - format\_state, 602
  - get\_dict, 603
  - get\_init\_state, 603
  - last\_support\_conditions\_input\_, 605
  - last\_support\_conditions\_output\_, 605
  - last\_support\_variables\_input\_, 605
  - last\_support\_variables\_output\_, 606
  - neg\_acceptance\_conditions, 603
  - num\_acc\_, 606
  - number\_of\_acceptance\_conditions, 603
  - project\_state, 604
  - succ\_iter, 604
  - support\_conditions, 604
  - support\_variables, 605
  - tgba, 602
  - transition\_annotation, 605
- spot::tgba\_bdd\_concrete, 606
  - ~tgba\_bdd\_concrete, 610
  - all\_acceptance\_conditions, 611
  - compute\_support\_conditions, 611
  - compute\_support\_variables, 611
  - data\_, 615
  - delete\_unaccepting\_scc, 611
  - format\_state, 611
  - get\_core\_data, 612
  - get\_dict, 612
  - get\_init\_bdd, 612
  - get\_init\_state, 612
  - init\_, 615
  - neg\_acceptance\_conditions, 612
  - number\_of\_acceptance\_conditions, 613
  - operator=, 613
  - project\_state, 613
  - set\_init\_state, 613
  - succ\_iter, 613
  - support\_conditions, 614
  - support\_variables, 614
  - tgba\_bdd\_concrete, 610, 611
  - transition\_annotation, 614
- spot::tgba\_bdd\_concrete\_factory, 615
  - ~tgba\_bdd\_concrete\_factory, 618
  - acc\_, 620
  - acc\_map\_, 618
  - constrain\_relation, 618
  - create\_anonymous\_state, 618
  - create\_atomic\_prop, 619
  - create\_state, 619
  - data\_, 620
  - declare\_acceptance\_condition, 619
  - finish, 619
  - get\_core\_data, 620
  - get\_dict, 620
  - tgba\_bdd\_concrete\_factory, 618
- spot::tgba\_bdd\_core\_data, 620
  - acc\_set, 624
  - acceptance\_conditions, 624
  - all\_acceptance\_conditions, 625
  - declare\_acceptance\_condition, 623
  - declare\_atomic\_prop, 623
  - declare\_now\_next, 624
  - delete\_unaccepting\_scc, 624
  - dict, 625
  - infinitely\_often, 624
  - negacc\_set, 625
  - next\_set, 625
  - notacc\_set, 625
  - notnext\_set, 625
  - notnow\_set, 625
  - notvar\_set, 626
  - now\_set, 626
  - nownext\_set, 626
  - operator=, 624
  - relation, 626
  - tgba\_bdd\_core\_data, 623
  - var\_set, 626
  - varandnext\_set, 626
- spot::tgba\_bdd\_factory, 627
  - ~tgba\_bdd\_factory, 628
  - get\_core\_data, 628
- spot::tgba\_explicit, 628
  - ~tgba\_explicit, 633
  - add\_acceptance\_condition, 633
  - add\_acceptance\_conditions, 633



- add\_condition, 633
- add\_conditions, 633
- add\_default\_init, 633
- all\_acceptance\_conditions, 633
- all\_acceptance\_conditions\_, 637
- all\_acceptance\_conditions\_computed\_, 637
- compute\_support\_conditions, 634
- compute\_support\_variables, 634
- copy\_acceptance\_conditions\_of, 634
- create\_transition, 634
- dict\_, 637
- format\_state, 634
- get\_acceptance\_condition, 634
- get\_dict, 635
- get\_init\_state, 635
- has\_acceptance\_condition, 635
- init\_, 638
- neg\_acceptance\_conditions, 635
- neg\_acceptance\_conditions\_, 638
- number\_of\_acceptance\_conditions, 635
- operator=, 635
- project\_state, 636
- set\_acceptance\_conditions, 636
- state, 632
- succ\_iter, 636
- support\_conditions, 636
- support\_variables, 637
- tgba\_explicit, 633
- transition\_annotation, 637
- spot::tgba\_explicit::transition, 806
  - acceptance\_conditions, 807
  - condition, 807
  - dest, 807
- spot::tgba\_explicit\_formula, 638
  - ~tgba\_explicit\_formula, 643
  - add\_acceptance\_condition, 643
  - add\_acceptance\_conditions, 643
  - add\_condition, 643
  - add\_conditions, 644
  - add\_default\_init, 644
  - add\_state, 644
  - all\_acceptance\_conditions, 644
  - all\_acceptance\_conditions\_, 649
  - all\_acceptance\_conditions\_computed\_, 649
  - complement\_all\_acceptance\_conditions, 644
  - compute\_support\_conditions, 644
  - compute\_support\_variables, 644
  - copy\_acceptance\_conditions\_of, 645
  - create\_transition, 645
  - declare\_acceptance\_condition, 645
  - dict\_, 649
  - format\_state, 645
  - get\_acceptance\_condition, 645
  - get\_dict, 645
  - get\_init\_state, 646
  - get\_label, 646
  - has\_acceptance\_condition, 646
  - has\_state, 646
  - init\_, 649
  - label\_t, 643
  - merge\_transitions, 646
  - name\_state\_map\_, 649
  - neg\_acceptance\_conditions, 646
  - neg\_acceptance\_conditions\_, 649
  - ns\_map, 643
  - number\_of\_acceptance\_conditions, 647
  - project\_state, 647
  - set\_acceptance\_conditions, 647
  - set\_init\_state, 647
  - sn\_map, 643
  - state, 643
  - state\_name\_map\_, 649
  - succ\_iter, 647
  - support\_conditions, 648
  - support\_variables, 648
  - tgba\_explicit\_formula, 643
  - transition\_annotation, 648
- spot::tgba\_explicit\_labelled, 650
  - ~tgba\_explicit\_labelled, 655
  - add\_acceptance\_condition, 655
  - add\_acceptance\_conditions, 655
  - add\_condition, 656
  - add\_conditions, 656
  - add\_default\_init, 656
  - add\_state, 656
  - all\_acceptance\_conditions, 656
  - all\_acceptance\_conditions\_, 661
  - all\_acceptance\_conditions\_computed\_, 661
  - complement\_all\_acceptance\_conditions, 656
  - compute\_support\_conditions, 657
  - compute\_support\_variables, 657
  - copy\_acceptance\_conditions\_of, 657
  - create\_transition, 657
  - declare\_acceptance\_condition, 657
  - dict\_, 661
  - format\_state, 657
  - get\_acceptance\_condition, 658
  - get\_dict, 658
  - get\_init\_state, 658
  - get\_label, 658
  - has\_acceptance\_condition, 658
  - has\_state, 659
  - init\_, 661
  - label\_t, 655
  - merge\_transitions, 659
  - name\_state\_map\_, 661
  - neg\_acceptance\_conditions, 659
  - neg\_acceptance\_conditions\_, 662

- ns\_map, 655
- number\_of\_acceptance\_conditions, 659
- project\_state, 659
- set\_acceptance\_conditions, 659
- set\_init\_state, 660
- sn\_map, 655
- state, 655
- state\_name\_map\_, 662
- succ\_iter, 660
- support\_conditions, 660
- support\_variables, 660
- tgba\_explicit\_labelled, 655
- transition\_annotation, 661
- spot::tgba\_explicit\_string, 662
  - ~tgba\_explicit\_string, 667
  - add\_acceptance\_condition, 667
  - add\_acceptance\_conditions, 667
  - add\_condition, 667
  - add\_conditions, 668
  - add\_default\_init, 668
  - add\_state, 668
  - all\_acceptance\_conditions, 668
  - all\_acceptance\_conditions\_, 673
  - all\_acceptance\_conditions\_computed\_, 673
  - complement\_all\_acceptance\_conditions, 668
  - compute\_support\_conditions, 668
  - compute\_support\_variables, 669
  - copy\_acceptance\_conditions\_of, 669
  - create\_transition, 669
  - declare\_acceptance\_condition, 669
  - dict\_, 673
  - format\_state, 669
  - get\_acceptance\_condition, 670
  - get\_dict, 670
  - get\_init\_state, 670
  - get\_label, 670
  - has\_acceptance\_condition, 670
  - has\_state, 671
  - init\_, 673
  - label\_t, 667
  - merge\_transitions, 671
  - name\_state\_map\_, 674
  - neg\_acceptance\_conditions, 671
  - neg\_acceptance\_conditions\_, 674
  - ns\_map, 667
  - number\_of\_acceptance\_conditions, 671
  - project\_state, 671
  - set\_acceptance\_conditions, 672
  - set\_init\_state, 672
  - sn\_map, 667
  - state, 667
  - state\_name\_map\_, 674
  - succ\_iter, 672
  - support\_conditions, 672
  - support\_variables, 673
  - tgba\_explicit\_string, 667
  - transition\_annotation, 673
- spot::tgba\_explicit\_succ\_iterator, 674
  - all\_acceptance\_conditions\_, 678
  - current\_acceptance\_conditions, 677
  - current\_condition, 677
  - current\_state, 677
  - done, 678
  - first, 678
  - i\_, 678
  - next, 678
  - s\_, 679
  - tgba\_explicit\_succ\_iterator, 677
- spot::tgba\_kv\_complement, 679
  - ~tgba\_kv\_complement, 683
  - acc\_list\_, 686
  - all\_acceptance\_conditions, 683
  - automaton\_, 686
  - compute\_support\_conditions, 683
  - compute\_support\_variables, 684
  - format\_state, 684
  - get\_acc\_list, 684
  - get\_dict, 684
  - get\_init\_state, 684
  - nb\_states\_, 687
  - neg\_acceptance\_conditions, 684
  - number\_of\_acceptance\_conditions, 685
  - project\_state, 685
  - succ\_iter, 685
  - support\_conditions, 686
  - support\_variables, 686
  - tgba\_kv\_complement, 683
  - the\_acceptance\_cond\_, 687
  - transition\_annotation, 686
- spot::tgba\_product, 687
  - ~tgba\_product, 691
  - all\_acceptance\_conditions, 691
  - all\_acceptance\_conditions\_, 695
  - compute\_support\_conditions, 692
  - compute\_support\_variables, 692
  - dict\_, 695
  - format\_state, 692
  - get\_dict, 692
  - get\_init\_state, 692
  - left\_, 695
  - left\_acc\_complement\_, 695
  - neg\_acceptance\_conditions, 692
  - neg\_acceptance\_conditions\_, 695
  - number\_of\_acceptance\_conditions, 693
  - operator=, 693
  - project\_state, 693
  - right\_, 695
  - right\_acc\_complement\_, 695

- right\_common\_acc\_, 695
- succ\_iter, 693
- support\_conditions, 694
- support\_variables, 694
- tgba\_product, 691
- transition\_annotation, 694
- spot::tgba\_reachable\_iterator, 695
  - ~tgba\_reachable\_iterator, 698
  - add\_state, 699
  - automata\_, 700
  - end, 699
  - next\_state, 699
  - process\_link, 699
  - process\_state, 699
  - run, 699
  - seen, 700
  - seen\_map, 698
  - start, 700
  - tgba\_reachable\_iterator, 698
  - want\_state, 700
- spot::tgba\_reachable\_iterator\_breadth\_first, 700
  - add\_state, 703
  - automata\_, 705
  - end, 703
  - next\_state, 704
  - process\_link, 704
  - process\_state, 704
  - run, 704
  - seen, 705
  - seen\_map, 703
  - start, 704
  - tgba\_reachable\_iterator\_breadth\_first, 703
  - todo, 705
  - want\_state, 705
- spot::tgba\_reachable\_iterator\_depth\_first, 705
  - add\_state, 708
  - automata\_, 710
  - end, 708
  - next\_state, 709
  - process\_link, 709
  - process\_state, 709
  - run, 709
  - seen, 710
  - seen\_map, 708
  - start, 709
  - tgba\_reachable\_iterator\_depth\_first, 708
  - todo, 710
  - want\_state, 710
- spot::tgba\_reduc, 710
  - ~tgba\_reduc, 717
  - add\_acceptance\_condition, 717
  - add\_acceptance\_conditions, 717
  - add\_condition, 717
  - add\_conditions, 717
  - add\_default\_init, 717
  - add\_state, 717
  - all\_acceptance\_conditions, 718
  - all\_acceptance\_conditions\_, 726
  - all\_acceptance\_conditions\_computed\_, 726
  - automata\_, 727
  - complement\_all\_acceptance\_conditions, 718
  - compute\_support\_conditions, 718
  - compute\_support\_variables, 718
  - copy\_acceptance\_conditions\_of, 718
  - create\_transition, 719
  - declare\_acceptance\_condition, 719
  - delete\_scc, 719
  - delete\_transitions, 719
  - dict\_, 727
  - display\_rel\_sim, 719
  - display\_scc, 719
  - end, 720
  - format\_state, 720
  - get\_acceptance\_condition, 720
  - get\_dict, 720
  - get\_init\_state, 720
  - get\_label, 720, 721
  - has\_acceptance\_condition, 721
  - has\_state, 721
  - init\_, 727
  - is\_not\_accepting, 721
  - is\_terminal, 721
  - label\_t, 716
  - merge\_state, 721
  - merge\_state\_delayed, 721
  - merge\_transitions, 722
  - name\_state\_map\_, 727
  - nb\_set\_acc\_cond, 722
  - neg\_acceptance\_conditions, 722
  - neg\_acceptance\_conditions\_, 727
  - next\_state, 722
  - ns\_map, 716
  - number\_of\_acceptance\_conditions, 722
  - process\_link, 722, 723
  - process\_state, 723
  - project\_state, 723
  - quotient\_state, 723
  - redirect\_transition, 724
  - remove\_acc, 724
  - remove\_component, 724
  - remove\_predecessor\_state, 724
  - remove\_scc, 724
  - remove\_state, 724
  - run, 724
  - seen, 727
  - seen\_map, 716
  - set\_acceptance\_conditions, 724
  - set\_init\_state, 725

- sn\_map, 716
- sp\_map, 716
- start, 725
- state, 716
- state\_name\_map\_, 727
- state\_predecessor\_map\_, 727
- succ\_iter, 725
- support\_conditions, 725
- support\_variables, 726
- tgba\_reduc, 717
- todo, 727
- transition\_annotation, 726
- want\_state, 726
- spot::tgba\_run, 728
  - ~tgba\_run, 729
  - cycle, 729
  - operator=, 729
  - prefix, 729
  - steps, 728
  - tgba\_run, 729
- spot::tgba\_run::step, 544
  - acc, 545
  - label, 545
  - s, 545
- spot::tgba\_run\_dotty\_decorator, 729
  - ~tgba\_run\_dotty\_decorator, 732
  - instance, 732
  - link\_decl, 733
  - map\_, 733
  - run\_, 733
  - state\_decl, 733
  - step\_map, 732
  - step\_num, 732
  - step\_set, 732
  - tgba\_run\_dotty\_decorator, 732
- spot::tgba\_safracomplement, 734
  - ~tgba\_safracomplement, 737
  - acceptance\_cond\_vec\_, 740
  - all\_acceptance\_cond\_, 740
  - all\_acceptance\_conditions, 737
  - automaton\_, 740
  - compute\_support\_conditions, 737
  - compute\_support\_variables, 737
  - format\_state, 737
  - get\_dict, 738
  - get\_init\_state, 738
  - get\_safracomplement, 738
  - neg\_acceptance\_cond\_, 740
  - neg\_acceptance\_conditions, 738
  - number\_of\_acceptance\_conditions, 738
  - project\_state, 738
  - safracomplement\_, 740
  - succ\_iter, 739
  - support\_conditions, 739
  - support\_variables, 740
  - tgba\_safracomplement, 737
  - transition\_annotation, 740
- spot::tgba\_sba\_proxy, 741
  - a\_, 747
  - acc\_cycle\_, 747
  - all\_acceptance\_conditions, 744
  - compute\_support\_conditions, 744
  - compute\_support\_variables, 744
  - cycle\_list, 744
  - cycle\_start\_, 747
  - format\_state, 745
  - get\_dict, 745
  - get\_init\_state, 745
  - neg\_acceptance\_conditions, 745
  - number\_of\_acceptance\_conditions, 745
  - project\_state, 746
  - state\_is\_accepting, 746
  - succ\_iter, 746
  - support\_conditions, 746
  - support\_variables, 747
  - tgba\_sba\_proxy, 744
  - transition\_annotation, 747
- spot::tgba\_scc, 748
  - ~tgba\_scc, 751
  - all\_acceptance\_conditions, 751
  - aut\_, 754
  - compute\_support\_conditions, 751
  - compute\_support\_variables, 751
  - format\_state, 751
  - get\_dict, 752
  - get\_init\_state, 752
  - neg\_acceptance\_conditions, 752
  - number\_of\_acceptance\_conditions, 752
  - project\_state, 752
  - scc\_map\_, 754
  - scc\_of\_state, 753
  - show\_, 754
  - succ\_iter, 753
  - support\_conditions, 753
  - support\_variables, 753
  - tgba\_scc, 751
  - transition\_annotation, 754
- spot::tgba\_sgba\_proxy, 754
  - ~tgba\_sgba\_proxy, 758
  - a\_, 761
  - acceptance\_condition\_, 761
  - all\_acceptance\_conditions, 758
  - compute\_support\_conditions, 758
  - compute\_support\_variables, 758
  - emulate\_acc\_cond\_, 761
  - format\_state, 758
  - get\_dict, 759
  - get\_init\_state, 759

- neg\_acceptance\_conditions, 759
- number\_of\_acceptance\_conditions, 759
- operator=, 759
- project\_state, 760
- state\_acceptance\_conditions, 760
- succ\_iter, 760
- support\_conditions, 760
- support\_variables, 761
- tgba\_sgba\_proxy, 758
- transition\_annotation, 761
- spot::tgba\_statistics, 762
  - dump, 762
  - states, 762
  - transitions, 762
- spot::tgba\_succ\_iterator, 762
  - ~tgba\_succ\_iterator, 764
  - current\_acceptance\_conditions, 764
  - current\_condition, 764
  - current\_state, 764
  - done, 764
  - first, 765
  - next, 765
- spot::tgba\_succ\_iterator\_concrete, 765
  - ~tgba\_succ\_iterator\_concrete, 768
  - current\_, 770
  - current\_acc\_, 770
  - current\_acceptance\_conditions, 769
  - current\_condition, 769
  - current\_state, 769
  - current\_state\_, 770
  - data\_, 770
  - done, 769
  - first, 769
  - next, 770
  - succ\_set\_, 770
  - succ\_set\_left\_, 770
  - tgba\_succ\_iterator\_concrete, 768
- spot::tgba\_succ\_iterator\_product, 771
  - ~tgba\_succ\_iterator\_product, 773
  - current\_acceptance\_conditions, 774
  - current\_cond\_, 775
  - current\_condition, 774
  - current\_state, 774
  - done, 774
  - first, 774
  - left\_, 775
  - left\_neg\_, 775
  - next, 775
  - next\_non\_false\_, 775
  - right\_, 775
  - right\_common\_acc\_, 776
  - right\_neg\_, 776
  - step\_, 775
  - tgba\_product, 775
  - tgba\_succ\_iterator\_product, 773
- spot::tgba\_succ\_iterator\_union, 776
  - ~tgba\_succ\_iterator\_union, 779
  - current\_acceptance\_conditions, 779
  - current\_cond\_, 781
  - current\_condition, 779
  - current\_state, 780
  - done, 780
  - first, 780
  - left\_, 781
  - left\_missing\_, 781
  - left\_neg\_, 781
  - next, 780
  - right\_, 781
  - right\_missing\_, 781
  - right\_neg\_, 781
  - tgba\_succ\_iterator\_union, 779
  - tgba\_union, 781
- spot::tgba\_tba\_proxy, 782
  - ~tgba\_tba\_proxy, 785
  - a\_, 789
  - acc\_cycle\_, 789
  - all\_acceptance\_conditions, 785
  - compute\_support\_conditions, 785
  - compute\_support\_variables, 786
  - cycle\_list, 785
  - format\_state, 786
  - get\_dict, 786
  - get\_init\_state, 786
  - neg\_acceptance\_conditions, 786
  - number\_of\_acceptance\_conditions, 787
  - operator=, 787
  - project\_state, 787
  - succ\_iter, 787
  - support\_conditions, 788
  - support\_variables, 788
  - tgba\_tba\_proxy, 785
  - the\_acceptance\_cond\_, 789
  - transition\_annotation, 788
- spot::tgba\_union, 789
  - ~tgba\_union, 793
  - all\_acceptance\_conditions, 794
  - all\_acceptance\_conditions\_, 797
  - compute\_support\_conditions, 794
  - compute\_support\_variables, 794
  - dict\_, 797
  - format\_state, 794
  - get\_dict, 794
  - get\_init\_state, 794
  - left\_, 797
  - left\_acc\_complement\_, 797
  - left\_acc\_missing\_, 797
  - left\_var\_missing\_, 797
  - neg\_acceptance\_conditions, 795

- neg\_acceptance\_conditions\_, 797
- number\_of\_acceptance\_conditions, 795
- operator=, 795
- project\_state, 795
- right\_, 797
- right\_acc\_complement\_, 797
- right\_acc\_missing\_, 798
- right\_common\_acc\_, 798
- right\_var\_missing\_, 798
- succ\_iter, 795
- support\_conditions, 796
- support\_variables, 796
- tgba\_union, 793
- transition\_annotation, 796
- spot::time\_info, 798
  - stime, 799
  - time\_info, 798
  - utime, 799
- spot::timer, 799
  - is\_running, 801
  - running, 802
  - start, 801
  - start\_, 802
  - stime, 801
  - stop, 801
  - timer, 801
  - total\_, 802
  - utime, 802
- spot::timer\_map, 802
  - cancel, 803
  - empty, 804
  - item\_type, 803
  - print, 804
  - start, 804
  - stop, 804
  - timer, 804
  - tm, 805
  - tm\_type, 803
- spot::unsigned\_statistics, 827
  - ~unsigned\_statistics, 829
  - get, 829
  - stats, 829
  - stats\_map, 829
  - unsigned\_fun, 829
- spot::unsigned\_statistics\_copy, 829
  - operator==, 830
  - set, 831
  - seteq, 831
  - stats, 831
  - stats\_map, 830
  - unsigned\_statistics\_copy, 830
- spot::weight, 833
  - dec\_weight\_handler, 835
  - inc\_weight\_handler, 835
  - m, 836
  - neg\_all\_acc, 836
  - operator<<, 835
  - operator+=, 835
  - operator-, 835
  - operator=, 835
  - pm, 836
  - weight, 835
  - weight\_vector, 834
- srand
  - random, 23
- ss\_vec
  - spot::taa\_tgba, 560
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 592
- stack
  - eltlyy::stack, 510
  - ltlyy::stack, 508
  - sautyy::stack, 506
- stack\_
  - eltlyy::slice, 492
  - ltlyy::slice, 493
  - sautyy::slice, 495
- stack\_type
  - spot::scc\_map, 478
  - spot::scc\_stack, 483
- start
  - spot::evtgba\_reachable\_iterator, 263
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 267
  - spot::evtgba\_reachable\_iterator\_depth\_first, 272
  - spot::parity\_game\_graph, 398
  - spot::parity\_game\_graph\_delayed, 406
  - spot::parity\_game\_graph\_direct, 413
  - spot::saba\_reachable\_iterator, 452
  - spot::saba\_reachable\_iterator\_breadth\_first, 457
  - spot::saba\_reachable\_iterator\_depth\_first, 462
  - spot::tgba\_reachable\_iterator, 700
  - spot::tgba\_reachable\_iterator\_breadth\_first, 704
  - spot::tgba\_reachable\_iterator\_depth\_first, 709
  - spot::tgba\_reduc, 725
  - spot::timer, 801
  - spot::timer\_map, 804
- start\_
  - spot::timer, 802
- state
  - spot::ltl::nfa, 359
  - spot::taa\_tgba, 560
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581

- spot::taa\_tgba\_string, 592
- spot::tgba\_explicit, 632
- spot::tgba\_explicit\_formula, 643
- spot::tgba\_explicit\_labelled, 655
- spot::tgba\_explicit\_string, 667
- spot::tgba\_reduc, 716
- state\_
  - spot::state\_bdd, 518
  - spot::state\_evtgba\_explicit, 522
  - spot::state\_explicit, 526
- state\_acceptance\_conditions
  - spot::tgba\_sgba\_proxy, 760
- state\_bdd
  - spot::state\_bdd, 517
- state\_couple
  - spot, 84
- state\_decl
  - spot::dotty\_decorator, 212
  - spot::tgba\_run\_dotty\_decorator, 733
- state\_evtgba\_explicit
  - spot::state\_evtgba\_explicit, 521
- state\_explicit
  - spot::state\_explicit, 525
- state\_index
  - spot::numbered\_state\_heap, 376
  - spot::numbered\_state\_heap\_hash\_map, 383
- state\_index\_p
  - spot::numbered\_state\_heap, 376
  - spot::numbered\_state\_heap\_hash\_map, 383
- state\_is\_accepting
  - spot::tgba\_sba\_proxy, 746
- state\_name\_map\_
  - spot::evtgba\_explicit, 252
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 597
  - spot::tgba\_explicit\_formula, 649
  - spot::tgba\_explicit\_labelled, 662
  - spot::tgba\_explicit\_string, 674
  - spot::tgba\_reduc, 727
- state\_predecessor\_map\_
  - spot::tgba\_reduc, 727
- state\_product
  - spot::state\_product, 529
- state\_set
  - spot::state\_set, 536
  - spot::taa\_tgba, 560
  - spot::taa\_tgba\_formula, 570
  - spot::taa\_tgba\_labelled, 581
  - spot::taa\_tgba\_string, 592
- state\_set\_vec\_
  - spot::taa\_tgba, 565
  - spot::taa\_tgba\_formula, 575
  - spot::taa\_tgba\_labelled, 587
  - spot::taa\_tgba\_string, 598
- state\_union
  - spot::state\_union, 542
- states
  - spot::connected\_component\_hash\_set, 165
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_shy, 197
  - spot::couvreur99\_check\_status, 201
  - spot::ec\_statistics, 228
  - spot::scc\_map::scc, 474
  - spot::tgba\_statistics, 762
- states\_
  - spot::ec\_statistics, 229
- states\_of
  - spot::scc\_map, 481
- statistics
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 197
  - spot::emptiness\_check, 233
  - spot::emptiness\_check\_result, 241
- stats
  - spot::acss\_statistics, 102
  - spot::ars\_statistics, 111
  - spot::couvreur99\_check, 183
  - spot::couvreur99\_check\_result, 189
  - spot::couvreur99\_check\_shy, 199
  - spot::ec\_statistics, 229
  - spot::unsigned\_statistics, 829
  - spot::unsigned\_statistics\_copy, 831
- stats\_map
  - spot::acss\_statistics, 101
  - spot::ars\_statistics, 110
  - spot::couvreur99\_check, 179
  - spot::couvreur99\_check\_result, 187
  - spot::couvreur99\_check\_shy, 194
  - spot::ec\_statistics, 227
  - spot::unsigned\_statistics, 829
  - spot::unsigned\_statistics\_copy, 830
- stats\_reachable
  - tgba\_misc, 37
- step
  - eltlyy::location, 334
  - ltlyy::location, 340
  - sautyy::location, 337
  - spot::minato\_isop::local\_vars, 332
- step\_
  - spot::tgba\_succ\_iterator\_product, 775
- step\_map
  - spot::tgba\_run\_dotty\_decorator, 732
- step\_num
  - spot::tgba\_run\_dotty\_decorator, 732
- step\_set
  - spot::tgba\_run\_dotty\_decorator, 732

- steps
  - spot::tgba\_run, 728
- stime
  - spot::time\_info, 799
  - spot::timer, 801
- stop
  - spot::timer, 801
  - spot::timer\_map, 804
- sub\_set\_acc\_cond\_
  - spot::parity\_game\_graph\_delayed, 407
- succ
  - spot::scc\_map, 481
  - spot::scc\_map::scc, 474
- succ\_
  - spot::taa\_succ\_iterator, 555
- succ\_iter
  - spot::evtgba, 246
  - spot::evtgba\_explicit, 251
  - spot::evtgba\_product, 257
  - spot::fair\_kripke, 289
  - spot::future\_conditions\_collector, 311
  - spot::kripke, 325
  - spot::saba, 442
  - spot::saba\_complement\_tgba, 447
  - spot::taa\_tgba, 563
  - spot::taa\_tgba\_formula, 573
  - spot::taa\_tgba\_labelled, 585
  - spot::taa\_tgba\_string, 595
  - spot::tgba, 604
  - spot::tgba\_bdd\_concrete, 613
  - spot::tgba\_explicit, 636
  - spot::tgba\_explicit\_formula, 647
  - spot::tgba\_explicit\_labelled, 660
  - spot::tgba\_explicit\_string, 672
  - spot::tgba\_kv\_complement, 685
  - spot::tgba\_product, 693
  - spot::tgba\_reduc, 725
  - spot::tgba\_safra\_complement, 739
  - spot::tgba\_sba\_proxy, 746
  - spot::tgba\_scc, 753
  - spot::tgba\_sgba\_proxy, 760
  - spot::tgba\_tba\_proxy, 787
  - spot::tgba\_union, 795
- succ\_iterator
  - spot::ltl::succ\_iterator, 546
- succ\_queue
  - spot::couvreur99\_check\_shy, 194
- succ\_set\_
  - spot::tgba\_succ\_iterator\_concrete, 770
- succ\_set\_left\_
  - spot::tgba\_succ\_iterator\_concrete, 770
- succ\_to\_string
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 504
- succ\_type
  - spot::scc\_map, 478
- successor
  - spot::couvreur99\_check\_shy::successor, 547
- super
  - spot::ltl::simplify\_f\_g\_visitor, 490
  - spot::ltl::unabbreviate\_logic\_visitor, 815
  - spot::ltl::unabbreviate\_ltl\_visitor, 819
- supp
  - spot::scc\_map::scc, 475
- supp\_rec
  - spot::scc\_map::scc, 475
- support\_conditions
  - spot::fair\_kripke, 289
  - spot::future\_conditions\_collector, 311
  - spot::kripke, 325
  - spot::taa\_tgba, 563
  - spot::taa\_tgba\_formula, 574
  - spot::taa\_tgba\_labelled, 586
  - spot::taa\_tgba\_string, 596
  - spot::tgba, 604
  - spot::tgba\_bdd\_concrete, 614
  - spot::tgba\_explicit, 636
  - spot::tgba\_explicit\_formula, 648
  - spot::tgba\_explicit\_labelled, 660
  - spot::tgba\_explicit\_string, 672
  - spot::tgba\_kv\_complement, 686
  - spot::tgba\_product, 694
  - spot::tgba\_reduc, 725
  - spot::tgba\_safra\_complement, 739
  - spot::tgba\_sba\_proxy, 746
  - spot::tgba\_scc, 753
  - spot::tgba\_sgba\_proxy, 760
  - spot::tgba\_tba\_proxy, 788
  - spot::tgba\_union, 796
- support\_variables
  - spot::fair\_kripke, 290
  - spot::future\_conditions\_collector, 311
  - spot::kripke, 326
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 574
  - spot::taa\_tgba\_labelled, 586
  - spot::taa\_tgba\_string, 596
  - spot::tgba, 605
  - spot::tgba\_bdd\_concrete, 614
  - spot::tgba\_explicit, 637
  - spot::tgba\_explicit\_formula, 648
  - spot::tgba\_explicit\_labelled, 660
  - spot::tgba\_explicit\_string, 673
  - spot::tgba\_kv\_complement, 686
  - spot::tgba\_product, 694
  - spot::tgba\_reduc, 726



- spot::tgba\_safra\_complement, 740
- spot::tgba\_sba\_proxy, 747
- spot::tgba\_scc, 753
- spot::tgba\_sgba\_proxy, 761
- spot::tgba\_tba\_proxy, 788
- spot::tgba\_union, 796
- symb\_merge\_
  - spot::ltl::language\_containment\_checker, 330
- symbol
  - spot::symbol, 549
- symbol\_set
  - spot, 84
- syntactic\_implication
  - ltl\_misc, 17
- syntactic\_implication\_neg
  - ltl\_misc, 17
- taa\_succ\_iterator
  - spot::taa\_succ\_iterator, 554
- taa\_tgba
  - spot::taa\_tgba, 561
- taa\_tgba\_formula
  - spot::taa\_tgba\_formula, 570
- taa\_tgba\_labelled
  - spot::taa\_tgba\_labelled, 581
- taa\_tgba\_string
  - spot::taa\_tgba\_string, 592
- tgba
  - spot::tgba, 602
- TGBA (Transition-based Generalized Büchi Automata), 3
- TGBA algorithms, 25
- TGBA on-the-fly algorithms, 26
- TGBA representations, 25
- TGBA runs and supporting functions, 48
- TGBA simplifications, 33
- tgba/ Directory Reference, 59
- tgba/bdddict.hh, 910
- tgba/bddprint.hh, 911
- tgba/formula2bdd.hh, 912
- tgba/futurecondcol.hh, 913
- tgba/public.hh, 851
- tgba/state.hh, 913
- tgba/statebdd.hh, 915
- tgba/succiter.hh, 915
- tgba/succiterconcrete.hh, 916
- tgba/taatgba.hh, 917
- tgba/tgba.hh, 918
- tgba/tgababddconcrete.hh, 919
- tgba/tgababddconcretefactory.hh, 920
- tgba/tgababddconcreteproduct.hh, 920
- tgba/tgababddcoredata.hh, 921
- tgba/tgababddfactory.hh, 922
- tgba/tgbaexplicit.hh, 923
- tgba/tgbakvcomplement.hh, 924
- tgba/tgbaproduct.hh, 925
- tgba/tgbareduc.hh, 926
- tgba/tgbasafracomplement.hh, 927
- tgba/tgbascc.hh, 928
- tgba/tgbasgba.hh, 929
- tgba/tgbatba.hh, 930
- tgba/tgbaunion.hh, 931
- tgba\_reduction
  - Reduce\_All, 35
  - Reduce\_None, 35
  - Reduce\_quotient\_Del\_Sim, 35
  - Reduce\_quotient\_Dir\_Sim, 35
  - Reduce\_Scc, 35
  - Reduce\_transition\_Del\_Sim, 35
  - Reduce\_transition\_Dir\_Sim, 35
- tgba\_algorithms
  - product, 26
- tgba\_bdd\_concrete
  - spot::tgba\_bdd\_concrete, 610, 611
- tgba\_bdd\_concrete\_factory
  - spot::tgba\_bdd\_concrete\_factory, 618
- tgba\_bdd\_core\_data
  - spot::tgba\_bdd\_core\_data, 623
- tgba\_dupexp\_bfs
  - tgba\_misc, 37
- tgba\_dupexp\_dfs
  - tgba\_misc, 38
- tgba\_explicit
  - spot::tgba\_explicit, 633
- tgba\_explicit\_formula
  - spot::tgba\_explicit\_formula, 643
- tgba\_explicit\_labelled
  - spot::tgba\_explicit\_labelled, 655
- tgba\_explicit\_string
  - spot::tgba\_explicit\_string, 667
- tgba\_explicit\_succ\_iterator
  - spot::tgba\_explicit\_succ\_iterator, 677
- tgba\_io
  - dotty\_reachable, 28
  - format\_tgba\_parse\_errors, 28
  - lbt\_reachable, 28
  - never\_claim\_reachable, 28
  - tgba\_parse, 28
  - tgba\_parse\_error, 27
  - tgba\_parse\_error\_list, 27
  - tgba\_save\_reachable, 29
- tgba\_kv\_complement
  - spot::tgba\_kv\_complement, 683
- tgba\_ltl
  - eltl\_to\_tgba\_lacim, 30
  - ltl\_to\_taa, 30
  - ltl\_to\_tgba\_fm, 31
  - ltl\_to\_tgba\_lacim, 32

- tgba\_misc
  - random\_graph, 37
  - stats\_reachable, 37
  - tgba\_dupexp\_bfs, 37
  - tgba\_dupexp\_dfs, 38
  - tgba\_powerset, 38
- tgba\_parse
  - tgba\_io, 28
- tgba\_parse\_error
  - tgba\_io, 27
- tgba\_parse\_error\_list
  - tgba\_io, 27
- tgba\_powerset
  - tgba\_misc, 38
- tgba\_product
  - spot::tgba\_product, 691
  - spot::tgba\_succ\_iterator\_product, 775
- tgba\_reachable\_iterator
  - spot::tgba\_reachable\_iterator, 698
- tgba\_reachable\_iterator\_breadth\_first
  - spot::tgba\_reachable\_iterator\_breadth\_first, 703
- tgba\_reachable\_iterator\_depth\_first
  - spot::tgba\_reachable\_iterator\_depth\_first, 708
- tgba\_reduc
  - spot::tgba\_reduc, 717
- tgba\_reduction
  - dn\_v, 34
  - free\_relation\_simulation, 35
  - get\_delayed\_relation\_simulation, 35
  - get\_direct\_relation\_simulation, 35
  - reduc\_tgba\_sim, 35
  - reduce\_tgba\_options, 35
  - s\_v, 34
  - sn\_v, 34
- tgba\_run
  - print\_tgba\_run, 49
  - project\_tgba\_run, 49
  - reduce\_run, 49
  - replay\_tgba\_run, 49
  - spot::tgba\_run, 729
  - tgba\_run\_to\_tgba, 50
- tgba\_run\_dotty\_decorator
  - spot::tgba\_run\_dotty\_decorator, 732
- tgba\_run\_to\_tgba
  - tgba\_run, 50
- tgba\_safra\_complement
  - spot::tgba\_safra\_complement, 737
- tgba\_save\_reachable
  - tgba\_io, 29
- tgba\_sba\_proxy
  - spot::tgba\_sba\_proxy, 744
- tgba\_scc
  - spot::tgba\_scc, 751
- tgba\_sgba\_proxy
  - spot::tgba\_sgba\_proxy, 758
- tgba\_state\_
  - spot::parity\_game\_graph, 399
  - spot::parity\_game\_graph\_delayed, 407
  - spot::parity\_game\_graph\_direct, 414
- tgba\_succ\_iterator\_concrete
  - spot::tgba\_succ\_iterator\_concrete, 768
- tgba\_succ\_iterator\_product
  - spot::tgba\_succ\_iterator\_product, 773
- tgba\_succ\_iterator\_union
  - spot::tgba\_succ\_iterator\_union, 779
- tgba\_tba\_proxy
  - spot::tgba\_tba\_proxy, 785
- tgba\_to\_evtgba
  - spot, 91
- tgba\_union
  - spot::tgba\_succ\_iterator\_union, 781
  - spot::tgba\_union, 793
- tgbaalgos/ Directory Reference, 60
- tgbaalgos/bfssteps.hh, 932
- tgbaalgos/cutsgcc.hh, 932
- tgbaalgos/dotty.hh, 860
- tgbaalgos/dottydec.hh, 933
- tgbaalgos/dupexp.hh, 934
- tgbaalgos/eltl2tgba\_lacim.hh, 935
- tgbaalgos/emptiness.hh, 936
- tgbaalgos/emptiness\_stats.hh, 937
- tgbaalgos/gtec/ Directory Reference, 53
- tgbaalgos/gtec/ce.hh, 938
- tgbaalgos/gtec/explsgcc.hh, 939
- tgbaalgos/gtec/gtec.hh, 939
- tgbaalgos/gtec/nsheap.hh, 940
- tgbaalgos/gtec/sccstack.hh, 941
- tgbaalgos/gtec/status.hh, 942
- tgbaalgos/gv04.hh, 943
- tgbaalgos/lbtt.hh, 943
- tgbaalgos/ltl2taa.hh, 944
- tgbaalgos/ltl2tgba\_fm.hh, 945
- tgbaalgos/ltl2tgba\_lacim.hh, 946
- tgbaalgos/magic.hh, 947
- tgbaalgos/neverclaim.hh, 948
- tgbaalgos/powerset.hh, 949
- tgbaalgos/projrun.hh, 949
- tgbaalgos/randomgraph.hh, 950
- tgbaalgos/reachiter.hh, 861
- tgbaalgos/reducerun.hh, 950
- tgbaalgos/reducttgba\_sim.hh, 951
- tgbaalgos/replayrun.hh, 953
- tgbaalgos/rundotdec.hh, 953
- tgbaalgos/save.hh, 863
- tgbaalgos/scc.hh, 954
- tgbaalgos/sccfilter.hh, 955
- tgbaalgos/se05.hh, 955

- tgbaalgos/stats.hh, 956
- tgbaalgos/tau03.hh, 957
- tgbaalgos/tau03opt.hh, 957
- tgbaalgos/weight.hh, 958
- tgparse/ Directory Reference, 61
- tgparse/public.hh, 852
- tgbsafracomplement.hh
  - TRANSFORM\_TO\_TGBA, 928
- the\_acceptance\_cond\_
  - spot::saba\_complement\_tgba, 447
  - spot::tgba\_kv\_complement, 687
  - spot::tgba\_tba\_proxy, 789
- ThirdStep
  - spot::minato\_isop::local\_vars, 331
- time\_info
  - spot::time\_info, 798
- timer
  - spot::timer, 801
  - spot::timer\_map, 804
- tm
  - spot::timer\_map, 805
- tm\_type
  - spot::timer\_map, 803
- to\_spin\_string
  - ltl\_io, 9
- to\_string
  - ltl\_io, 9
  - spot::duplicator\_node, 218
  - spot::duplicator\_node\_delayed, 223
  - spot::spoiler\_node, 498
  - spot::spoiler\_node\_delayed, 504
- todo
  - spot::couvreur99\_check\_shy, 199
  - spot::evtgba\_reachable\_iterator\_breadth\_first, 268
  - spot::evtgba\_reachable\_iterator\_depth\_first, 273
  - spot::parity\_game\_graph, 399
  - spot::parity\_game\_graph\_delayed, 407
  - spot::parity\_game\_graph\_direct, 414
  - spot::saba\_reachable\_iterator\_breadth\_first, 457
  - spot::saba\_reachable\_iterator\_depth\_first, 462
  - spot::tgba\_reachable\_iterator\_breadth\_first, 705
  - spot::tgba\_reachable\_iterator\_depth\_first, 710
  - spot::tgba\_reduc, 727
- todo\_
  - spot::minato\_isop, 348
  - spot::scc\_map, 482
- todo\_item
  - spot::couvreur99\_check\_shy::todo\_item, 806
- todo\_list
  - spot::couvreur99\_check\_shy, 194
- top
  - spot::scc\_stack, 484
- total\_
  - spot::timer, 802
- total\_1\_
  - spot::ltl::random\_ltl, 429
- total\_2\_
  - spot::ltl::random\_ltl, 429
- total\_2\_and\_more\_
  - spot::ltl::random\_ltl, 429
- trans\_map
  - spot::ltl::language\_containment\_checker, 328
- TRANSFORM\_TO\_TGBA
  - tgbsafracomplement.hh, 928
- transition\_annotation
  - spot::fair\_kripke, 290
  - spot::future\_conditions\_collector, 312
  - spot::kripke, 326
  - spot::taa\_tgba, 564
  - spot::taa\_tgba\_formula, 574
  - spot::taa\_tgba\_labelled, 586
  - spot::taa\_tgba\_string, 596
  - spot::tgba, 605
  - spot::tgba\_bdd\_concrete, 614
  - spot::tgba\_explicit, 637
  - spot::tgba\_explicit\_formula, 648
  - spot::tgba\_explicit\_labelled, 661
  - spot::tgba\_explicit\_string, 673
  - spot::tgba\_kv\_complement, 686
  - spot::tgba\_product, 694
  - spot::tgba\_reduc, 726
  - spot::tgba\_safra\_complement, 740
  - spot::tgba\_sba\_proxy, 747
  - spot::tgba\_scc, 754
  - spot::tgba\_sgba\_proxy, 761
  - spot::tgba\_tba\_proxy, 788
  - spot::tgba\_union, 796
- transition\_list
  - spot::evtgba\_explicit, 249
- transitions
  - spot::couvreur99\_check, 182
  - spot::couvreur99\_check\_shy, 198
  - spot::ec\_statistics, 229
  - spot::tgba\_statistics, 762
- transitions\_
  - spot::ec\_statistics, 229
- translated\_
  - spot::ltl::language\_containment\_checker, 330
- Translating LTL formulae into TGBA, 29
- translation
  - spot::ltl::language\_containment\_checker::record\_, 431
- triplet
  - spot::ltl::automatop, 121

- trivial
  - spot::scc\_map::scc, 475
- True
  - spot::ltl::constant, 173
  - spot::ltl::formula\_tree, 98
- true\_instance
  - spot::ltl::constant, 174
- type
  - spot::ltl::binop, 152
  - spot::ltl::constant, 173
  - spot::ltl::multop, 353
  - spot::ltl::unop, 824
- U
  - spot::ltl::binop, 152
- unabbreviate\_logic
  - ltl\_rewriting, 14
- unabbreviate\_logic\_visitor
  - spot::ltl::unabbreviate\_logic\_visitor, 815
- unabbreviate\_ltl
  - spot::ltl, 96
- unabbreviate\_ltl\_visitor
  - spot::ltl::unabbreviate\_ltl\_visitor, 819
- unop
  - spot::ltl::unop, 825
- unref
  - spot::symbol, 550
- unref\_
  - spot::ltl::atomic\_prop, 117
  - spot::ltl::automatop, 124
  - spot::ltl::binop, 154
  - spot::ltl::constant, 174
  - spot::ltl::formula, 299
  - spot::ltl::multop, 356
  - spot::ltl::ref\_formula, 436
  - spot::ltl::unop, 827
- unregister\_all\_my\_variables
  - spot::bdd\_dict, 141
- unregister\_variable
  - spot::bdd\_dict, 141
- unsigned\_fun
  - spot::acss\_statistics, 101
  - spot::ars\_statistics, 110
  - spot::couvreur99\_check, 179
  - spot::couvreur99\_check\_result, 187
  - spot::couvreur99\_check\_shy, 194
  - spot::ec\_statistics, 227
  - spot::unsigned\_statistics, 829
- unsigned\_statistics\_copy
  - spot::unsigned\_statistics\_copy, 830
- update\_sums
  - spot::ltl::random\_ltl, 428
- update\_supp\_rec
  - spot::scc\_map, 481
- useful\_acc
  - spot::scc\_map::scc, 475
  - spot::scc\_stats, 486
- useful\_acc\_of
  - spot::scc\_map, 481
- useless\_scc\_map
  - spot::scc\_stats, 486
- utime
  - spot::time\_info, 799
  - spot::timer, 802
- v1
  - spot::minato\_isop::local\_vars, 332
- val
  - spot::ltl::constant, 174
- val\_
  - spot::ltl::constant, 175
- val\_name
  - spot::ltl::constant, 175
- var\_formula\_map
  - spot::bdd\_dict, 142
- var\_map
  - spot::bdd\_dict, 142
- var\_refs
  - spot::bdd\_dict, 143
- var\_set
  - spot::tgba\_bdd\_core\_data, 626
- varandnext\_set
  - spot::tgba\_bdd\_core\_data, 626
- vars
  - spot::minato\_isop::local\_vars, 332
- vec
  - spot::ltl::automatop, 121
  - spot::ltl::multop, 352
- version
  - misc\_tools, 20
- vf\_map
  - spot::bdd\_dict, 137
- visit
  - spot::ltl::clone\_visitor, 158, 159
  - spot::ltl::const\_visitor, 168, 169
  - spot::ltl::postfix\_visitor, 424
  - spot::ltl::simplify\_f\_g\_visitor, 490, 491
  - spot::ltl::unabbreviate\_logic\_visitor, 815, 816
  - spot::ltl::unabbreviate\_ltl\_visitor, 819, 820
  - spot::ltl::visitor, 833
- vr\_map
  - spot::bdd\_dict, 138
- W
  - spot::ltl::binop, 152
- wang32\_hash
  - hash\_funcs, 21
- want\_state

- spot::parity\_game\_graph, [398](#)
- spot::parity\_game\_graph\_delayed, [406](#)
- spot::parity\_game\_graph\_direct, [413](#)
- spot::saba\_reachable\_iterator, [452](#)
- spot::saba\_reachable\_iterator\_breadth\_first,  
[457](#)
- spot::saba\_reachable\_iterator\_depth\_first, [462](#)
- spot::tgba\_reachable\_iterator, [700](#)
- spot::tgba\_reachable\_iterator\_breadth\_first,  
[705](#)
- spot::tgba\_reachable\_iterator\_depth\_first, [710](#)
- spot::tgba\_reduc, [726](#)
- weight
  - spot::weight, [835](#)
- weight\_vector
  - spot::weight, [834](#)
- where\_
  - spot::gspn\_exception, [313](#)
  - spot::nips\_exception, [363](#)
- X
  - spot::ltl::unop, [824](#)
- Xor
  - spot::ltl::binop, [152](#)