

spot
0.7

Generated by Doxygen 1.7.1

Tue Feb 1 2011 12:00:52

Contents

1	Main Page	1
1.1	The Spot Library	1
1.2	This Document	1
1.3	Handy starting points	1
2	Deprecated List	2
3	Bug List	2
4	Module Documentation	2
4.1	LTL formulae	2
4.1.1	Detailed Description	2
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	3
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	Kripke Structures	10
4.7	Essential LTL types	10
4.7.1	Function Documentation	11
4.8	LTL Abstract Syntax Tree	11
4.9	LTL environments	12
4.9.1	Detailed Description	12
4.10	Algorithms for LTL formulae	12
4.11	Derivable visitors	12
4.12	Rewriting LTL formulae	13
4.12.1	Enumeration Type Documentation	13
4.12.2	Function Documentation	14
4.13	Miscellaneous algorithms for LTL formulae	15
4.13.1	Typedef Documentation	16
4.13.2	Function Documentation	16
4.14	Miscellaneous helper algorithms	18

4.14.1 Detailed Description	19
4.14.2 Function Documentation	19
4.15 Hashing functions	20
4.15.1 Function Documentation	21
4.16 Random functions	21
4.16.1 Function Documentation	22
4.17 Input/Output of TGBA	23
4.17.1 Typedef Documentation	24
4.17.2 Function Documentation	25
4.18 Essential SABA types	27
4.19 Essential TGBA types	28
4.20 TGBA representations	29
4.21 TGBA algorithms	30
4.21.1 Function Documentation	30
4.22 TGBA on-the-fly algorithms	30
4.22.1 Function Documentation	31
4.23 Translating LTL formulae into TGBA	31
4.23.1 Function Documentation	32
4.24 Algorithm patterns	35
4.25 TGBA simplifications	35
4.25.1 Typedef Documentation	37
4.25.2 Enumeration Type Documentation	37
4.25.3 Function Documentation	37
4.26 Miscellaneous algorithms on TGBA	40
4.26.1 Function Documentation	41
4.27 Decorating the dot output	42
4.28 Emptiness-checks	43
4.28.1 Detailed Description	43
4.29 Emptiness-check algorithms	43
4.29.1 Function Documentation	44
4.30 TGBA runs and supporting functions	52
4.30.1 Function Documentation	53
4.31 Emptiness-check statistics	54
5 Directory Documentation	55
5.1 eltlparse/ Directory Reference	55
5.2 evtgba/ Directory Reference	56

5.3	evtgbalgorithms/ Directory Reference	57
5.4	evtgbaparse/ Directory Reference	58
5.5	gspn/ Directory Reference	59
5.6	tgbalgorithms/gtec/ Directory Reference	60
5.7	kripke/ Directory Reference	61
5.8	ltlast/ Directory Reference	61
5.9	ltlenv/ Directory Reference	63
5.10	ltlparse/ Directory Reference	64
5.11	ltlvisit/ Directory Reference	65
5.12	misc/ Directory Reference	66
5.13	neverparse/ Directory Reference	67
5.14	nips/ Directory Reference	67
5.15	saba/ Directory Reference	68
5.16	sabalgorithms/ Directory Reference	69
5.17	sautparse/ Directory Reference	69
5.18	tgba/ Directory Reference	70
5.19	tgbalgorithms/ Directory Reference	71
5.20	tgbaparse/ Directory Reference	73
6	Namespace Documentation	73
6.1	eltly Namespace Reference	73
6.1.1	Function Documentation	74
6.2	ltly Namespace Reference	76
6.2.1	Function Documentation	78
6.3	neverclaimy Namespace Reference	80
6.3.1	Function Documentation	81
6.4	sauty Namespace Reference	83
6.4.1	Function Documentation	84
6.5	spot Namespace Reference	85
6.5.1	Typedef Documentation	100
6.5.2	Function Documentation	101
6.6	spot::ltl Namespace Reference	108
6.7	spot::ltl Namespace Reference	109
6.7.1	Function Documentation	113
6.8	spot::ltl::formula_tree Namespace Reference	114
6.8.1	Detailed Description	114
6.8.2	Typedef Documentation	114

6.8.3	Enumeration Type Documentation	115
6.8.4	Function Documentation	115
7	Class Documentation	115
7.1	spot::acss_statistics Class Reference	115
7.1.1	Detailed Description	118
7.1.2	Member Typedef Documentation	118
7.1.3	Constructor & Destructor Documentation	118
7.1.4	Member Function Documentation	118
7.1.5	Member Data Documentation	119
7.2	spot::bdd_dict::anon_free_list Class Reference	119
7.2.1	Member Typedef Documentation	123
7.2.2	Constructor & Destructor Documentation	123
7.2.3	Member Function Documentation	123
7.2.4	Member Data Documentation	124
7.3	spot::ars_statistics Class Reference	124
7.3.1	Detailed Description	127
7.3.2	Member Typedef Documentation	127
7.3.3	Constructor & Destructor Documentation	127
7.3.4	Member Function Documentation	127
7.3.5	Member Data Documentation	128
7.4	spot::ltdl::atomic_prop Class Reference	128
7.4.1	Detailed Description	132
7.4.2	Member Typedef Documentation	132
7.4.3	Constructor & Destructor Documentation	132
7.4.4	Member Function Documentation	133
7.4.5	Member Data Documentation	135
7.5	spot::ltdl::automatop Class Reference	135
7.5.1	Detailed Description	140
7.5.2	Member Typedef Documentation	140
7.5.3	Constructor & Destructor Documentation	140
7.5.4	Member Function Documentation	140
7.5.5	Member Data Documentation	143
7.6	spot::barand< gen > Class Template Reference	143
7.6.1	Detailed Description	144
7.6.2	Constructor & Destructor Documentation	144
7.6.3	Member Function Documentation	144

7.6.4	Member Data Documentation	144
7.7	spot::bdd_allocator Class Reference	145
7.7.1	Detailed Description	149
7.7.2	Member Typedef Documentation	149
7.7.3	Constructor & Destructor Documentation	149
7.7.4	Member Function Documentation	149
7.7.5	Member Data Documentation	151
7.8	spot::bdd_dict Class Reference	151
7.8.1	Detailed Description	156
7.8.2	Member Typedef Documentation	156
7.8.3	Constructor & Destructor Documentation	157
7.8.4	Member Function Documentation	157
7.8.5	Member Data Documentation	161
7.9	spot::bdd_less_than Struct Reference	162
7.9.1	Detailed Description	162
7.9.2	Member Function Documentation	163
7.10	spot::bdd_ordered Class Reference	163
7.10.1	Constructor & Destructor Documentation	163
7.10.2	Member Function Documentation	163
7.10.3	Member Data Documentation	164
7.11	spot::bfs_steps Class Reference	164
7.11.1	Detailed Description	166
7.11.2	Constructor & Destructor Documentation	166
7.11.3	Member Function Documentation	166
7.11.4	Member Data Documentation	167
7.12	spot::ltl::binop Class Reference	168
7.12.1	Detailed Description	173
7.12.2	Member Typedef Documentation	173
7.12.3	Member Enumeration Documentation	173
7.12.4	Constructor & Destructor Documentation	173
7.12.5	Member Function Documentation	174
7.12.6	Member Data Documentation	176
7.13	spot::char_ptr_less_than Struct Reference	177
7.13.1	Detailed Description	177
7.13.2	Member Function Documentation	177
7.14	spot::ltl::clone_visitor Class Reference	177

7.14.1 Detailed Description	180
7.14.2 Constructor & Destructor Documentation	180
7.14.3 Member Function Documentation	180
7.14.4 Member Data Documentation	181
7.15 spot::scc_stack::connected_component Struct Reference	181
7.15.1 Constructor & Destructor Documentation	183
7.15.2 Member Data Documentation	183
7.16 spot::connected_component_hash_set Class Reference	183
7.16.1 Detailed Description	186
7.16.2 Member Typedef Documentation	186
7.16.3 Constructor & Destructor Documentation	186
7.16.4 Member Function Documentation	186
7.16.5 Member Data Documentation	187
7.17 spot::connected_component_hash_set_factory Class Reference	187
7.17.1 Detailed Description	190
7.17.2 Constructor & Destructor Documentation	190
7.17.3 Member Function Documentation	190
7.18 spot::ltl::const_visitor Struct Reference	190
7.18.1 Detailed Description	191
7.18.2 Constructor & Destructor Documentation	191
7.18.3 Member Function Documentation	191
7.19 spot::ltl::constant Class Reference	192
7.19.1 Detailed Description	195
7.19.2 Member Enumeration Documentation	195
7.19.3 Constructor & Destructor Documentation	195
7.19.4 Member Function Documentation	195
7.19.5 Member Data Documentation	197
7.20 spot::couvreur99_check Class Reference	197
7.20.1 Detailed Description	201
7.20.2 Member Typedef Documentation	201
7.20.3 Constructor & Destructor Documentation	201
7.20.4 Member Function Documentation	202
7.20.5 Member Data Documentation	205
7.21 spot::couvreur99_check_result Class Reference	206
7.21.1 Detailed Description	210
7.21.2 Member Typedef Documentation	210

7.21.3	Constructor & Destructor Documentation	210
7.21.4	Member Function Documentation	210
7.21.5	Member Data Documentation	212
7.22	spot::couvreur99_check_shy Class Reference	213
7.22.1	Detailed Description	218
7.22.2	Member Typedef Documentation	218
7.22.3	Constructor & Destructor Documentation	218
7.22.4	Member Function Documentation	219
7.22.5	Member Data Documentation	222
7.23	spot::couvreur99_check_status Class Reference	224
7.23.1	Detailed Description	226
7.23.2	Constructor & Destructor Documentation	226
7.23.3	Member Function Documentation	226
7.23.4	Member Data Documentation	226
7.24	spot::ltl::declarative_environment Class Reference	227
7.24.1	Detailed Description	230
7.24.2	Member Typedef Documentation	230
7.24.3	Constructor & Destructor Documentation	230
7.24.4	Member Function Documentation	230
7.24.5	Member Data Documentation	231
7.25	spot::ltl::default_environment Class Reference	231
7.25.1	Detailed Description	234
7.25.2	Constructor & Destructor Documentation	234
7.25.3	Member Function Documentation	234
7.26	spot::delayed_simulation_relation Class Reference	235
7.27	spot::direct_simulation_relation Class Reference	235
7.28	spot::taa_succ_iterator::distance_sort Struct Reference	235
7.28.1	Member Function Documentation	235
7.29	spot::dotty_decorator Class Reference	235
7.29.1	Detailed Description	237
7.29.2	Constructor & Destructor Documentation	237
7.29.3	Member Function Documentation	237
7.30	spot::duplicator_node Class Reference	238
7.30.1	Detailed Description	241
7.30.2	Constructor & Destructor Documentation	241
7.30.3	Member Function Documentation	242

7.30.4	Member Data Documentation	243
7.31	spot::duplicator_node_delayed Class Reference	244
7.31.1	Detailed Description	247
7.31.2	Constructor & Destructor Documentation	248
7.31.3	Member Function Documentation	248
7.31.4	Member Data Documentation	250
7.32	spot::ec_statistics Class Reference	251
7.32.1	Detailed Description	254
7.32.2	Member Typedef Documentation	254
7.32.3	Constructor & Destructor Documentation	254
7.32.4	Member Function Documentation	255
7.32.5	Member Data Documentation	256
7.33	spot::emptiness_check Class Reference	257
7.33.1	Detailed Description	260
7.33.2	Constructor & Destructor Documentation	260
7.33.3	Member Function Documentation	261
7.33.4	Member Data Documentation	262
7.34	spot::emptiness_check_instantiator Class Reference	262
7.34.1	Constructor & Destructor Documentation	264
7.34.2	Member Function Documentation	264
7.34.3	Member Data Documentation	265
7.35	spot::emptiness_check_result Class Reference	265
7.35.1	Detailed Description	268
7.35.2	Constructor & Destructor Documentation	268
7.35.3	Member Function Documentation	269
7.35.4	Member Data Documentation	270
7.36	spot::ltl::environment Class Reference	270
7.36.1	Detailed Description	271
7.36.2	Constructor & Destructor Documentation	272
7.36.3	Member Function Documentation	272
7.37	spot::evtgba Class Reference	272
7.37.1	Constructor & Destructor Documentation	274
7.37.2	Member Function Documentation	274
7.38	spot::evtgba_explicit Class Reference	276
7.38.1	Member Typedef Documentation	280
7.38.2	Constructor & Destructor Documentation	280

7.38.3	Member Function Documentation	280
7.38.4	Member Data Documentation	282
7.39	spot::evtgba_iterator Class Reference	282
7.39.1	Constructor & Destructor Documentation	283
7.39.2	Member Function Documentation	283
7.40	spot::evtgba_product Class Reference	284
7.40.1	Member Typedef Documentation	286
7.40.2	Constructor & Destructor Documentation	286
7.40.3	Member Function Documentation	287
7.40.4	Member Data Documentation	288
7.41	spot::evtgba_reachable_iterator Class Reference	288
7.41.1	Detailed Description	291
7.41.2	Member Typedef Documentation	291
7.41.3	Constructor & Destructor Documentation	292
7.41.4	Member Function Documentation	292
7.41.5	Member Data Documentation	293
7.42	spot::evtgba_reachable_iterator_breadth_first Class Reference	293
7.42.1	Detailed Description	296
7.42.2	Member Typedef Documentation	296
7.42.3	Constructor & Destructor Documentation	297
7.42.4	Member Function Documentation	297
7.42.5	Member Data Documentation	298
7.43	spot::evtgba_reachable_iterator_depth_first Class Reference	298
7.43.1	Detailed Description	301
7.43.2	Member Typedef Documentation	301
7.43.3	Constructor & Destructor Documentation	302
7.43.4	Member Function Documentation	302
7.43.5	Member Data Documentation	303
7.44	spot::explicit_connected_component Class Reference	303
7.44.1	Detailed Description	305
7.44.2	Constructor & Destructor Documentation	306
7.44.3	Member Function Documentation	306
7.44.4	Member Data Documentation	306
7.45	spot::explicit_connected_component_factory Class Reference	307
7.45.1	Detailed Description	307
7.45.2	Constructor & Destructor Documentation	307

7.45.3	Member Function Documentation	308
7.46	spot::explicit_state_conjunction Class Reference	308
7.46.1	Detailed Description	311
7.46.2	Member Typedef Documentation	311
7.46.3	Constructor & Destructor Documentation	312
7.46.4	Member Function Documentation	312
7.46.5	Member Data Documentation	313
7.47	spot::fair_kripke Class Reference	313
7.47.1	Detailed Description	317
7.47.2	Member Function Documentation	317
7.48	spot::fair_kripke_succ_iterator Class Reference	321
7.48.1	Detailed Description	324
7.48.2	Constructor & Destructor Documentation	324
7.48.3	Member Function Documentation	325
7.48.4	Member Data Documentation	326
7.49	spot::ltl::formula Class Reference	326
7.49.1	Detailed Description	328
7.49.2	Constructor & Destructor Documentation	328
7.49.3	Member Function Documentation	329
7.49.4	Member Data Documentation	330
7.50	spot::ltl::formula_ptr_hash Struct Reference	330
7.50.1	Detailed Description	331
7.50.2	Member Function Documentation	331
7.51	spot::ltl::formula_ptr_less_than Struct Reference	331
7.51.1	Detailed Description	331
7.51.2	Member Function Documentation	331
7.52	spot::free_list Class Reference	332
7.52.1	Detailed Description	334
7.52.2	Member Typedef Documentation	335
7.52.3	Constructor & Destructor Documentation	335
7.52.4	Member Function Documentation	335
7.52.5	Member Data Documentation	336
7.53	spot::future_conditions_collector Class Reference	336
7.53.1	Detailed Description	340
7.53.2	Member Typedef Documentation	340
7.53.3	Constructor & Destructor Documentation	340

7.53.4	Member Function Documentation	341
7.53.5	Member Data Documentation	344
7.54	spot::gspn_exception Class Reference	345
7.54.1	Detailed Description	345
7.54.2	Constructor & Destructor Documentation	345
7.54.3	Member Function Documentation	345
7.54.4	Member Data Documentation	345
7.55	spot::gspn_interface Class Reference	346
7.55.1	Constructor & Destructor Documentation	348
7.55.2	Member Function Documentation	348
7.55.3	Member Data Documentation	348
7.56	spot::gspn_ssp_interface Class Reference	349
7.56.1	Constructor & Destructor Documentation	351
7.56.2	Member Function Documentation	351
7.56.3	Member Data Documentation	351
7.57	spot::identity_hash< T > Struct Template Reference	351
7.57.1	Detailed Description	352
7.57.2	Member Function Documentation	352
7.58	spot::kripke Class Reference	352
7.58.1	Detailed Description	356
7.58.2	Constructor & Destructor Documentation	356
7.58.3	Member Function Documentation	356
7.59	spot::kripke_succ_iterator Class Reference	360
7.59.1	Detailed Description	363
7.59.2	Constructor & Destructor Documentation	363
7.59.3	Member Function Documentation	364
7.59.4	Member Data Documentation	365
7.60	spot::ltl::language_containment_checker Class Reference	365
7.60.1	Member Typedef Documentation	367
7.60.2	Constructor & Destructor Documentation	368
7.60.3	Member Function Documentation	368
7.60.4	Member Data Documentation	369
7.61	spot::minato_isop::local_vars Struct Reference	369
7.61.1	Detailed Description	370
7.61.2	Member Enumeration Documentation	370
7.61.3	Constructor & Destructor Documentation	370

7.61.4	Member Data Documentation	370
7.62	neverclaimy::location Class Reference	371
7.62.1	Detailed Description	373
7.62.2	Constructor & Destructor Documentation	373
7.62.3	Member Function Documentation	373
7.62.4	Member Data Documentation	374
7.63	eltlyy::location Class Reference	374
7.63.1	Detailed Description	376
7.63.2	Constructor & Destructor Documentation	376
7.63.3	Member Function Documentation	376
7.63.4	Member Data Documentation	377
7.64	ltlyy::location Class Reference	377
7.64.1	Detailed Description	379
7.64.2	Constructor & Destructor Documentation	379
7.64.3	Member Function Documentation	379
7.64.4	Member Data Documentation	380
7.65	sautyy::location Class Reference	380
7.65.1	Detailed Description	382
7.65.2	Constructor & Destructor Documentation	382
7.65.3	Member Function Documentation	382
7.65.4	Member Data Documentation	383
7.66	spot::loopless_modular_mixed_radix_gray_code Class Reference	383
7.66.1	Detailed Description	384
7.66.2	Constructor & Destructor Documentation	385
7.66.3	Member Function Documentation	385
7.66.4	Member Data Documentation	386
7.67	spot::ltl::ltl_file Class Reference	387
7.67.1	Detailed Description	387
7.67.2	Constructor & Destructor Documentation	387
7.67.3	Member Function Documentation	388
7.67.4	Member Data Documentation	388
7.68	spot::minato_isop Class Reference	388
7.68.1	Detailed Description	389
7.68.2	Constructor & Destructor Documentation	389
7.68.3	Member Function Documentation	389
7.68.4	Member Data Documentation	390

7.69	spot::ltl::multop Class Reference	390
7.69.1	Detailed Description	395
7.69.2	Member Typedef Documentation	395
7.69.3	Member Enumeration Documentation	395
7.69.4	Constructor & Destructor Documentation	395
7.69.5	Member Function Documentation	395
7.69.6	Member Data Documentation	398
7.70	spot::ltl::nfa Class Reference	399
7.70.1	Detailed Description	400
7.70.2	Member Typedef Documentation	400
7.70.3	Constructor & Destructor Documentation	401
7.70.4	Member Function Documentation	401
7.70.5	Member Data Documentation	403
7.71	spot::nips_exception Class Reference	403
7.71.1	Detailed Description	404
7.71.2	Constructor & Destructor Documentation	404
7.71.3	Member Function Documentation	404
7.71.4	Member Data Documentation	404
7.72	spot::nips_interface Class Reference	405
7.72.1	Detailed Description	407
7.72.2	Constructor & Destructor Documentation	407
7.72.3	Member Function Documentation	407
7.72.4	Member Data Documentation	407
7.73	spot::ltl::formula_tree::node Struct Reference	408
7.73.1	Constructor & Destructor Documentation	408
7.74	spot::ltl::formula_tree::node_atomic Struct Reference	408
7.74.1	Member Data Documentation	410
7.75	spot::ltl::formula_tree::node_binop Struct Reference	410
7.75.1	Member Data Documentation	411
7.76	spot::ltl::formula_tree::node_multop Struct Reference	412
7.76.1	Member Data Documentation	413
7.77	spot::ltl::formula_tree::node_nfa Struct Reference	414
7.77.1	Member Data Documentation	415
7.78	spot::ltl::formula_tree::node_unop Struct Reference	415
7.78.1	Member Data Documentation	417
7.79	spot::numbered_state_heap Class Reference	417

7.79.1	Detailed Description	419
7.79.2	Member Typedef Documentation	419
7.79.3	Constructor & Destructor Documentation	419
7.79.4	Member Function Documentation	419
7.80	spot::numbered_state_heap_const_iterator Class Reference	421
7.80.1	Detailed Description	421
7.80.2	Constructor & Destructor Documentation	421
7.80.3	Member Function Documentation	421
7.81	spot::numbered_state_heap_factory Class Reference	422
7.81.1	Detailed Description	423
7.81.2	Constructor & Destructor Documentation	423
7.81.3	Member Function Documentation	424
7.82	spot::numbered_state_heap_hash_map Class Reference	424
7.82.1	Detailed Description	427
7.82.2	Member Typedef Documentation	427
7.82.3	Constructor & Destructor Documentation	428
7.82.4	Member Function Documentation	428
7.82.5	Member Data Documentation	429
7.83	spot::numbered_state_heap_hash_map_factory Class Reference	429
7.83.1	Detailed Description	431
7.83.2	Constructor & Destructor Documentation	432
7.83.3	Member Function Documentation	432
7.84	spot::ltl::random_ltl::op_proba Struct Reference	433
7.84.1	Member Typedef Documentation	434
7.84.2	Member Function Documentation	434
7.84.3	Member Data Documentation	434
7.85	spot::option_map Class Reference	434
7.85.1	Detailed Description	435
7.85.2	Member Function Documentation	435
7.85.3	Friends And Related Function Documentation	437
7.85.4	Member Data Documentation	437
7.86	spot::ltl::multop::paircmp Struct Reference	437
7.86.1	Detailed Description	437
7.86.2	Member Function Documentation	437
7.87	spot::parity_game_graph Class Reference	437
7.87.1	Detailed Description	441

7.87.2	Member Typedef Documentation	441
7.87.3	Constructor & Destructor Documentation	441
7.87.4	Member Function Documentation	441
7.87.5	Member Data Documentation	443
7.88	spot::parity_game_graph_delayed Class Reference	444
7.88.1	Detailed Description	448
7.88.2	Member Typedef Documentation	448
7.88.3	Constructor & Destructor Documentation	449
7.88.4	Member Function Documentation	449
7.88.5	Member Data Documentation	452
7.89	spot::parity_game_graph_direct Class Reference	452
7.89.1	Detailed Description	456
7.89.2	Member Typedef Documentation	456
7.89.3	Constructor & Destructor Documentation	456
7.89.4	Member Function Documentation	456
7.89.5	Member Data Documentation	458
7.90	neverclaimy::position Class Reference	459
7.90.1	Detailed Description	460
7.90.2	Constructor & Destructor Documentation	460
7.90.3	Member Function Documentation	460
7.90.4	Member Data Documentation	461
7.91	ltlyy::position Class Reference	461
7.91.1	Detailed Description	462
7.91.2	Constructor & Destructor Documentation	462
7.91.3	Member Function Documentation	462
7.91.4	Member Data Documentation	463
7.92	eltlyy::position Class Reference	463
7.92.1	Detailed Description	464
7.92.2	Constructor & Destructor Documentation	464
7.92.3	Member Function Documentation	464
7.92.4	Member Data Documentation	465
7.93	sautyy::position Class Reference	465
7.93.1	Detailed Description	466
7.93.2	Constructor & Destructor Documentation	466
7.93.3	Member Function Documentation	466
7.93.4	Member Data Documentation	467

7.94	spot::ltl::postfix_visitor Class Reference	467
7.94.1	Detailed Description	470
7.94.2	Constructor & Destructor Documentation	470
7.94.3	Member Function Documentation	470
7.95	spot::power_map Struct Reference	472
7.95.1	Member Typedef Documentation	472
7.95.2	Constructor & Destructor Documentation	472
7.95.3	Member Function Documentation	473
7.95.4	Member Data Documentation	473
7.96	spot::ptr_hash< T > Struct Template Reference	473
7.96.1	Detailed Description	473
7.96.2	Member Function Documentation	474
7.97	spot::ltl::random_ltl Class Reference	474
7.97.1	Detailed Description	476
7.97.2	Constructor & Destructor Documentation	477
7.97.3	Member Function Documentation	477
7.97.4	Member Data Documentation	478
7.98	spot::ltl::language_containment_checker::record_ Struct Reference	481
7.98.1	Member Typedef Documentation	482
7.98.2	Member Data Documentation	482
7.99	spot::ltl::ref_formula Class Reference	482
7.99.1	Detailed Description	485
7.99.2	Constructor & Destructor Documentation	485
7.99.3	Member Function Documentation	486
7.99.4	Member Data Documentation	487
7.100	spot::rsymbol Class Reference	487
7.100.1	Constructor & Destructor Documentation	489
7.100.2	Member Function Documentation	489
7.100.3	Member Data Documentation	490
7.101	spot::saba Class Reference	490
7.101.1	Detailed Description	492
7.101.2	Constructor & Destructor Documentation	492
7.101.3	Member Function Documentation	492
7.101.4	Member Data Documentation	494
7.102	spot::saba_complement_tgba Class Reference	494
7.102.1	Detailed Description	497

7.102.2 Constructor & Destructor Documentation	498
7.102.3 Member Function Documentation	498
7.102.4 Member Data Documentation	499
7.103spot::saba_reachable_iterator Class Reference	499
7.103.1 Detailed Description	502
7.103.2 Member Typedef Documentation	502
7.103.3 Constructor & Destructor Documentation	503
7.103.4 Member Function Documentation	503
7.103.5 Member Data Documentation	505
7.104spot::saba_reachable_iterator_breadth_first Class Reference	505
7.104.1 Detailed Description	508
7.104.2 Member Typedef Documentation	508
7.104.3 Constructor & Destructor Documentation	509
7.104.4 Member Function Documentation	509
7.104.5 Member Data Documentation	510
7.105spot::saba_reachable_iterator_depth_first Class Reference	511
7.105.1 Detailed Description	514
7.105.2 Member Typedef Documentation	514
7.105.3 Constructor & Destructor Documentation	515
7.105.4 Member Function Documentation	515
7.105.5 Member Data Documentation	516
7.106spot::saba_state Class Reference	517
7.106.1 Detailed Description	517
7.106.2 Constructor & Destructor Documentation	517
7.106.3 Member Function Documentation	518
7.107spot::saba_state_conjunction Class Reference	519
7.107.1 Detailed Description	521
7.107.2 Constructor & Destructor Documentation	521
7.107.3 Member Function Documentation	521
7.108spot::saba_state_ptr_equal Struct Reference	523
7.108.1 Detailed Description	523
7.108.2 Member Function Documentation	523
7.109spot::saba_state_ptr_hash Struct Reference	523
7.109.1 Detailed Description	523
7.109.2 Member Function Documentation	524
7.110spot::saba_state_ptr_less_than Struct Reference	524

7.110.1 Detailed Description	524
7.110.2 Member Function Documentation	524
7.111spot::saba_state_shared_ptr_equal Struct Reference	525
7.111.1 Detailed Description	525
7.111.2 Member Function Documentation	525
7.112spot::saba_state_shared_ptr_hash Struct Reference	525
7.112.1 Detailed Description	525
7.112.2 Member Function Documentation	526
7.113spot::saba_state_shared_ptr_less_than Struct Reference	526
7.113.1 Detailed Description	526
7.113.2 Member Function Documentation	526
7.114spot::saba_succ_iterator Class Reference	527
7.114.1 Detailed Description	527
7.114.2 Constructor & Destructor Documentation	527
7.114.3 Member Function Documentation	527
7.115spot::scc_map::scc Struct Reference	528
7.115.1 Constructor & Destructor Documentation	529
7.115.2 Member Data Documentation	529
7.116spot::scc_map Class Reference	530
7.116.1 Detailed Description	533
7.116.2 Member Typedef Documentation	533
7.116.3 Constructor & Destructor Documentation	534
7.116.4 Member Function Documentation	534
7.116.5 Member Data Documentation	538
7.117spot::scc_stack Class Reference	538
7.117.1 Member Typedef Documentation	539
7.117.2 Member Function Documentation	539
7.117.3 Member Data Documentation	540
7.118spot::scc_stats Struct Reference	541
7.118.1 Member Function Documentation	541
7.118.2 Member Data Documentation	541
7.119spot::sccs_set Struct Reference	542
7.119.1 Member Data Documentation	543
7.120spot::ltdl::simplify_f_g_visitor Class Reference	543
7.120.1 Detailed Description	546
7.120.2 Member Typedef Documentation	546

7.120.3 Constructor & Destructor Documentation	546
7.120.4 Member Function Documentation	547
7.120.5 Member Data Documentation	548
7.121eltlyy::slice< T, S > Class Template Reference	548
7.121.1 Detailed Description	548
7.121.2 Constructor & Destructor Documentation	548
7.121.3 Member Function Documentation	548
7.121.4 Member Data Documentation	549
7.122sautyy::slice< T, S > Class Template Reference	549
7.122.1 Detailed Description	549
7.122.2 Constructor & Destructor Documentation	549
7.122.3 Member Function Documentation	550
7.122.4 Member Data Documentation	550
7.123neverclaimyy::slice< T, S > Class Template Reference	550
7.123.1 Detailed Description	550
7.123.2 Constructor & Destructor Documentation	551
7.123.3 Member Function Documentation	551
7.123.4 Member Data Documentation	551
7.124ltlyy::slice< T, S > Class Template Reference	551
7.124.1 Detailed Description	552
7.124.2 Constructor & Destructor Documentation	552
7.124.3 Member Function Documentation	552
7.124.4 Member Data Documentation	552
7.125spot::spoiler_node Class Reference	552
7.125.1 Detailed Description	554
7.125.2 Constructor & Destructor Documentation	554
7.125.3 Member Function Documentation	555
7.125.4 Member Data Documentation	556
7.126spot::spoiler_node_delayed Class Reference	556
7.126.1 Detailed Description	559
7.126.2 Constructor & Destructor Documentation	559
7.126.3 Member Function Documentation	560
7.126.4 Member Data Documentation	561
7.127neverclaimyy::stack< T, S > Class Template Reference	562
7.127.1 Member Typedef Documentation	563
7.127.2 Constructor & Destructor Documentation	563

7.127.3 Member Function Documentation	563
7.127.4 Member Data Documentation	564
7.128sautyy::stack< T, S > Class Template Reference	565
7.128.1 Member Typedef Documentation	565
7.128.2 Constructor & Destructor Documentation	565
7.128.3 Member Function Documentation	566
7.128.4 Member Data Documentation	567
7.129eltlyy::stack< T, S > Class Template Reference	567
7.129.1 Member Typedef Documentation	568
7.129.2 Constructor & Destructor Documentation	568
7.129.3 Member Function Documentation	568
7.129.4 Member Data Documentation	569
7.130ltlyy::stack< T, S > Class Template Reference	569
7.130.1 Member Typedef Documentation	570
7.130.2 Constructor & Destructor Documentation	570
7.130.3 Member Function Documentation	570
7.130.4 Member Data Documentation	571
7.131spot::evtgba_explicit::state Struct Reference	572
7.131.1 Member Data Documentation	572
7.132spot::state Class Reference	572
7.132.1 Detailed Description	573
7.132.2 Constructor & Destructor Documentation	573
7.132.3 Member Function Documentation	573
7.133spot::state_bdd Class Reference	574
7.133.1 Detailed Description	577
7.133.2 Constructor & Destructor Documentation	577
7.133.3 Member Function Documentation	577
7.133.4 Member Data Documentation	578
7.134spot::state_evtgba_explicit Class Reference	578
7.134.1 Detailed Description	581
7.134.2 Constructor & Destructor Documentation	581
7.134.3 Member Function Documentation	581
7.134.4 Member Data Documentation	582
7.135spot::state_explicit Class Reference	582
7.135.1 Detailed Description	585
7.135.2 Constructor & Destructor Documentation	585

7.135.3 Member Function Documentation	585
7.135.4 Member Data Documentation	586
7.136spot::state_product Class Reference	586
7.136.1 Detailed Description	589
7.136.2 Constructor & Destructor Documentation	589
7.136.3 Member Function Documentation	590
7.136.4 Member Data Documentation	591
7.137spot::state_ptr_equal Struct Reference	591
7.137.1 Detailed Description	591
7.137.2 Member Function Documentation	592
7.138spot::state_ptr_hash Struct Reference	592
7.138.1 Detailed Description	592
7.138.2 Member Function Documentation	592
7.139spot::state_ptr_less_than Struct Reference	593
7.139.1 Detailed Description	593
7.139.2 Member Function Documentation	593
7.140spot::state_set Class Reference	593
7.140.1 Detailed Description	596
7.140.2 Constructor & Destructor Documentation	596
7.140.3 Member Function Documentation	596
7.140.4 Member Data Documentation	597
7.141spot::state_shared_ptr_equal Struct Reference	598
7.141.1 Detailed Description	598
7.141.2 Member Function Documentation	598
7.142spot::state_shared_ptr_hash Struct Reference	598
7.142.1 Detailed Description	598
7.142.2 Member Function Documentation	599
7.143spot::state_shared_ptr_less_than Struct Reference	599
7.143.1 Detailed Description	599
7.143.2 Member Function Documentation	599
7.144spot::state_union Class Reference	599
7.144.1 Detailed Description	602
7.144.2 Constructor & Destructor Documentation	602
7.144.3 Member Function Documentation	603
7.144.4 Member Data Documentation	604
7.145spot::tgba_run::step Struct Reference	604

7.145.1 Member Data Documentation	605
7.146spot::string_hash Struct Reference	606
7.146.1 Detailed Description	606
7.146.2 Member Function Documentation	606
7.147spot::ltdl::succ_iterator Class Reference	606
7.147.1 Constructor & Destructor Documentation	607
7.147.2 Member Function Documentation	607
7.147.3 Member Data Documentation	607
7.148spot::couvreur99_check_shy::successor Struct Reference	607
7.148.1 Constructor & Destructor Documentation	608
7.148.2 Member Data Documentation	609
7.149spot::symbol Class Reference	609
7.149.1 Member Typedef Documentation	610
7.149.2 Constructor & Destructor Documentation	610
7.149.3 Member Function Documentation	610
7.149.4 Member Data Documentation	611
7.150spot::taa_succ_iterator Class Reference	611
7.150.1 Member Typedef Documentation	614
7.150.2 Constructor & Destructor Documentation	615
7.150.3 Member Function Documentation	615
7.150.4 Member Data Documentation	616
7.151spot::taa_tgba Class Reference	617
7.151.1 Detailed Description	621
7.151.2 Member Typedef Documentation	621
7.151.3 Constructor & Destructor Documentation	622
7.151.4 Member Function Documentation	622
7.151.5 Member Data Documentation	625
7.152spot::taa_tgba_formula Class Reference	626
7.152.1 Member Typedef Documentation	630
7.152.2 Constructor & Destructor Documentation	631
7.152.3 Member Function Documentation	631
7.152.4 Member Data Documentation	636
7.153spot::taa_tgba_labelled< label, label_hash > Class Template Reference	637
7.153.1 Detailed Description	641
7.153.2 Member Typedef Documentation	642
7.153.3 Constructor & Destructor Documentation	642

7.153.4 Member Function Documentation	642
7.153.5 Member Data Documentation	648
7.154spot::taa_tgba_string Class Reference	649
7.154.1 Member Typedef Documentation	653
7.154.2 Constructor & Destructor Documentation	654
7.154.3 Member Function Documentation	654
7.154.4 Member Data Documentation	659
7.155spot::tgba Class Reference	660
7.155.1 Detailed Description	664
7.155.2 Constructor & Destructor Documentation	664
7.155.3 Member Function Documentation	664
7.155.4 Member Data Documentation	668
7.156spot::tgba_bdd_concrete Class Reference	668
7.156.1 Detailed Description	672
7.156.2 Constructor & Destructor Documentation	672
7.156.3 Member Function Documentation	673
7.156.4 Member Data Documentation	677
7.157spot::tgba_bdd_concrete_factory Class Reference	677
7.157.1 Detailed Description	680
7.157.2 Member Typedef Documentation	680
7.157.3 Constructor & Destructor Documentation	680
7.157.4 Member Function Documentation	681
7.157.5 Member Data Documentation	682
7.158spot::tgba_bdd_core_data Struct Reference	683
7.158.1 Detailed Description	686
7.158.2 Constructor & Destructor Documentation	686
7.158.3 Member Function Documentation	687
7.158.4 Member Data Documentation	687
7.159spot::tgba_bdd_factory Class Reference	690
7.159.1 Detailed Description	691
7.159.2 Constructor & Destructor Documentation	692
7.159.3 Member Function Documentation	692
7.160spot::tgba_explicit Class Reference	692
7.160.1 Detailed Description	696
7.160.2 Member Typedef Documentation	696
7.160.3 Constructor & Destructor Documentation	697

7.160.4 Member Function Documentation	697
7.160.5 Member Data Documentation	701
7.161spot::tgba_explicit_formula Class Reference	702
7.161.1 Member Typedef Documentation	707
7.161.2 Constructor & Destructor Documentation	707
7.161.3 Member Function Documentation	707
7.161.4 Member Data Documentation	713
7.162spot::tgba_explicit_labelled< label, label_hash > Class Template Reference	714
7.162.1 Detailed Description	719
7.162.2 Member Typedef Documentation	719
7.162.3 Constructor & Destructor Documentation	719
7.162.4 Member Function Documentation	719
7.162.5 Member Data Documentation	725
7.163spot::tgba_explicit_number Class Reference	726
7.163.1 Member Typedef Documentation	731
7.163.2 Constructor & Destructor Documentation	731
7.163.3 Member Function Documentation	731
7.163.4 Member Data Documentation	737
7.164spot::tgba_explicit_string Class Reference	738
7.164.1 Member Typedef Documentation	743
7.164.2 Constructor & Destructor Documentation	743
7.164.3 Member Function Documentation	743
7.164.4 Member Data Documentation	749
7.165spot::tgba_explicit_succ_iterator Class Reference	750
7.165.1 Detailed Description	753
7.165.2 Constructor & Destructor Documentation	753
7.165.3 Member Function Documentation	753
7.165.4 Member Data Documentation	755
7.166spot::tgba_kv_complement Class Reference	755
7.166.1 Detailed Description	759
7.166.2 Constructor & Destructor Documentation	759
7.166.3 Member Function Documentation	759
7.166.4 Member Data Documentation	763
7.167spot::tgba_product Class Reference	763
7.167.1 Detailed Description	767
7.167.2 Constructor & Destructor Documentation	767

7.167.3 Member Function Documentation	767
7.167.4 Member Data Documentation	771
7.168spot::tgba_product_init Class Reference	771
7.168.1 Detailed Description	775
7.168.2 Constructor & Destructor Documentation	775
7.168.3 Member Function Documentation	775
7.168.4 Member Data Documentation	778
7.169spot::tgba_reachable_iterator Class Reference	778
7.169.1 Detailed Description	781
7.169.2 Member Typedef Documentation	781
7.169.3 Constructor & Destructor Documentation	782
7.169.4 Member Function Documentation	782
7.169.5 Member Data Documentation	783
7.170spot::tgba_reachable_iterator_breadth_first Class Reference	784
7.170.1 Detailed Description	787
7.170.2 Member Typedef Documentation	787
7.170.3 Constructor & Destructor Documentation	788
7.170.4 Member Function Documentation	788
7.170.5 Member Data Documentation	789
7.171spot::tgba_reachable_iterator_depth_first Class Reference	790
7.171.1 Detailed Description	793
7.171.2 Member Typedef Documentation	793
7.171.3 Constructor & Destructor Documentation	794
7.171.4 Member Function Documentation	794
7.171.5 Member Data Documentation	795
7.172spot::tgba_reduc Class Reference	796
7.172.1 Detailed Description	802
7.172.2 Member Typedef Documentation	802
7.172.3 Constructor & Destructor Documentation	803
7.172.4 Member Function Documentation	803
7.172.5 Member Data Documentation	813
7.173spot::tgba_run Struct Reference	814
7.173.1 Detailed Description	814
7.173.2 Member Typedef Documentation	815
7.173.3 Constructor & Destructor Documentation	815
7.173.4 Member Function Documentation	815

7.173.5 Member Data Documentation	815
7.174spot::tgba_run_dotty_decorator Class Reference	815
7.174.1 Detailed Description	818
7.174.2 Member Typedef Documentation	818
7.174.3 Constructor & Destructor Documentation	818
7.174.4 Member Function Documentation	819
7.174.5 Member Data Documentation	820
7.175spot::tgba_saфра_complement Class Reference	820
7.175.1 Detailed Description	824
7.175.2 Constructor & Destructor Documentation	824
7.175.3 Member Function Documentation	824
7.175.4 Member Data Documentation	828
7.176spot::tgba_sba_proxy Class Reference	828
7.176.1 Detailed Description	832
7.176.2 Member Typedef Documentation	832
7.176.3 Constructor & Destructor Documentation	832
7.176.4 Member Function Documentation	832
7.176.5 Member Data Documentation	836
7.177spot::tgba_scc Class Reference	836
7.177.1 Detailed Description	840
7.177.2 Constructor & Destructor Documentation	840
7.177.3 Member Function Documentation	840
7.177.4 Member Data Documentation	843
7.178spot::tgba_sgba_proxy Class Reference	844
7.178.1 Detailed Description	848
7.178.2 Constructor & Destructor Documentation	848
7.178.3 Member Function Documentation	848
7.178.4 Member Data Documentation	852
7.179spot::tgba_statistics Struct Reference	852
7.179.1 Member Function Documentation	852
7.179.2 Member Data Documentation	852
7.180spot::tgba_succ_iterator Class Reference	853
7.180.1 Detailed Description	854
7.180.2 Constructor & Destructor Documentation	854
7.180.3 Member Function Documentation	854
7.181spot::tgba_succ_iterator_concrete Class Reference	855

7.181.1 Detailed Description	858
7.181.2 Constructor & Destructor Documentation	859
7.181.3 Member Function Documentation	859
7.181.4 Member Data Documentation	860
7.182spot::tgba_succ_iterator_product Class Reference	861
7.182.1 Detailed Description	864
7.182.2 Constructor & Destructor Documentation	865
7.182.3 Member Function Documentation	865
7.182.4 Friends And Related Function Documentation	866
7.182.5 Member Data Documentation	867
7.183spot::tgba_succ_iterator_union Class Reference	867
7.183.1 Detailed Description	870
7.183.2 Constructor & Destructor Documentation	870
7.183.3 Member Function Documentation	871
7.183.4 Friends And Related Function Documentation	872
7.183.5 Member Data Documentation	872
7.184spot::tgba_tba_proxy Class Reference	873
7.184.1 Detailed Description	877
7.184.2 Member Typedef Documentation	877
7.184.3 Constructor & Destructor Documentation	878
7.184.4 Member Function Documentation	878
7.184.5 Member Data Documentation	881
7.185spot::tgba_union Class Reference	882
7.185.1 Detailed Description	886
7.185.2 Constructor & Destructor Documentation	886
7.185.3 Member Function Documentation	887
7.185.4 Member Data Documentation	890
7.186spot::time_info Struct Reference	891
7.186.1 Detailed Description	891
7.186.2 Constructor & Destructor Documentation	891
7.186.3 Member Data Documentation	892
7.187spot::timer Class Reference	892
7.187.1 Detailed Description	894
7.187.2 Constructor & Destructor Documentation	894
7.187.3 Member Function Documentation	894
7.187.4 Member Data Documentation	895

7.188spot::timer_map Class Reference	895
7.188.1 Detailed Description	896
7.188.2 Member Typedef Documentation	896
7.188.3 Member Function Documentation	896
7.188.4 Member Data Documentation	898
7.189spot::couvreur99_check_shy::todo_item Struct Reference	898
7.189.1 Constructor & Destructor Documentation	899
7.189.2 Member Data Documentation	899
7.190spot::taa_tgba::transition Struct Reference	899
7.190.1 Detailed Description	899
7.190.2 Member Data Documentation	900
7.191spot::ltl::nfa::transition Struct Reference	900
7.191.1 Detailed Description	900
7.191.2 Member Data Documentation	900
7.192spot::tgba_explicit::transition Struct Reference	901
7.192.1 Detailed Description	901
7.192.2 Member Data Documentation	901
7.193spot::evtgba_explicit::transition Struct Reference	901
7.193.1 Detailed Description	903
7.193.2 Member Data Documentation	903
7.194spot::ltl::automatop::triplecmp Struct Reference	903
7.194.1 Detailed Description	903
7.194.2 Member Function Documentation	903
7.195spot::ltl::unabbreviate_logic_visitor Class Reference	904
7.195.1 Detailed Description	907
7.195.2 Member Typedef Documentation	907
7.195.3 Constructor & Destructor Documentation	907
7.195.4 Member Function Documentation	908
7.195.5 Member Data Documentation	909
7.196spot::ltl::unabbreviate_ltl_visitor Class Reference	909
7.196.1 Detailed Description	912
7.196.2 Member Typedef Documentation	912
7.196.3 Constructor & Destructor Documentation	912
7.196.4 Member Function Documentation	913
7.196.5 Member Data Documentation	914
7.197spot::ltl::unop Class Reference	914

7.197.1 Detailed Description	918
7.197.2 Member Typedef Documentation	918
7.197.3 Member Enumeration Documentation	919
7.197.4 Constructor & Destructor Documentation	919
7.197.5 Member Function Documentation	919
7.197.6 Member Data Documentation	921
7.198spot::unsigned_statistics Struct Reference	922
7.198.1 Member Typedef Documentation	924
7.198.2 Constructor & Destructor Documentation	924
7.198.3 Member Function Documentation	924
7.198.4 Member Data Documentation	924
7.199spot::unsigned_statistics_copy Class Reference	925
7.199.1 Detailed Description	925
7.199.2 Member Typedef Documentation	925
7.199.3 Constructor & Destructor Documentation	925
7.199.4 Member Function Documentation	926
7.199.5 Member Data Documentation	926
7.200spot::ltdl::visitor Struct Reference	926
7.200.1 Detailed Description	928
7.200.2 Constructor & Destructor Documentation	928
7.200.3 Member Function Documentation	928
7.201spot::weight Class Reference	929
7.201.1 Detailed Description	930
7.201.2 Member Typedef Documentation	930
7.201.3 Constructor & Destructor Documentation	930
7.201.4 Member Function Documentation	930
7.201.5 Friends And Related Function Documentation	931
7.201.6 Member Data Documentation	931
8 File Documentation	931
8.1 mainpage.dox File Reference	931
8.2 gspn/common.hh File Reference	931
8.3 nips/common.hh File Reference	933
8.4 gspn/gspn.hh File Reference	934
8.5 gspn/ssp.hh File Reference	934
8.6 nips/nips.hh File Reference	935
8.6.1 Typedef Documentation	936

8.7	eltlparse/location.hh File Reference	936
8.8	ltlparse/location.hh File Reference	938
8.9	neverparse/location.hh File Reference	940
8.10	sautparse/location.hh File Reference	941
8.11	eltlparse/position.hh File Reference	943
8.12	ltlparse/position.hh File Reference	945
8.13	neverparse/position.hh File Reference	946
8.14	sautparse/position.hh File Reference	948
8.15	eltlparse/public.hh File Reference	949
8.16	evtgbaparse/public.hh File Reference	951
8.17	ltlparse/public.hh File Reference	952
8.18	neverparse/public.hh File Reference	953
8.19	tgba/public.hh File Reference	954
8.20	tgbaparse/public.hh File Reference	955
8.21	eltlparse/stack.hh File Reference	956
8.22	ltlparse/stack.hh File Reference	957
8.23	neverparse/stack.hh File Reference	958
8.24	sautparse/stack.hh File Reference	958
8.25	evtgba/evtgba.hh File Reference	959
8.26	evtgba/evtgbaiter.hh File Reference	960
8.27	evtgba/explicit.hh File Reference	961
8.28	evtgba/product.hh File Reference	962
8.29	evtgba/symbol.hh File Reference	963
8.30	evtgbaalgos/dotty.hh File Reference	965
8.31	ltlvisit/dotty.hh File Reference	965
8.32	tgbaalgos/dotty.hh File Reference	966
8.33	evtgbaalgos/reachiter.hh File Reference	967
8.34	tgbaalgos/reachiter.hh File Reference	968
8.35	evtgbaalgos/save.hh File Reference	970
8.36	tgbaalgos/save.hh File Reference	970
8.37	evtgbaalgos/tgba2evtgba.hh File Reference	971
8.38	kripke/fairkripke.hh File Reference	971
8.39	kripke/kripke.hh File Reference	973
8.40	ltlast/allnodes.hh File Reference	973
	8.40.1 Detailed Description	974
8.41	ltlast/atomic_prop.hh File Reference	974

8.41.1 Detailed Description	976
8.42 Itlast/automatop.hh File Reference	976
8.42.1 Detailed Description	977
8.43 Itlast/binop.hh File Reference	977
8.43.1 Detailed Description	979
8.44 Itlast/constant.hh File Reference	979
8.44.1 Detailed Description	980
8.45 Itlast/formula.hh File Reference	980
8.45.1 Detailed Description	981
8.46 Itlast/formula_tree.hh File Reference	982
8.46.1 Detailed Description	983
8.47 Itlast/multop.hh File Reference	983
8.47.1 Detailed Description	985
8.48 Itlast/nfa.hh File Reference	985
8.48.1 Detailed Description	986
8.49 Itlast/predecl.hh File Reference	986
8.49.1 Detailed Description	987
8.50 Itlast/reformula.hh File Reference	987
8.50.1 Detailed Description	988
8.51 Itlast/unop.hh File Reference	988
8.51.1 Detailed Description	990
8.52 Itlast/visitor.hh File Reference	990
8.52.1 Detailed Description	991
8.53 Itlenv/declenv.hh File Reference	991
8.54 Itlenv/defaultenv.hh File Reference	993
8.55 Itlenv/environment.hh File Reference	994
8.56 Itlparse/Itlfile.hh File Reference	995
8.57 Itlvisit/apcollect.hh File Reference	996
8.58 Itlvisit/basicreduce.hh File Reference	998
8.59 Itlvisit/clone.hh File Reference	999
8.60 Itlvisit/contain.hh File Reference	1000
8.61 Itlvisit/destroy.hh File Reference	1001
8.62 Itlvisit/dump.hh File Reference	1002
8.63 Itlvisit/length.hh File Reference	1003
8.64 Itlvisit/lunabbrev.hh File Reference	1004
8.65 Itlvisit/nenoform.hh File Reference	1006

8.66	ltlvisit/postfix.hh File Reference	1007
8.67	ltlvisit/randomltl.hh File Reference	1008
8.68	ltlvisit/reduce.hh File Reference	1009
8.69	ltlvisit/simpfg.hh File Reference	1011
8.70	ltlvisit/syntimpl.hh File Reference	1011
8.71	ltlvisit/tostring.hh File Reference	1012
8.72	ltlvisit/tunabbrev.hh File Reference	1013
8.73	misc/bareword.hh File Reference	1015
8.74	misc/bddalloc.hh File Reference	1015
8.75	misc/bddltl.hh File Reference	1016
8.76	misc/bddop.hh File Reference	1017
8.77	misc/escape.hh File Reference	1018
8.78	misc/freelist.hh File Reference	1019
8.79	misc/hash.hh File Reference	1020
8.80	misc/hashfunc.hh File Reference	1021
8.81	misc/ltstr.hh File Reference	1022
8.82	misc/memusage.hh File Reference	1023
8.83	misc/minato.hh File Reference	1023
8.84	misc/modgray.hh File Reference	1024
8.85	misc/optionmap.hh File Reference	1024
8.86	misc/random.hh File Reference	1025
8.87	misc/timer.hh File Reference	1026
8.88	misc/version.hh File Reference	1027
8.89	saba/explicitstateconjunction.hh File Reference	1027
8.90	saba/saba.hh File Reference	1028
8.91	saba/sabacomplementtgba.hh File Reference	1029
8.92	saba/sabastate.hh File Reference	1030
8.93	saba/sabasucciter.hh File Reference	1032
8.94	sabaalgos/sabadotty.hh File Reference	1034
8.95	sabaalgos/sabareachiter.hh File Reference	1034
8.96	tgba/bdddict.hh File Reference	1035
8.97	tgba/bddprint.hh File Reference	1036
8.98	tgba/formula2bdd.hh File Reference	1038
8.99	tgba/futurecondcol.hh File Reference	1039
8.100	tgba/state.hh File Reference	1040
8.101	tgba/statebdd.hh File Reference	1042

8.102tgba/succiter.hh File Reference	1043
8.103tgba/succiterconcrete.hh File Reference	1044
8.104tgba/taatgba.hh File Reference	1046
8.105tgba/tgba.hh File Reference	1047
8.106tgba/tgbabddconcrete.hh File Reference	1048
8.107tgba/tgbabddconcretefactory.hh File Reference	1049
8.108tgba/tgbabddconcreteproduct.hh File Reference	1050
8.109tgba/tgbabddcoredata.hh File Reference	1052
8.110tgba/tgbabddfactory.hh File Reference	1053
8.111tgba/tgbaexplicit.hh File Reference	1055
8.112tgba/tgbakvcomplement.hh File Reference	1056
8.113tgba/tgbaproduct.hh File Reference	1057
8.114tgba/tgbareduc.hh File Reference	1058
8.115tgba/tgbasafracomplement.hh File Reference	1059
8.115.1 Define Documentation	1060
8.116tgba/tgbascc.hh File Reference	1060
8.117tgba/tgbasgba.hh File Reference	1061
8.118tgba/tgbatba.hh File Reference	1063
8.119tgba/tgbaunion.hh File Reference	1064
8.120tgba/wdbacomp.hh File Reference	1065
8.121tgbaalgos/bfssteps.hh File Reference	1065
8.122tgbaalgos/cutscscc.hh File Reference	1066
8.123tgbaalgos/dottydec.hh File Reference	1067
8.124tgbaalgos/dupeexp.hh File Reference	1068
8.125tgbaalgos/eltl2tgba_lacim.hh File Reference	1069
8.126tgbaalgos/emptiness.hh File Reference	1070
8.127tgbaalgos/emptiness_stats.hh File Reference	1072
8.128tgbaalgos/gtec/ce.hh File Reference	1073
8.129tgbaalgos/gtec/explscscc.hh File Reference	1074
8.130tgbaalgos/gtec/gtec.hh File Reference	1075
8.131tgbaalgos/gtec/nsheap.hh File Reference	1077
8.132tgbaalgos/gtec/scsccstack.hh File Reference	1078
8.133tgbaalgos/gtec/status.hh File Reference	1079
8.134tgbaalgos/gv04.hh File Reference	1080
8.135tgbaalgos/lbtt.hh File Reference	1081
8.136tgbaalgos/ltl2taa.hh File Reference	1082

8.137tgbalgorithms/ltl2tgb_fm.hh File Reference	1083
8.138tgbalgorithms/ltl2tgb_lacim.hh File Reference	1084
8.139tgbalgorithms/magic.hh File Reference	1085
8.140tgbalgorithms/minimize.hh File Reference	1086
8.141tgbalgorithms/neverclaim.hh File Reference	1087
8.142tgbalgorithms/powerset.hh File Reference	1088
8.143tgbalgorithms/projrun.hh File Reference	1089
8.144tgbalgorithms/randomgraph.hh File Reference	1090
8.145tgbalgorithms/reducerun.hh File Reference	1092
8.146tgbalgorithms/reductgba_sim.hh File Reference	1092
8.147tgbalgorithms/replayrun.hh File Reference	1094
8.148tgbalgorithms/rundotdec.hh File Reference	1094
8.149tgbalgorithms/safety.hh File Reference	1095
8.150tgbalgorithms/scc.hh File Reference	1096
8.151tgbalgorithms/sccfilter.hh File Reference	1097
8.152tgbalgorithms/se05.hh File Reference	1098
8.153tgbalgorithms/stats.hh File Reference	1099
8.154tgbalgorithms/tau03.hh File Reference	1100
8.155tgbalgorithms/tau03opt.hh File Reference	1101
8.156tgbalgorithms/weight.hh File Reference	1102

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.

Member [spot::state::~~state](#)() Client code should now call `s->destroy()` ; instead of `delete s;`.

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

- [Kripke Structures](#)
- [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 ELTL 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 <l_string, [parse_error_list](#) &error_list, environment &env=default_environment::instance(), bool debug=false)

Build a formula from an LTL string.

- bool [spot::ltl::format_parse_errors](#) (std::ostream &os, const std::string <l_string, [parse_error_list](#) &error_list)

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

- std::ostream & [spot::ltl::dotty](#) (std::ostream &os, const formula *f)

Write a formula tree using dot's syntax.

- std::ostream & [spot::ltl::dump](#) (std::ostream &os, const formula *f)

Dump a formula tree.

- std::ostream & [spot::ltl::to_string](#) (const formula *f, std::ostream &os, 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 [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).

- std::ostream & [spot::ltl::to_spin_string](#) (const formula *f, std::ostream &os, bool full_parent=false)

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

- std::string [spot::ltl::to_spin_string](#) (const formula *f, bool full_parent=false)

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

4.5.1 Typedef Documentation

4.5.1.1 `typedef std::pair<eltlyy::location, spair> spot::eltl::parse_error`

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

4.5.1.2 `typedef std::pair<ltlyy::location, std::string> spot::ltl::parse_error`

A parse diagnostic with its location.

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 unparseable.

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

Warning

This function is not reentrant.

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::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 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 unparseable.

Warning

This function is not reentrant.

4.5.2.8 `std::ostream& spot::ltl::to_spin_string (const formula * f, std::ostream & os, bool full_parent = false)`

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

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.

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

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

Parameters

f The formula to translate.

full_parent Whether or not the string should be fully parenthesized.

4.5.2.10 `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.

4.5.2.11 `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.

4.6 Kripke Structures

Classes

- class `spot::fair_kripke_succ_iterator`
*Iterator code for a Fair Kripke structure.
This iterator can be used to simplify the writing of an iterator on a Fair Kripke structure (or lookalike).*
- class `spot::fair_kripke`
*Interface for a Fair Kripke structure.
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition, and a set of acceptance conditions.*
- class `spot::kripke_succ_iterator`
*Iterator code for Kripke structure
This iterator can be used to simplify the writing of an iterator on a Kripke structure (or lookalike).*
- class `spot::kripke`
*Interface for a Kripke structure
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition.*

4.7 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.

4.7.1 Function Documentation

4.7.1.1 formula* spot::ltl::clone (const formula * f)

Clone a formula.

Deprecated

Use `f->clone()` instead.

4.7.1.2 void spot::ltl::destroy (const formula * f)

Destroys a formula.

Deprecated

Use `f->destroy()` instead.

4.8 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.

4.9 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.

4.9.1 Detailed Description

LTL environment implementations.

4.10 Algorithms for LTL formulae

Modules

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

4.11 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 $true \vee f$ and $false \wedge g$ by f and $G \wedge g$.
- 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](#).

4.12 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 U f` and `false R g` by `F f` and `G g`.

4.12.1 Enumeration Type Documentation

4.12.1.1 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.

4.12.2 Function Documentation

4.12.2.1 `formula* spot::ltl::basic_reduce (const formula * f)`

Basic rewritings.

4.12.2.2 `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.)

4.12.2.3 `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

4.12.2.4 `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`

4.12.2.5 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`.

4.13 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.

4.13.1 Typedef Documentation

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

Set of atomic propositions.

4.13.2 Function Documentation

4.13.2.1 `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.

4.13.2.2 `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.

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

Whether a formula starts with FG.

4.13.2.4 bool spot::ltl::is_GF (const formula * f)

Whether a formula starts with GF.

4.13.2.5 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.

4.13.2.6 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).

4.13.2.7 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}
/// }
///

```

4.13.2.8 `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`

4.14 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 derecursived 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.

4.14.1 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.

4.14.2 Function Documentation

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

Escape " and \ characters in *str*.

4.14.2.2 `std::string spot::escape_str (const std::string & str)`

Escape " and \ characters in *str*.

4.14.2.3 `bool spot::is_bare_word (const char * str)`

4.14.2.4 `std::string spot::quote_unless_bare_word (const std::string & str)`

Double-quote words that are not bare.

See also

`is_bare_word`

4.14.2.5 `const char* spot::version ()`

Return Spot's version.

4.15 Hashing functions

Classes

- struct [spot::ltd::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::identity_hash< T >](#)
A hash function that returns identity.
- 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.

4.15.1 Function Documentation

4.15.1.1 `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::ptr_hash< T >::operator()`.

4.15.1.2 `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>

4.16 Random functions

Classes

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

Functions

- void `spot::srand (unsigned int seed)`
Reset the seed of the pseudo-random number generator.
- int `spot::rrand (int min, int max)`
Compute a pseudo-random integer value between min and max included.
- int `spot::mrand (int max)`
Compute a pseudo-random integer value between 0 and max-1 included.

- double `spot::drand()`
Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).
- double `spot::nrand()`
Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).
- double `spot::bmrnd()`
Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).
- int `spot::prand(double p)`
Return a pseudo-random positive integer value following a Poisson distribution with parameter p .

4.16.1 Function Documentation

4.16.1.1 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.

4.16.1.2 double `spot::drand()`

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

See also

`mrnd`, `rrand`, `srnd`

4.16.1.3 int `spot::mrnd(int max)`

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

See also

`drand`, `rrand`, `srnd`

4.16.1.4 double `spot::nrand()`

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

4.16.1.5 `int spot::prand (double p)`

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

Precondition

$p > 0$

4.16.1.6 `int spot::rrand (int min, int max)`

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

See also

`drand`, `mrnd`, `srnd`

4.16.1.7 `void spot::srand (unsigned int seed)`

Reset the seed of the pseudo-random number generator.

See also

`drand`, `mrnd`, `rrand`

4.17 Input/Output of TGBA

Modules

- [Decorating the dot output](#)

Typedefs

- `typedef std::pair< neverclaimyy::location, std::string > spot::neverclaim_parse_error`
A parse diagnostic with its location.
- `typedef std::list< neverclaim_parse_error > spot::neverclaim_parse_error_list`
A list of parser diagnostics, as filled by parse.
- `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::neverclaim_parse` (`const std::string &filename`, `neverclaim_parse_error_list &error_list`, `bdd_dict *dict`, `ltl::environment &env=ltl::default_environment::instance()`, `bool debug=false`)
Build a `spot::tgba_explicit` from a Spin never claim file.
- `bool spot::format_neverclaim_parse_errors` (`std::ostream &os`, `const std::string &filename`, `neverclaim_parse_error_list &error_list`)
Format diagnostics produced by `spot::neverclaim_parse`.
- `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::lbtt_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`.

4.17.1 Typedef Documentation

4.17.1.1 `typedef std::pair<neverclaimmy::location, std::string> spot::neverclaim_parse_error`

A parse diagnostic with its location.

4.17.1.2 `typedef std::list<neverclaim_parse_error> spot::neverclaim_parse_error_list`

A list of parser diagnostics, as filled by `parse`.

4.17.1.3 `typedef std::pair<tgba::location, std::string> spot::tgba_parse_error`

A parse diagnostic with its location.

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

A list of parser diagnostics, as filled by `parse`.

4.17.2 Function Documentation

4.17.2.1 `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.

4.17.2.2 `bool spot::format_neverclaim_parse_errors (std::ostream & os, const std::string & filename, neverclaim_parse_error_list & error_list)`

Format diagnostics produced by `spot::neverclaim_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.

4.17.2.3 `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.

4.17.2.4 `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.

4.17.2.5 `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.

4.17.2.6 `tgba_explicit_string* spot::neverclaim_parse (const std::string & filename, neverclaim_parse_error_list & error_list, bdd_dict * dict, ltl::environment & env = ltl::default_environment::instance(), bool debug = false)`

Build a [spot::tgba_explicit](#) from a Spin never claim 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.

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

Returns

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

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

Warning

This function is not reentrant.

```
4.17.2.7 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 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.

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

Save reachable states in text format.

4.18 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),.*

4.19 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).*

4.20 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`

4.21 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.

4.21.1 Function Documentation

4.21.1.1 `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.

4.22 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.

Functions

- `tgba * spot::wdba_complement (const tgba *aut)`

Complement a weak deterministic Büchi automaton.

4.22.1 Function Documentation

4.22.1.1 `tgba* spot::wdba_complement (const tgba * aut)`

Complement a weak deterministic Büchi automaton.

Parameters

aut a weak deterministic Büchi automaton to complement

Returns

a new automaton that recognizes the complement language

4.23 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 ELTL 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:

4.23.1 Function Documentation

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

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

4.23.1.2 `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*.

4.23.13 [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 .

4.23.1.4 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{\`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*.

4.24 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.

4.25 TGBA simplifications

Classes

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

- class `spot::duplicator_node_delayed`
Duplicator node of parity game graph for delayed simulation.
- class `spot::parity_game_graph_delayed`

Typedefs

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

Enumerations

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

Functions

- `tgba_explicit_number * spot::minimize_monitor` (const `tgba *a`)
Construct a minimal deterministic monitor.
- `tgba_explicit_number * spot::minimize_wdba` (const `tgba *a`)
Minimize a Büchi automaton in the WDBA class.
- const `tgba * spot::minimize_obligation` (const `tgba *aut_f`, const `ltl::formula *f=0`, const `tgba *aut_neg_f=0`)
Minimize an automaton if it represents an obligation property.
- 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.

4.25.1 Typedef Documentation

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

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

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

4.25.2 Enumeration Type Documentation

4.25.2.1 `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.

4.25.3 Function Documentation

4.25.3.1 `void spot::free_relation_simulation (direct_simulation_relation * rel)`

To free a simulation relation.

4.25.3.2 `void spot::free_relation_simulation (delayed_simulation_relation * rel)`

To free a simulation relation.

4.25.3.3 `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.

4.25.3.4 `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*.

4.25.3.5 `tgba_explicit_number* spot::minimize_monitor (const tgba * a)`

Construct a minimal deterministic monitor.

The automaton will be converted into minimal deterministic monitor. All useless SCCs should have been previously removed (using `scc_filter()` for instance). Then the automaton will be determinized and minimized using the standard DFA construction as if all states were accepting states.

For more detail about monitors, see the following paper:

```
/// @InProceedings{ tabakov.10.rv,
///   author   = {Deian Tabakov and Moshe Y. Vardi},
///   title    = {Optimized Temporal Monitors for SystemC{^*}},
///   booktitle = {Proceedings of the 10th International Conference
///     on Runtime Verification},
///   pages    = {436--451},
///   year     = 2010,
///   volume   = {6418},
///   series   = {Lecture Notes in Computer Science},
///   month    = nov,
///   publisher = {Spring-Verlag}
/// }
```

(Note: although the above paper uses Spot, this function did not exist in Spot at that time.)

Parameters

a the automaton to convert into a minimal deterministic monitor

Precondition

Dead SCCs should have been removed from *a* before calling this function.

4.25.3.6 `const tgba* spot::minimize_obligation (const tgba * aut_f, const ltl::formula * f = 0, const tgba * aut_neg_f = 0)`

Minimize an automaton if it represents an obligation property.

This function attempts to minimize the automaton *aut_f* using the algorithm implemented in the [minimize_wdba\(\)](#) function, and presented by the following paper:

```

/// @InProceedings{dax.07.atva,
///   author   = {Christian Dax and Jochen Eisinger and Felix Klaedtke},
///   title    = {Mechanizing the Powerset Construction for Restricted
///               Classes of  $\{\omega\}$ -Automata},
///   year     = 2007,
///   series   = {Lecture Notes in Computer Science},
///   publisher = {Springer-Verlag},
///   volume   = 4762,
///   booktitle = {Proceedings of the 5th International Symposium on
///               Automated Technology for Verification and Analysis
///               (ATVA'07)},
///   editor    = {Kedar S. Namjoshi and Tomohiro Yoneda and Teruo Higashino
///               and Yoshio Okamura},
///   month     = oct
/// }
///

```

Because it is hard to determine if an automaton corresponds to an obligation property, you should supply either the formula *f* expressed by the automaton *aut_f*, or *aut_neg_f* the negation of the automaton *aut_neg_f*.

Parameters

aut_f the automaton to minimize

f the LTL formula represented by the automaton *aut_f*

aut_neg_f an automaton representing the negation of *aut_f*

Returns

a new tgba if the automaton could be minimized, *aut_f* if the automaton cannot be minimized, 0 if we do not if the minimization is correct because neither *f* nor *aut_neg_f* were supplied.

The function proceeds as follows. If the formula *f* or the automaton *aut* can easily be proved to represent an obligation formula, then the result of `minimize(aut)` is returned. Otherwise, if *aut_neg_f* was not supplied but *f* was, *aut_neg_f* is built from the negation of *f*. Then we check that `product(aut, !minimize(aut_f))` and `product(aut_neg_f, minimize(aut))` are both empty. If they are, the minimization was sound. (See the paper for full details.)

4.25.3.7 tgba_explicit_number* spot::minimize_wdba (const tgba * a)

Minimize a Büchi automaton in the WDBA class.

This takes a TGBA whose language is representable by a Weak Deterministic Büchi Automaton, and construct a minimal WDBA for this language.

If the input automaton does not represent a WDBA language, the resulting automaton is still a WDBA, but it will not be equivalent to the original automaton. Use the [minimize_obligation\(\)](#) function if you are not sure whether it is safe to call this function.

Please see the following paper for a discussion of this technique.

```

/// @InProceedings{ dax.07.atva,
///   author   = {Christian Dax and Jochen Eisinger and Felix Klaedtke},
///   title    = {Mechanizing the Powerset Construction for Restricted
///     Classes of {\omega$}-Automata},
///   year     = 2007,
///   series   = {Lecture Notes in Computer Science},
///   publisher = {Springer-Verlag},
///   volume   = 4762,
///   booktitle = {Proceedings of the 5th International Symposium on
///     Automated Technology for Verification and Analysis
///     (ATVA'07)},
///   editor   = {Kedar S. Namjoshi and Tomohiro Yoneda and Teruo Higashino
///     and Yoshio Okamura},
///   month    = oct
/// }
///

```

4.25.3.8 `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::reduc_tgba_options` specifying which optimizations to apply.

Returns

the reduced automata.

4.26 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 * 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=<tl::default_environment::instance())`
Construct a tgba randomly.

- `tgba_statistics spot::stats_reachable` (const `tgba *g`)
Compute statistics for an automaton.
- `tgba_explicit_number * spot::tgba_powerset` (const `tgba *aut`, `power_map &pm`)
*Build a deterministic automaton, ignoring acceptance conditions.
This create a deterministic automaton that recognizes the same language as aut would if its acceptance conditions were ignored. This is the classical powerset algorithm.*
- `tgba_explicit_number * spot::tgba_powerset` (const `tgba *aut`)

4.26.1 Function Documentation

4.26.1.1 `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 = <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.)

4.26.1.2 tgba_statistics spot::stats_reachable (const tgba * g)

Compute statistics for an automaton.

4.26.1.3 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.

4.26.1.4 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.

4.26.1.5 tgba_explicit_number* spot::tgba_powerset (const tgba * aut, power_map & pm)

Build a deterministic automaton, ignoring acceptance conditions.

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

If *pm* is supplied it will be filled with the set of original states associated to each state of the deterministic automaton.

4.26.1.6 tgba_explicit_number* spot::tgba_powerset (const tgba * aut)

4.27 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`.

4.28 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](#)

4.28.1 Detailed Description

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

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

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

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

4.29 Emptiness-check algorithms

Classes

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

Functions

- `emptiness_check * spot::couvreur99` (`const tgba *a`, `option_map options=option_map()`, `const numbered_state_heap_factory *nshf=numbered_state_heap_hash_map_factory::instance()`)
Check whether the language of an automate is empty.
- `emptiness_check * spot::explicit_gv04_check` (`const tgba *a`, `option_map o=option_map()`)
Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.
- `emptiness_check * spot::explicit_magic_search` (`const tgba *a`, `option_map o=option_map()`)
*Returns an emptiness checker on the `spot::tgba` automaton *a*.*
- `emptiness_check * spot::bit_state_hashing_magic_search` (`const tgba *a`, `size_t size`, `option_map o=option_map()`)
*Returns an emptiness checker on the `spot::tgba` automaton *a*.*
- `emptiness_check * spot::magic_search` (`const tgba *a`, `option_map o=option_map()`)
Wrapper for the two `magic_search` implementations.
- `emptiness_check * spot::explicit_se05_search` (`const tgba *a`, `option_map o=option_map()`)
*Returns an emptiness check on the `spot::tgba` automaton *a*.*
- `emptiness_check * spot::bit_state_hashing_se05_search` (`const tgba *a`, `size_t size`, `option_map o=option_map()`)
*Returns an emptiness checker on the `spot::tgba` automaton *a*.*
- `emptiness_check * spot::se05` (`const tgba *a`, `option_map o`)
Wrapper for the two `se05` implementations.
- `emptiness_check * spot::explicit_tau03_search` (`const tgba *a`, `option_map o=option_map()`)
*Returns an emptiness checker on the `spot::tgba` automaton *a*.*
- `emptiness_check * spot::explicit_tau03_opt_search` (`const tgba *a`, `option_map o=option_map()`)
*Returns an emptiness checker on the `spot::tgba` automaton *a*.*

4.29.1 Function Documentation

4.29.1.1 `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`

4.29.1.2 `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`

4.29.1.3 `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, SCC is and ACC are two stacks.)

```

/// check(<Sigma, Q, delta, q, F>, H, SCC, ACC)
///   if q is not in H // new state
///     H[q] = H.size + 1
///     SCC.push(<H[q], {}>)
///     forall <a, s> : <q, _, a, s> in delta
///       ACC.push(a)
///       res = check(<Sigma, Q, delta, s, F>, H, SCC, ACC)
///       if res
///         return res
///     <n, _> = SCC.top()
///     if n = H[q]
///       SCC.pop()
///       mark_reachable_states_as_dead(<Sigma, Q, delta, q, F>, H)
///     return 0
///   else
///     if H[q] = 0 // dead state
///       ACC.pop()
///       return true
///     else // state in stack: merge SCC
///       all = {}
///       do
///         <n, a> = SCC.pop()
///         all = all union a union { ACC.pop() }
///       until n <= H[q]
///       SCC.push(<n, all>)
///       if all != F
///         return 0
///     return new emptiness_check_result(necessary data)
///

```

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

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

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

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

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

The following options are available.

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

4.29.1.4 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}
/// }
///

```

4.29.1.5 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{
///   author      = {Costas Courcoubetis and Moshe Y. Vardi and Pierre
///                 Wolper and Mihalis Yannakakis},
///   title       = {Memory-Efficient Algorithm for the Verification of
///                 Temporal Properties},
///   journal     = {Formal Methods in System Design},
///   pages       = {275--288},
///   year       = {1992},
///   volume     = {1}
/// }
///

```

Bug

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

4.29.1.6 `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{"at"} 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`

4.29.1.7 `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.

4.29.1.8 `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}
/// }
///

```

4.29.1.9 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_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.

4.29.1.10 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.

4.30 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.*

4.30.1 Function Documentation

4.30.1.1 `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).

4.30.1.2 `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

4.30.1.3 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*.

4.30.1.4 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

4.30.1.5 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*.

4.31 Emptiness-check statistics

Classes

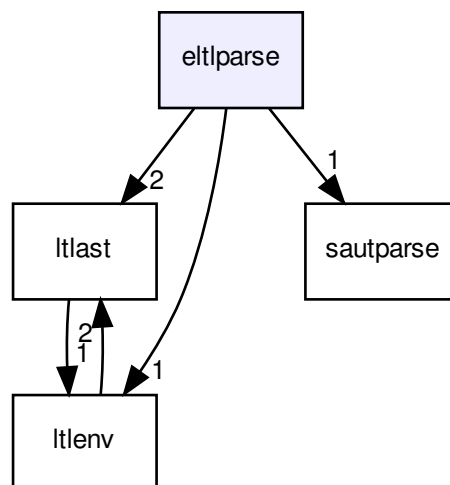
- struct [spot::unsigned_statistics](#)

- 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

Directory dependency graph for eltlparse/:

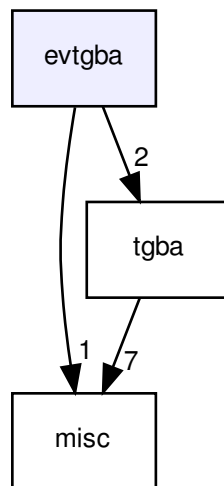


Files

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

5.2 evtgba/ Directory Reference

Directory dependency graph for evtgba/:

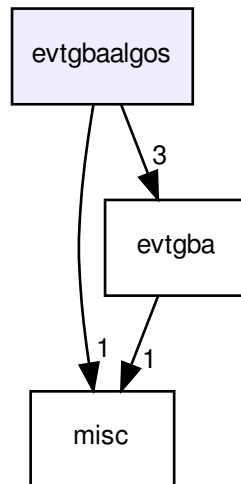


Files

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

5.3 evtgbaalgos/ Directory Reference

Directory dependency graph for evtgbaalgos/:

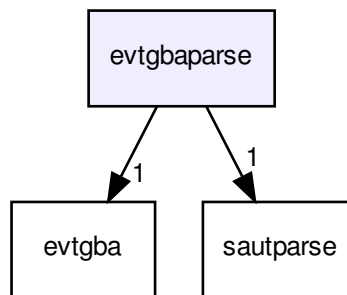


Files

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

5.4 evtgbaparse/ Directory Reference

Directory dependency graph for evtgbaparse/:

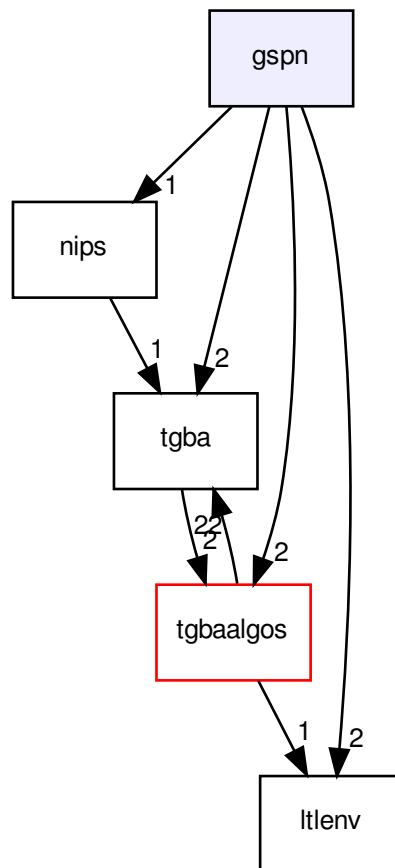


Files

- file [public.hh](#)

5.5 gspn/ Directory Reference

Directory dependency graph for gspn/:

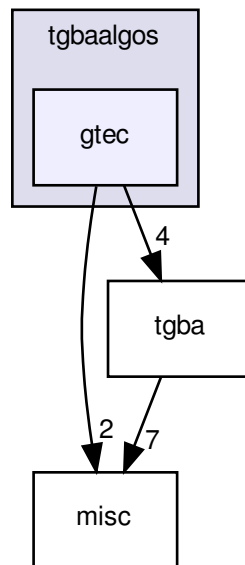


Files

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

5.6 tgbaalgos/gtec/ Directory Reference

Directory dependency graph for tgbaalgos/gtec/:

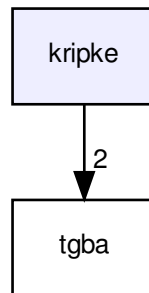


Files

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

5.7 kripke/ Directory Reference

Directory dependency graph for kripke/:

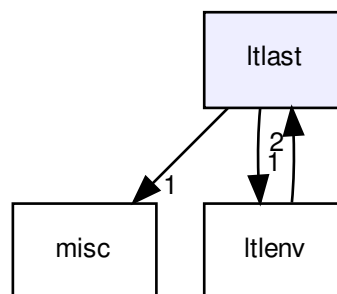


Files

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

5.8 ltlast/ Directory Reference

Directory dependency graph for ltlast/:



Files

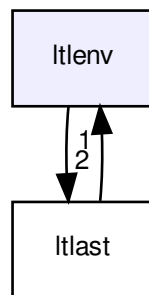
- file [allnodes.hh](#)

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

Directory dependency graph for Itlenv/:

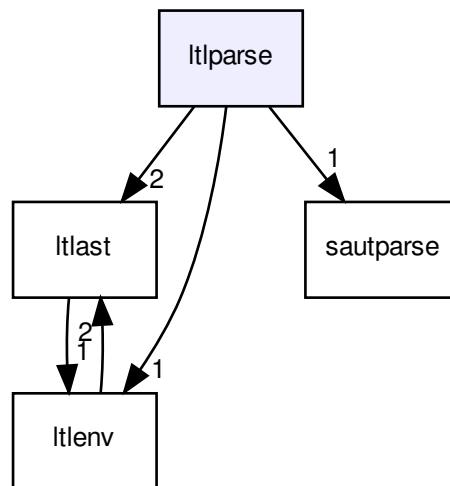


Files

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

5.10 Itlparse/ Directory Reference

Directory dependency graph for Itlparse/:

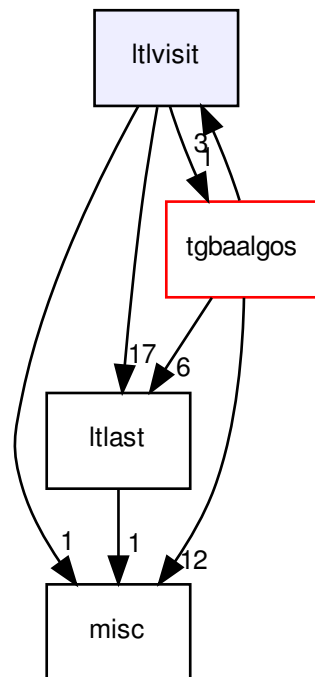


Files

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

5.11 Itlvisit/ Directory Reference

Directory dependency graph for Itlvisit/:



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 [nenofom.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

Directory dependency graph for misc/:

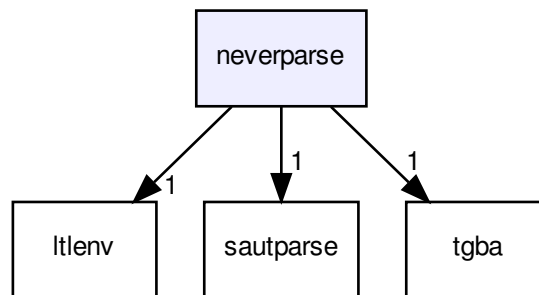


Files

- 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 neverparse/ Directory Reference

Directory dependency graph for neverparse/:

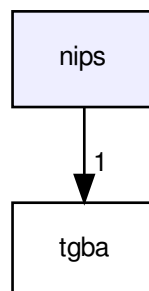


Files

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

5.14 nips/ Directory Reference

Directory dependency graph for nips/:

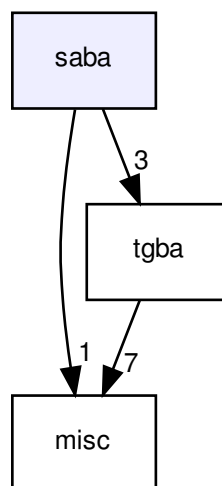


Files

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

5.15 saba/ Directory Reference

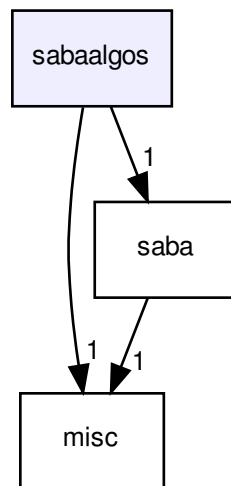
Directory dependency graph for saba/:

**Files**

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

5.16 sbaalgos/ Directory Reference

Directory dependency graph for sbaalgos/:

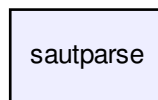


Files

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

5.17 sautparse/ Directory Reference

Directory dependency graph for sautparse/:



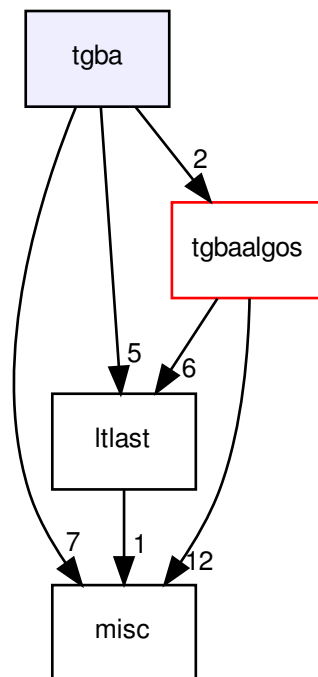
Files

- file [location.hh](#)

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

5.18 tgba/ Directory Reference

Directory dependency graph for tgba/:



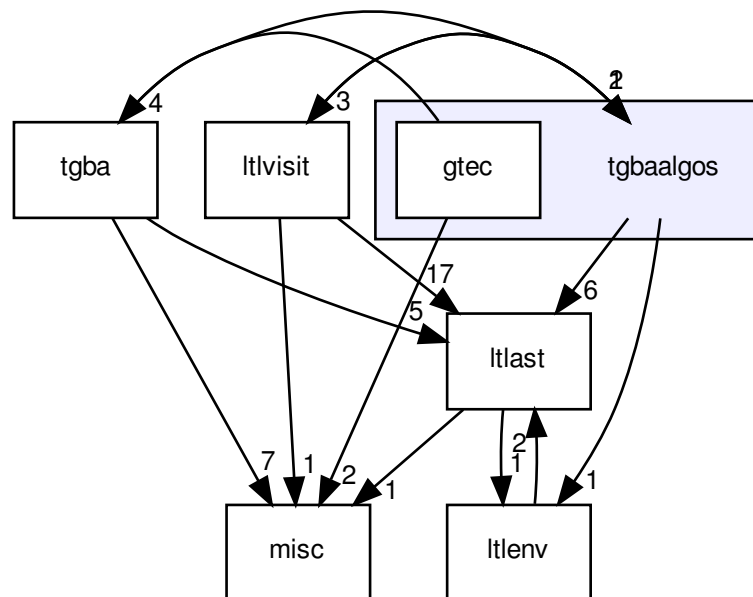
Files

- 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 [tgbabddfacy.hh](#)
- file [tgbaexplicit.hh](#)
- file [tgbakvcomplement.hh](#)
- file [tgbaproduct.hh](#)
- file [tgbaeduc.hh](#)
- file [tgbafracomplement.hh](#)
- file [tgbascc.hh](#)
- file [tgbaagba.hh](#)
- file [tgbatba.hh](#)
- file [tgbaunion.hh](#)
- file [wdbacomp.hh](#)

5.19 tgbaalgos/ Directory Reference

Directory dependency graph for tgbaalgos/:



Directories

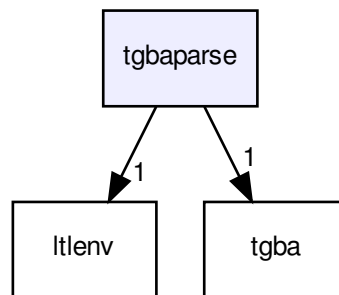
- directory [gtec](#)

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 [minimize.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 [safety.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](#)

5.20 tgbaparse/ Directory Reference

Directory dependency graph for tgbaparse/:



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 location & loc1, const location & loc2) [inline]`

Compare two location objects.

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

Compare two position objects.

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

Join two location objects to create a location.

References eltly::location::end.

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

Add two position objects.

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

Add two location objects.

References eltly::location::columns().

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

Add and assign a location.

References eltly::location::columns().

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

Add and assign a position.

References eltly::position::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& eltlyy::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.

6.1.1.11 `std::ostream& eltlyy::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 eltlyy::position::column, eltlyy::position::filename, and eltlyy::position::line.

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

Compare two location objects.

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

Compare two position objects.

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

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 Itlyy::operator!=(const location & loc1, const location & loc2) [inline]`

Compare two location objects.

6.2.1.2 `bool Itlyy::operator!=(const position & pos1, const position & pos2) [inline]`

Compare two position objects.

6.2.1.3 `const location Itlyy::operator+ (const location & begin, const location & end) [inline]`

Join two location objects to create a location.

References `Itlyy::location::end`.

6.2.1.4 `const position Itlyy::operator+ (const position & begin, const int width) [inline]`

Add two position objects.

6.2.1.5 `const location Itlyy::operator+ (const location & begin, unsigned int width) [inline]`

Add two location objects.

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

6.2.1.6 `location& Itlyy::operator+= (location & res, unsigned int width) [inline]`

Add and assign a location.

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

6.2.1.7 `const position& Itlyy::operator+= (position & res, const int width) [inline]`

Add and assign a position.

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

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

Add two position objects.

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

Add and assign a position.

6.2.1.10 std::ostream& Itlyy::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.

6.2.1.11 std::ostream& Itlyy::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 Itlyy::position::column, Itlyy::position::filename, and Itlyy::position::line.

6.2.1.12 bool Itlyy::operator== (const location & *loc1*, const location & *loc2*) [inline]

Compare two location objects.

6.2.1.13 bool Itlyy::operator== (const position & *pos1*, const position & *pos2*) [inline]

Compare two position objects.

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

6.3 neverclaimyy 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.

6.3.1 Function Documentation

6.3.1.1 `bool neverclaimyy::operator!= (const location & loc1, const location & loc2)` `[inline]`

Compare two location objects.

6.3.1.2 `bool neverclaimyy::operator!= (const position & pos1, const position & pos2)` `[inline]`

Compare two position objects.

6.3.1.3 `const location neverclaimyy::operator+ (const location & begin, const location & end)` `[inline]`

Join two location objects to create a location.

References `neverclaimyy::location::end`.

6.3.1.4 `const position neverclaimyy::operator+ (const position & begin, const int width)` `[inline]`

Add two position objects.

6.3.1.5 `const location neverclaimyy::operator+ (const location & begin, unsigned int width)` `[inline]`

Add two location objects.

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

6.3.1.6 `location& neverclaimyy::operator+= (location & res, unsigned int width)` `[inline]`

Add and assign a location.

References neverclaimyy::location::columns().

6.3.1.7 `const position& neverclaimyy::operator+=(position & res, const int width)`
[inline]

Add and assign a position.

References neverclaimyy::position::columns().

6.3.1.8 `const position neverclaimyy::operator- (const position & begin, const int width)`
[inline]

Add two position objects.

6.3.1.9 `const position& neverclaimyy::operator-= (position & res, const int width)` **[inline]**

Add and assign a position.

6.3.1.10 `std::ostream& neverclaimyy::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.

6.3.1.11 `std::ostream& neverclaimyy::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 neverclaimyy::position::column, neverclaimyy::position::filename, and neverclaimyy::position::line.

6.3.1.12 `bool neverclaimyy::operator==(const location & loc1, const location & loc2)`
`[inline]`

Compare two location objects.

6.3.1.13 `bool neverclaimyy::operator==(const position & pos1, const position & pos2)`
`[inline]`

Compare two position objects.

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

6.4 sautty 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.

6.4.1 Function Documentation

6.4.1.1 `const location sauty::operator+ (const location & begin, const location & end) [inline]`

Join two location objects to create a location.

References `sauty::location::end`.

6.4.1.2 `const location sauty::operator+ (const location & begin, unsigned int width) [inline]`

Add two location objects.

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

6.4.1.3 `const position sauty::operator+ (const position & begin, const int width) [inline]`

Add two position objects.

6.4.1.4 `const position& sauty::operator+= (position & res, const int width) [inline]`

Add and assign a position.

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

6.4.1.5 `location& sauty::operator+= (location & res, unsigned int width) [inline]`

Add and assign a location.

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

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

Add two position objects.

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

Add and assign a position.

6.4.1.8 std::ostream& sautyy::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.

6.4.1.9 std::ostream& sautyy::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 sautyy::position::column, sautyy::position::filename, and sautyy::position::line.

6.5 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](#)
*Iterator code for a Fair Kripke structure.
This iterator can be used to simplify the writing of an iterator on a Fair Kripke structure (or lookalike).*
- class [fair_kripke](#)
*Interface for a Fair Kripke structure.
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition, and a set of acceptance conditions.*
- class [kripke_succ_iterator](#)
*Iterator code for Kripke structure
This iterator can be used to simplify the writing of an iterator on a Kripke structure (or lookalike).*
- class [kripke](#)
*Interface for a Kripke structure
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition.*
- 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 [identity_hash](#)
A hash function that returns identity.
- 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 [tgba_explicit_number](#)
- 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 [tgba_product_init](#)
A lazy product with different initial states.
- 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.
- struct [power_map](#)
- 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< [evtgba::location](#), std::string > [evtgba_parse_error](#)
A parse diagnostic with its location.

- typedef std::list< [evtgba_parse_error](#) > [evtgba_parse_error_list](#)
A list of parser diagnostics, as filled by parse.
- typedef std::pair< [neverclaimyy::location](#), std::string > [neverclaim_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< [neverclaim_parse_error](#) > [neverclaim_parse_error_list](#)
A list of parser diagnostics, as filled by parse.
- 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< [tgba::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.
- `tgba_explicit_string * neverclaim_parse (const std::string &filename, neverclaim_parse_error_list &error_list, bdd_dict *dict, ltl::environment &env=ltl::default_environment::instance(), bool debug=false)`
Build a `spot::tgba_explicit` from a Spin never claim file.
- `bool format_neverclaim_parse_errors (std::ostream &os, const std::string &filename, neverclaim_parse_error_list &error_list)`
Format diagnostics produced by `spot::neverclaim_parse`.
- `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)`

- void [shared_state_deleter](#) (state *s)
- [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.
- [tgba](#) * [wdba_complement](#) (const [tgba](#) *aut)

Complement a weak deterministic Büchi automaton.
- 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 ELTL 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](#) * [couvereur99](#) (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](#) * [explicit_gv04_check](#) (const [tgba](#) *a, [option_map](#) o=[option_map](#)())

Emptiness check based on Geldenhuys and Valmari's TACAS'04 paper.
- std::ostream & [lbt_t_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.
- `tgba_explicit_number * minimize_monitor` (const `tgba` *a)
Construct a minimal deterministic monitor.
- `tgba_explicit_number * minimize_wdba` (const `tgba` *a)
Minimize a Büchi automaton in the WDBA class.
- const `tgba` * `minimize_obligation` (const `tgba` *aut_f, const `ltl::formula` *f=0, const `tgba` *aut_neg_f=0)
Minimize an automaton if it represents an obligation property.
- `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_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.
- `bool is_guarantee_automaton` (const `tgba` *aut, const `scc_map` *sm=0)
Whether an automaton represents a guarantee property.
- `bool is_safety_mwdba` (const `tgba` *aut)
Whether a minimized WDBA represents a safety property.
- `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)

- `tgba_explicit_number * tgba_powerset` (const `tgba` *aut, `power_map` &pm)

Build a deterministic automaton, ignoring acceptance conditions.

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

- `tgba_explicit_number * tgba_powerset` (const `tgba` *aut)

6.5.1 Typedef Documentation

6.5.1.1 typedef std::vector<bdd_ordered> spot::acc_list_t

6.5.1.2 typedef std::pair<evtgba::location, std::string> spot::evtgba_parse_error

A parse diagnostic with its location.

6.5.1.3 typedef std::list<evtgba_parse_error> spot::evtgba_parse_error_list

A list of parser diagnostics, as filled by parse.

6.5.1.4 typedef std::set<rsymbol> spot::rsymbol_set

6.5.1.5 typedef boost::shared_ptr<const saba_state> spot::shared_saba_state

6.5.1.6 `typedef boost::shared_ptr<const state> spot::shared_state`

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

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

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

6.5.2 Function Documentation

6.5.2.1 `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.

6.5.2.2 `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.

6.5.2.3 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.

6.5.2.4 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.

6.5.2.5 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.

6.5.2.6 `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.

6.5.2.7 `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.

6.5.2.8 `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.

6.5.2.9 `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.

6.5.2.10 `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.

6.5.2.11 `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.

6.5.2.12 `const ltl::formula* spot::bdd_to_formula (bdd f, const bdd_dict * d)`

6.5.2.13 `scc_stats spot::build_scc_stats (const tgba * a)`

6.5.2.14 `scc_stats spot::build_scc_stats (const scc_map & m)`

6.5.2.15 `bdd spot::compute_all_acceptance_conditions (bdd neg_acceptance_conditions)`

Compute all acceptance conditions from all neg acceptance conditions.

6.5.2.16 `bdd spot::compute_neg_acceptance_conditions (bdd all_acceptance_conditions)`

Compute neg acceptance conditions from all acceptance conditions.

6.5.2.17 `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

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

Print reachable states in dot format.

6.5.2.19 `std::ostream& spot::dump_scc_dot (const tgba * a, std::ostream & out, bool verbose = false)`**6.5.2.20** `std::ostream& spot::dump_scc_dot (const scc_map & m, std::ostream & out, bool verbose = false)`**6.5.2.21** `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 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.

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

Save reachable states in text format.

6.5.2.23 `std::vector<std::vector<scs_set* > >* spot::find_paths (tgba * a, const scc_map & m)`**6.5.2.24** `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.

6.5.2.25 `bdd spot::formula_to_bdd (const ltl::formula * f, bdd_dict * d, void * for_me)`**6.5.2.26** `bool spot::is_guarantee_automaton (const tgba * aut, const scc_map * sm = 0)`

Whether an automaton represents a guarantee property.

A weak deterministic TGBA represents a guarantee property if any accepting path ends on an accepting state with only one transition that is a self-loop labelled by `true`.

Note that in the general case, this is only a sufficient condition : some guarantee automata might not be recognized with this check e.g. because of some non-determinism in the automaton. In that case, you should interpret a `false` return value as "I don't know".

If you apply this function on a weak deterministic TGBA (e.g. after a successful minimization with [minimize_obligation\(\)](#)), then the result leaves no doubt: `false` really means that the automaton is not a guarantee property.

Parameters

aut the automaton to check

sm an [scc_map](#) of the automaton if available (it will be built otherwise. If you supply an [scc_map](#) you should call `build_map()` before passing it to this function.

6.5.2.27 bool spot::is_safety_mwdba (const tgba * *aut*)

Whether a minimized WDBA represents a safety property.

A minimized WDBA (as returned by a successful run of [minimize_obligation\(\)](#)) represent safety property if it contains only accepting transitions.

Parameters

aut the automaton to check

6.5.2.28 unsigned spot::max_spanning_paths (std::vector< sccs_set * > * *paths*, scc_map & *m*)**6.5.2.29 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.

6.5.2.30 std::ostream& spot::operator<< (std::ostream & *os*, const nips_exception & *e*)**6.5.2.31 std::ostream& spot::operator<< (std::ostream & *os*, const gspn_exception & *e*)****6.5.2.32 std::ostream& spot::saba_dotty_reachable (std::ostream & *os*, const saba * *g*)**

Print reachable states in dot format.

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

Prune unaccepting SCCs and remove superfluous acceptance conditions.

This function will explore the SCCs of the automaton and remove dead SCCs (i.e. SCC that are not accepting, and those 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 transitions going to an accepting SCC are accepting.

6.5.2.34 void spot::shared_state_deleter (state * s) [inline]

References spot::state::destroy().

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

6.5.2.36 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.)

6.6 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.

6.7 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 U f` and `false R g` by `F f` and `G g`.

- class `unabbreviate_ltl_visitor`

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

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 <l_string`, `parse_error_list &error_list`, `environment &env=default_environment::instance()`, `bool debug=false`)
Build a formula from an LTL string.
- `bool format_parse_errors` (`std::ostream &os`, `const std::string <l_string`, `parse_error_list &error_list`)
Format diagnostics produced by `spot::ltl::parse`.
- `atomic_prop_set * atomic_prop_collect` (`const formula *f`, `atomic_prop_set *s=0`)
Return the set of atomic propositions occurring in a formula.
- `formula * basic_reduce` (`const formula *f`)
Basic rewritings.
- `bool is_GF` (`const formula *f`)
Whether a formula starts with GF.

- `bool is_FG (const formula *f)`
Whether a formula starts with FG.
- `formula * clone (const formula *f)`
Clone a formula.
- `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 U f` and `false R g` by `F f` and `G 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 ECTL 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, bool full_parent=false)
Output a formula as a (parsable by Spin) string.
- std::string `to_spin_string` (const formula *f, bool full_parent=false)
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.

6.7.1 Function Documentation

6.7.1.1 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.
///

```

6.7.1.2 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`.

6.8 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.

6.8.1 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`.

6.8.2 Typedef Documentation

6.8.2.1 typedef boost::shared_ptr<node> spot::ltl::formula_tree::node_ptr

We use `boost::shared_ptr` to easily handle deletion.

6.8.3 Enumeration Type Documentation

6.8.3.1 anonymous enum

Integer values for True and False used in [node_atomic](#).

Enumerator:

True

False

6.8.4 Function Documentation

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

Get the arity.

6.8.4.2 `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.

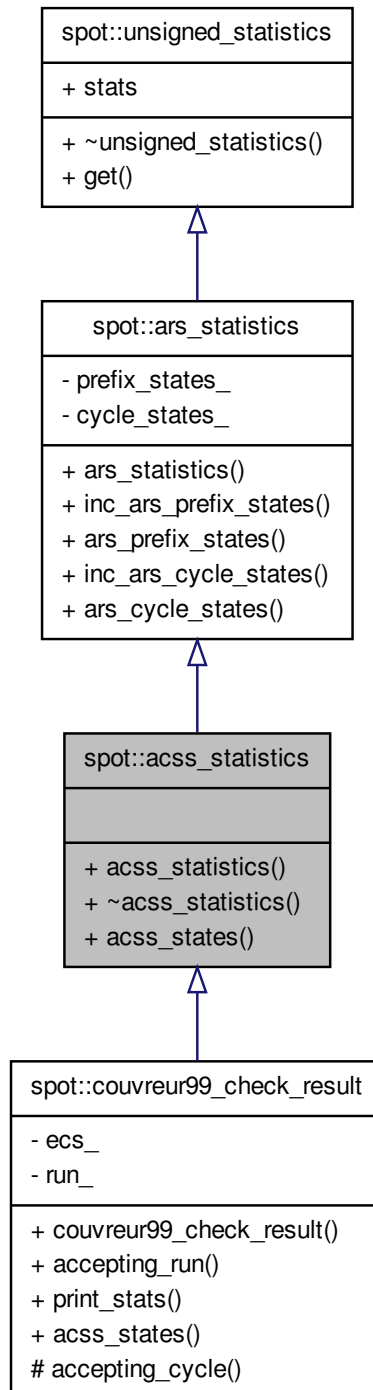
7 Class Documentation

7.1 `spot::acss_statistics` Class Reference

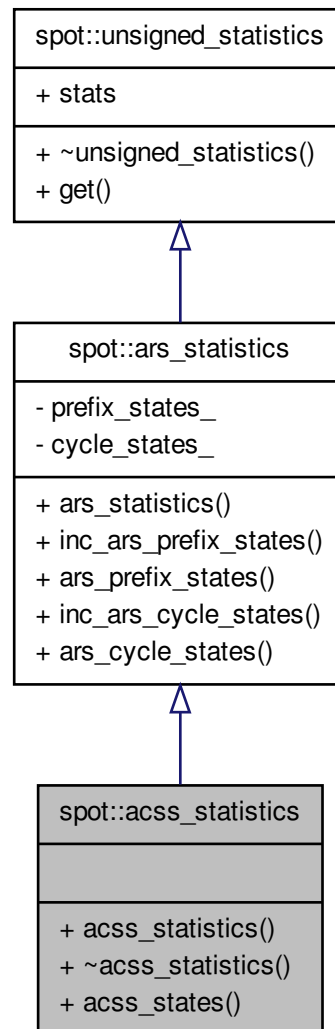
Accepting Cycle Search Space statistics.

```
#include <tgbalgs/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 `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](#).

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

References spot::ars_statistics::cycle_states_.

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

References spot::ars_statistics::prefix_states_.

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()`.

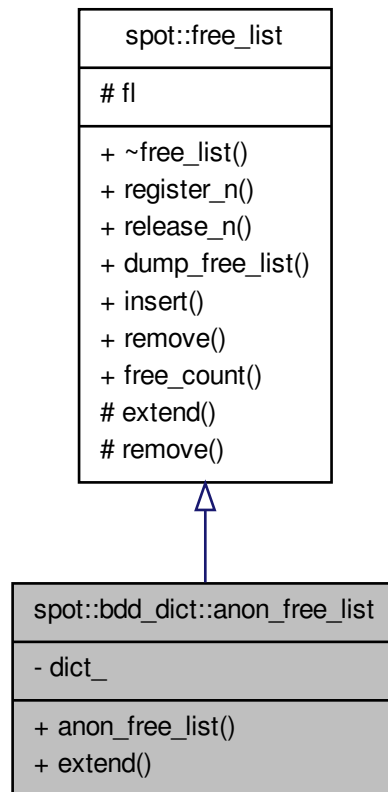
The documentation for this class was generated from the following file:

- [tgbaalgos/emptiness_stats.hh](#)

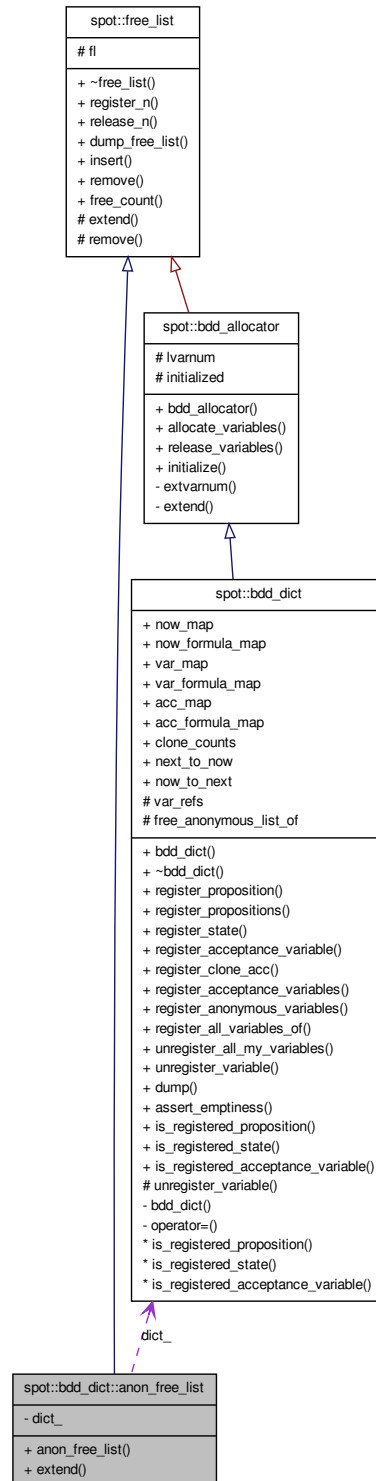
7.2 spot::bdd_dict::anon_free_list Class Reference

```
#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-lenght 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 (int *base*, int *n* = 0) [inherited]

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

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

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

7.2.4 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:

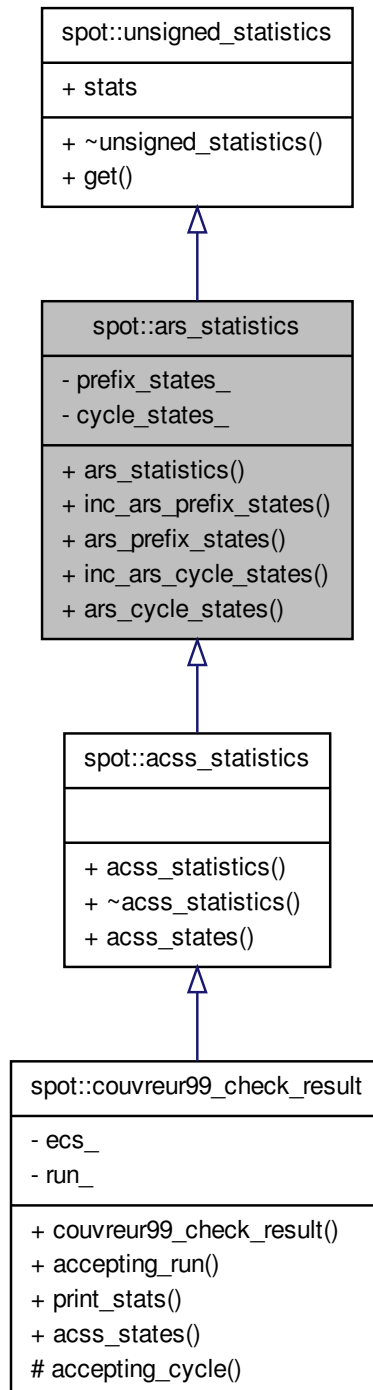
- [tgba/bdddict.hh](#)

7.3 spot::ars_statistics Class Reference

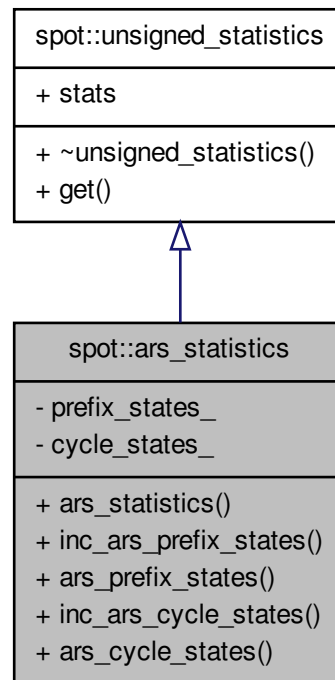
Accepting Run Search statistics.

```
#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 `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_`.

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

References `prefix_states_`.

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:

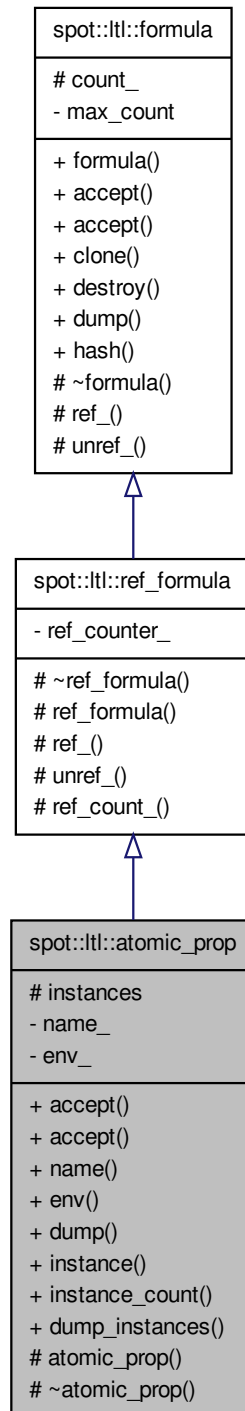
- tgbaalgorithms/[emptiness_stats.hh](#)

7.4 spot::ltl::atomic_prop Class Reference

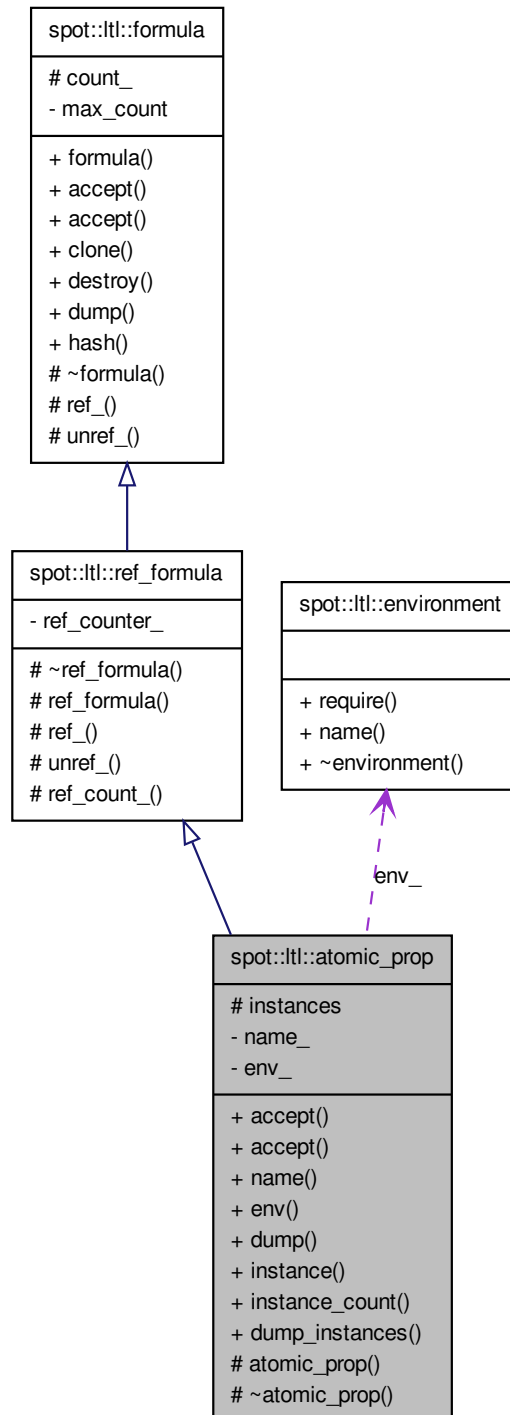
Atomic propositions.

```
#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 (visitor & v) [virtual]

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

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

7.4.4.2 virtual void spot::ltl::atomic_prop::accept (const_visitor & v) const [virtual]

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

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

7.4.4.3 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_.

Referenced by spot::ltl::formula_ptr_hash::operator>(), and spot::ltl::formula_ptr_less_than::operator>().

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:

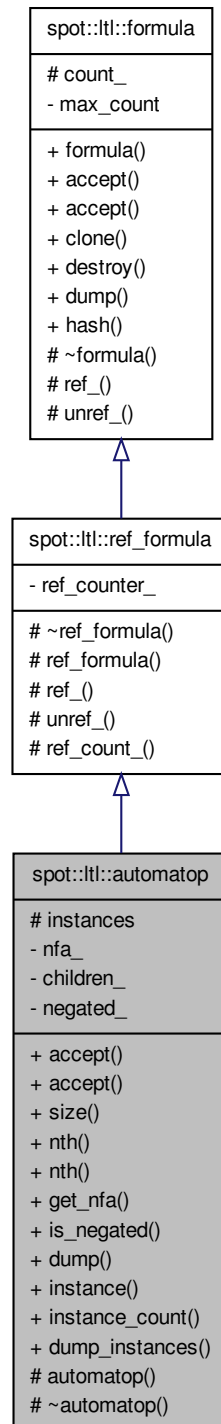
- ltlast/[atomic_prop.hh](#)

7.5 spot::ltl::automatop Class Reference

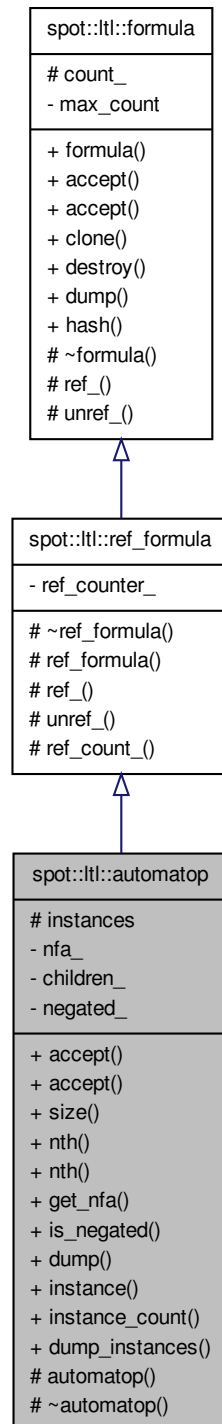
Automaton operators.

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

Inheritance diagram for spot::ltl::automatop:



Collaboration diagram for spot::ltl::automatop:



Classes

- struct [tripletcmp](#)
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 (visitor & v)` [virtual]

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

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

7.5.4.2 `virtual void spot::ltl::automatop::accept (const_visitor & v) const` [virtual]

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

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

7.5.4.3 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_`.

Referenced by `spot::ltl::formula_ptr_hash::operator()()`, and `spot::ltl::formula_ptr_less_than::operator()()`.

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

Get the n th argument.

Starting with $n = 0$.

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

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:

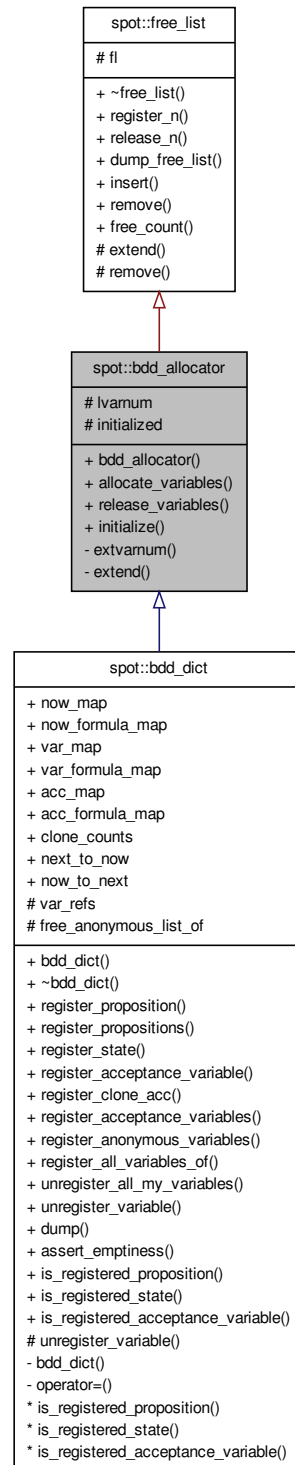
- [misc/random.hh](#)

7.7 spot::bdd_allocator Class Reference

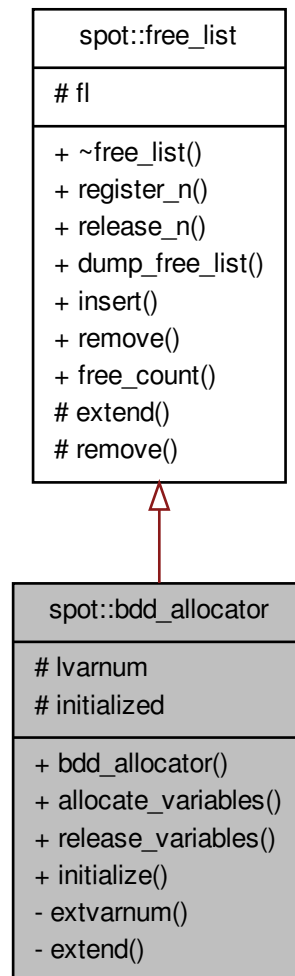
Manage ranges of variables.

```
#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 (int *base*, int *n* = 0) [inherited]

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

7.7.4.12 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*.

7.7.5 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.

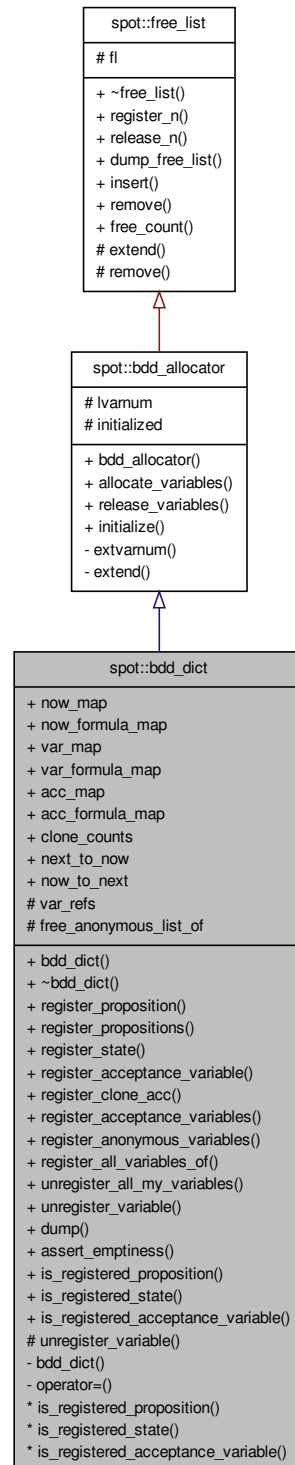
The documentation for this class was generated from the following file:

- [misc/bddalloc.hh](#)

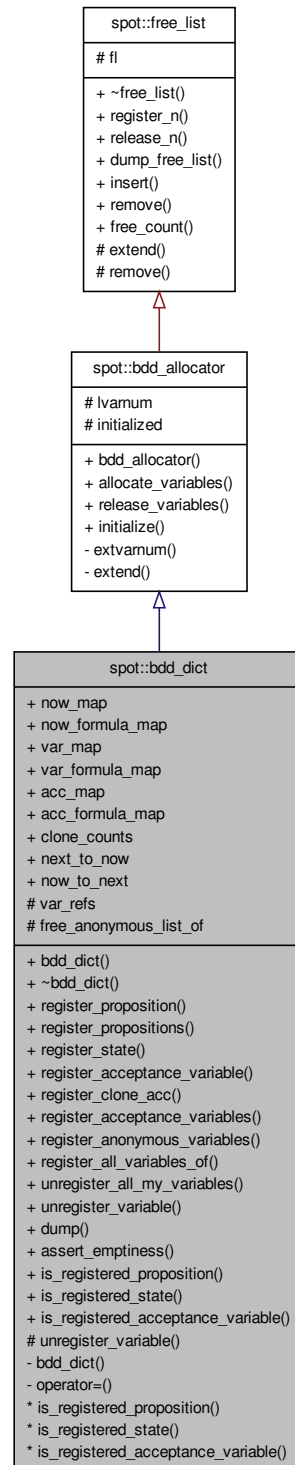
7.8 spot::bdd_dict Class Reference

```
#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_emptiness](#) () const
Make sure the dictionary is empty.
- int [allocate_variables](#) (int n)
Allocate n BDD variables.
- void [release_variables](#) (int base, int n)
Release n BDD variables starting at base.
- bool [is_registered_proposition](#) (const ltl::formula *f, const void *by_me)
- bool [is_registered_state](#) (const ltl::formula *f, const void *by_me)
- bool [is_registered_acceptance_variable](#) (const ltl::formula *f, const void *by_me)

Static Public Member Functions

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

Public Attributes

- [fv_map now_map](#)
Maps formulae to "Now" BDD variables.
- [vf_map now_formula_map](#)
Maps "Now" BDD variables to formulae.
- [fv_map var_map](#)
Maps atomic propositions to BDD variables.
- [vf_map var_formula_map](#)
Maps BDD variables to atomic propositions.
- [fv_map acc_map](#)
Maps acceptance conditions to BDD variables.
- [vf_map acc_formula_map](#)
Maps BDD variables to acceptance conditions.
- [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_emptiness () 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 (int var, const void * me)`

Release a variable used by *me*.

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

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 () const [inline]

7.10.2.3 unsigned& spot::bdd_ordered::order () [inline]

References order_.

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](#)

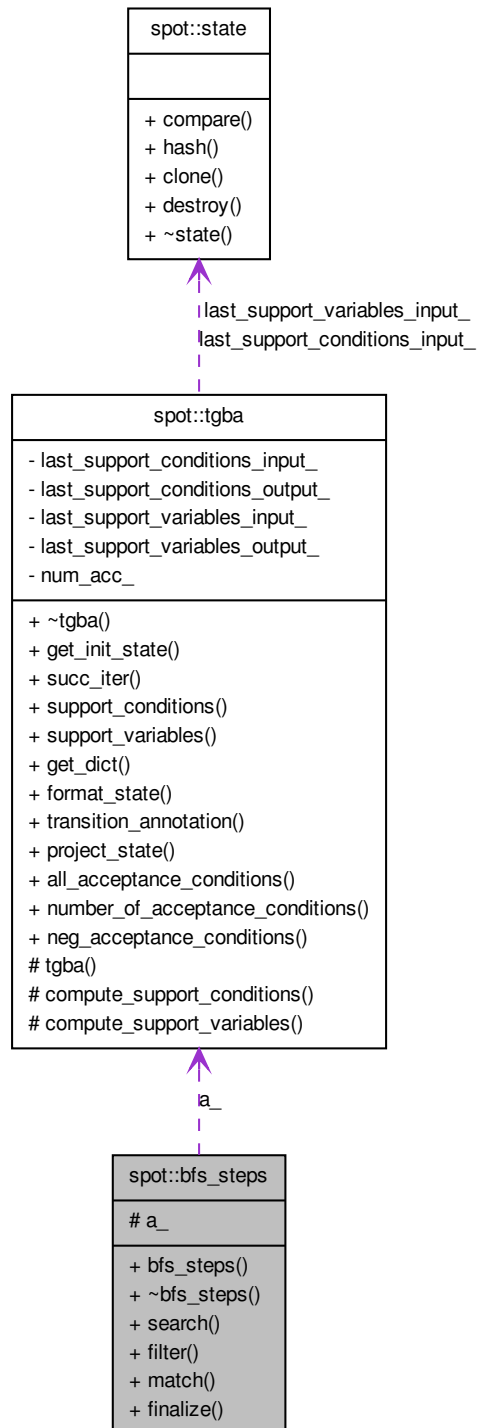
7.11 spot::bfs_steps Class Reference

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 destroy `s`. Otherwise you will have to define such a set, just to be able to destroy 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:

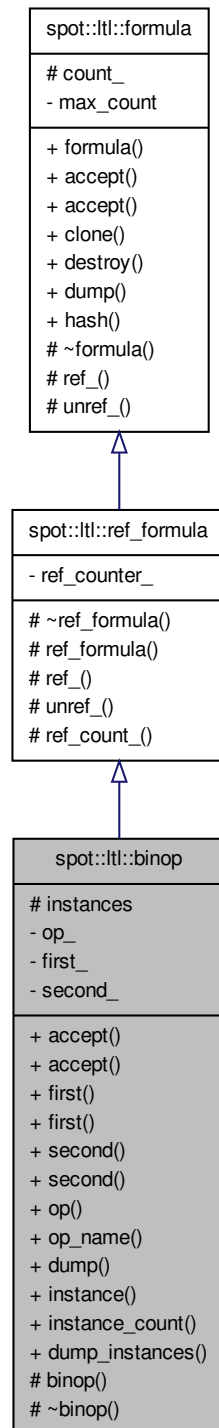
- [tgbaalgos/bfssteps.hh](#)

7.12 spot::ltl::binop Class Reference

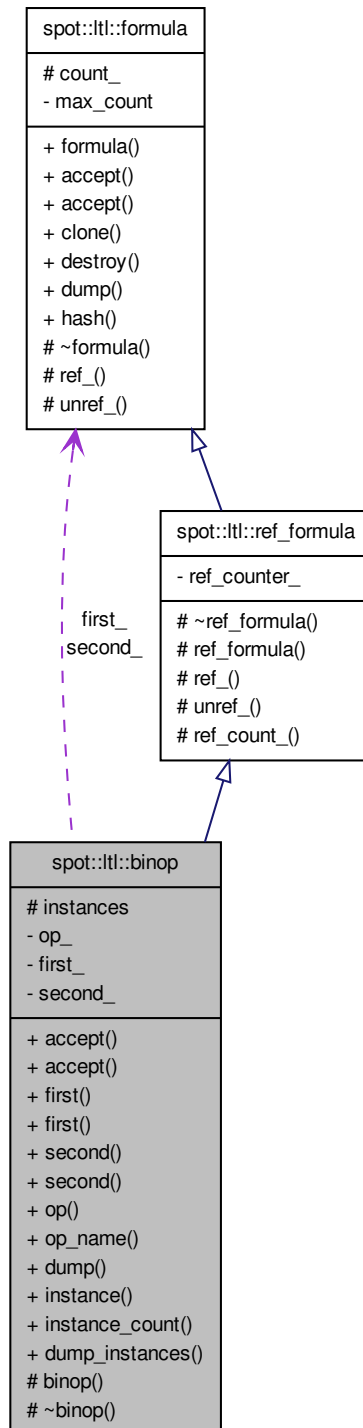
Binary operator.

```
#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 opertaors

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 (visitor & v) [virtual]

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

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

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

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

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

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

Get the first operand.

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

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_.

Referenced by spot::ltl::formula_ptr_hash::operator>(), and spot::ltl::formula_ptr_less_than::operator>().

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

Get the second operand.

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

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`

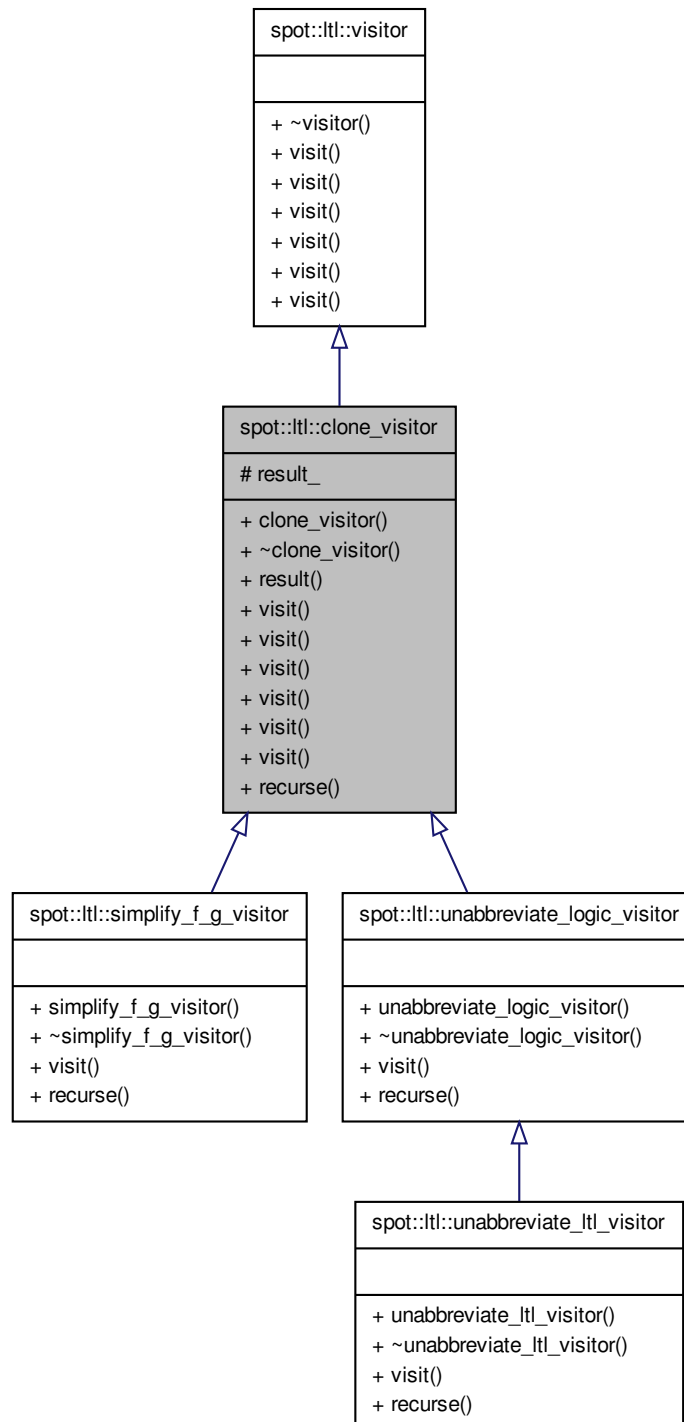
7.14 spot::ltl::clone_visitor Class Reference

Clone a formula.

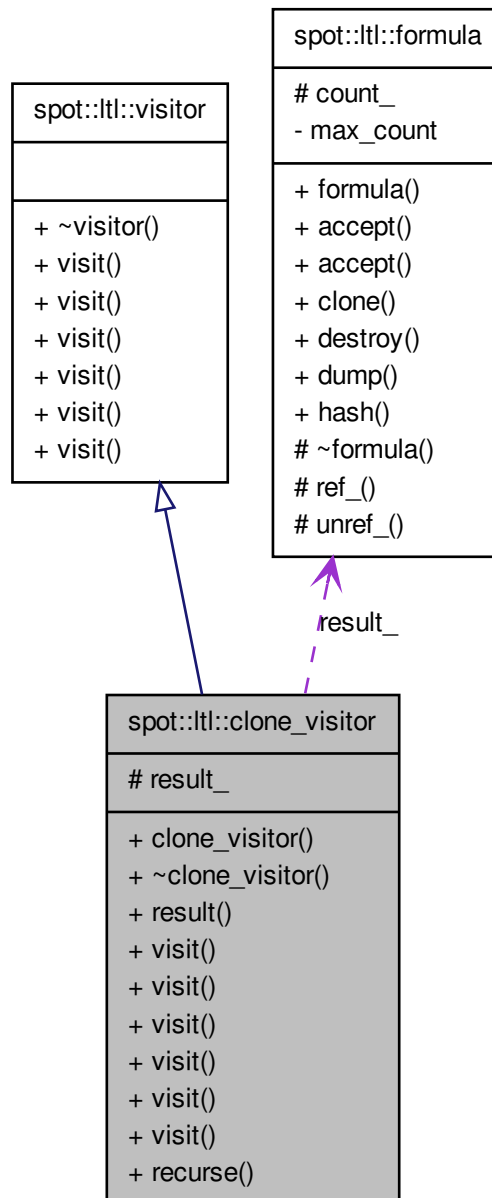
This visitor is public, because it's convenient to derive from it and override part of its methods. But if you just want the functionality, consider using `spot::ltl::formula::clone` instead, it is way faster.

```
#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 (atomic_prop * ap) [virtual]`

Implements [spot::ltl::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 (binop * bo) [virtual]`

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

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

7.14.4 Member Data Documentation

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

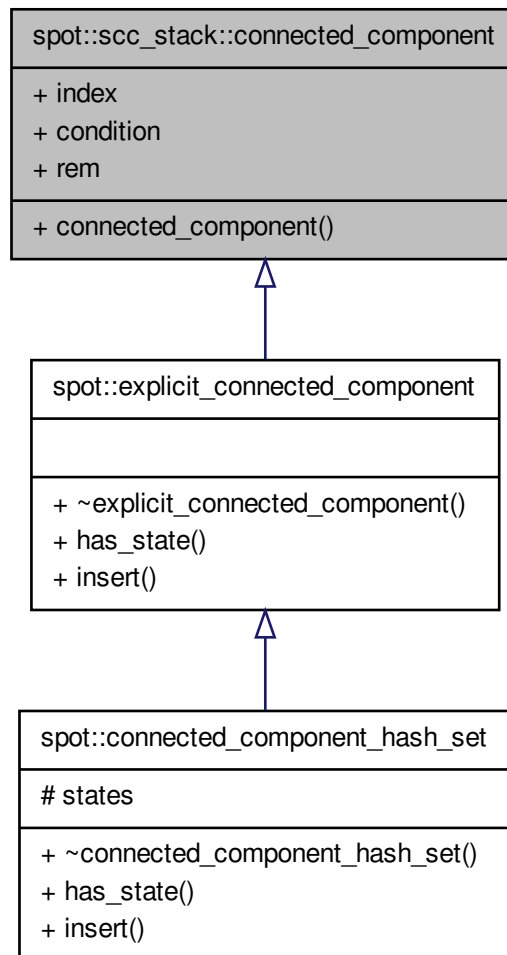
The documentation for this class was generated from the following file:

- [ltlvisit/clone.hh](#)

7.15 `spot::scc_stack::connected_component` Struct Reference

```
#include <tgbaalgorithms/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

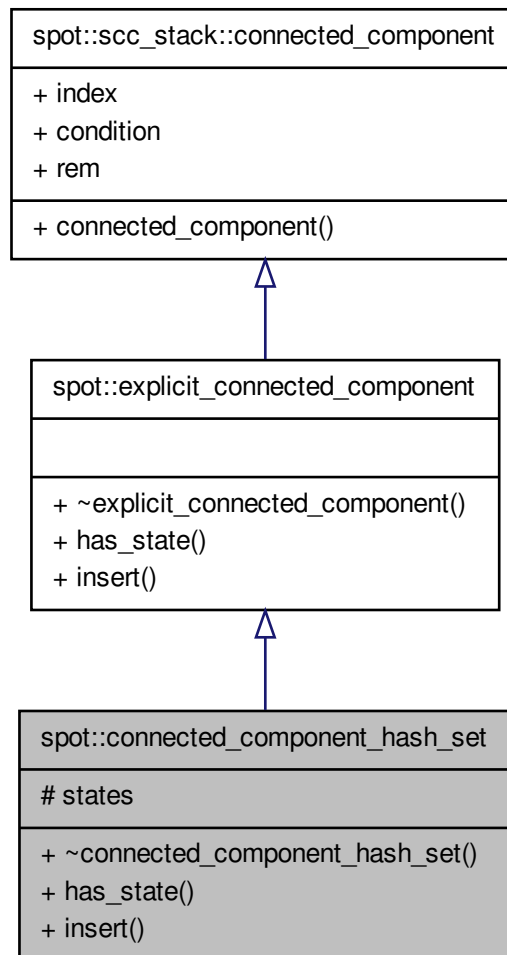
The documentation for this struct was generated from the following file:

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

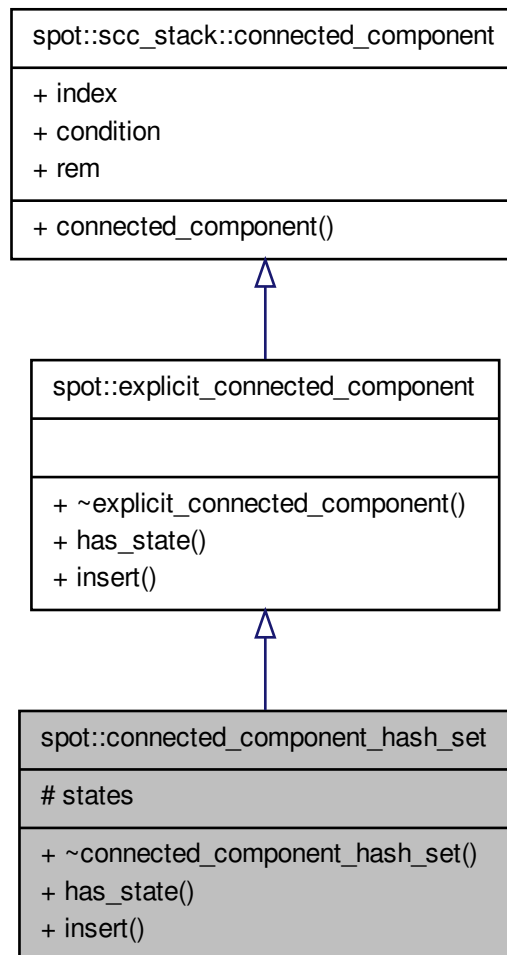
7.16 spot::connected_component_hash_set Class Reference

```
#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 destroy *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:

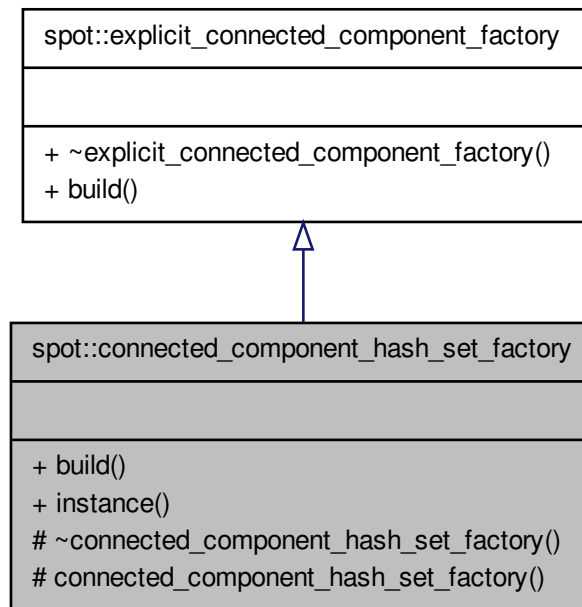
- [tgbaalgos/gtec/explscs.hh](#)

7.17 spot::connected_component_hash_set_factory Class Reference

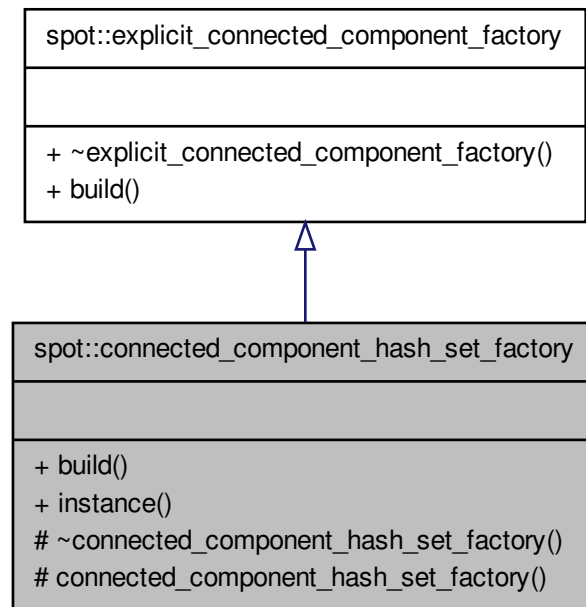
Factory for [connected_component_hash_set](#).

```
#include <tgbaalgos/gtec/explscs.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 [atomic_prop](#) * *node*) [[pure](#) [virtual](#)]

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

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

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

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

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

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

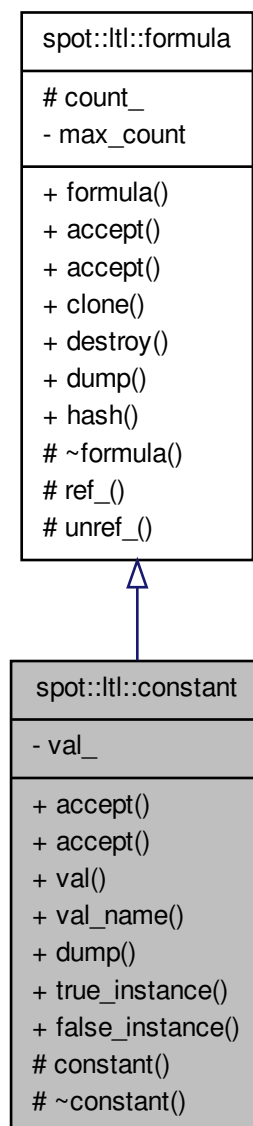
- [ltlast/visitor.hh](#)

7.19 spot::ltl::constant Class Reference

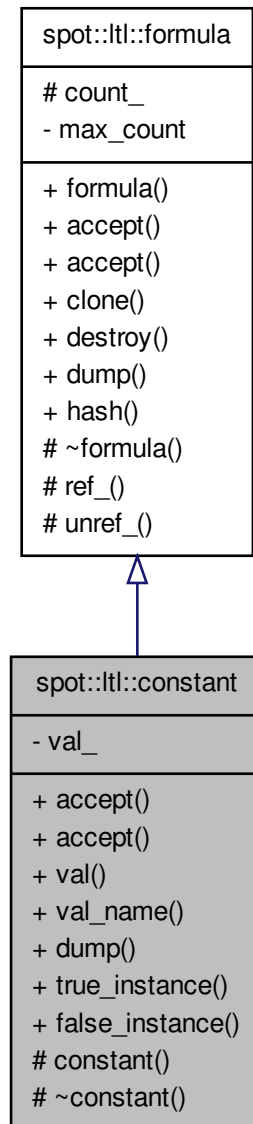
A constant (True or False).

```
#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 (visitor & *v*) [virtual]**

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

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

7.19.4.2 virtual void spot::ltl::constant::accept (const_visitor & *v*) const [virtual]

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

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

7.19.4.3 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_.

Referenced by spot::ltl::formula_ptr_hash::operator()(), and spot::ltl::formula_ptr_less_than::operator()().

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:

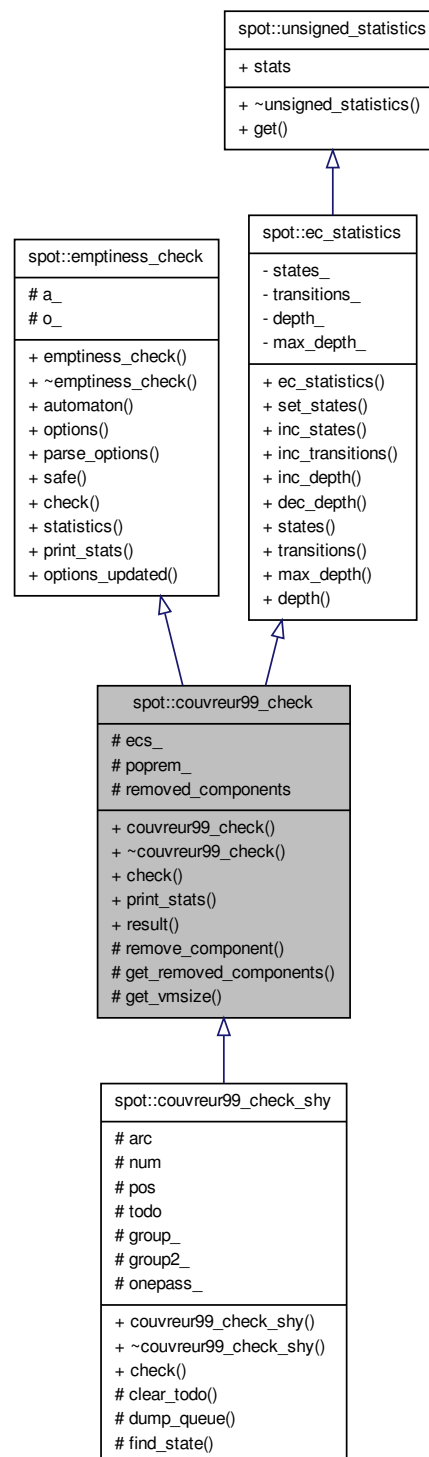
- [ltlast/constant.hh](#)

7.20 `spot::couvreur99_check` Class Reference

An implementation of the Couvreur99 emptiness-check algorithm.

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

Inheritance diagram for spot::couvreur99_check:



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
- 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 `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.
- 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::emptiness_check::check () [pure virtual, inherited]

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\(\)](#)

7.20.4.3 virtual emptiness_check_result* spot::couvreur99_check::check () [virtual]

Check whether the automaton's language is empty.

Reimplemented in [spot::couvreur99_check_shy](#).

7.20.4.4 void spot::ec_statistics::dec_depth (unsigned n = 1) [inline, inherited]

References spot::ec_statistics::depth_.

7.20.4.5 unsigned spot::ec_statistics::depth () const [inline, inherited]

References spot::ec_statistics::depth_.

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

References spot::unsigned_statistics::stats.

7.20.4.7 unsigned spot::couvreur99_check::get_removed_components () const [protected]

7.20.4.8 unsigned spot::couvreur99_check::get_vmsize () const [protected]

7.20.4.9 void spot::ec_statistics::inc_depth (unsigned *n* = 1) [inline, inherited]

References spot::ec_statistics::depth_, and spot::ec_statistics::max_depth_.

7.20.4.10 void spot::ec_statistics::inc_states () [inline, inherited]

References spot::ec_statistics::states_.

7.20.4.11 void spot::ec_statistics::inc_transitions () [inline, inherited]

References spot::ec_statistics::transitions_.

7.20.4.12 unsigned spot::ec_statistics::max_depth () const [inline, inherited]

References spot::ec_statistics::max_depth_.

7.20.4.13 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_.

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

Notify option updates.

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

Modify the algorithm options.

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

7.20.4.17 `virtual std::ostream& spot::emptiness_check::print_stats (std::ostream & os) const [virtual, inherited]`

Print statistics, if any.

7.20.4.18 `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.

7.20.4.19 `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.

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

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

7.20.4.21 `void spot::ec_statistics::set_states (unsigned n) [inline, inherited]`

References `spot::ec_statistics::states_`.

7.20.4.22 unsigned spot::ec_statistics::states () const [inline, inherited]

References spot::ec_statistics::states_.

7.20.4.23 virtual const unsigned_statistics* spot::emptiness_check::statistics () const [virtual, inherited]

Return statistics, if available.

7.20.4.24 unsigned spot::ec_statistics::transitions () const [inline, inherited]

References spot::ec_statistics::transitions_.

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:

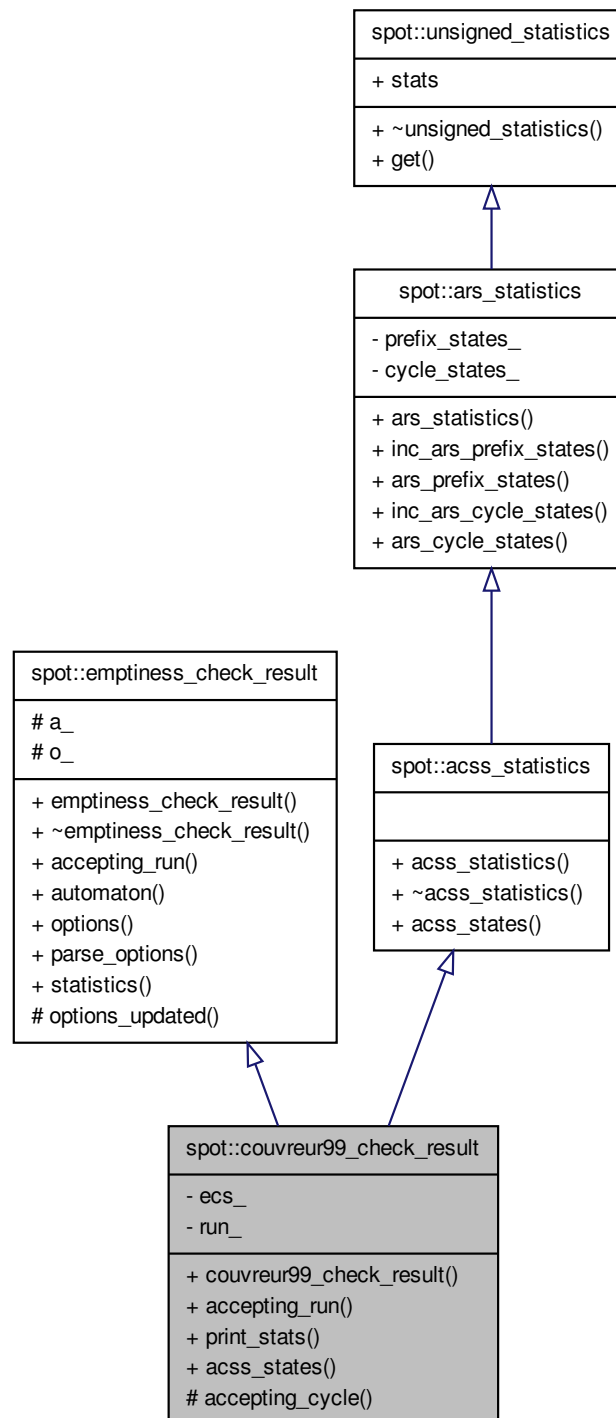
- tgbaalgos/gtec/[gtec.hh](#)

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:



- void [print_stats](#) (std::ostream &os) const
- virtual unsigned [acss_states](#) () const
Number of states in the search space for the accepting cycle.
- 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.
- 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]`

7.21.4.3 `virtual tgba_run* spot::emptiness_check_result::accepting_run () [virtual,
inherited]`

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).

7.21.4.4 `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](#).

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

References [spot::ars_statistics::cycle_states_](#).

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

References [spot::ars_statistics::prefix_states_](#).

7.21.4.7 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_](#).

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

References [spot::unsigned_statistics::stats](#).

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

References [spot::ars_statistics::cycle_states_](#).

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

References [spot::ars_statistics::prefix_states_](#).

7.21.4.11 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_](#).

7.21.4.12 `virtual void spot::emptiness_check_result::options_updated (const option_map & old)` `[protected, virtual, inherited]`

Notify option updates.

7.21.4.13 `const char* spot::emptiness_check_result::parse_options (char * options)` `[inherited]`

Modify the algorithm options.

7.21.4.14 `void spot::couvreur99_check_result::print_stats (std::ostream & os) const`

7.21.4.15 `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:

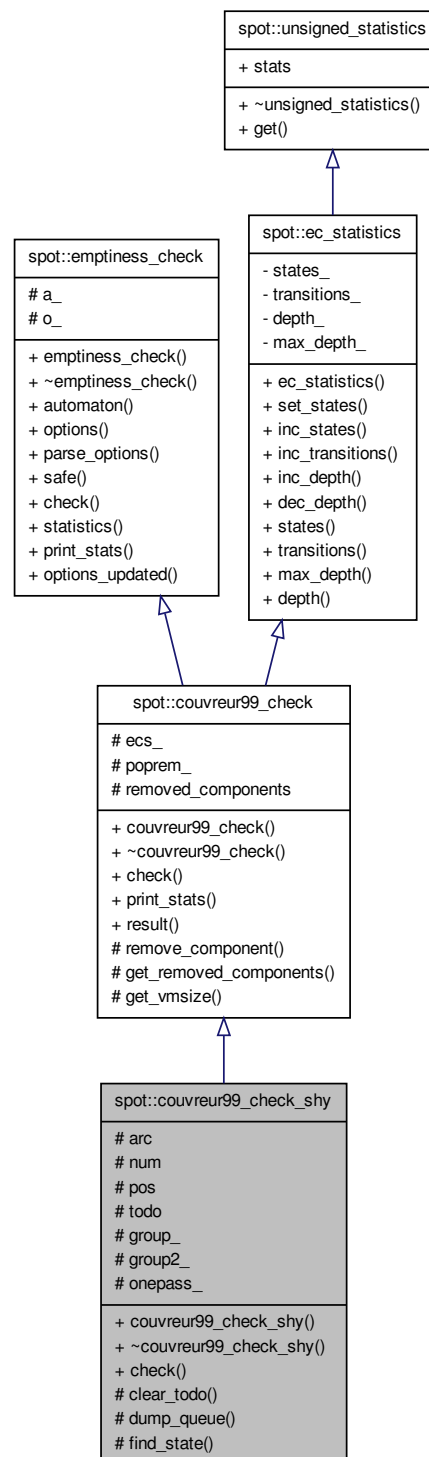
- [tgbaalgos/gtec/ce.hh](#)

7.22 spot::couvreur99_check_shy Class Reference

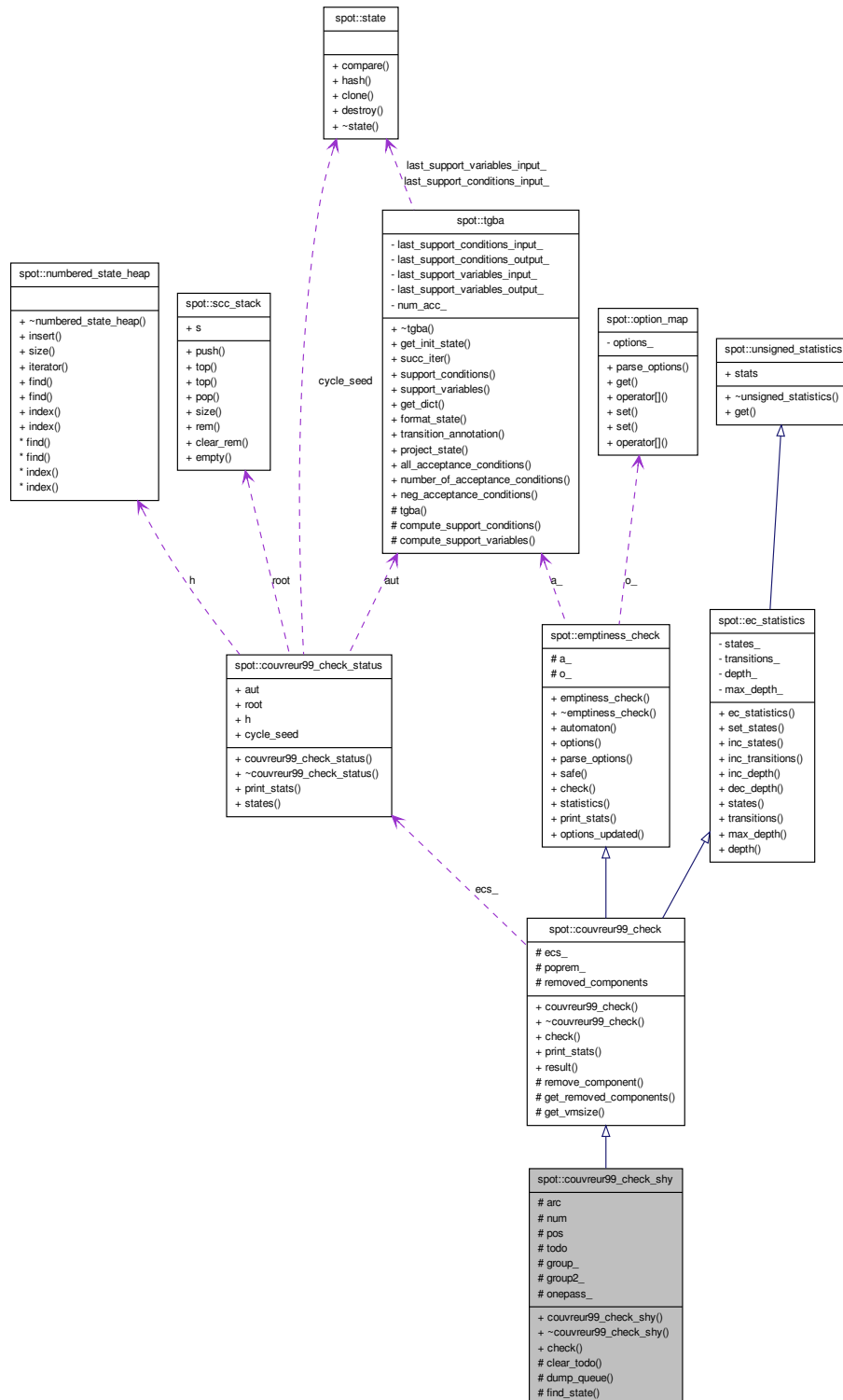
A version of [spot::couvreur99_check](#) that tries to visit known states first.

```
#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 [emptiness_check_result](#) * [check](#) ()=0
Check whether the automaton contain an accepting run.
- virtual std::ostream & [print_stats](#) (std::ostream &os) const
- 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::emptiness_check::check ()` [`pure virtual`, `inherited`]

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\(\)](#)

7.22.4.3 `virtual emptiness_check_result* spot::couvreur99_check_shy::check ()` [`virtual`]

Check whether the automaton's language is empty.

Reimplemented from `spot::couvreur99_check`.

7.22.4.4 `void spot::couvreur99_check_shy::clear_todo ()` [`protected`]

7.22.4.5 `void spot::ec_statistics::dec_depth (unsigned n = 1)` [`inline`, `inherited`]

References `spot::ec_statistics::depth_`.

7.22.4.6 `unsigned spot::ec_statistics::depth () const` [`inline`, `inherited`]

References `spot::ec_statistics::depth_`.

7.22.4.7 `void spot::couvreur99_check_shy::dump_queue (std::ostream & os = std::cerr)`
[protected]

Dump the queue for debugging.

7.22.4.8 `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.

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

References `spot::unsigned_statistics::stats`.

7.22.4.10 `unsigned spot::couvreur99_check::get_removed_components () const`
[protected, inherited]

7.22.4.11 `unsigned spot::couvreur99_check::get_vmsize () const` **[protected, inherited]**

7.22.4.12 `void spot::ec_statistics::inc_depth (unsigned n = 1)` **[inline, inherited]**

References `spot::ec_statistics::depth_`, and `spot::ec_statistics::max_depth_`.

7.22.4.13 `void spot::ec_statistics::inc_states ()` **[inline, inherited]**

References `spot::ec_statistics::states_`.

7.22.4.14 `void spot::ec_statistics::inc_transitions ()` **[inline, inherited]**

References `spot::ec_statistics::transitions_`.

7.22.4.15 unsigned spot::ec_statistics::max_depth () const [inline, inherited]

References spot::ec_statistics::max_depth_.

7.22.4.16 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_.

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

Notify option updates.

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

Modify the algorithm options.

7.22.4.19 virtual std::ostream& spot::emptiness_check::print_stats (std::ostream & os) const [virtual, inherited]

Print statistics, if any.

7.22.4.20 virtual std::ostream& spot::couvreur99_check::print_stats (std::ostream & os) const [virtual, inherited]**7.22.4.21 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.

7.22.4.22 `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.

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

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

7.22.4.24 `void spot::ec_statistics::set_states (unsigned n)` [inline, inherited]

References `spot::ec_statistics::states_`.

7.22.4.25 `unsigned spot::ec_statistics::states () const` [inline, inherited]

References `spot::ec_statistics::states_`.

7.22.4.26 `virtual const unsigned_statistics* spot::emptiness_check::statistics () const`
[virtual, inherited]

Return statistics, if available.

7.22.4.27 `unsigned spot::ec_statistics::transitions () const` [inline, inherited]

References `spot::ec_statistics::transitions_`.

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:

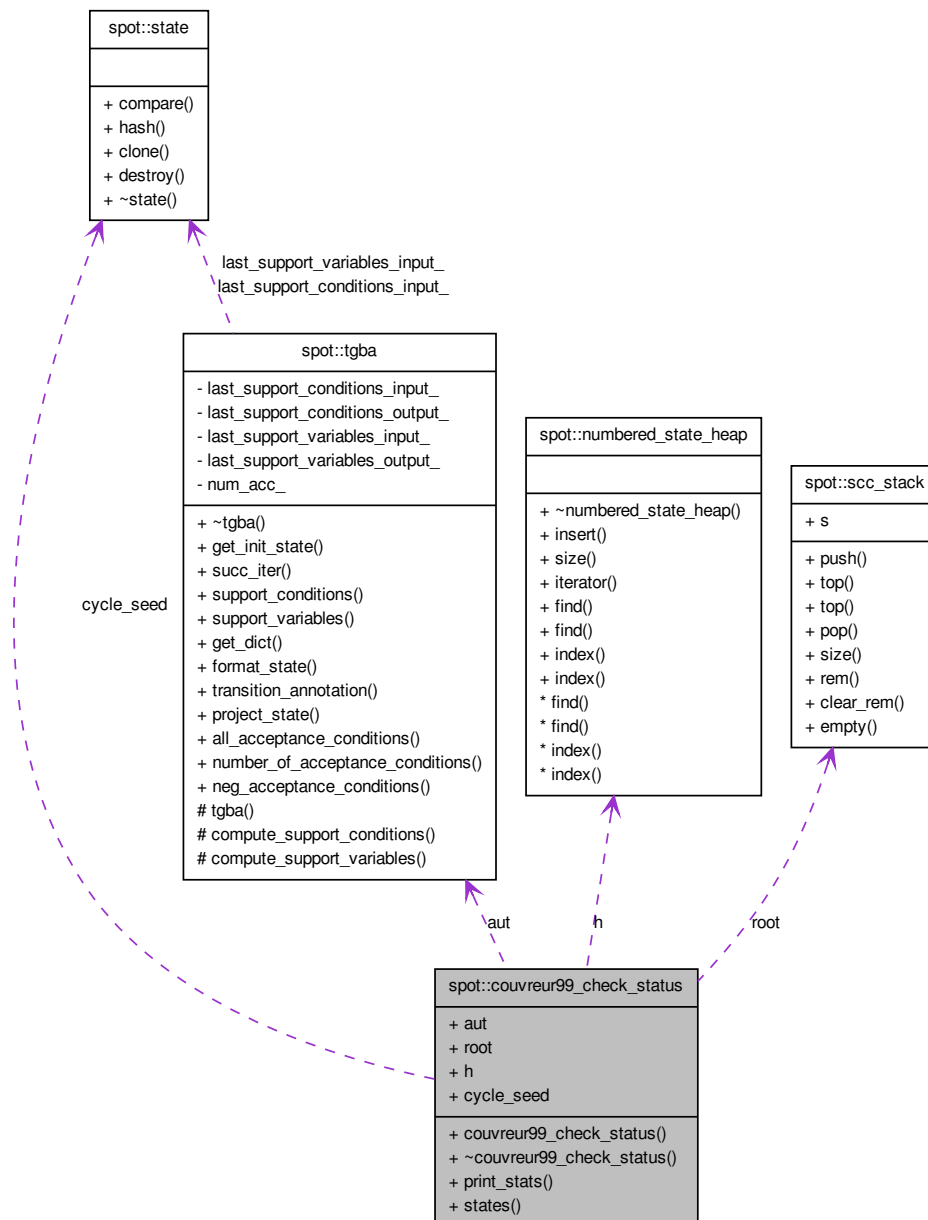
- tgbaalgos/gtec/[gtec.hh](#)

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`

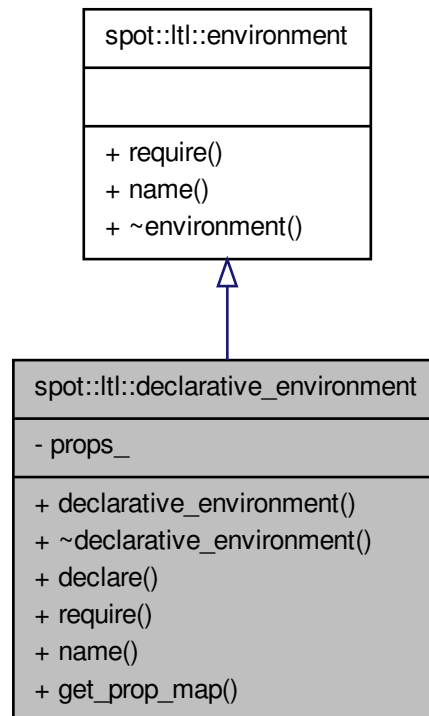
7.24 spot::ltl::declarative_environment Class Reference

A declarative environment.

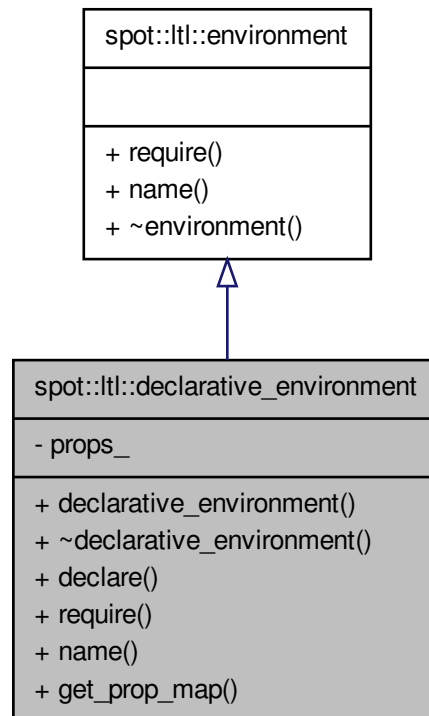
This environment recognizes all atomic propositions that have been previously declared. It will reject other.

```
#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`

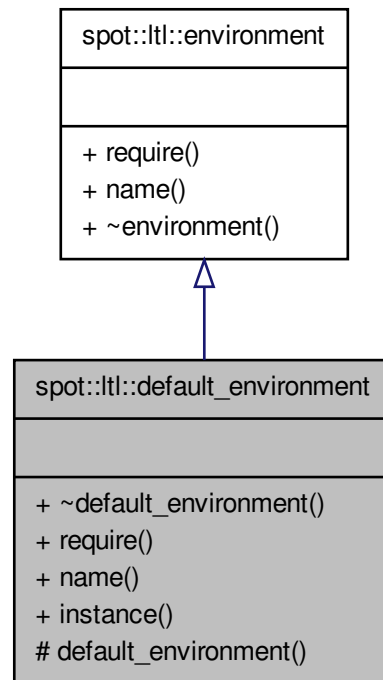
7.25 `spot::ltl::default_environment` Class Reference

A laxist environment.

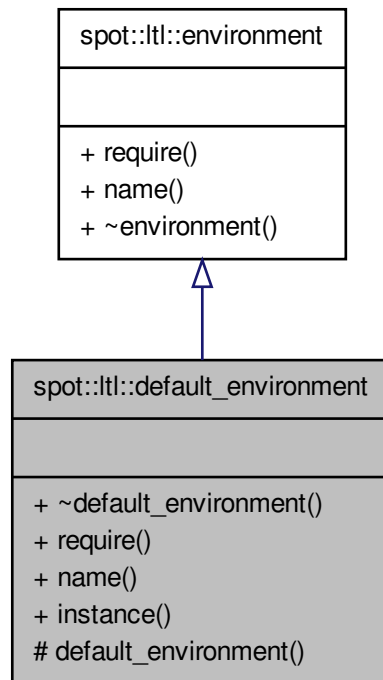
This environment recognizes all atomic propositions.

```
#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:

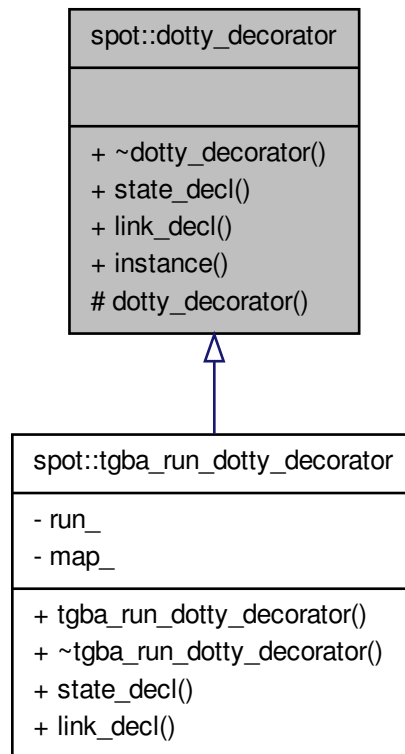
- [tgba/taatgba.hh](#)

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:

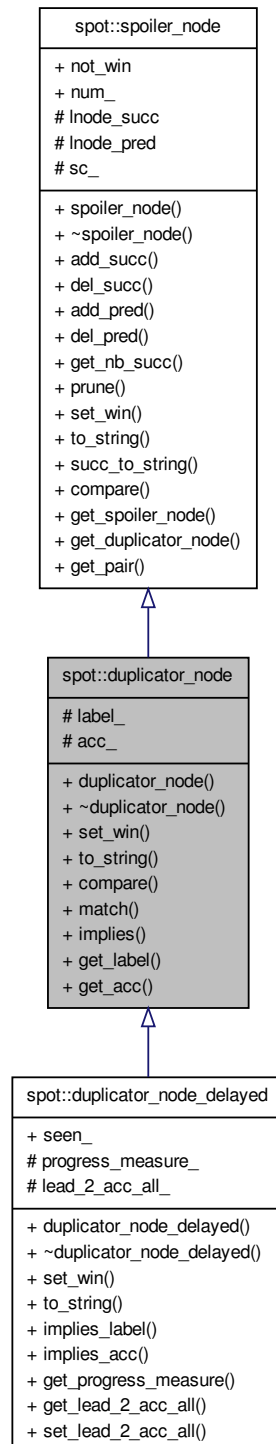
- [tgbaalgos/dottydec.hh](#)

7.30 spot::duplicator_node Class Reference

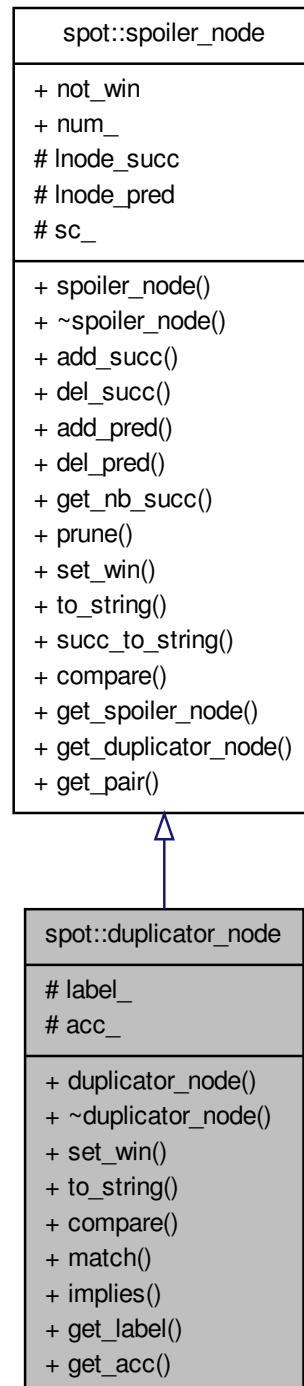
Duplicator node of parity game graph.

```
#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:

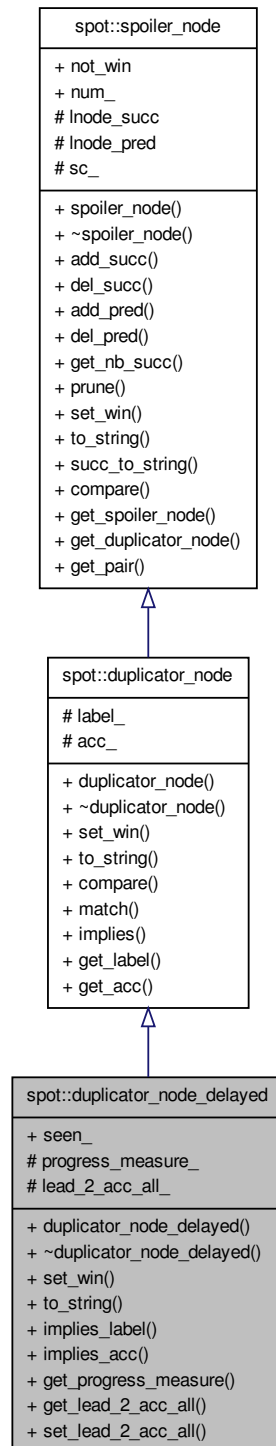
- `tgbaalgos/`[reductgba_sim.hh](#)

7.31 `spot::duplicator_node_delayed` Class Reference

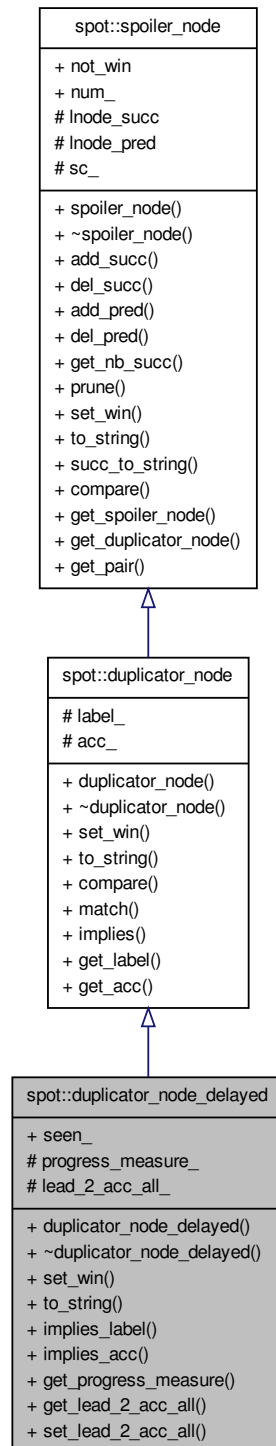
Duplicator node of parity game graph for delayed simulation.

```
#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:

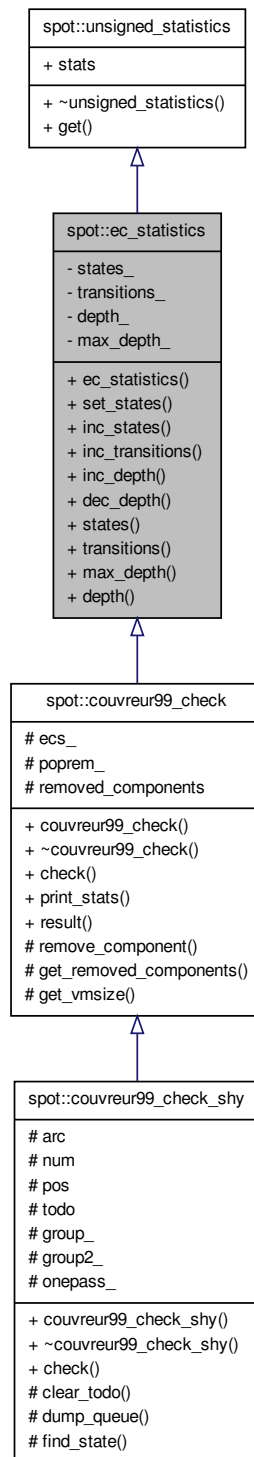
- tgbaalgos/[reductgba_sim.hh](#)

7.32 spot::ec_statistics Class Reference

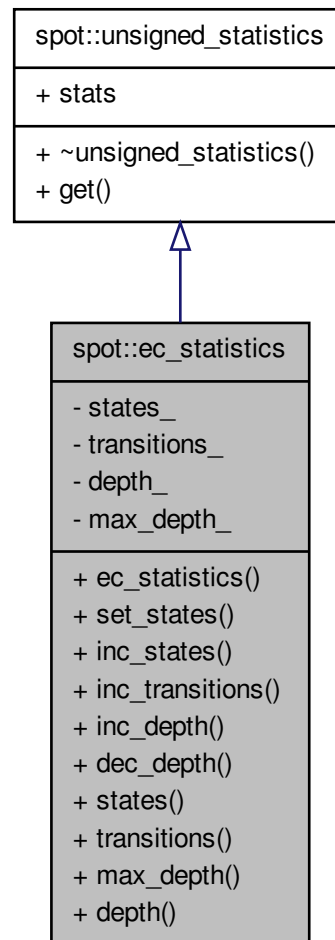
Emptiness-check statistics.

```
#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 discint 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 [spot::unsigned_statistics::stats](#).

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_`.

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_.

7.32.4.10 unsigned spot::ec_statistics::transitions () const [inline]

References transitions_.

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:

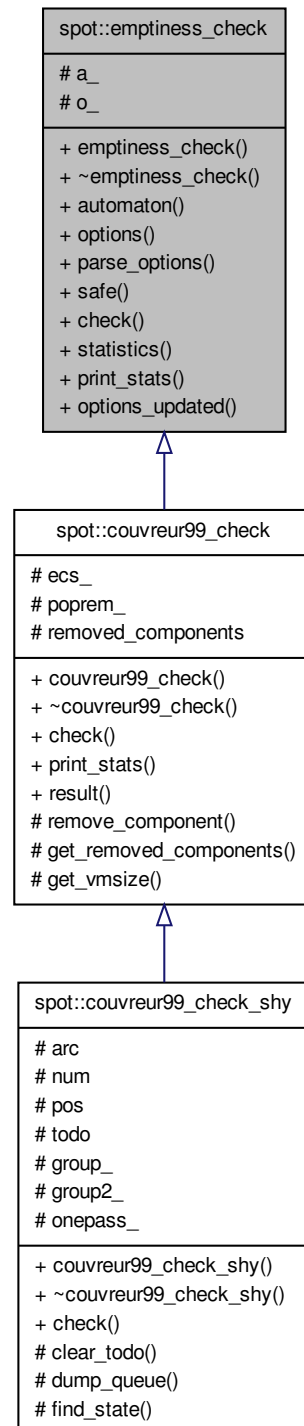
- [tgbaalgorithms/emptiness_stats.hh](#)

7.33 spot::emptiness_check Class Reference

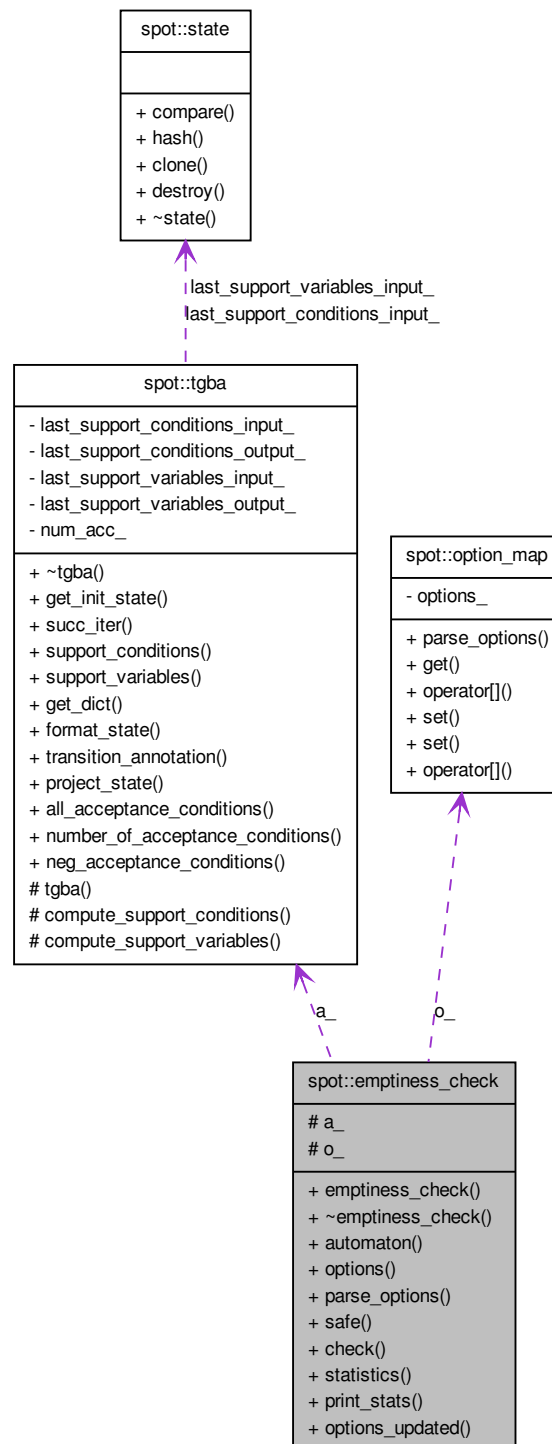
Common interface to emptiness check algorithms.

```
#include <tgbalgorithms/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\(\)](#)

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.

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().

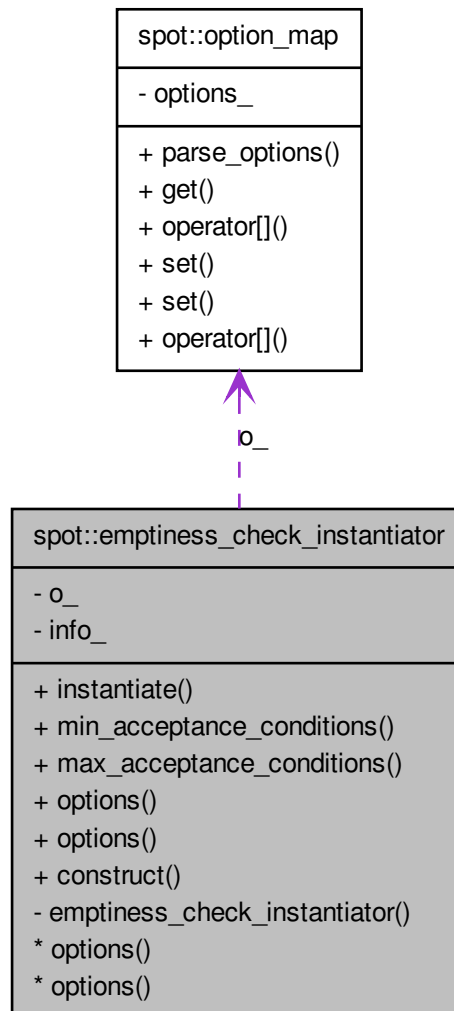
The documentation for this class was generated from the following file:

- [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 const option_map& spot::emptiness_check_instantiator::options () const [inline]

Accessor to the options.

References o_.

7.34.2.6 option_map& spot::emptiness_check_instantiator::options () [inline]

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:

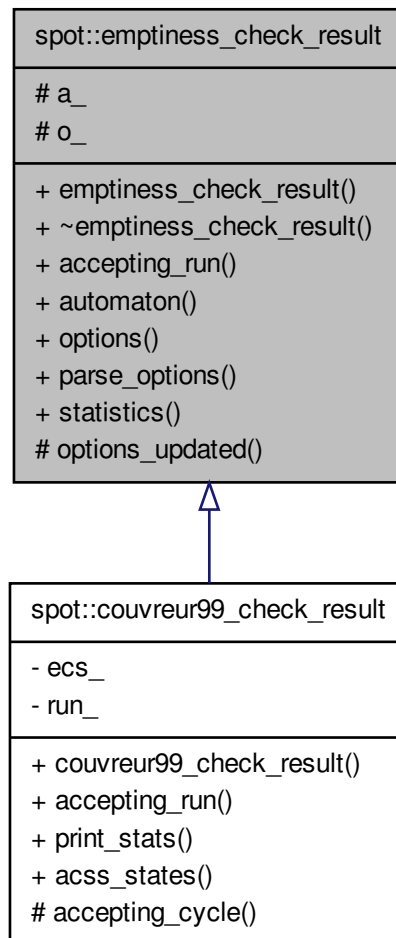
- [tgbaalgos/emptiness.hh](#)

7.35 spot::emptiness_check_result Class Reference

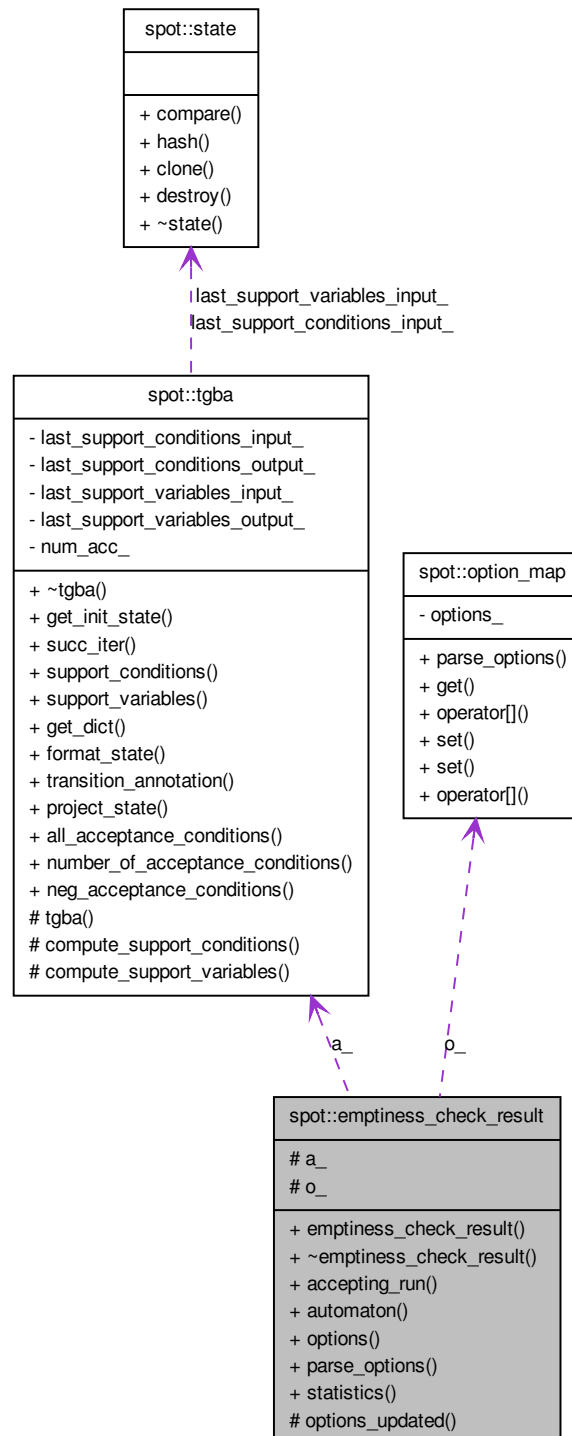
The result of an emptiness check.

```
#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).

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:

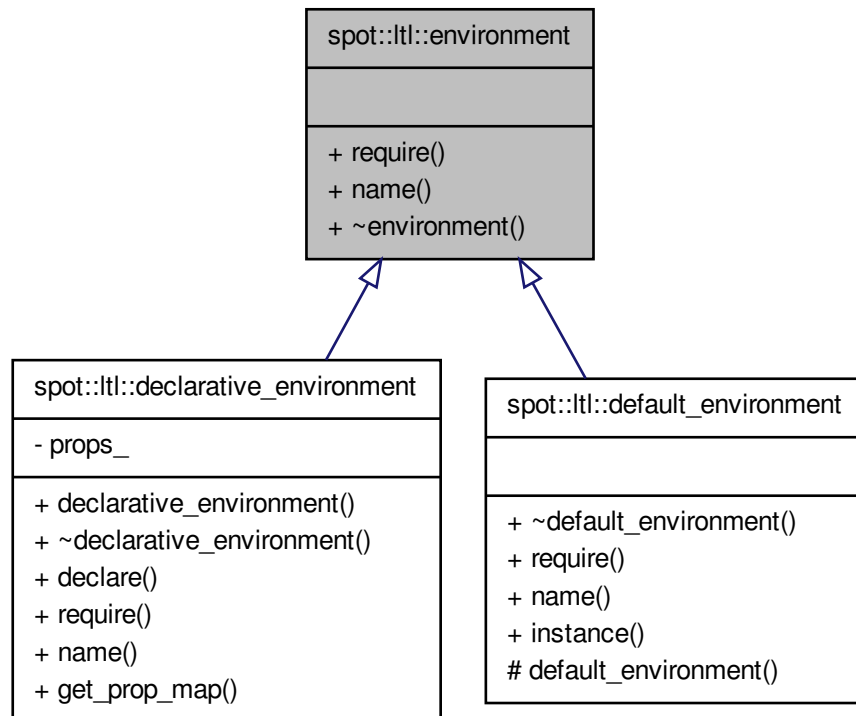
- tgbaalgos/[emptiness.hh](#)

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](#).

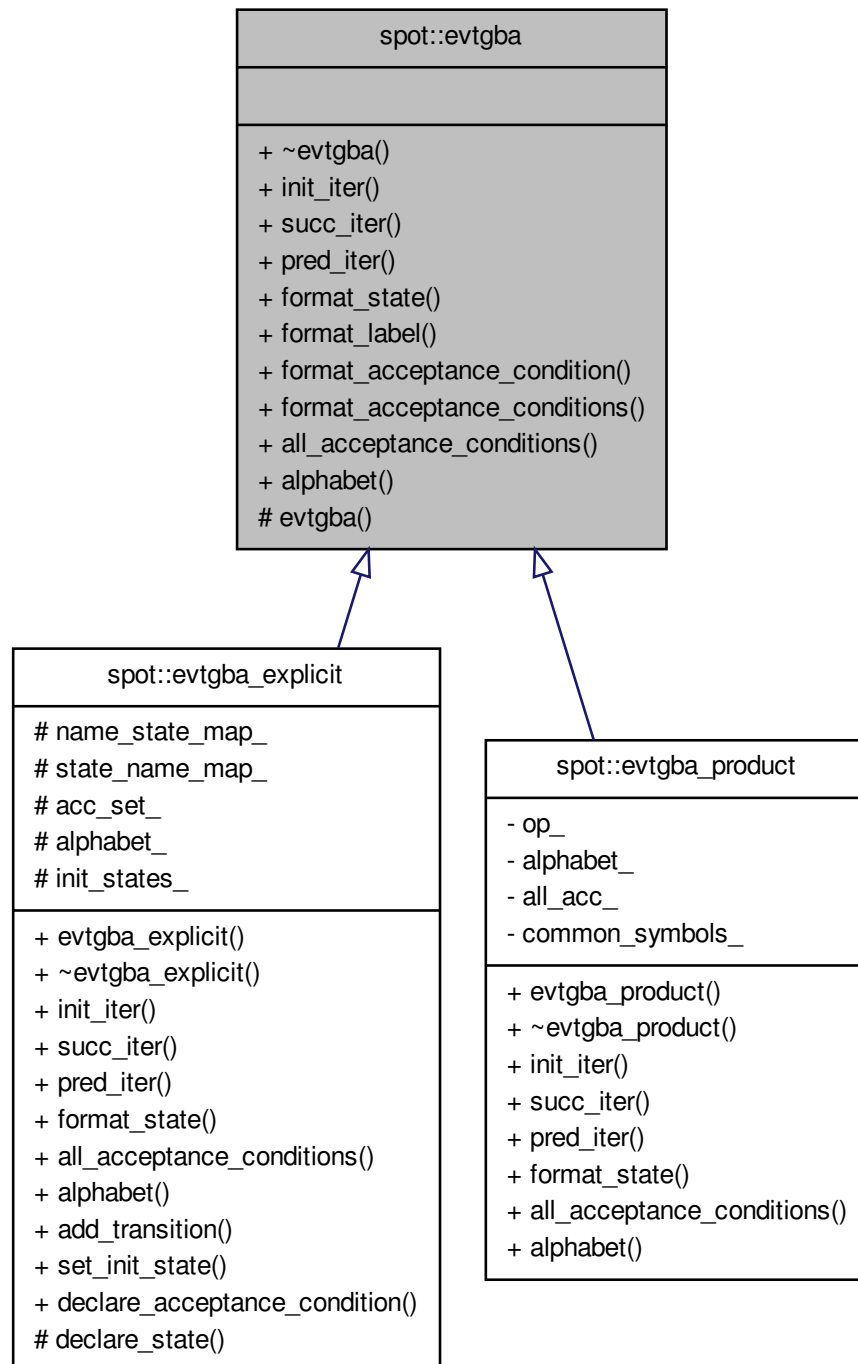
The documentation for this class was generated from the following file:

- [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](#).

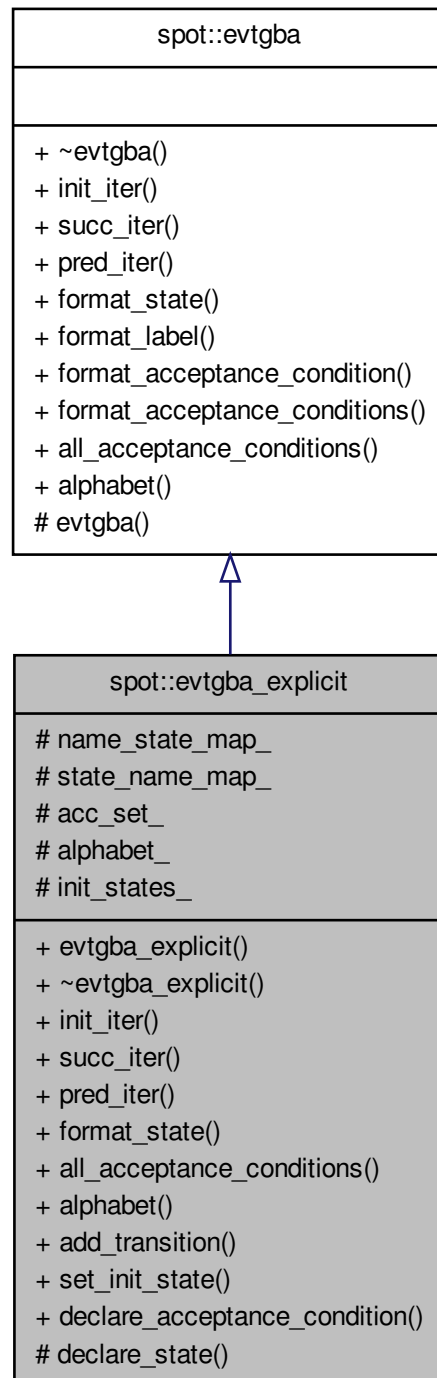
The documentation for this class was generated from the following file:

- [evtgba/evtgba.hh](#)

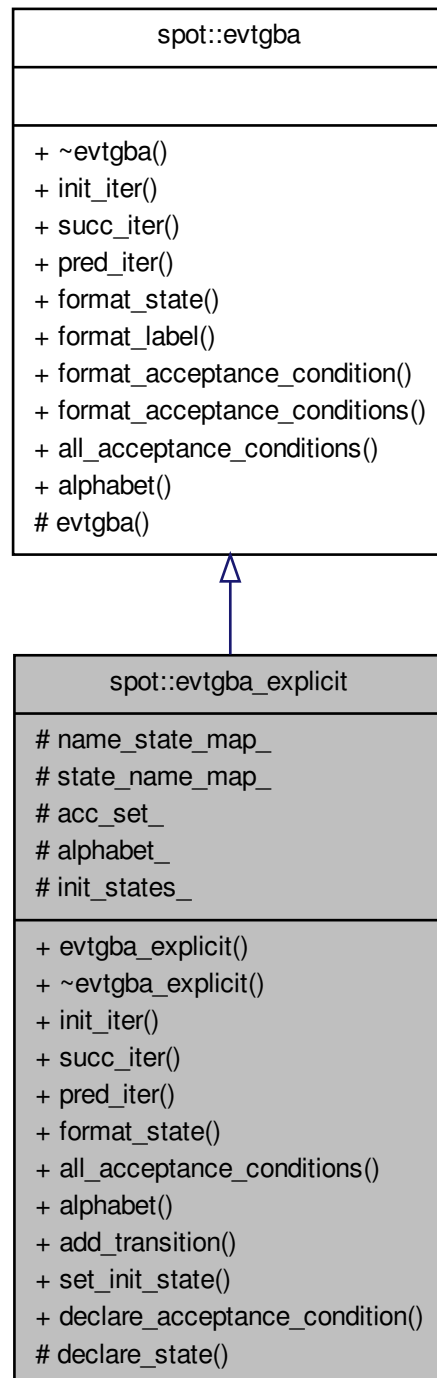
7.38 spot::evtgba_explicit Class Reference

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

Implements [spot::evtgba](#).

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**]

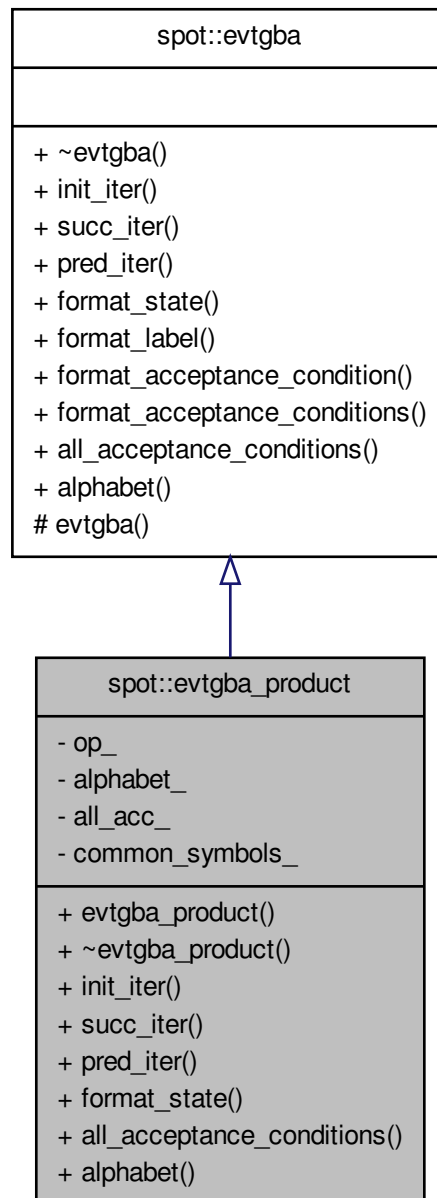
The documentation for this class was generated from the following file:

- evtgba/[evtgbaiter.hh](#)

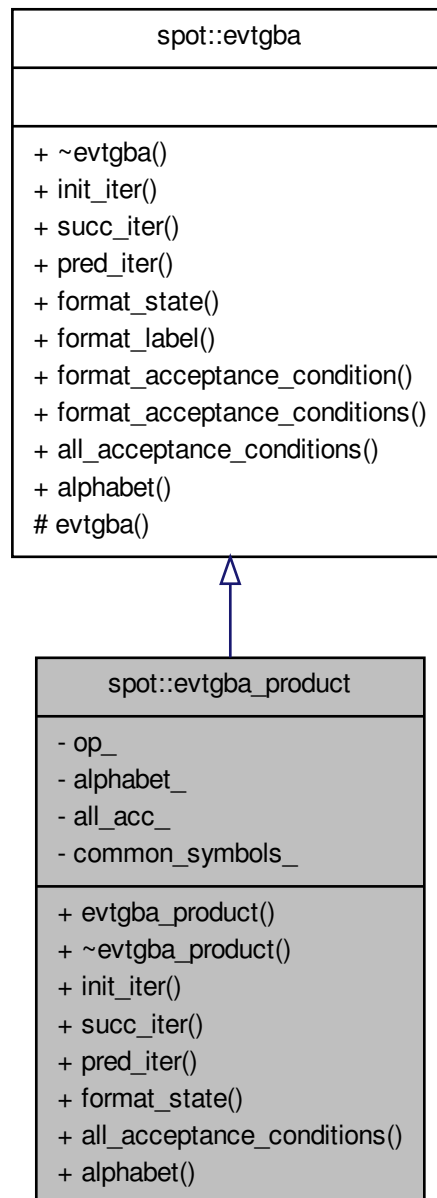
7.40 spot::evtgba_product Class Reference

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

Implements [spot::evtgba](#).

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:

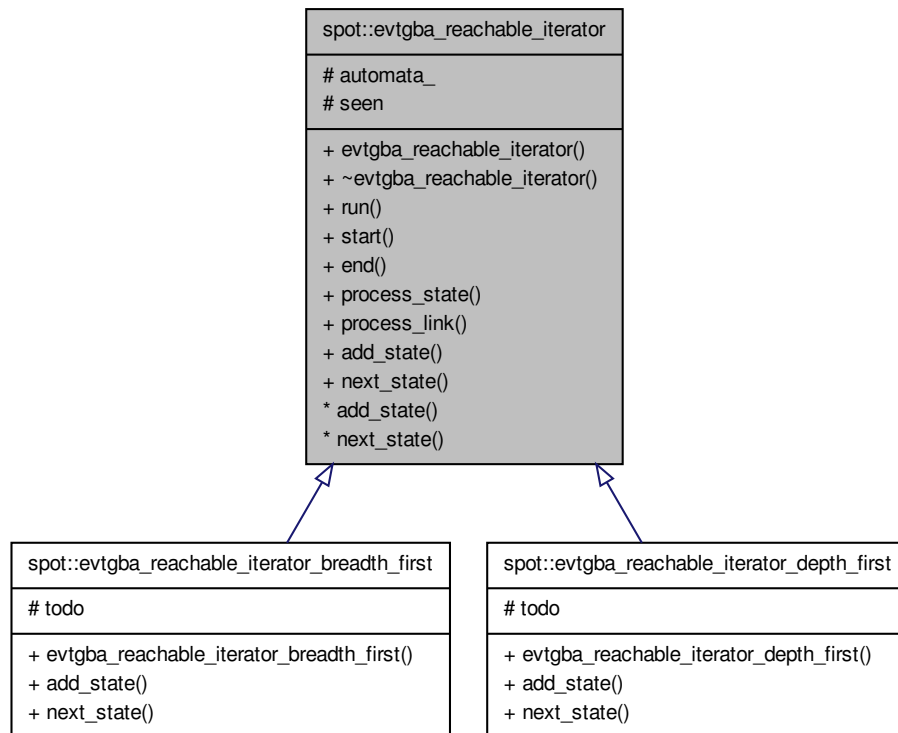
- [evtgba/product.hh](#)

7.41 spot::evtgba_reachable_iterator Class Reference

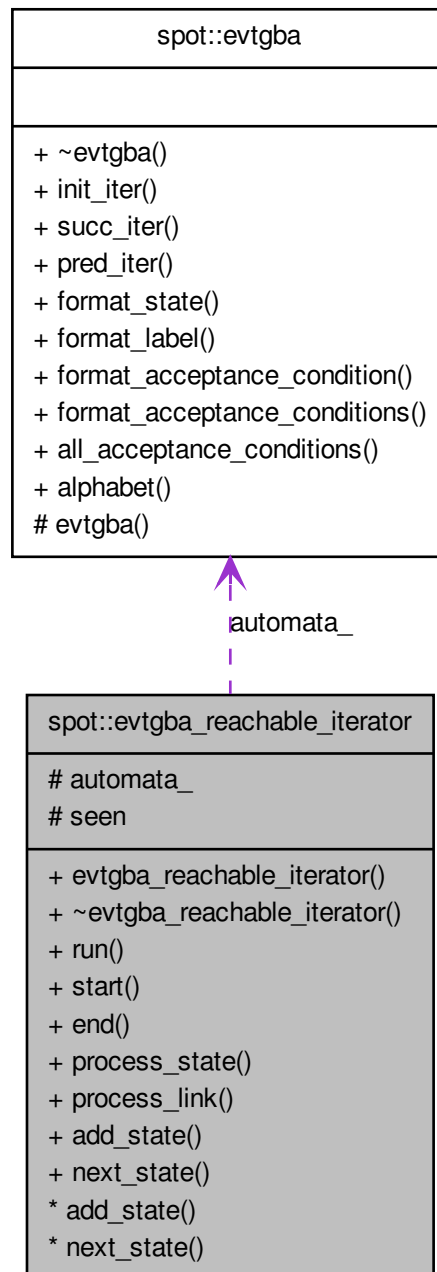
Iterate over all reachable states of a [spot::evtgba](#).

```
#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:

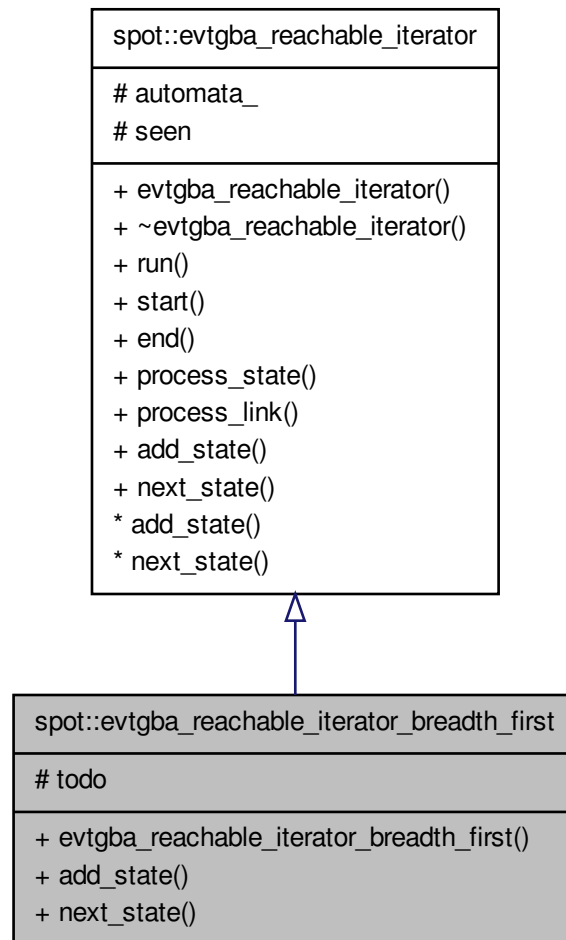
- `evtgbaalgos/reachiter.hh`

7.42 `spot::evtgba_reachable_iterator_breadth_first` Class Reference

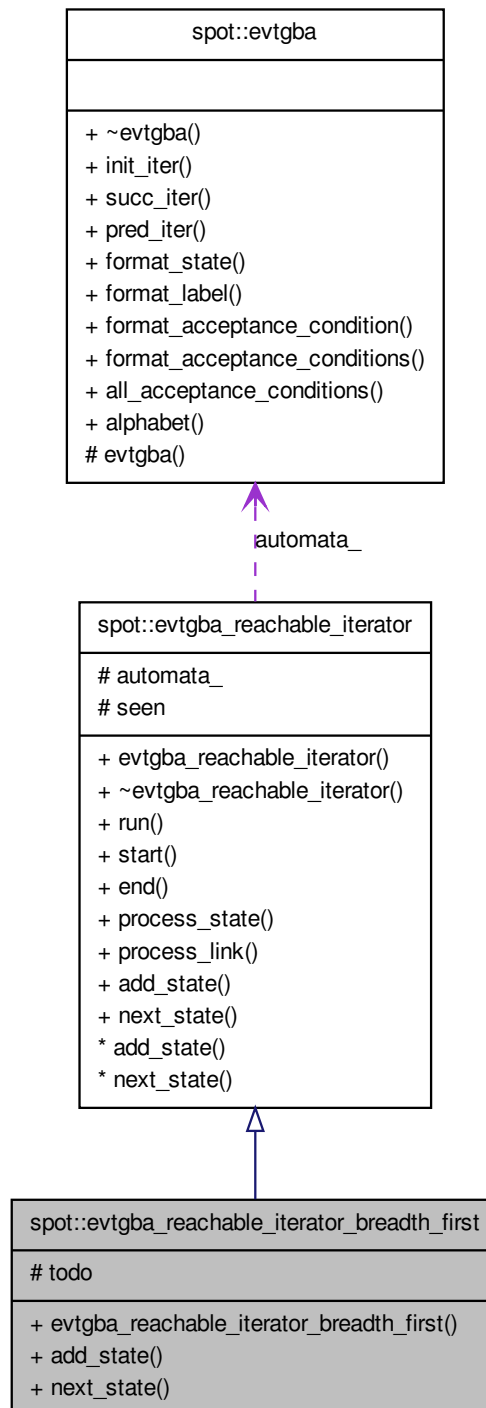
An implementation of `spot::evtgba_reachable_iterator` that browses states breadth first.

```
#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:

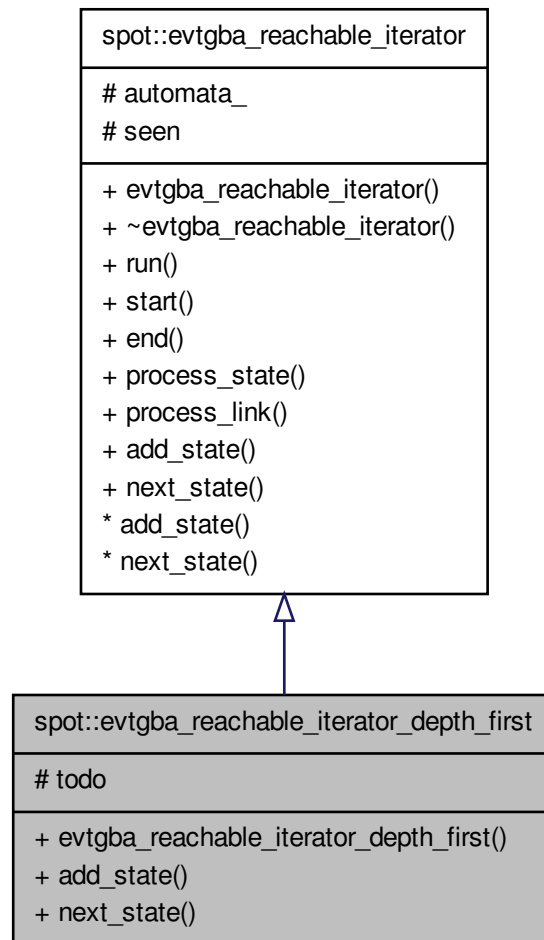
- [evtgbaalgos/reachiter.hh](#)

7.43 `spot::evtgba_reachable_iterator_depth_first` Class Reference

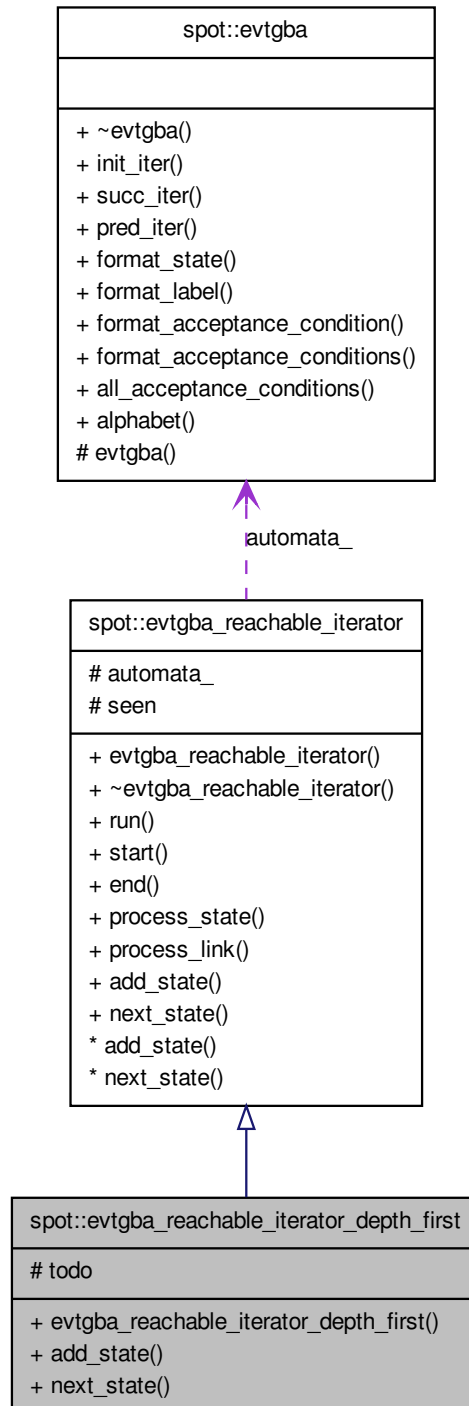
An implementation of [spot::evtgba_reachable_iterator](#) that browses states depth first.

```
#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:

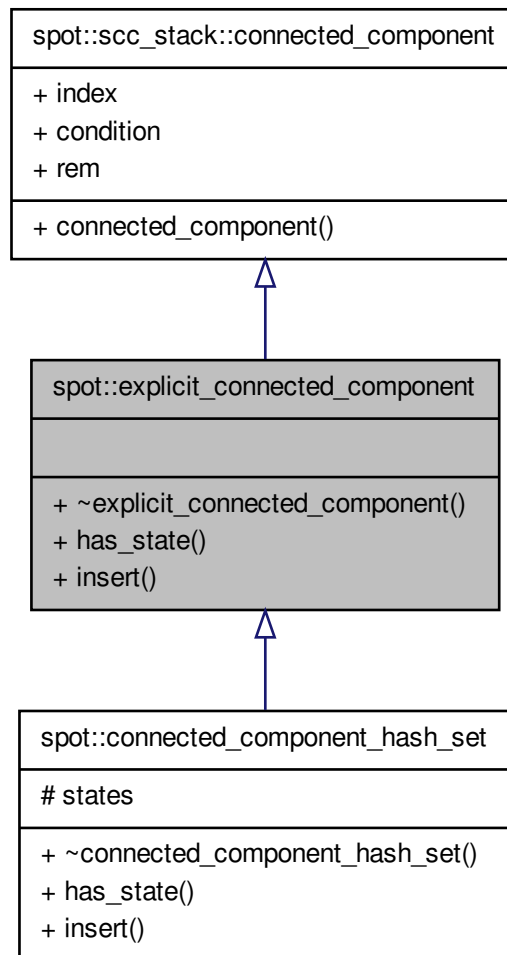
- [evtgbalgorithms/reachiter.hh](#)

7.44 `spot::explicit_connected_component` Class Reference

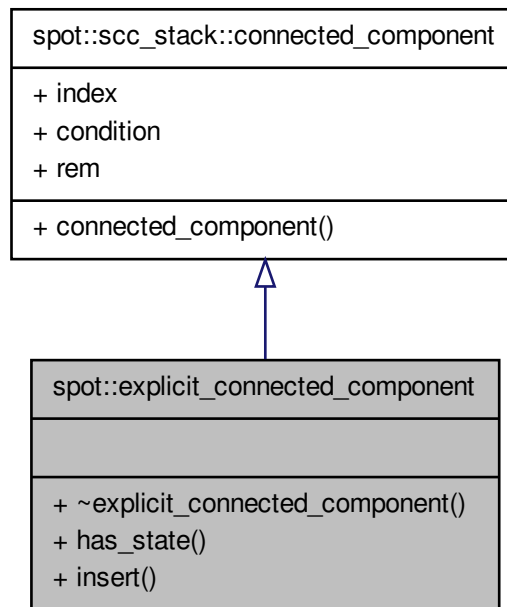
An SCC storing all its states explicitly.

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

Inheritance diagram for spot::explicit_connected_component:



Collaboration diagram for spot::explicit_connected_component:



Public Member Functions

- virtual `~explicit_connected_component()`
- virtual const `state * has_state (const state *s)` const =0
Check if the SCC contains states s.
- virtual void `insert (const state *s)`=0
Insert a new state in the SCC.

Public Attributes

- int `index`
Index of the SCC.
- bdd `condition`
- `std::list< const state * > rem`

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 destroy 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:

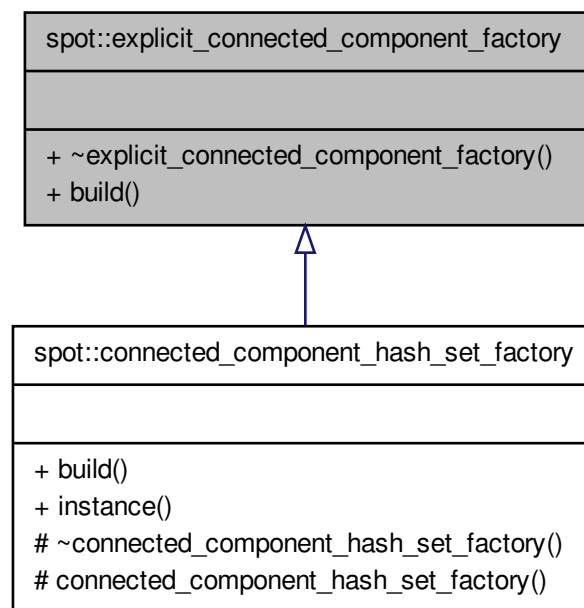
- [tgbaalgos/gtec/explsc.hh](#)

7.45 spot::explicit_connected_component_factory Class Reference

Abstract factory for [explicit_connected_component](#).

```
#include <tgbaaalgos/gtec/explscc.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](#)

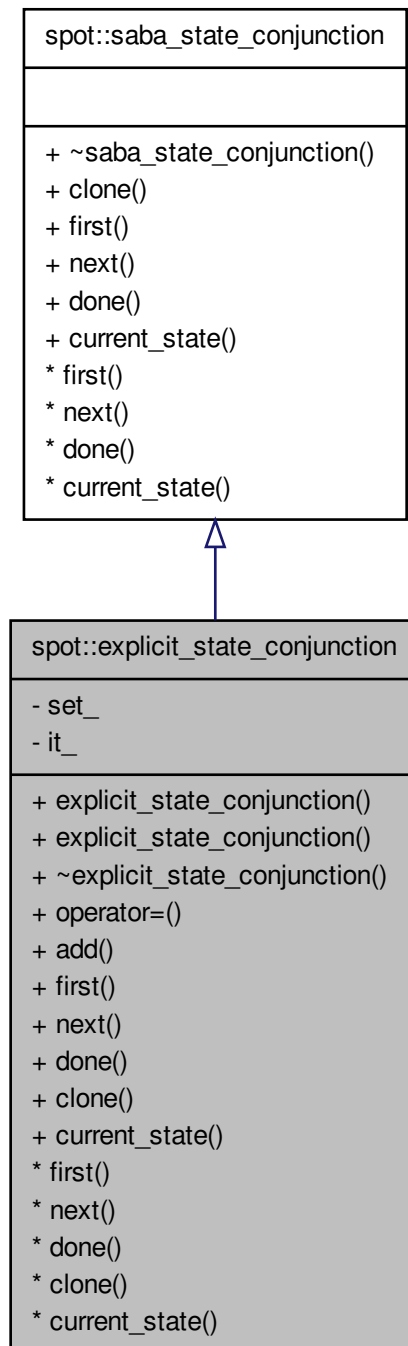
7.46 spot::explicit_state_conjunction Class Reference

Basic implementation of [saba_state_conjunction](#).

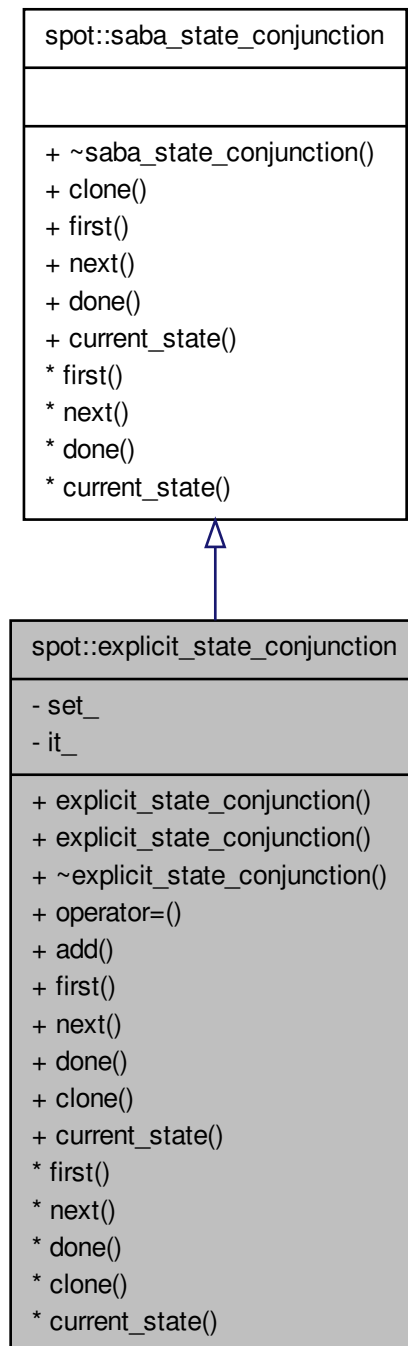
This class provides a basic implementation to iterate over a conjunction of states of a saba.

```
#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`

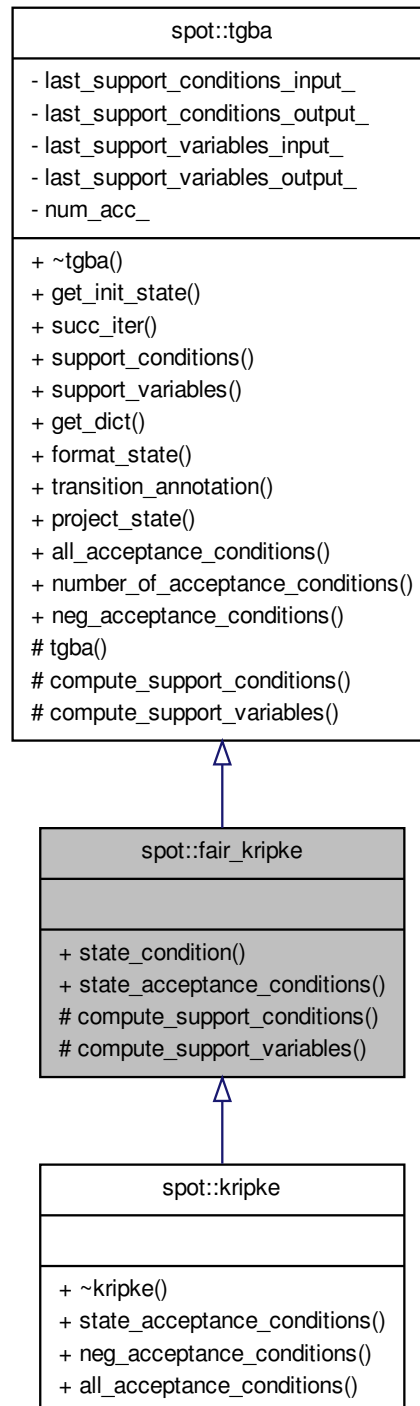
7.47 `spot::fair_kripke` Class Reference

Interface for a Fair Kripke structure.

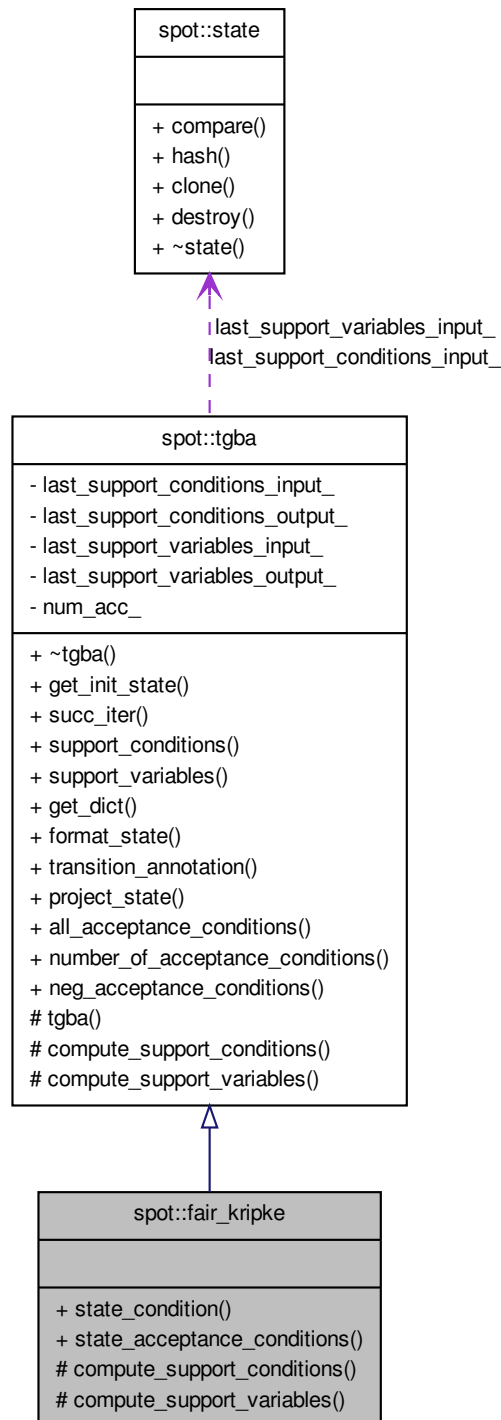
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition, and a set of acceptance conditions.

```
#include <kripke/fairkripke.hh>
```

Inheritance diagram for spot::fair_kripke:



Collaboration diagram for spot::fair_kripke:



Public Member Functions

- virtual bdd [state_condition](#) (const [state](#) *s) const =0
The condition that label the state s.
- virtual bdd [state_acceptance_conditions](#) (const [state](#) *s) const =0
The set of acceptance conditions that label the state s.
- 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 Detailed Description

Interface for a Fair Kripke structure.

A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition, and a set of acceptance conditions. Such a structure can be seen as [spot::tgba](#) by pushing all labels to the outgoing transitions.

A programmer that develops an instance of Fair Kripke structure needs just provide an implementation for the following methods:

- [kripke::get_init_state\(\)](#)
- [kripke::succ_iter\(\)](#)
- [kripke::state_condition\(\)](#)
- [kripke::state_acceptance_conditions\(\)](#)
- [kripke::format_state\(\)](#)
- and optionally [kripke::transition_annotation\(\)](#)

The other methods of the `tgba` interface are supplied by this class and need not be defined.

See also [spot::fair_kripke_succ_iterator](#).

7.47.2 Member Function Documentation

7.47.2.1 `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](#).

7.47.2.2 `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](#).

7.47.2.3 `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`.

7.47.2.4 `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_explicit_number`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_safera_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 >`.

7.47.2.5 `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_safera_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.47.2.6 `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 `destroy` 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_product_init`, `spot::tgba_safera_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, `spot::tgba_sba_proxy`, and `spot::tgba_union`.

7.47.2.7 `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_safra_complement](#), [spot::tgba_scc](#), [spot::tgba_sgba_proxy](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.47.2.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

7.47.2.9 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.47.2.10 `virtual bdd spot::fair_kripke::state_acceptance_conditions (const state * s) const`
`[pure virtual]`

The set of acceptance conditions that label the state *s*.

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

7.47.2.11 `virtual bdd spot::fair_kripke::state_condition (const state * s) const` `[pure virtual]`

The condition that label the state *s*.

This should be a conjunction of atomic propositions.

7.47.2.12 `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 synchronised 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 destroy it when appropriate (this can be done during the lifetime of the iterator).

global_state In a product, the state of the global product automaton. Otherwise, 0. Like *local_state*, *global_state* is not adopted by `succ_iter`.

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

Implemented in `spot::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`.

7.47.2.13 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.

7.47.2.14 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.

7.47.2.15 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](#), and [spot::tgba_scc](#).

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

- [kripke/fairkripke.hh](#)

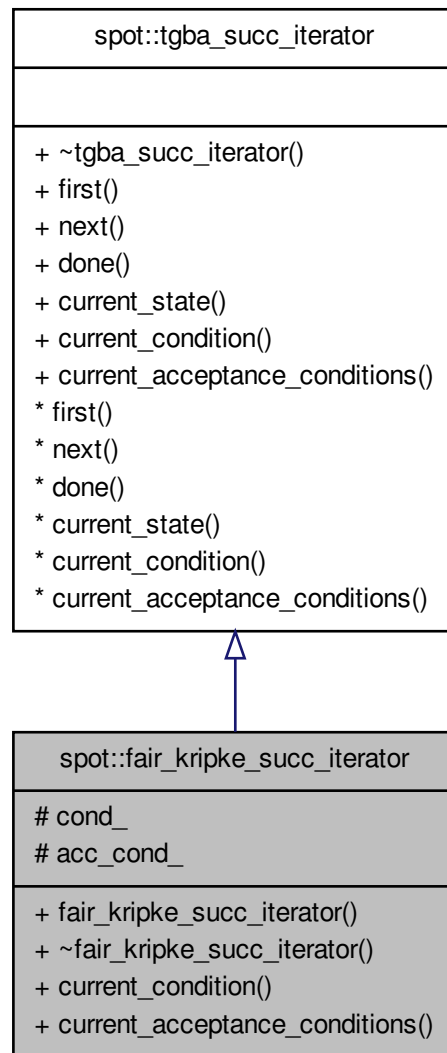
7.48 `spot::fair_kripke_succ_iterator` Class Reference

Iterator code for a Fair Kripke structure.

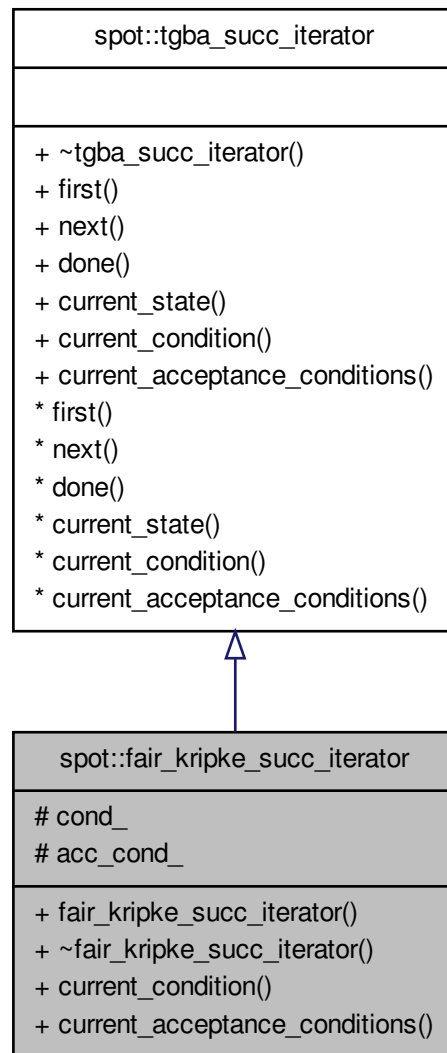
This iterator can be used to simplify the writing of an iterator on a Fair Kripke structure (or lookalike).

```
#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 bdd &cond, const bdd &acc_cond)

Constructor.

- virtual [~fair_kripke_succ_iterator](#) ()
- 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.

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.

Protected Attributes

- bdd `cond_`
- bdd `acc_cond_`

7.48.1 Detailed Description

Iterator code for a Fair Kripke structure.

This iterator can be used to simplify the writing of an iterator on a Fair Kripke structure (or lookalike). If you inherit from this iterator, you should only redefine

- `fair_kripke_succ_iterator::first()`
- `fair_kripke_succ_iterator::next()`
- `fair_kripke_succ_iterator::done()`
- `fair_kripke_succ_iterator::current_state()`

This class implements `fair_kripke_succ_iterator::current_condition()`, and `fair_kripke_succ_iterator::current_acceptance_conditions()`.

7.48.2 Constructor & Destructor Documentation

7.48.2.1 `spot::fair_kripke_succ_iterator::fair_kripke_succ_iterator (const bdd & cond, const bdd & acc_cond)`

Constructor.

The `cond` and `acc_cond` arguments will be those returned by `fair_kripke_succ_iterator::current_condition()`, and `fair_kripke_succ_iterator::current_acceptance_conditions()`.

7.48.2.2 `virtual spot::fair_kripke_succ_iterator::~~fair_kripke_succ_iterator () [virtual]`**7.48.3 Member Function Documentation****7.48.3.1** `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](#).

7.48.3.2 `virtual bdd spot::fair_kripke_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](#).

7.48.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

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](#).

7.48.3.4 `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](#).

7.48.3.5 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`.

7.48.3.6 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`.

7.48.4 Member Data Documentation**7.48.4.1 bdd spot::fair_kripke_succ_iterator::acc_cond_ [protected]****7.48.4.2 bdd spot::fair_kripke_succ_iterator::cond_ [protected]**

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

- `kripke/fairkripke.hh`

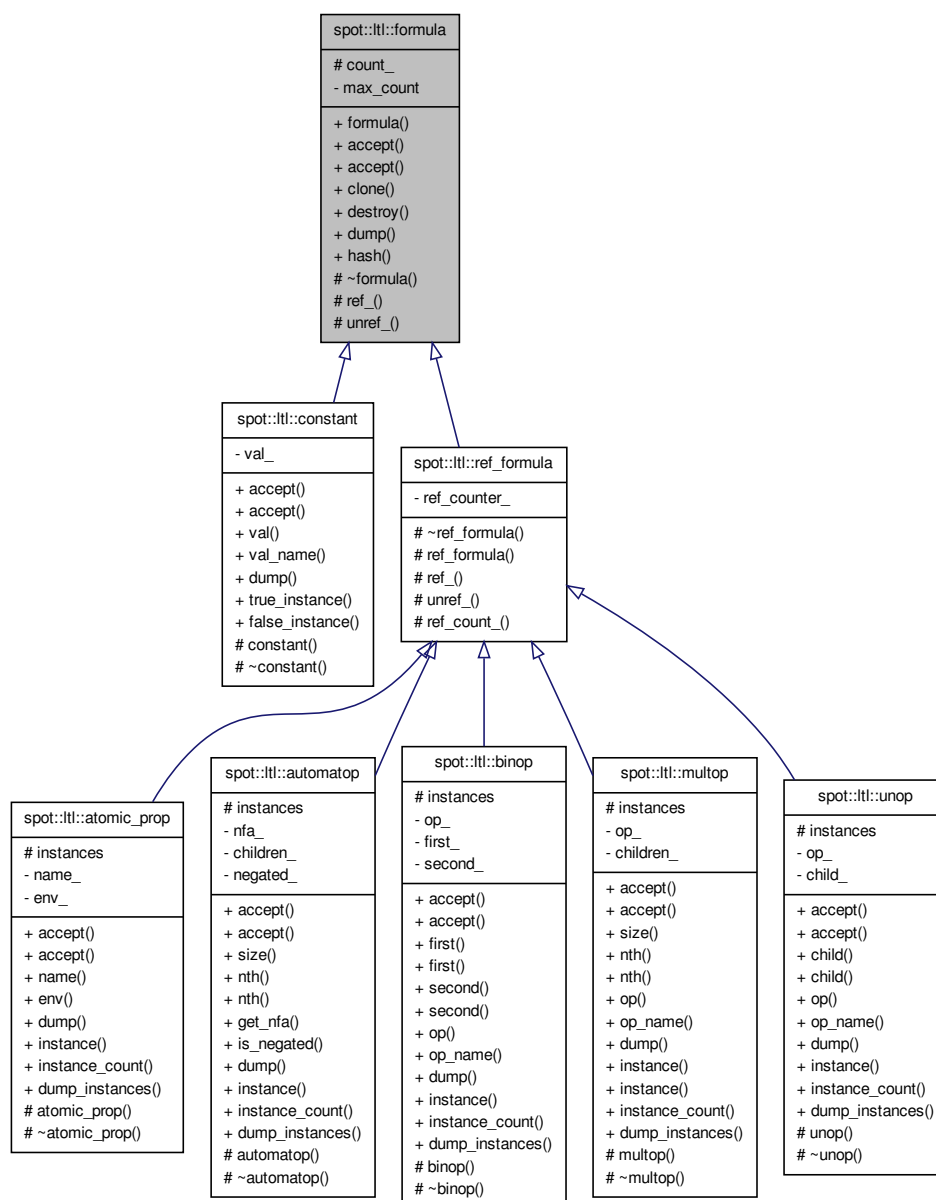
7.49 spot::ltl::formula Class Reference

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 (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](#).

7.49.3.2 `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](#).

7.49.3.3 `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](#).

Referenced by `spot::ltl::formula_ptr_less_than::operator()`.

7.49.3.6 `size_t spot::ltl::formula::hash () const [inline]`

Return a hash key for the formula.

References `count_`.

Referenced by `spot::ltl::formula_ptr_hash::operator()()`, and `spot::ltl::formula_ptr_less_than::operator()()`.

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]`

References `spot::ltl::formula::hash()`.

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]`

References `spot::ltl::formula::dump()`, and `spot::ltl::formula::hash()`.

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

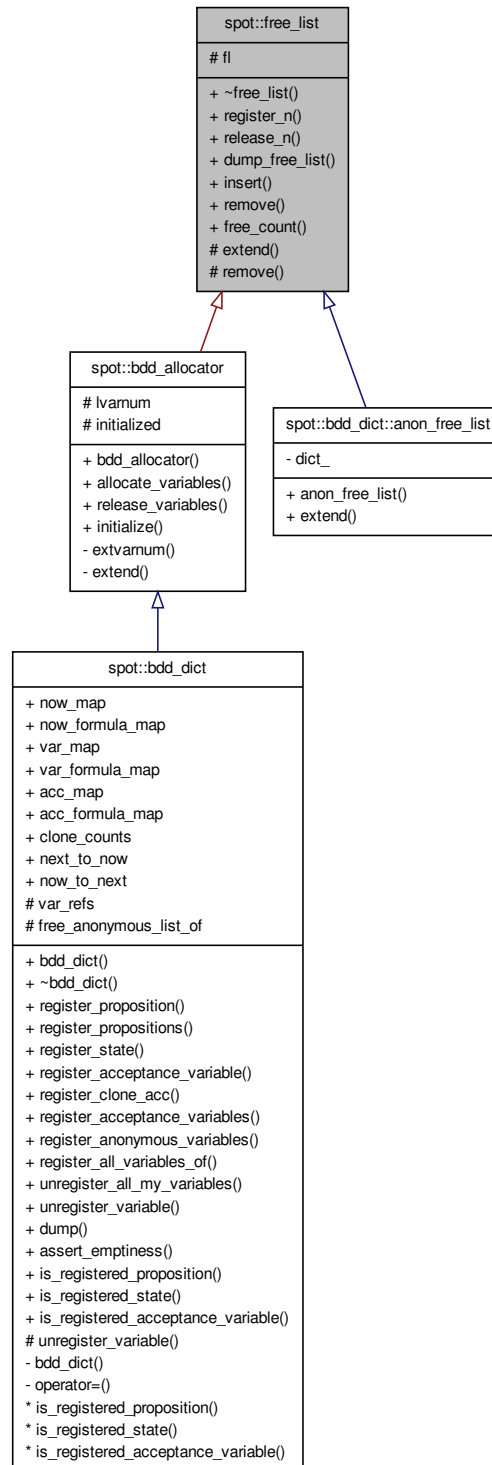
- [ltlast/formula.hh](#)

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-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

- 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 (int base, int n = 0)`

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

7.52.4.8 `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*.

7.52.5 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](#)

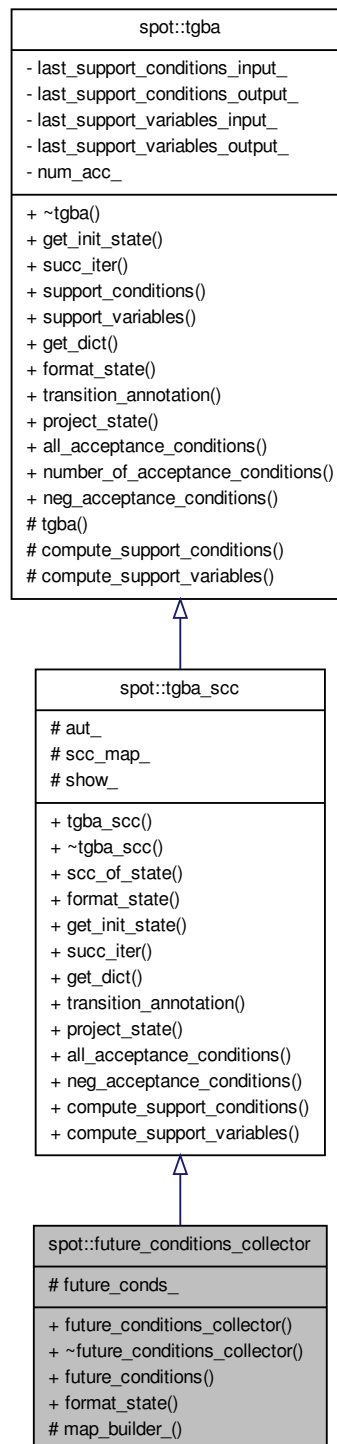
7.53 `spot::future_conditions_collector` Class Reference

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

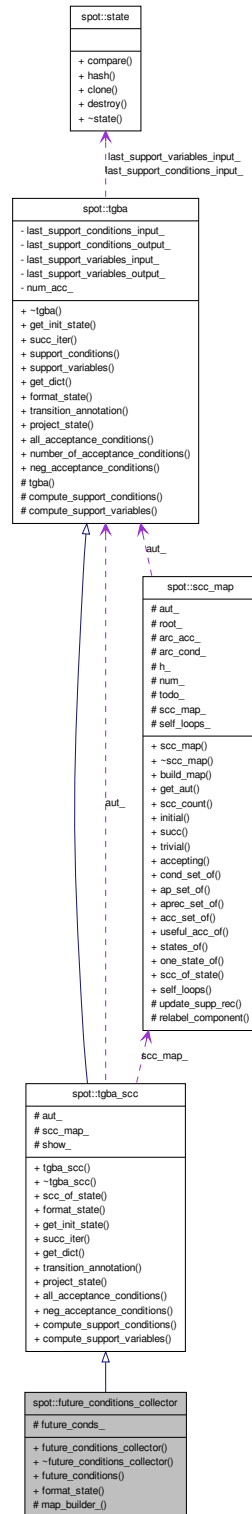
This class is a `spot::tgba` wrapper that simply add a new method, [future_conditions\(\)](#), to any `spot::tgba`.

```
#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 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](#).

7.53.4.2 `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 `destroy` 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 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 destroyed 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 destroy 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()`.

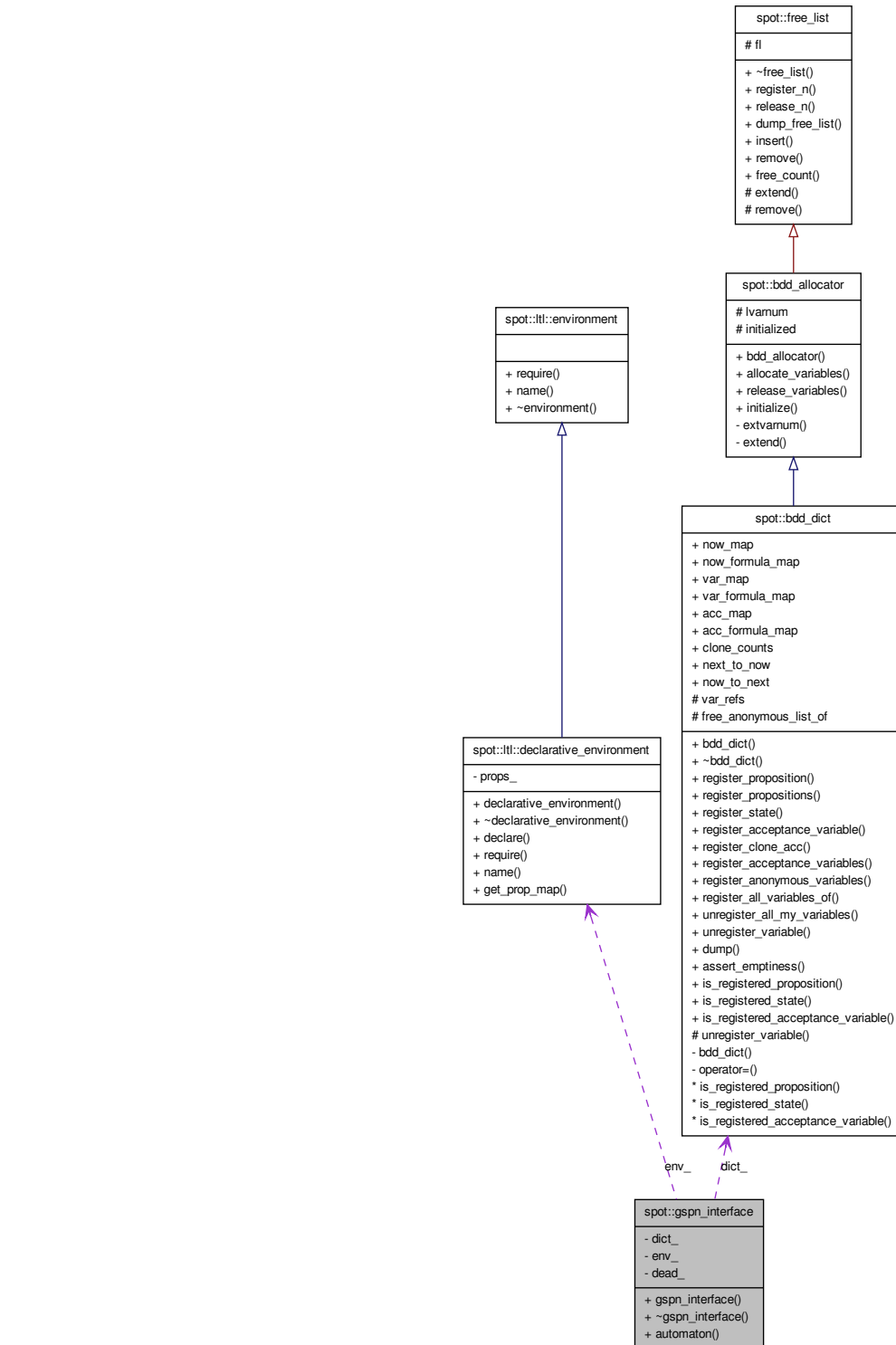
The documentation for this class was generated from the following file:

- [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]`

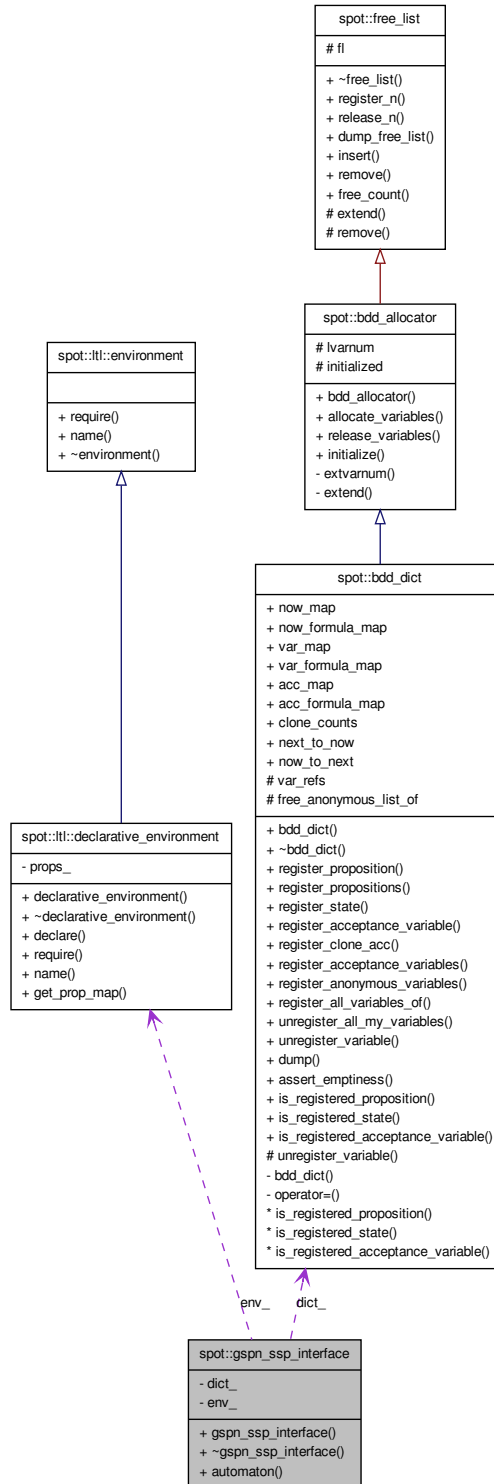
The documentation for this class was generated from the following file:

- [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::identity_hash< T >` Struct Template Reference

A hash function that returns identity.

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

Public Member Functions

- `size_t operator()` (const T &s) const

7.57.1 Detailed Description

`template<typename T> struct spot::identity_hash< T >`

A hash function that returns identity.

7.57.2 Member Function Documentation

7.57.2.1 `template<typename T> size_t spot::identity_hash< T >::operator() (const T & s)`
`const [inline]`

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

- `misc/hash.hh`

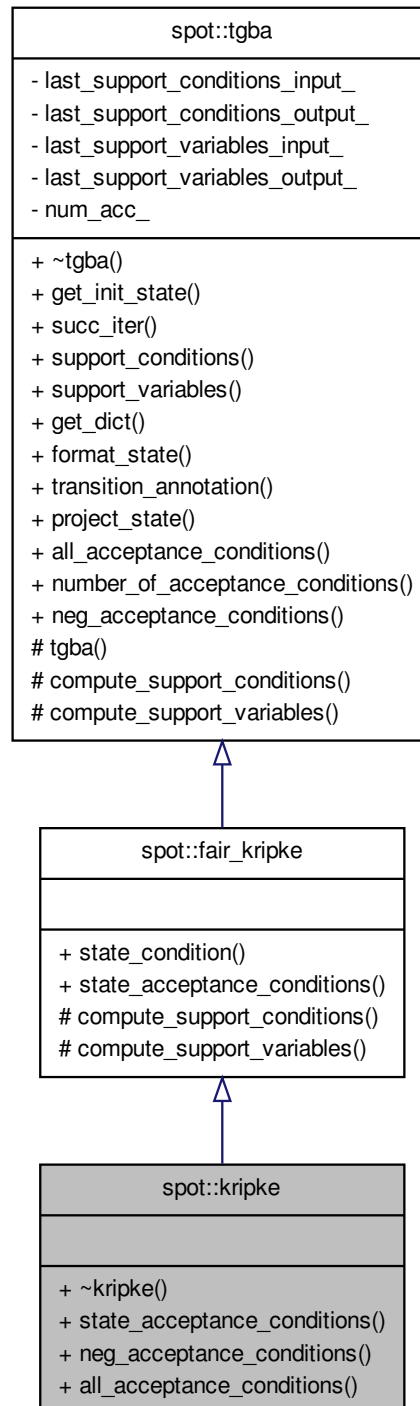
7.58 spot::kripke Class Reference

Interface for a Kripke structure

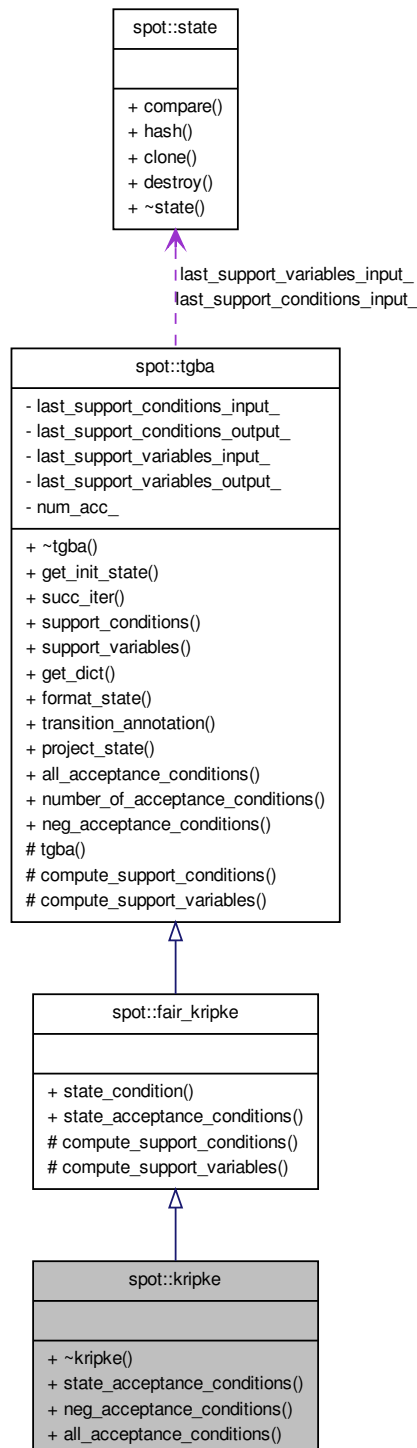
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition.

```
#include <kripke/kripke.hh>
```

Inheritance diagram for spot::kripke:



Collaboration diagram for spot::kripke:



Public Member Functions

- virtual `~kripke ()`
- virtual bdd `state_acceptance_conditions (const state *)` const
The set of acceptance conditions that label the state s.
- 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 `state_condition (const state *s)` const =0
The condition that label the state s.
- 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 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()`.

7.58.1 Detailed Description

Interface for a Kripke structure

A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition. Such a structure can be seen as [spot::tgba](#) without any acceptance condition.

A programmer that develops an instance of Kripke structure needs just provide an implementation for the following methods:

- [kripke::get_init_state\(\)](#)
- [kripke::succ_iter\(\)](#)
- [kripke::state_condition\(\)](#)
- [kripke::format_state\(\)](#)
- and optionally [kripke::transition_annotation\(\)](#)

The other methods of the [tgba](#) interface (like those dealing with acceptance conditions) are supplied by this [kripke](#) class and need not be defined.

See also [spot::kripke_succ_iterator](#).

7.58.2 Constructor & Destructor Documentation

7.58.2.1 virtual spot::kripke::~~kripke () [virtual]

7.58.3 Member Function Documentation

7.58.3.1 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](#).

7.58.3.2 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](#).

7.58.3.3 `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`.

7.58.3.4 `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_explicit_number`, `spot::tgba_kv_complement`, `spot::tgba_product`, `spot::tgba_saфра_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 >`.

7.58.3.5 `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_saфра_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.58.3.6 `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 `destroy` 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_product_init`, `spot::tgba_saфра_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, `spot::tgba_sba_proxy`, and `spot::tgba_union`.

7.58.3.7 `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](#).

7.58.3.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
[virtual, inherited]

The number of acceptance conditions.

7.58.3.9 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.58.3.10 `virtual bdd spot::kripke::state_acceptance_conditions (const state * s) const`
[virtual]

The set of acceptance conditions that label the state *s*.

Implements [spot::fair_kripke](#).

7.58.3.11 `virtual bdd spot::fair_kripke::state_condition (const state * s) const` **[pure virtual, inherited]**

The condition that label the state *s*.

This should be a conjunction of atomic propositions.

7.58.3.12 `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 destroy 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`.

7.58.3.13 `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.

7.58.3.14 `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.

7.58.3.15 `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](#), and [spot::tgba_scc](#).

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

- [kripke/kripke.hh](#)

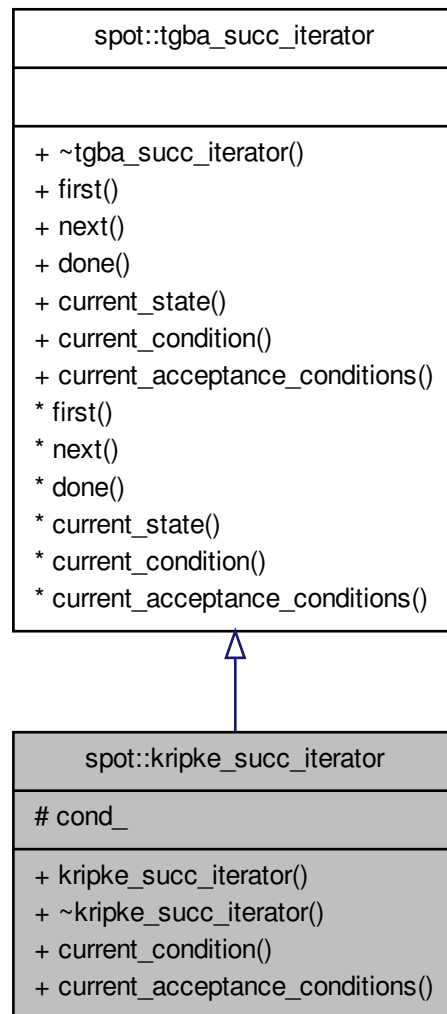
7.59 spot::kripke_succ_iterator Class Reference

Iterator code for Kripke structure

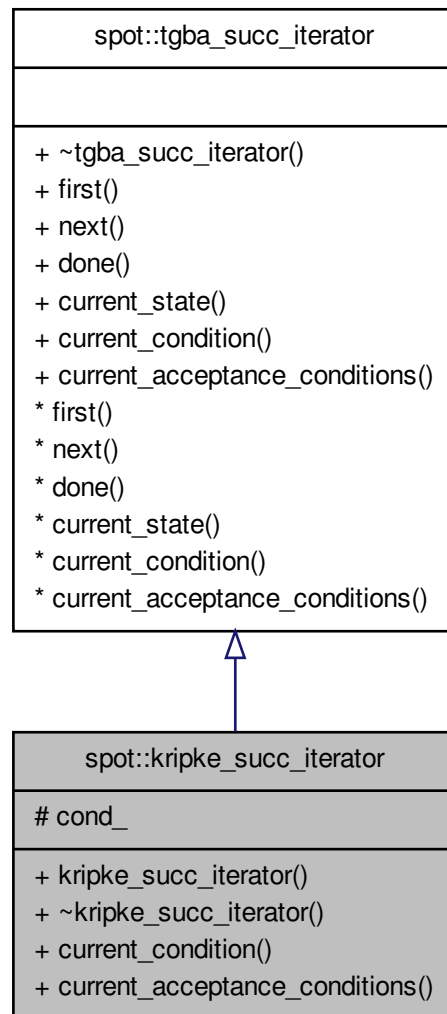
This iterator can be used to simplify the writing of an iterator on a Kripke structure (or lookalike).

```
#include <kripke/kripke.hh>
```

Inheritance diagram for spot::kripke_succ_iterator:



Collaboration diagram for spot::kripke_succ_iterator:



Public Member Functions

- [kripke_succ_iterator](#) (const bdd &cond)

Constructor:

- virtual [~kripke_succ_iterator](#) ()
- 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.

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.

Protected Attributes

- bdd `cond_`

7.59.1 Detailed Description

Iterator code for Kripke structure

This iterator can be used to simplify the writing of an iterator on a Kripke structure (or lookalike). If you inherit from this iterator, you should only redefine

- `kripke_succ_iterator::first()`
- `kripke_succ_iterator::next()`
- `kripke_succ_iterator::done()`
- `kripke_succ_iterator::current_state()`

This class implements `kripke_succ_iterator::current_condition()`, and `kripke_succ_iterator::current_acceptance_conditions()`.

7.59.2 Constructor & Destructor Documentation

7.59.2.1 `spot::kripke_succ_iterator::kripke_succ_iterator (const bdd & cond)`

Constructor.

The `cond` argument will be the one returned by `kripke_succ_iterator::current_condition()`.

7.59.2.2 `virtual spot::kripke_succ_iterator::~~kripke_succ_iterator () [virtual]`

7.59.3 Member Function Documentation

7.59.3.1 `virtual bdd spot::kripke_succ_iterator::current_acceptance_conditions () const [virtual]`

Get the acceptance conditions on the transition leading to this successor.

Implements [spot::tgba_succ_iterator](#).

7.59.3.2 `virtual bdd spot::kripke_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](#).

7.59.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

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](#).

7.59.3.4 `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](#).

7.59.3.5 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`.

7.59.3.6 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`.

7.59.4 Member Data Documentation

7.59.4.1 bdd spot::kripke_succ_iterator::cond_ [protected]

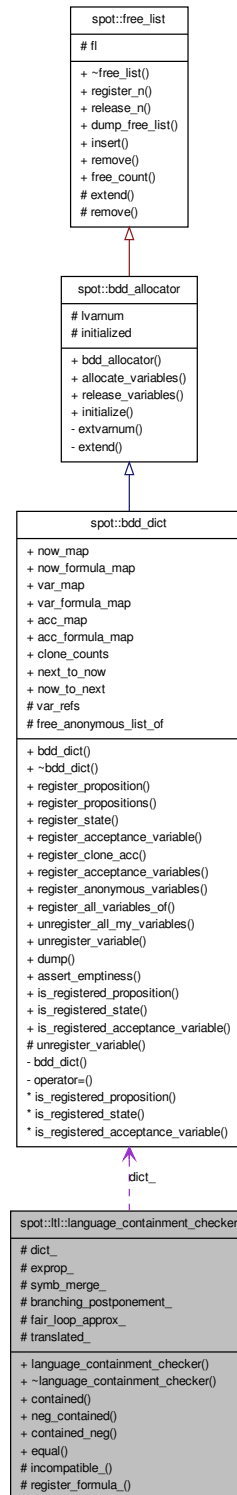
The documentation for this class was generated from the following file:

- [kripke/kripke.hh](#)

7.60 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](#)

7.60.1 Member Typedef Documentation

- 7.60.1.1** typedef Sgi::hash_map<const [formula](#)*, [record_](#), [formula_ptr_hash](#)>
spot::ltl::language_containment_checker::trans_map [private]

7.60.2 Constructor & Destructor Documentation

7.60.2.1 `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.

7.60.2.2 `spot::ltl::language_containment_checker::~~language_containment_checker ()`

7.60.3 Member Function Documentation

7.60.3.1 `bool spot::ltl::language_containment_checker::contained (const formula * l, const formula * g)`

Check whether $L(l)$ is a subset of $L(g)$.

7.60.3.2 `bool spot::ltl::language_containment_checker::contained_neg (const formula * l, const formula * g)`

Check whether $L(l)$ is a subset of $L(!g)$.

7.60.3.3 `bool spot::ltl::language_containment_checker::equal (const formula * l, const formula * g)`

Check whether $L(l) = L(g)$.

7.60.3.4 `bool spot::ltl::language_containment_checker::incompatible_ (record_ * l, record_ * g) [protected]`

7.60.3.5 `bool spot::ltl::language_containment_checker::neg_contained (const formula * l, const formula * g)`

Check whether $L(!l)$ is a subset of $L(g)$.

7.60.3.6 `record_* spot::ltl::language_containment_checker::register_formula_ (const formula * f)` `[protected]`

7.60.4 Member Data Documentation

7.60.4.1 `bool spot::ltl::language_containment_checker::branching_postponement_` `[protected]`

7.60.4.2 `bdd_dict* spot::ltl::language_containment_checker::dict_` `[protected]`

7.60.4.3 `bool spot::ltl::language_containment_checker::exprop_` `[protected]`

7.60.4.4 `bool spot::ltl::language_containment_checker::fair_loop_approx_` `[protected]`

7.60.4.5 `bool spot::ltl::language_containment_checker::symb_merge_` `[protected]`

7.60.4.6 `trans_map spot::ltl::language_containment_checker::translated_` `[protected]`

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

- `ltlvisit/contain.hh`

7.61 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](#)

7.61.1 Detailed Description

Internal variables for [minato_isop](#).

7.61.2 Member Enumeration Documentation

7.61.2.1 anonymous enum

Enumerator:

FirstStep
SecondStep
ThirdStep
FourthStep

7.61.3 Constructor & Destructor Documentation

7.61.3.1 spot::minato_isop::local_vars::local_vars (bdd *f_min*, bdd *f_max*, bdd *vars*) [inline]

7.61.4 Member Data Documentation

7.61.4.1 bdd spot::minato_isop::local_vars::f0_max

7.61.4.2 bdd spot::minato_isop::local_vars::f0_min

7.61.4.3 bdd spot::minato_isop::local_vars::f1_max

7.61.4.4 bdd spot::minato_isop::local_vars::f1_min

7.61.4.5 bdd spot::minato_isop::local_vars::f_max

7.61.4.6 bdd spot::minato_isop::local_vars::f_min

7.61.4.7 bdd spot::minato_isop::local_vars::g0

7.61.4.8 bdd spot::minato_isop::local_vars::g1

7.61.4.9 enum { ... } spot::minato_isop::local_vars::step

7.61.4.10 bdd spot::minato_isop::local_vars::v1

7.61.4.11 bdd spot::minato_isop::local_vars::vars

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

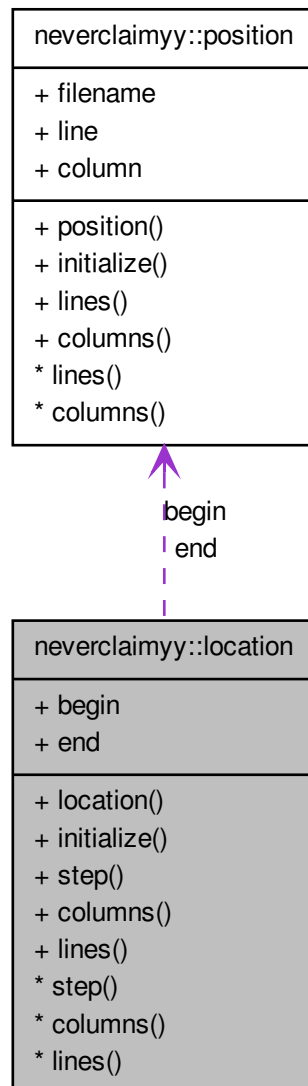
- misc/[minato.hh](#)

7.62 neverclaimyy::location Class Reference

Abstract a location.

```
#include <neverparse/location.hh>
```

Collaboration diagram for neverclaimyy::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 neverclaimyy::location::location () [inline]

Construct a location.

7.62.3 Member Function Documentation

7.62.3.1 void neverclaimyy::location::columns (unsigned int count = 1) [inline]

Extend the current location to the COUNT next columns.

Referenced by neverclaimyy::operator+(), and neverclaimyy::operator+=().

7.62.3.2 void neverclaimyy::location::initialize (std::string * fn) [inline]

Initialization.

7.62.3.3 void neverclaimyy::location::lines (unsigned int *count* = 1) [inline]

Extend the current location to the COUNT next lines.

7.62.3.4 void neverclaimyy::location::step () [inline]

Reset initial location to final location.

References begin, end, and neverclaimyy::position::initialize().

7.62.4 Member Data Documentation

7.62.4.1 position neverclaimyy::location::begin

Beginning of the located region.

Referenced by step().

7.62.4.2 position neverclaimyy::location::end

End of the located region.

Referenced by neverclaimyy::operator+(), and step().

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

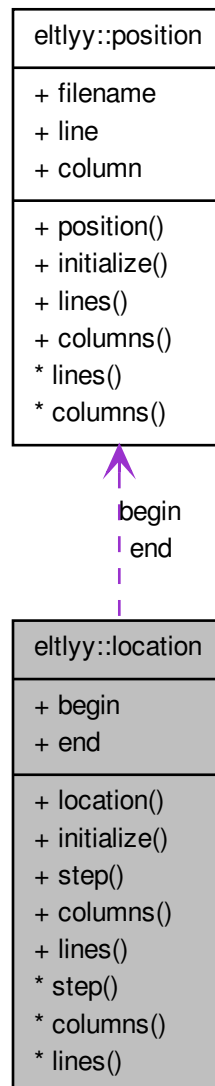
- neverparse/[location.hh](#)

7.63 eltly::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.

7.63.1 Detailed Description

Abstract a location.

7.63.2 Constructor & Destructor Documentation

7.63.2.1 eltlyy::location::location () [inline]

Construct a location.

7.63.3 Member Function Documentation

7.63.3.1 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+=().

7.63.3.2 void eltlyy::location::initialize (std::string * *fn*) [inline]

Initialization.

7.63.3.3 void eltlyy::location::lines (unsigned int *count* = 1) [inline]

Extend the current location to the COUNT next lines.

7.63.3.4 void eltlyy::location::step () [inline]

Reset initial location to final location.

References begin, end, and eltlyy::position::initialize().

7.63.4 Member Data Documentation

7.63.4.1 position eltlyy::location::begin

Beginning of the located region.

Referenced by step().

7.63.4.2 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:

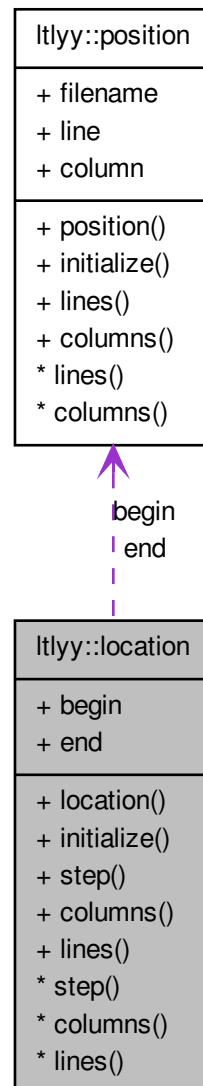
- [eltlparse/location.hh](#)

7.64 Itlly::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.

7.64.1 Detailed Description

Abstract a location.

7.64.2 Constructor & Destructor Documentation

7.64.2.1 Itlyy::location::location () [inline]

Construct a location.

7.64.3 Member Function Documentation

7.64.3.1 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+=().

7.64.3.2 void Itlyy::location::initialize (std::string * fn) [inline]

Initialization.

7.64.3.3 void ltlyy::location::lines (unsigned int *count* = 1) [inline]

Extend the current location to the COUNT next lines.

7.64.3.4 void ltlyy::location::step () [inline]

Reset initial location to final location.

References begin, end, and ltlyy::position::initialize().

7.64.4 Member Data Documentation

7.64.4.1 position ltlyy::location::begin

Beginning of the located region.

Referenced by step().

7.64.4.2 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:

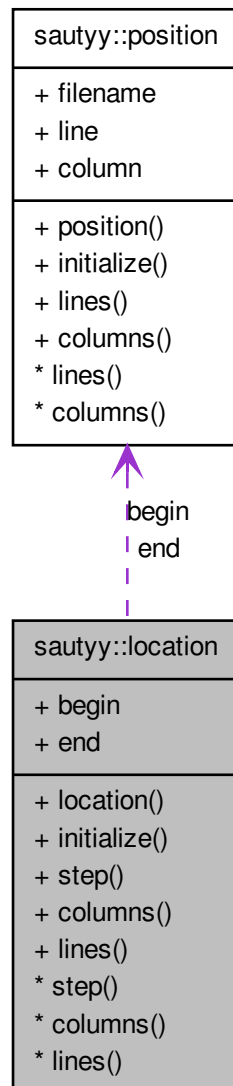
- [ltlparse/location.hh](#)

7.65 sautyy::location Class Reference

Abstract a location.

```
#include <sautparse/location.hh>
```

Collaboration diagram for sautty::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.

7.65.1 Detailed Description

Abstract a location.

7.65.2 Constructor & Destructor Documentation

7.65.2.1 sautty::location::location () [inline]

Construct a location.

7.65.3 Member Function Documentation

7.65.3.1 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+=\(\)](#).

7.65.3.2 void sautty::location::initialize (std::string * fn) [inline]

Initialization.

7.65.3.3 `void sautyy::location::lines (unsigned int count = 1) [inline]`

Extend the current location to the COUNT next lines.

References end.

7.65.3.4 `void sautyy::location::step () [inline]`

Reset initial location to final location.

7.65.4 Member Data Documentation**7.65.4.1** `position sautyy::location::begin`

Beginning of the located region.

Referenced by `columns()`.

7.65.4.2 `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`

7.66 `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_](#)

7.66.1 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 radices (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.)

7.66.2 Constructor & Destructor Documentation

7.66.2.1 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.

7.66.2.2 virtual spot::loopless_modular_mixed_radix_gray_code::~~loopless_modular_mixed_radix_gray_code () [virtual]

7.66.3 Member Function Documentation

7.66.3.1 virtual void spot::loopless_modular_mixed_radix_gray_code::a_first (int j) [pure virtual]

Reset a_j to its initial value.

7.66.3.2 virtual bool spot::loopless_modular_mixed_radix_gray_code::a_last (int j) const [pure virtual]

Whether a_j is on its last value.

7.66.3.3 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.

7.66.3.4 `bool spot::loopless_modular_mixed_radix_gray_code::done () const [inline]`

Whether all tuple have been explored.

References `done_`.

7.66.3.5 `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()`.

7.66.3.6 `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_`.

7.66.3.7 `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 .

7.66.4 Member Data Documentation**7.66.4.1** `int* spot::loopless_modular_mixed_radix_gray_code::a_ [protected]`**7.66.4.2** `bool spot::loopless_modular_mixed_radix_gray_code::done_ [protected]`

Referenced by `done()`.

7.66.4.3 `int* spot::loopless_modular_mixed_radix_gray_code::f_ [protected]`

Referenced by `last()`.

7.66.4.4 int* spot::loopless_modular_mixed_radix_gray_code::m_ [protected]

7.66.4.5 int spot::loopless_modular_mixed_radix_gray_code::n_ [protected]

Referenced by last().

7.66.4.6 int* spot::loopless_modular_mixed_radix_gray_code::non_one_radixes_ [protected]

7.66.4.7 int* spot::loopless_modular_mixed_radix_gray_code::s_ [protected]

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

- misc/[modgray.hh](#)

7.67 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](#)

7.67.1 Detailed Description

Read LTL formulae from a file, one by one.

7.67.2 Constructor & Destructor Documentation

7.67.2.1 spot::ltl::ltl_file::ltl_file (const std::string & filename)

7.67.2.2 spot::ltl::ltl_file (const char * filename)

7.67.3 Member Function Documentation

7.67.3.1 formula* spot::ltl::ltl_file::next ()

Return the next parsed LTL formula, and 0 at end of file.

7.67.4 Member Data Documentation

7.67.4.1 std::ifstream spot::ltl::ltl_file::in [private]

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

- [ltlparse/ltlfile.hh](#)

7.68 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_

7.68.1 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}
/// }
///

```

7.68.2 Constructor & Destructor Documentation

7.68.2.1 spot::minato_isop::minato_isop (bdd input)

Constructor.

- input The BDD function to translate in ISOP.

7.68.2.2 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*.

7.68.3 Member Function Documentation

7.68.3.1 bdd spot::minato_isop::next ()

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

7.68.4 Member Data Documentation

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

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

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

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

- [misc/minato.hh](#)

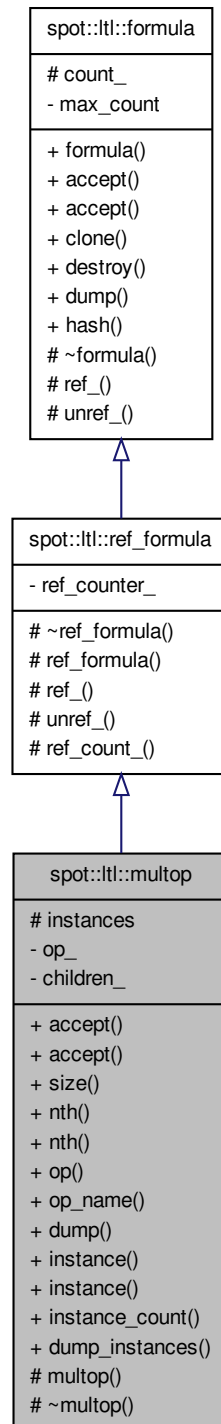
7.69 spot::ltl::multop Class Reference

Multi-operand operators.

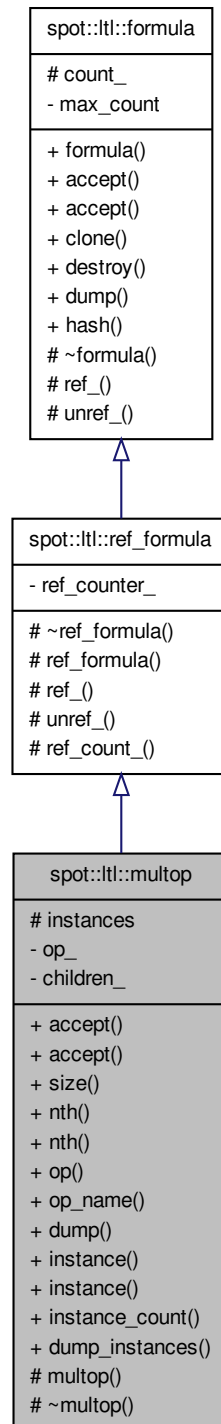
These operators are considered commutative and associative.

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

Inheritance diagram for spot::ltl::multop:



Collaboration diagram for spot::ltl::multop:



Classes

- struct [pairecmp](#)
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_`

7.69.1 Detailed Description

Multi-operand operators.

These operators are considered commutative and associative.

7.69.2 Member Typedef Documentation

7.69.2.1 `typedef std::map<pair, multop*, paircmp> spot::ltl::multop::map` `[protected]`

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

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

List of formulae.

7.69.3 Member Enumeration Documentation

7.69.3.1 `enum spot::ltl::multop::type`

Enumerator:

Or

And

7.69.4 Constructor & Destructor Documentation

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

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

7.69.5 Member Function Documentation

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

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

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

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

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

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

7.69.5.3 formula* spot::ltl::formula::clone () const [inherited]

clone this node

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

7.69.5.4 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().

7.69.5.5 virtual std::string spot::ltl::multop::dump () const [virtual]

Return a canonic representation of the atomic proposition.

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

7.69.5.6 static std::ostream& spot::ltl::multop::dump_instances (std::ostream & os) [static]

Dump all instances. For debugging.

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

Return a hash key for the formula.

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

Referenced by spot::ltl::formula_ptr_hash::operator()(), and spot::ltl::formula_ptr_less_than::operator()().

7.69.5.8 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.

7.69.5.9 `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.

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

Number of instantiated multi-operand operators. For debugging.

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

Get the nth children.

Starting with $n = 0$.

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

Get the nth children.

Starting with $n = 0$.

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

Get the type of this operator.

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

Get the type of this operator, as a string.

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

increment reference counter if any

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

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

Number of references to this formula.

7.69.5.17 unsigned spot::ltl::multop::size () const

Get the number of children.

7.69.5.18 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](#).

7.69.6 Member Data Documentation**7.69.6.1 vec* spot::ltl::multop::children_ [private]****7.69.6.2 size_t spot::ltl::formula::count_ [protected, inherited]**

The hash key of this formula.

Referenced by [spot::ltl::formula::hash\(\)](#).

7.69.6.3 map spot::ltl::multop::instances [static, protected]**7.69.6.4 type spot::ltl::multop::op_ [private]**

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

- [ltlast/multop.hh](#)

7.70 spot::ltl::nfa Class Reference

Nondeterministic Finite Automata used by automata operators.

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

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_](#)

7.70.1 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.

7.70.2 Member Typedef Documentation

7.70.2.1 typedef Sgi::hash_map<int, state*, Sgi::hash<int> > spot::ltl::nfa::is_map
[private]

7.70.2.2 typedef succ_iterator spot::ltl::nfa::iterator

Iterator over the successors of a state.

7.70.2.3 typedef boost::shared_ptr<formula_tree::node> spot::ltl::nfa::label

7.70.2.4 typedef boost::shared_ptr<nfa> spot::ltl::nfa::ptr

7.70.2.5 `typedef Sgi::hash_map<const state*, int, ptr_hash<state> > spot::ltl::nfa::si_map`
`[private]`

7.70.2.6 `typedef std::list<transition*> spot::ltl::nfa::state`

7.70.3 Constructor & Destructor Documentation

7.70.3.1 `spot::ltl::nfa::nfa ()`

7.70.3.2 `spot::ltl::nfa::~~nfa ()`

7.70.3.3 `spot::ltl::nfa::nfa (const nfa & other) [private]`

Explicitly disallow use of implicitly generated member functions we don't want.

7.70.4 Member Function Documentation

7.70.4.1 `state* spot::ltl::nfa::add_state (int name) [private]`

7.70.4.2 `void spot::ltl::nfa::add_transition (int src, int dst, const label lbl)`

7.70.4.3 `unsigned spot::ltl::nfa::arity ()`

Get the 'arity' i.e. max t.cost, for each transition t.

7.70.4.4 `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);
```

7.70.4.5 iterator spot::ltl::nfa::end (const state * s) const

Return an iterator just past the last succesor of *state*.

7.70.4.6 int spot::ltl::nfa::format_state (const state * s) const**7.70.4.7 const state* spot::ltl::nfa::get_init_state ()**

Get the initial state of the NFA.

7.70.4.8 const std::string& spot::ltl::nfa::get_name () const**7.70.4.9 bool spot::ltl::nfa::is_final (const state * s)**

Tell whether the given state is final or not.

7.70.4.10 bool spot::ltl::nfa::is_loop ()

Tell whether the NFA is 'loop', i.e. without any final state.

7.70.4.11 nfa& spot::ltl::nfa::operator= (const nfa & other) [private]**7.70.4.12 void spot::ltl::nfa::set_final (int name)****7.70.4.13 void spot::ltl::nfa::set_init_state (int name)****7.70.4.14 void spot::ltl::nfa::set_name (const std::string &)**

7.70.5 Member Data Documentation

7.70.5.1 `size_t spot::ltl::nfa::arity_` [private]

7.70.5.2 `std::set<int> spot::ltl::nfa::finals_` [private]

7.70.5.3 `state* spot::ltl::nfa::init_` [private]

7.70.5.4 `is_map spot::ltl::nfa::is_` [private]

7.70.5.5 `std::string spot::ltl::nfa::name_` [private]

7.70.5.6 `si_map spot::ltl::nfa::si_` [private]

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

- [ltlast/nfa.hh](#)

7.71 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_](#)

7.71.1 Detailed Description

An exception used to forward NIPS errors.

7.71.2 Constructor & Destructor Documentation

7.71.2.1 `spot::nips_exception::nips_exception (const std::string & where, int err) [inline]`

7.71.2.2 `spot::nips_exception::nips_exception (const std::string & where) [inline]`

7.71.3 Member Function Documentation

7.71.3.1 `int spot::nips_exception::get_err () const [inline]`

References `err_`.

7.71.3.2 `int spot::nips_exception::get_err_defined () const [inline]`

References `err_defined_`.

7.71.3.3 `std::string spot::nips_exception::get_where () const [inline]`

References `where_`.

7.71.4 Member Data Documentation

7.71.4.1 `int spot::nips_exception::err_ [private]`

Referenced by `get_err()`.

7.71.4.2 `bool spot::nips_exception::err_defined_ [private]`

Referenced by `get_err_defined()`.

7.71.4.3 std::string spot::nips_exception::where_ [private]

Referenced by get_where().

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

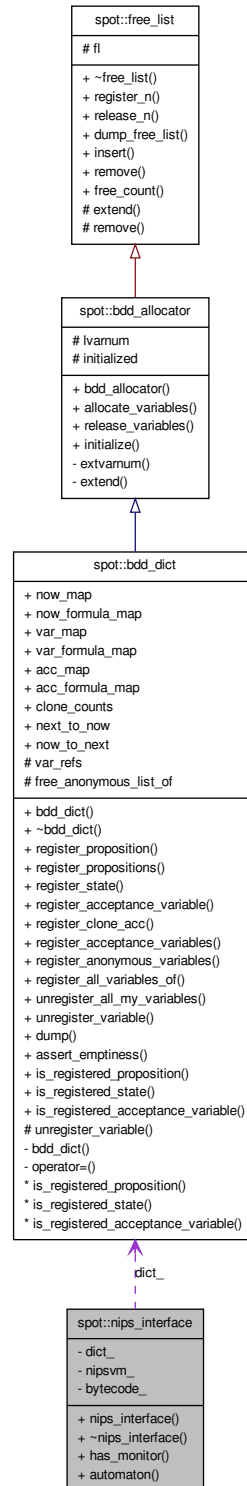
- [nips/common.hh](#)

7.72 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_](#)

7.72.1 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 provied 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).

7.72.2 Constructor & Destructor Documentation

7.72.2.1 `spot::nips_interface::nips_interface (bdd_dict * dict, const std::string & filename)`

7.72.2.2 `spot::nips_interface::~~nips_interface ()`

7.72.3 Member Function Documentation

7.72.3.1 `tgba* spot::nips_interface::automaton ()`

7.72.3.2 `bool spot::nips_interface::has_monitor () const`

7.72.4 Member Data Documentation

7.72.4.1 `nipsvm_bytecode_t* spot::nips_interface::bytecode_ [private]`

7.72.4.2 bdd_dict* spot::nips_interface::dict_ [private]

7.72.4.3 nipsvm_t* spot::nips_interface::nipsvm_ [private]

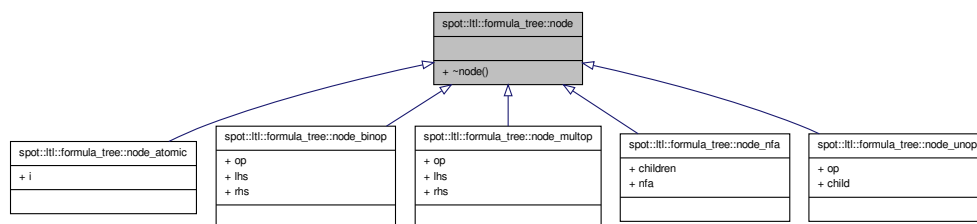
The documentation for this class was generated from the following file:

- [nips/nips.hh](#)

7.73 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](#) ()

7.73.1 Constructor & Destructor Documentation

7.73.1.1 virtual spot::ltl::formula_tree::node::~~node () [inline, virtual]

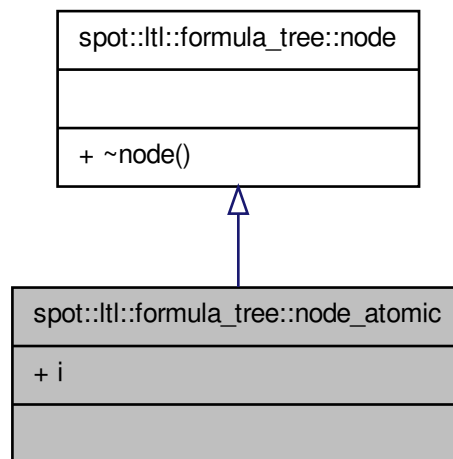
The documentation for this struct was generated from the following file:

- [ltlast/formula_tree.hh](#)

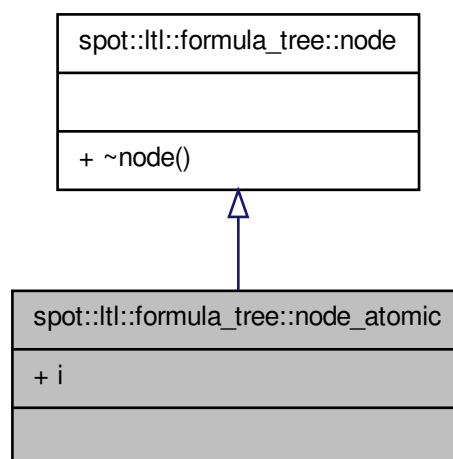
7.74 spot::ltl::formula_tree::node_atomic Struct Reference

```
#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`

7.74.1 Member Data Documentation

7.74.1.1 int spot::ltl::formula_tree::node_atomic::i

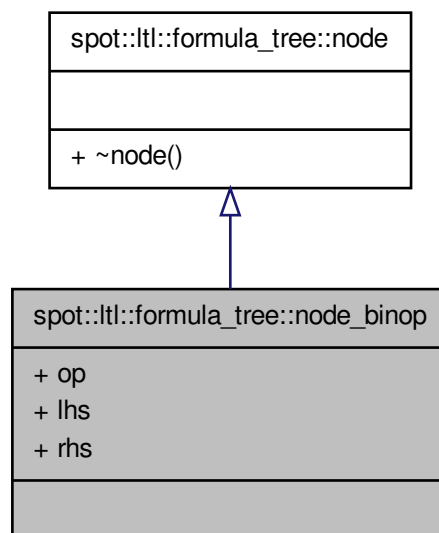
The documentation for this struct was generated from the following file:

- [ltlast/formula_tree.hh](#)

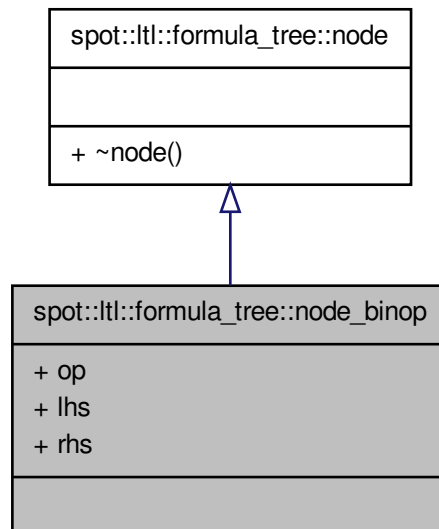
7.75 spot::ltl::formula_tree::node_binop Struct Reference

```
#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](#)

7.75.1 Member Data Documentation

7.75.1.1 `node_ptr` `spot::ltl::formula_tree::node_binop::lhs`

7.75.1.2 `binop::type` `spot::ltl::formula_tree::node_binop::op`

7.75.1.3 `node_ptr` `spot::ltl::formula_tree::node_binop::rhs`

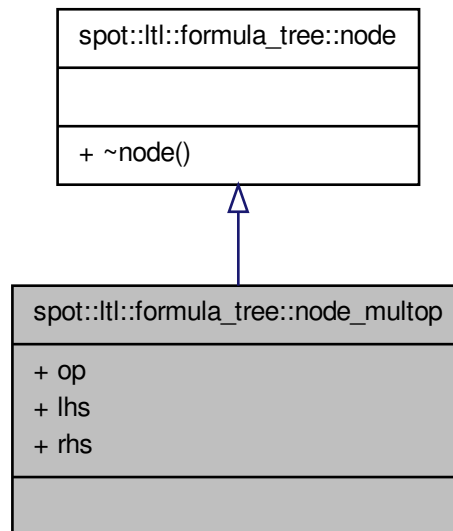
The documentation for this struct was generated from the following file:

- [ltlast/formula_tree.hh](#)

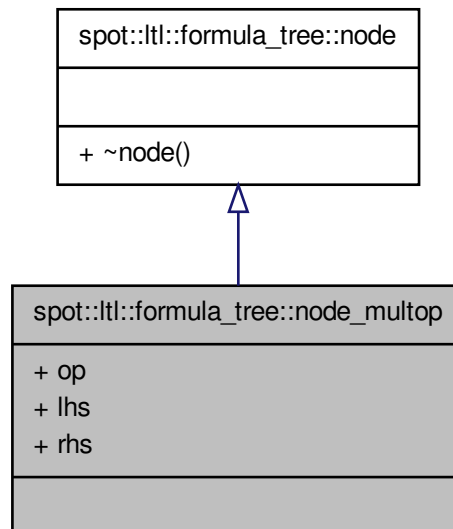
7.76 spot::ltl::formula_tree::node_multop Struct Reference

```
#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](#)

7.76.1 Member Data Documentation

7.76.1.1 `node_ptr` `spot::ltl::formula_tree::node_multop::lhs`

7.76.1.2 `multop::type` `spot::ltl::formula_tree::node_multop::op`

7.76.1.3 `node_ptr` `spot::ltl::formula_tree::node_multop::rhs`

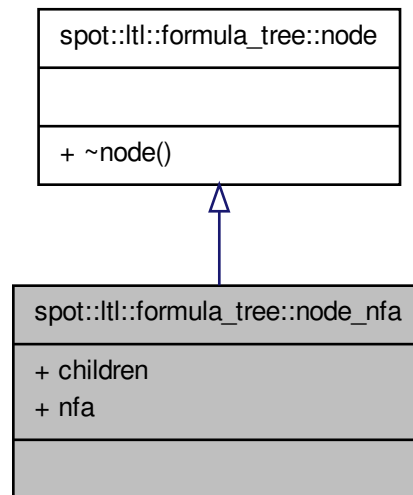
The documentation for this struct was generated from the following file:

- [ltlast/formula_tree.hh](#)

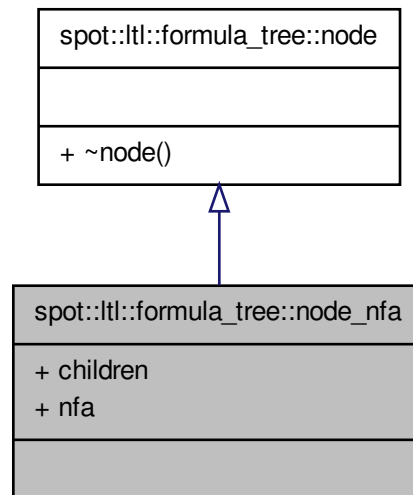
7.77 spot::ltl::formula_tree::node_nfa Struct Reference

```
#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`

7.77.1 Member Data Documentation

7.77.1.1 `std::vector<node_ptr> spot::ltl::formula_tree::node_nfa::children`

7.77.1.2 `spot::ltl::nfa::ptr spot::ltl::formula_tree::node_nfa::nfa`

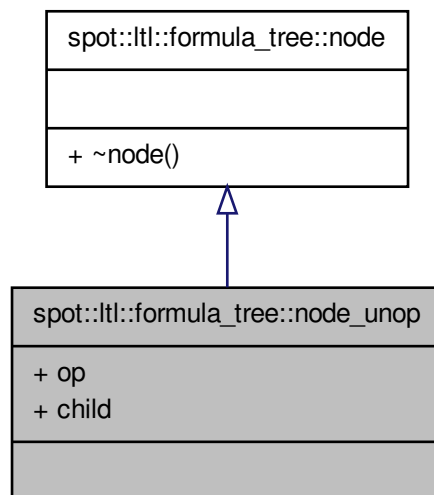
The documentation for this struct was generated from the following file:

- `ltlast/formula_tree.hh`

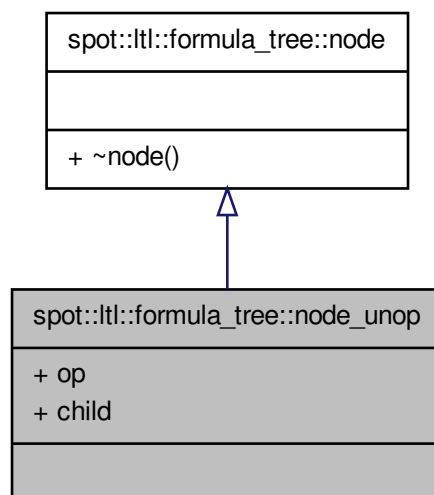
7.78 spot::ltl::formula_tree::node_unop Struct Reference

```
#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](#)

7.78.1 Member Data Documentation

7.78.1.1 node_ptr spot::ltl::formula_tree::node_unop::child

7.78.1.2 unop::type spot::ltl::formula_tree::node_unop::op

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

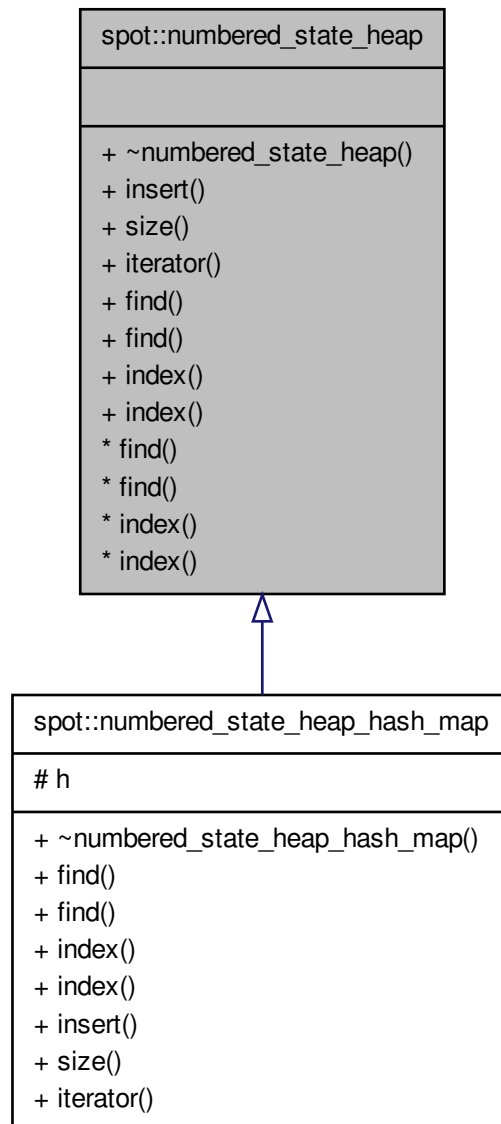
- [ltlast/formula_tree.hh](#)

7.79 spot::numbered_state_heap Class Reference

Keep track of a large quantity of indexed states.

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

Inheritance diagram for spot::numbered_state_heap:



Public Types

- typedef std::pair< const [state](#) *, int * > [state_index_p](#)
- typedef std::pair< const [state](#) *, int > [state_index](#)

Public Member Functions

- virtual `~numbered_state_heap()`
- virtual void `insert` (const `state` *s, int index)=0
Add a new state s with index index.
- virtual int `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

7.79.1 Detailed Description

Keep track of a large quantity of indexed states.

7.79.2 Member Typedef Documentation

7.79.2.1 `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index`

7.79.2.2 `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p`

7.79.3 Constructor & Destructor Documentation

7.79.3.1 `virtual spot::numbered_state_heap::~~numbered_state_heap()` [`inline`, `virtual`]

7.79.4 Member Function Documentation

7.79.4.1 `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](#).

7.79.4.2 `virtual state_index_p spot::numbered_state_heap::find (const state * s) [pure virtual]`

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

7.79.4.3 `virtual state_index_p spot::numbered_state_heap::index (const state * s) [pure virtual]`

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

7.79.4.4 `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](#).

7.79.4.5 `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](#).

7.79.4.6 `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](#).

7.79.4.7 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](#)

7.80 spot::numbered_state_heap_const_iterator Class Reference

Iterator on [numbered_state_heap](#) objects.

```
#include <tgbaalgos/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

7.80.1 Detailed Description

Iterator on [numbered_state_heap](#) objects.

7.80.2 Constructor & Destructor Documentation

7.80.2.1 virtual spot::numbered_state_heap_const_iterator::~~numbered_state_heap_const_iterator () [inline, virtual]

7.80.3 Member Function Documentation

7.80.3.1 virtual bool spot::numbered_state_heap_const_iterator::done () const [pure virtual]

7.80.3.2 virtual void spot::numbered_state_heap_const_iterator::first () [pure virtual]

Iteration.

7.80.3.3 virtual int spot::numbered_state_heap_const_iterator::get_index () const [pure virtual]

7.80.3.4 virtual const state* spot::numbered_state_heap_const_iterator::get_state () const [pure virtual]

Inspection.

7.80.3.5 virtual void spot::numbered_state_heap_const_iterator::next () [pure virtual]

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

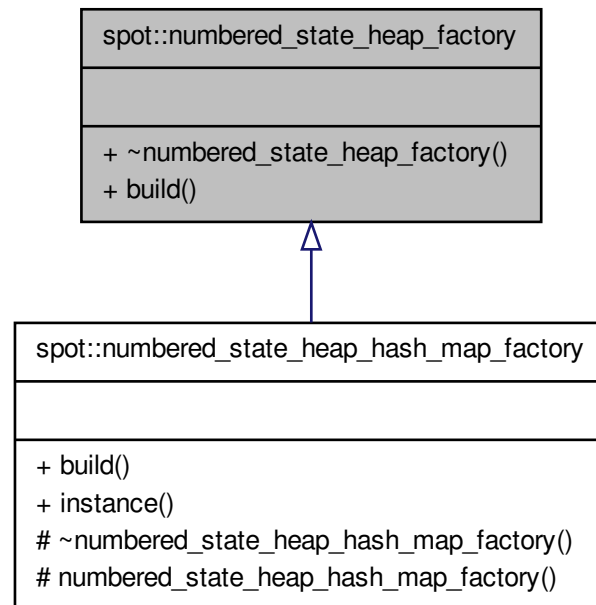
- tgbaalgos/gtec/[nsheap.hh](#)

7.81 spot::numbered_state_heap_factory Class Reference

Abstract factory for [numbered_state_heap](#).

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

Inheritance diagram for spot::numbered_state_heap_factory:



Public Member Functions

- virtual [~numbered_state_heap_factory](#) ()
- virtual [numbered_state_heap](#) * [build](#) () const =0

7.81.1 Detailed Description

Abstract factory for [numbered_state_heap](#).

7.81.2 Constructor & Destructor Documentation

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

7.81.3 Member Function Documentation

7.81.3.1 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:

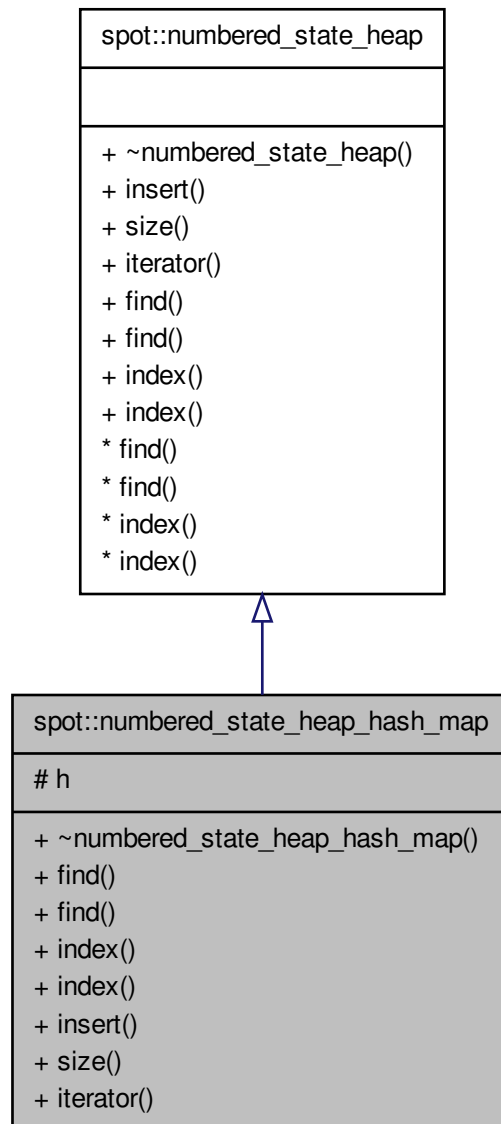
- [tgbaalgos/gtec/nsheap.hh](#)

7.82 spot::numbered_state_heap_hash_map Class Reference

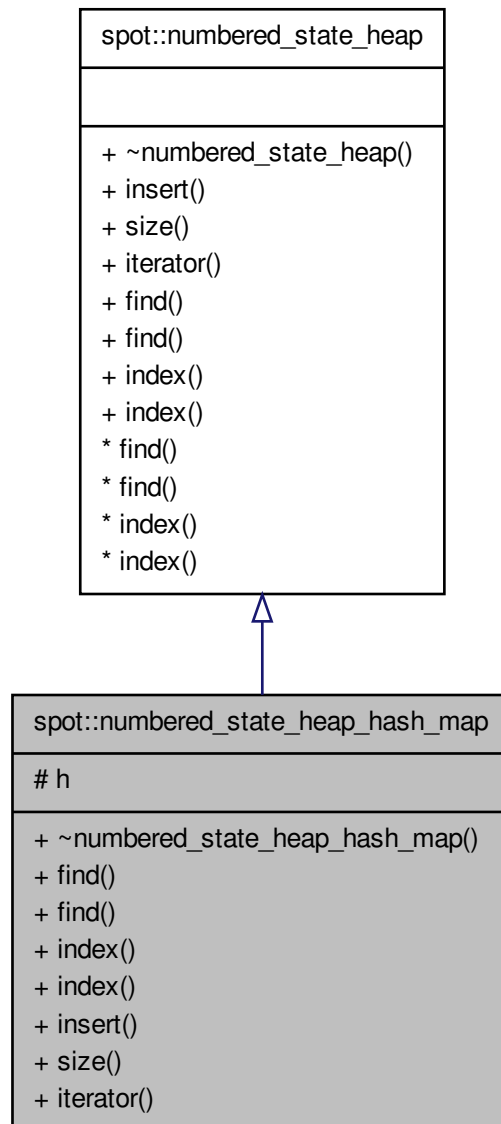
A straightforward implementation of [numbered_state_heap](#) with a hash map.

```
#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.

7.82.1 Detailed Description

A straightforward implementation of [numbered_state_heap](#) with a hash map.

7.82.2 Member Typedef Documentation

7.82.2.1 `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal>
spot::numbered_state_heap_hash_map::hash_type`

7.82.2.2 `typedef std::pair<const state*, int> spot::numbered_state_heap::state_index
[inherited]`

7.82.2.3 `typedef std::pair<const state*, int*> spot::numbered_state_heap::state_index_p
[inherited]`

7.82.3 Constructor & Destructor Documentation

7.82.3.1 virtual `spot::numbered_state_heap_hash_map::~~numbered_state_heap_hash_map ()`
[**virtual**]

7.82.4 Member Function Documentation

7.82.4.1 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](#).

7.82.4.2 virtual `state_index_p spot::numbered_state_heap_hash_map::find (const state * s)`
[**virtual**]

Implements [spot::numbered_state_heap](#).

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

Implements [spot::numbered_state_heap](#).

7.82.4.4 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](#).

7.82.4.5 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](#).

7.82.4.6 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](#).

7.82.4.7 virtual int spot::numbered_state_heap_hash_map::size () const [virtual]

The number of stored states.

Implements [spot::numbered_state_heap](#).

7.82.5 Member Data Documentation

7.82.5.1 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:

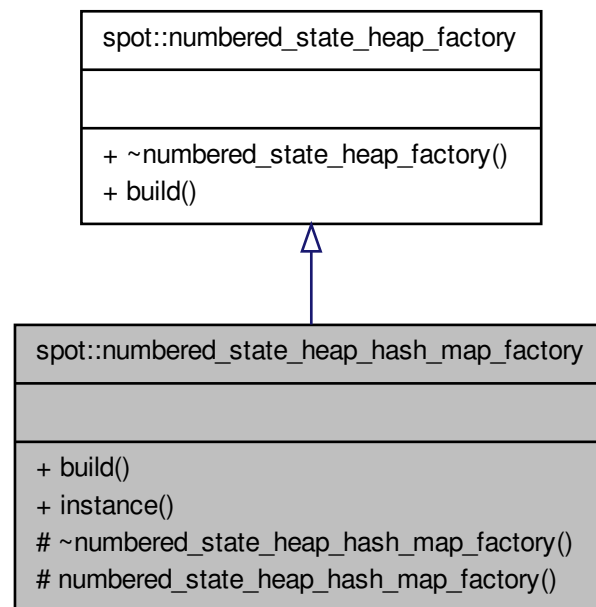
- [tgbaalgos/gtec/nsheap.hh](#)

7.83 spot::numbered_state_heap_hash_map_factory Class Reference

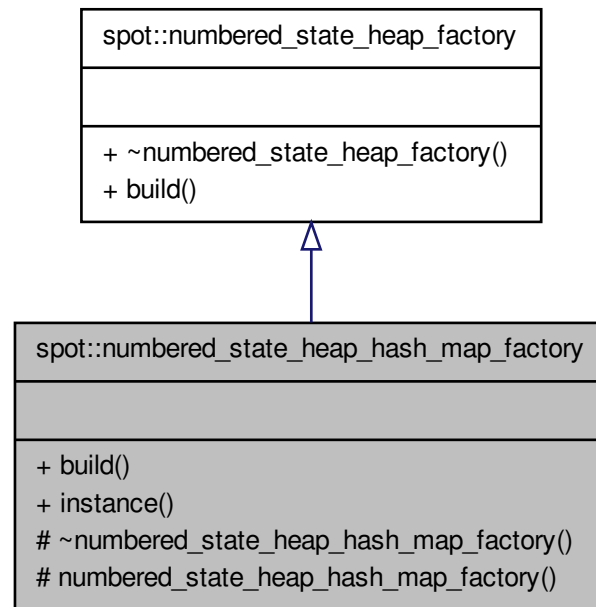
Factory for [numbered_state_heap_hash_map](#).

```
#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 ()`

7.83.1 Detailed Description

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

7.83.2 Constructor & Destructor Documentation

7.83.2.1 `virtual spot::numbered_state_heap_hash_map_factory::~~numbered_state_heap_hash_map_factory () [inline, protected, virtual]`

7.83.2.2 `spot::numbered_state_heap_hash_map_factory::numbered_state_heap_hash_map_factory () [protected]`

7.83.3 Member Function Documentation

7.83.3.1 `virtual numbered_state_heap_hash_map* spot::numbered_state_heap_hash_map_factory::build () const [virtual]`

Implements [spot::numbered_state_heap_factory](#).

7.83.3.2 `static const numbered_state_heap_hash_map_factory* spot::numbered_state_heap_hash_map_factory::instance () [static]`

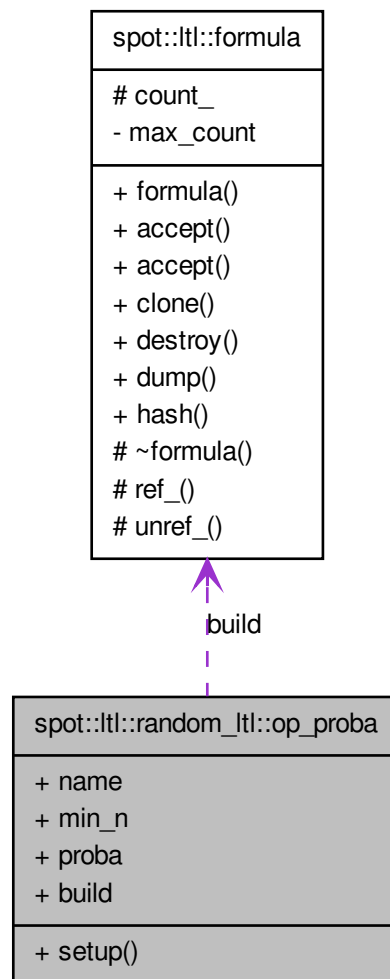
Get the unique instance of this class.

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

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

7.84 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](#)

7.84.1 Member Typedef Documentation

7.84.1.1 typedef formula*(* spot::ltl::random_ltl::op_proba::builder)(const random_ltl *rl, int n)

7.84.2 Member Function Documentation

7.84.2.1 void spot::ltl::random_ltl::op_proba::setup (const char * *name*, int *min_n*, builder *build*)

7.84.3 Member Data Documentation

7.84.3.1 builder spot::ltl::random_ltl::op_proba::build

7.84.3.2 int spot::ltl::random_ltl::op_proba::min_n

7.84.3.3 const char* spot::ltl::random_ltl::op_proba::name

7.84.3.4 double spot::ltl::random_ltl::op_proba::proba

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

- ltlvisit/[randomltl.hh](#)

7.85 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.

7.85.1 Detailed Description

Manage a map of options.

Each option is defined by a string and is associated to an integer value.

7.85.2 Member Function Documentation

7.85.2.1 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\[\]\(\)](#)

7.85.2.2 int& spot::option_map::operator[] (const char * *option*)

Get a reference to the current value of *option*.

7.85.2.3 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\(\)](#)

7.85.2.4 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.

7.85.2.5 void spot::option_map::set (const option_map & *o*)

Acquire all the settings of *o*.

7.85.2.6 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.

7.85.3 Friends And Related Function Documentation

7.85.3.1 std::ostream& operator<< (std::ostream & *os*, const option_map & *m*) [friend]

Print the [option_map](#) *m*.

7.85.4 Member Data Documentation

7.85.4.1 std::map<std::string, int> spot::option_map::options_ [private]

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

- [misc/optionmap.hh](#)

7.86 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

7.86.1 Detailed Description

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

7.86.2 Member Function Documentation

7.86.2.1 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:

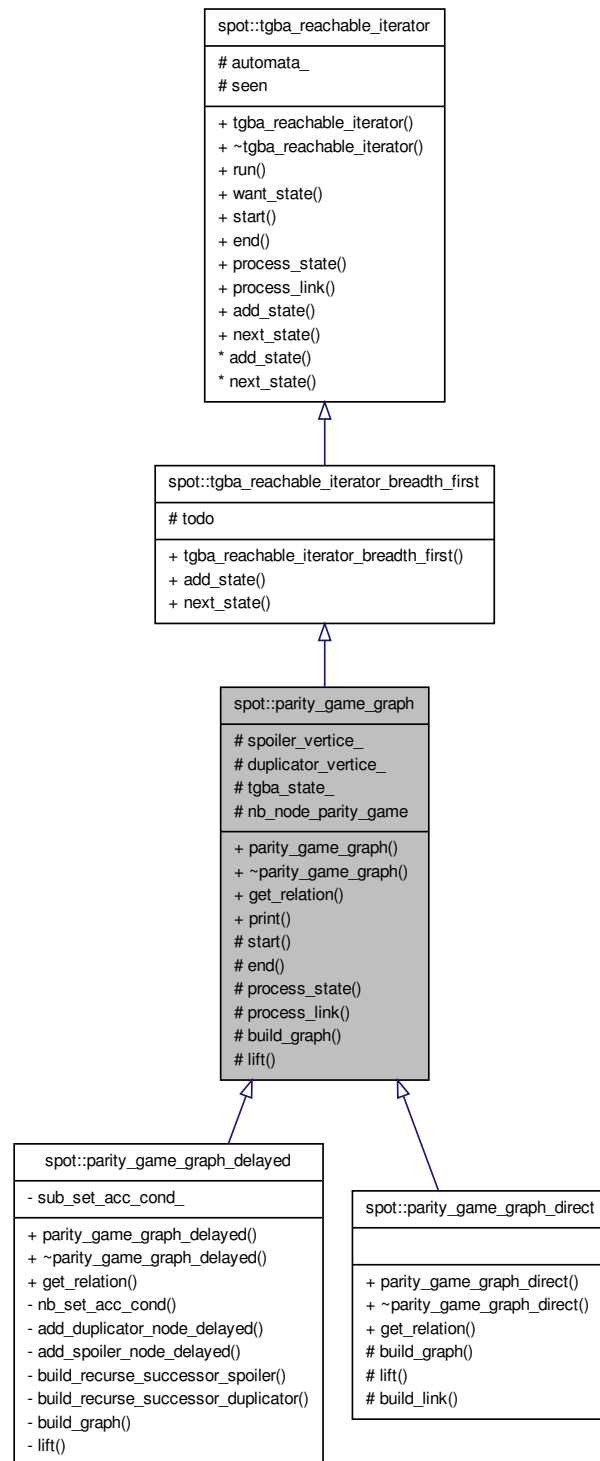
- [ltlast/multop.hh](#)

7.87 spot::parity_game_graph Class Reference

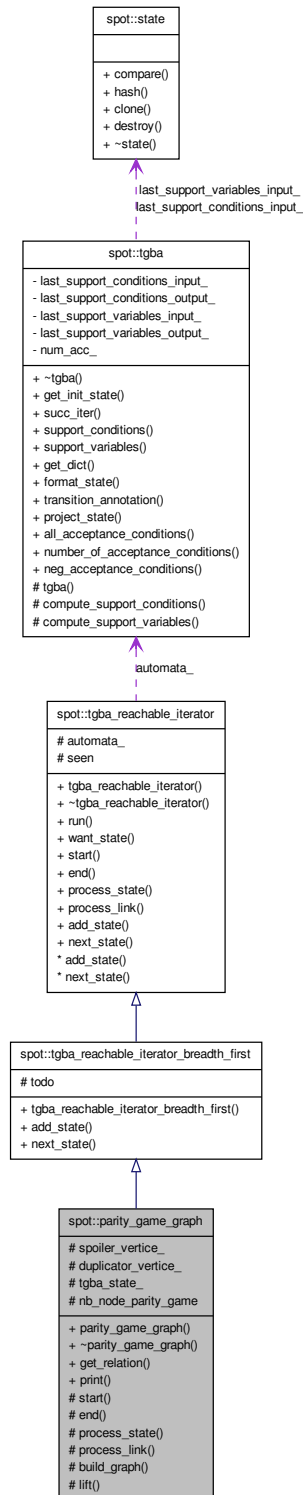
Parity game graph which compute a simulation relation.

```
#include <tgbalgorithms/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.

7.87.1 Detailed Description

Parity game graph which compute a simulation relation.

7.87.2 Member Typedef Documentation

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

7.87.3 Constructor & Destructor Documentation

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

7.87.3.2 `virtual spot::parity_game_graph::~~parity_game_graph () [virtual]`

7.87.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

7.87.4.2 `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](#).

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

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.87.4.4 virtual simulation_relation* spot::parity_game_graph::get_relation () [pure virtual]

Implemented in [spot::parity_game_graph_direct](#), and [spot::parity_game_graph_delayed](#).

7.87.4.5 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](#).

7.87.4.6 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](#).

7.87.4.7 void spot::parity_game_graph::print (std::ostream & os)

7.87.4.8 void spot::parity_game_graph::process_link (int in, int out, const tgba_succ_iterator * si) [protected]

7.87.4.9 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.

7.87.4.10 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](#).

7.87.4.11 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.

7.87.4.12 void spot::parity_game_graph::start () [protected, virtual]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.87.4.13 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.

7.87.5 Member Data Documentation

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

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

7.87.5.2 dn_v spot::parity_game_graph::duplicator_vertice_ [protected]

7.87.5.3 int spot::parity_game_graph::nb_node_parity_game [protected]

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

States already seen.

7.87.5.5 sn_v spot::parity_game_graph::spoiler_vertice_ [protected]

7.87.5.6 s_v spot::parity_game_graph::tgba_state_ [protected]

7.87.5.7 std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo [protected, inherited]

A queue of states yet to explore.

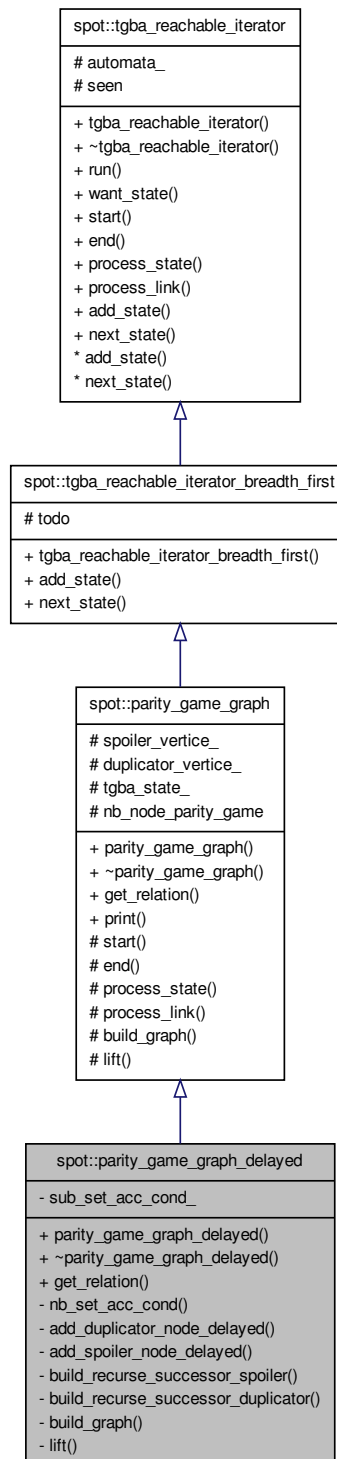
The documentation for this class was generated from the following file:

- tgbaalgos/[reductgba_sim.hh](#)

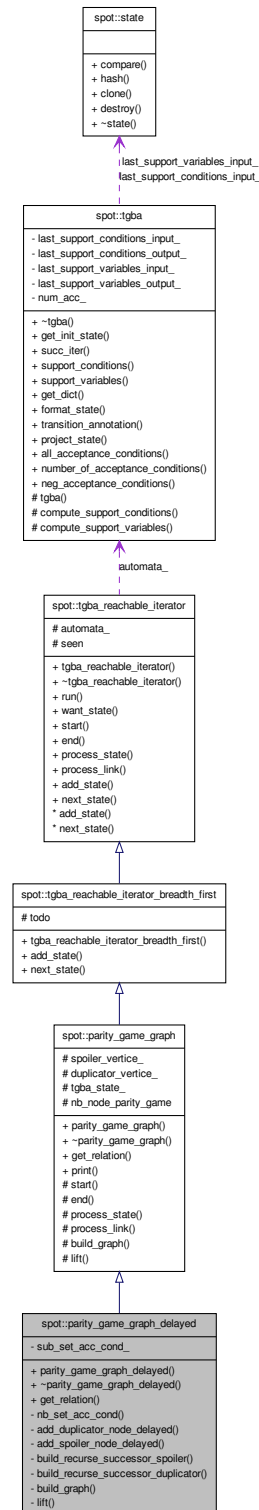
7.88 spot::parity_game_graph_delayed Class Reference

```
#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](#)

7.88.1 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}
/// }
```

7.88.2 Member Typedef Documentation

7.88.2.1 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.

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

7.88.3 Constructor & Destructor Documentation

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

7.88.3.2 `spot::parity_game_graph_delayed::~~parity_game_graph_delayed ()`

7.88.4 Member Function Documentation

7.88.4.1 `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]`

7.88.4.2 `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]`

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

Implements [spot::tgba_reachable_iterator](#).

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

Compute the couple as for direct simulation,.

Implements [spot::parity_game_graph](#).

7.88.4.5 `void spot::parity_game_graph_delayed::build_recurse_successor_duplicator (duplicator_node * dn, spoiler_node * sn, std::ostream & os)` `[private]`

7.88.4.6 void spot::parity_game_graph_delayed::build_recurse_successor_spoiler (spoiler_node * *sn*, std::ostream & *os*) [private]

7.88.4.7 void spot::parity_game_graph::end () [protected, virtual, inherited]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.88.4.8 virtual delayed_simulation_relation* spot::parity_game_graph_delayed::get_relation () [virtual]

Implements [spot::parity_game_graph](#).

7.88.4.9 virtual void spot::parity_game_graph_delayed::lift () [private, virtual]

The Jurdzinski's lifting algorithm.

Implements [spot::parity_game_graph](#).

7.88.4.10 int spot::parity_game_graph_delayed::nb_set_acc_cond () [private]

Return the number of acceptance condition.

7.88.4.11 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](#).

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

7.88.4.13 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.

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

7.88.4.15 `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](#).

7.88.4.16 `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.

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

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.88.4.18 `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.

7.88.5 Member Data Documentation

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

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

7.88.5.2 `dn_v spot::parity_game_graph::duplicator_vertice_` `[protected, inherited]`

7.88.5.3 `int spot::parity_game_graph::nb_node_parity_game` `[protected, inherited]`

7.88.5.4 `seen_map spot::tgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

7.88.5.5 `sn_v spot::parity_game_graph::spoiler_vertice_` `[protected, inherited]`

7.88.5.6 `bdd_v spot::parity_game_graph_delayed::sub_set_acc_cond_` `[private]`

7.88.5.7 `s_v spot::parity_game_graph::tgba_state_` `[protected, inherited]`

7.88.5.8 `std::deque<const state*> spot::tgba_reachable_iterator_breadth_first::todo`
`[protected, inherited]`

A queue of states yet to explore.

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

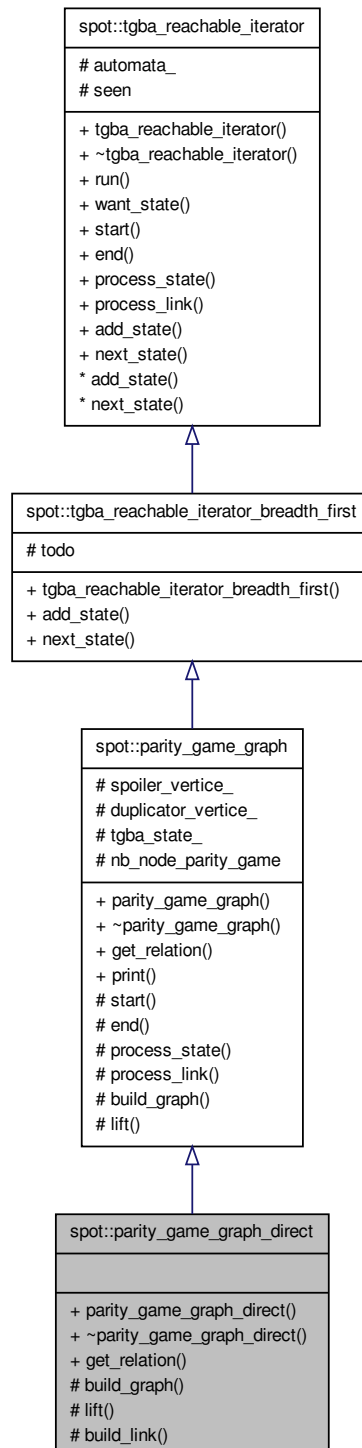
- [tgbaalgos/reductgba_sim.hh](#)

7.89 spot::parity_game_graph_direct Class Reference

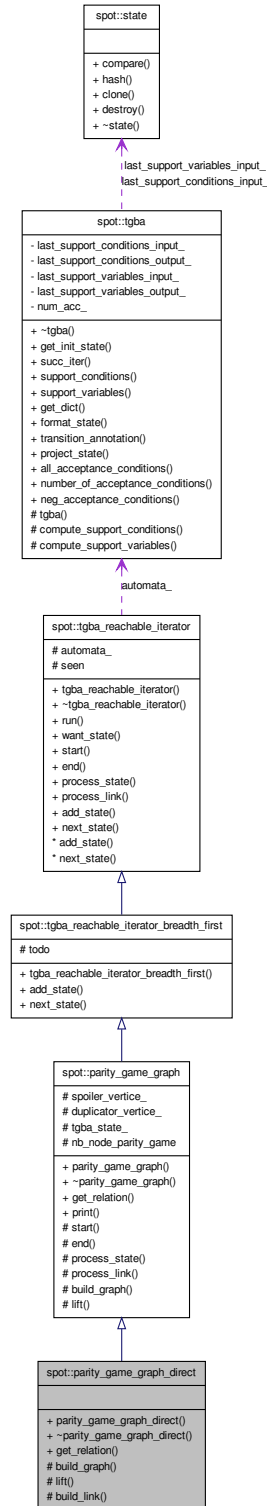
Parity game graph which compute the direct simulation relation.

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

Inheritance diagram for spot::parity_game_graph_direct:



Collaboration diagram for spot::parity_game_graph_direct:



Public Member Functions

- `parity_game_graph_direct` (const `tgba` *a)
- `~parity_game_graph_direct` ()
- virtual `direct_simulation_relation` * `get_relation` ()
- void `print` (std::ostream &os)
- virtual void `process_link` (const `state` *in_s, int in, const `state` *out_s, int out, const `tgba_succ_iterator` *si)
- virtual void `add_state` (const `state` *s)
- virtual const `state` * `next_state` ()
Called by `run()` to obtain the 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.

7.89.1 Detailed Description

Parity game graph which compute the direct simulation relation.

7.89.2 Member Typedef Documentation

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

7.89.3 Constructor & Destructor Documentation

7.89.3.1 `spot::parity_game_graph_direct::parity_game_graph_direct (const tgba * a)`

7.89.3.2 `spot::parity_game_graph_direct::~~parity_game_graph_direct ()`

7.89.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

7.89.4.2 `virtual void spot::parity_game_graph_direct::build_graph () [protected,
virtual]`

Compute each node of the graph.

Implements [spot::parity_game_graph](#).

7.89.4.3 `void spot::parity_game_graph_direct::build_link () [protected]`

7.89.4.4 void spot::parity_game_graph::end () [protected, virtual, inherited]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.89.4.5 virtual direct_simulation_relation* spot::parity_game_graph_direct::get_relation () [virtual]

Implements [spot::parity_game_graph](#).

7.89.4.6 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](#).

7.89.4.7 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](#).

7.89.4.8 void spot::parity_game_graph::print (std::ostream & os) [inherited]**7.89.4.9 void spot::parity_game_graph::process_link (int in, int out, const tgba_succ_iterator * si) [protected, inherited]****7.89.4.10 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.

7.89.4.11 `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](#).

7.89.4.12 `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.

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

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.89.4.14 `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.

7.89.5 Member Data Documentation

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

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

7.89.5.2 `dn_v spot::parity_game_graph::duplicator_vertice_` [protected, inherited]

7.89.5.3 `int spot::parity_game_graph::nb_node_parity_game` [protected, inherited]

7.89.5.4 `seen_map spot::tgba_reachable_iterator::seen` [protected, inherited]

States already seen.

7.89.5.5 `sn_v spot::parity_game_graph::spoiler_vertice_` [protected, inherited]

7.89.5.6 `s_v spot::parity_game_graph::tgba_state_` [protected, inherited]

7.89.5.7 `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](#)

7.90 neverclaimyy::position Class Reference

Abstract a position.

```
#include <neverparse/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.

7.90.1 Detailed Description

Abstract a position.

7.90.2 Constructor & Destructor Documentation

7.90.2.1 neverclaimyy::position::position () [inline]

Construct a position.

7.90.3 Member Function Documentation

7.90.3.1 void neverclaimyy::position::columns (int count = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by neverclaimyy::operator+=().

7.90.3.2 void neverclaimyy::position::initialize (std::string * fn) [inline]

Initialization.

References column, filename, and line.

Referenced by neverclaimyy::location::step().

7.90.3.3 void neverclaimyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References column, and line.

7.90.4 Member Data Documentation**7.90.4.1 unsigned int neverclaimyy::position::column**

Current column number.

Referenced by columns(), initialize(), lines(), neverclaimyy::operator<<(), and neverclaimyy::operator==().

7.90.4.2 std::string* neverclaimyy::position::filename

File name to which this position refers.

Referenced by initialize(), neverclaimyy::operator<<(), and neverclaimyy::operator==().

7.90.4.3 unsigned int neverclaimyy::position::line

Current line number.

Referenced by initialize(), lines(), neverclaimyy::operator<<(), and neverclaimyy::operator==().

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

- [neverparse/position.hh](#)

7.91 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.

7.91.1 Detailed Description

Abstract a position.

7.91.2 Constructor & Destructor Documentation

7.91.2.1 Itlyy::position::position () [inline]

Construct a position.

7.91.3 Member Function Documentation

7.91.3.1 void Itlyy::position::columns (int count = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by Itlyy::operator+=().

7.91.3.2 void Itlyy::position::initialize (std::string * fn) [inline]

Initialization.

References column, filename, and line.

Referenced by Itlyy::location::step().

7.91.3.3 `void ltlyy::position::lines (int count = 1) [inline]`

(line related) Advance to the COUNT next lines.

References column, and line.

7.91.4 Member Data Documentation

7.91.4.1 `unsigned int ltlyy::position::column`

Current column number.

Referenced by `columns()`, `initialize()`, `lines()`, `ltlyy::operator<<()`, and `ltlyy::operator==()`.

7.91.4.2 `std::string* ltlyy::position::filename`

File name to which this position refers.

Referenced by `initialize()`, `ltlyy::operator<<()`, and `ltlyy::operator==()`.

7.91.4.3 `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](#)

7.92 `eltlyy::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.

7.92.1 Detailed Description

Abstract a position.

7.92.2 Constructor & Destructor Documentation

7.92.2.1 eltly::position::position () [inline]

Construct a position.

7.92.3 Member Function Documentation

7.92.3.1 void eltly::position::columns (int count = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by eltly::operator+=().

7.92.3.2 void eltly::position::initialize (std::string * fn) [inline]

Initialization.

References column, filename, and line.

Referenced by eltly::location::step().

7.92.3.3 void eltlyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References column, and line.

7.92.4 Member Data Documentation

7.92.4.1 unsigned int eltlyy::position::column

Current column number.

Referenced by columns(), initialize(), lines(), eltlyy::operator<<(), and eltlyy::operator==().

7.92.4.2 std::string* eltlyy::position::filename

File name to which this position refers.

Referenced by initialize(), eltlyy::operator<<(), and eltlyy::operator==().

7.92.4.3 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](#)

7.93 sautyy::position Class Reference

Abstract a position.

```
#include <sautparse/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.

7.93.1 Detailed Description

Abstract a position.

7.93.2 Constructor & Destructor Documentation

7.93.2.1 sauty::position::position () [inline]

Construct a position.

7.93.3 Member Function Documentation

7.93.3.1 void sauty::position::columns (int count = 1) [inline]

(column related) Advance to the COUNT next columns.

References column.

Referenced by sauty::operator+=().

7.93.3.2 void sauty::position::initialize (std::string * fn) [inline]

Initialization.

References column, filename, and line.

7.93.3.3 void sautyy::position::lines (int *count* = 1) [inline]

(line related) Advance to the COUNT next lines.

References column, and line.

7.93.4 Member Data Documentation

7.93.4.1 unsigned int sautyy::position::column

Current column number.

Referenced by columns(), initialize(), lines(), and sautyy::operator<<().

7.93.4.2 std::string* sautyy::position::filename

File name to which this position refers.

Referenced by initialize(), and sautyy::operator<<().

7.93.4.3 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](#)

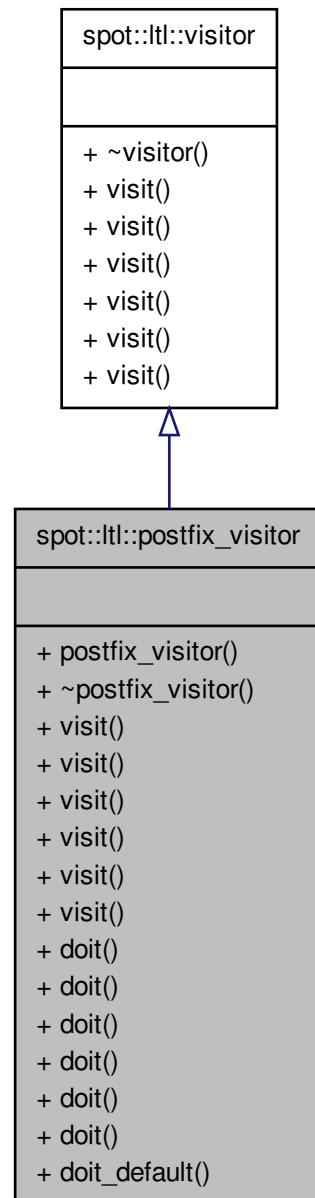
7.94 spot::ltl::postfix_visitor Class Reference

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

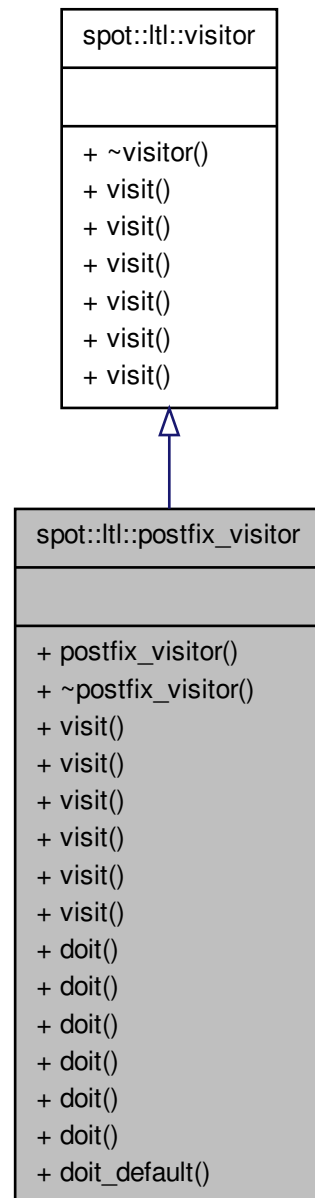
Override one or more of the postfix_visitor::doit methods with the algorithm to apply.

```
#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)

7.94.1 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.

7.94.2 Constructor & Destructor Documentation

7.94.2.1 spot::ltl::postfix_visitor::postfix_visitor ()

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

7.94.3 Member Function Documentation

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

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

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

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

7.94.3.5 virtual void spot::ltl::postfix_visitor::doit (automatop * *mo*) [virtual]

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

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

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

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

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

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

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

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

7.94.3.11 void spot::ltl::postfix_visitor::visit (atomic_prop * *ap*) [virtual]

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

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

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

7.94.3.13 void spot::ltl::postfix_visitor::visit (automatop * *c*) [virtual]

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

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

- [ltlvisit/postfix.hh](#)

7.95 spot::power_map Struct Reference

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

Public Types

- typedef std::set< const [state](#) *, [state_ptr_less_than](#) > [power_state](#)
- typedef std::map< int, [power_state](#) > [power_map_data](#)
- typedef Sgi::hash_set< const [state](#) *, [state_ptr_hash](#), [state_ptr_equal](#) > [state_set](#)

Public Member Functions

- [~power_map](#) ()
- const [power_state](#) & [states_of](#) (int s) const
- const [state](#) * [canonicalize](#) (const [state](#) *s)

Public Attributes

- [power_map_data](#) [map_](#)
- [state_set](#) [states](#)

7.95.1 Member Typedef Documentation

7.95.1.1 typedef std::map<int, power_state> spot::power_map::power_map_data

7.95.1.2 typedef std::set<const state*, state_ptr_less_than> spot::power_map::power_state

7.95.1.3 typedef Sgi::hash_set<const state*, state_ptr_hash, state_ptr_equal>
spot::power_map::state_set

7.95.2 Constructor & Destructor Documentation

7.95.2.1 spot::power_map::~power_map () [inline]

References [spot::state::destroy\(\)](#), and [states](#).

7.95.3 Member Function Documentation

7.95.3.1 const state* spot::power_map::canonicalize (const state * s) [inline]

References spot::state::destroy(), and states.

7.95.3.2 const power_state& spot::power_map::states_of (int s) const [inline]

References map_.

7.95.4 Member Data Documentation

7.95.4.1 power_map_data spot::power_map::map_

Referenced by states_of().

7.95.4.2 state_set spot::power_map::states

Referenced by canonicalize(), and ~power_map().

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

- [tgbaalgos/powerset.hh](#)

7.96 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

7.96.1 Detailed Description

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

A hash function for pointers.

7.96.2 Member Function Documentation

7.96.2.1 `template<class T > size_t spot::ptr_hash< T >::operator() (const T * p) const` `[inline]`

References `spot::knuth32_hash()`.

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

- [misc/hash.hh](#)

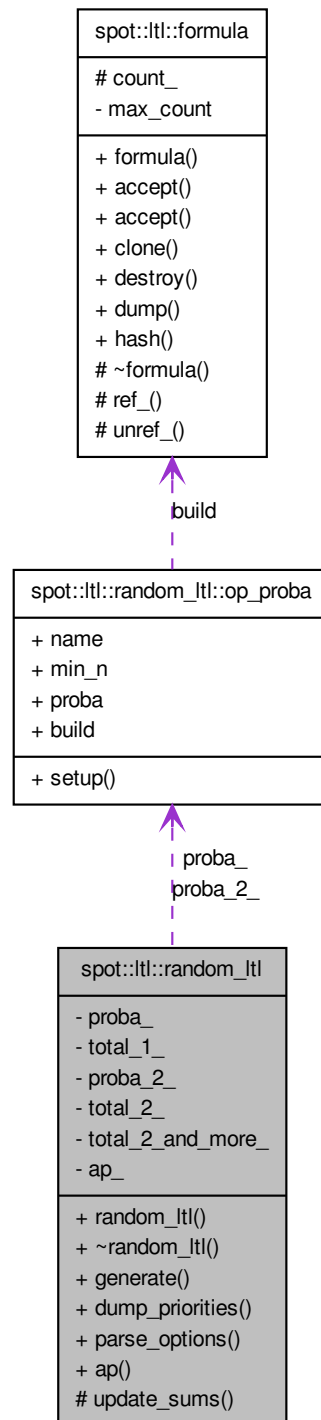
7.97 spot::ltl::random_ltl Class Reference

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_](#)

7.97.1 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\(\)](#).

7.97.2 Constructor & Destructor Documentation

7.97.2.1 `spot::ltl::random_ltl::random_ltl (const atomic_prop_set * ap)`

Create a random LTL generator using atomic propositions from *ap*.

7.97.2.2 `spot::ltl::random_ltl::~~random_ltl ()`

7.97.3 Member Function Documentation

7.97.3.1 `const atomic_prop_set* spot::ltl::random_ltl::ap () const [inline]`

Return the set of atomic proposition used to build formulae.

References `ap_`.

7.97.3.2 `std::ostream& spot::ltl::random_ltl::dump_priorities (std::ostream & os) const`

Print the priorities of each operator, constants, and atomic propositions.

7.97.3.3 `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()`.

7.97.3.4 `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.

7.97.3.5 void spot::ltl::random_ltl::update_sums () [protected]

7.97.4 Member Data Documentation

7.97.4.1 const atomic_prop_set* spot::ltl::random_ltl::ap_ [private]

Referenced by ap().

7.97.4.2 op_proba* spot::ltl::random_ltl::proba_ [private]

7.97.4.3 op_proba* spot::ltl::random_ltl::proba_2_ [private]

7.97.4.4 double spot::ltl::random_ltl::total_1_ [private]

7.97.4.5 double spot::ltl::random_ltl::total_2_ [private]

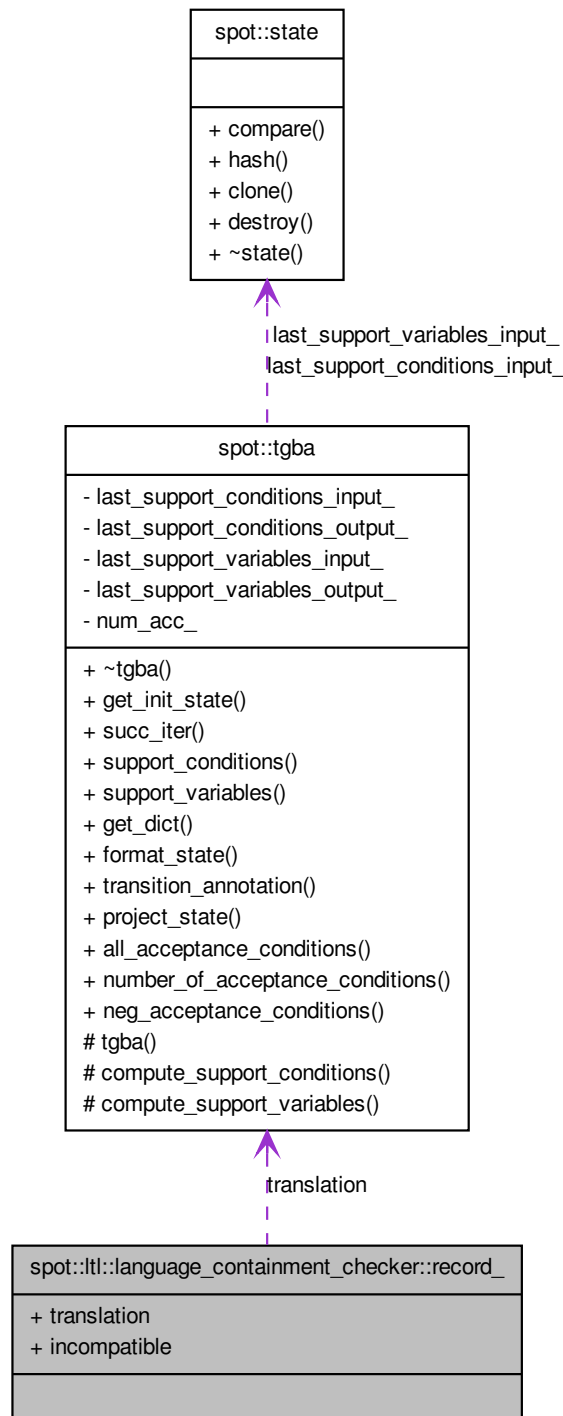
7.97.4.6 double spot::ltl::random_ltl::total_2_and_more_ [private]

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

- [ltlvisit/randomltl.hh](#)

7.98 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](#)

7.98.1 Member Typedef Documentation

7.98.1.1 typedef std::map<const [record_*](#), bool> [spot::ltl::language_containment_checker::record_::incomp_map](#)

7.98.2 Member Data Documentation

7.98.2.1 [incomp_map](#) [spot::ltl::language_containment_checker::record_::incompatible](#)

7.98.2.2 const [tgba*](#) [spot::ltl::language_containment_checker::record_::translation](#)

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

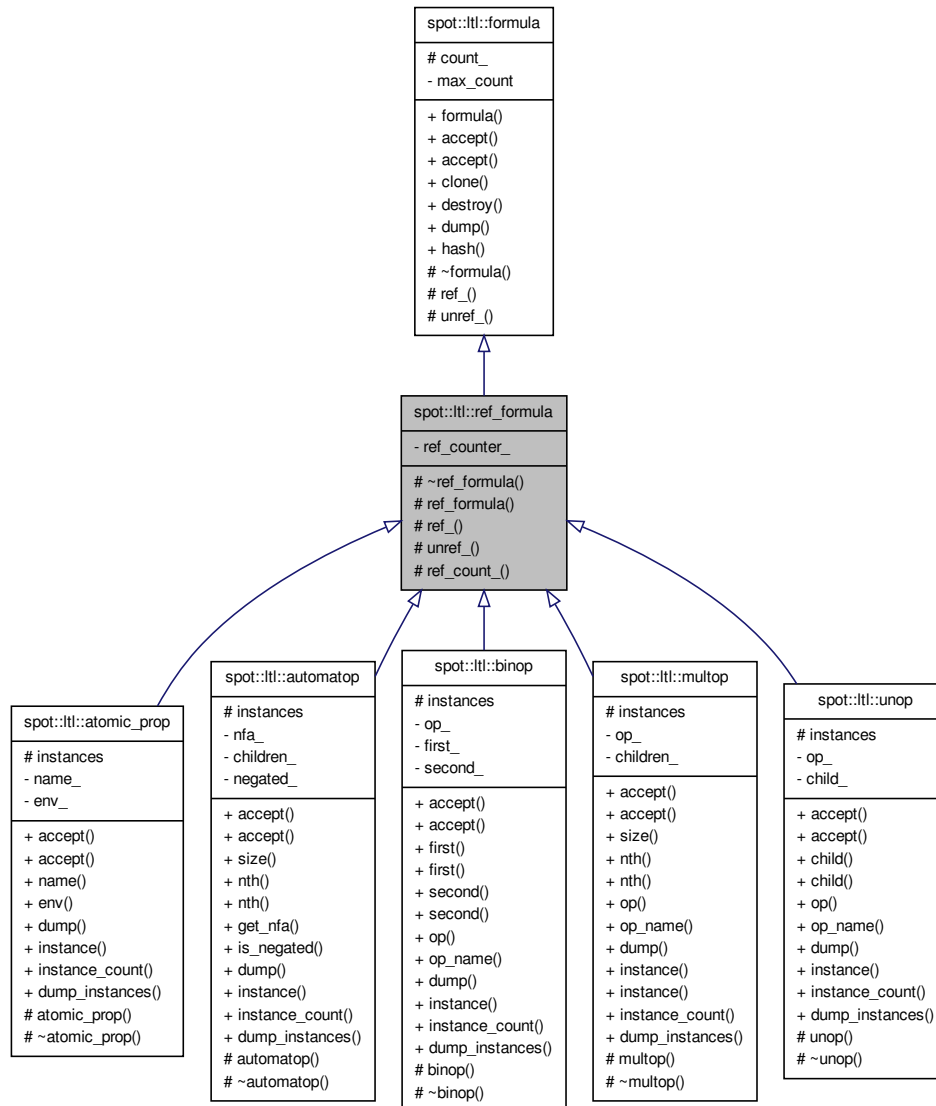
- [ltlvisit/contain.hh](#)

7.99 spot::ltl::ref_formula Class Reference

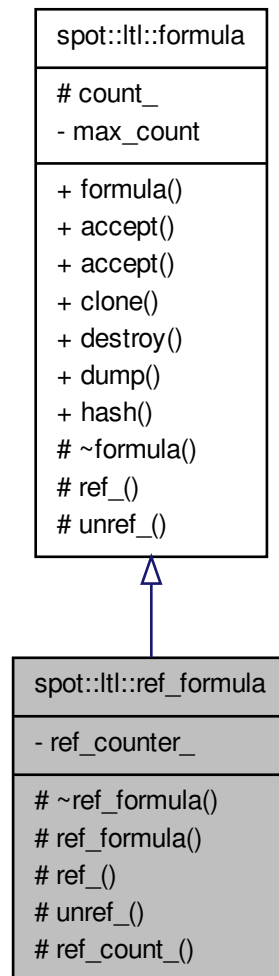
A reference-counted LTL formula.

```
#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_](#)

7.99.1 Detailed Description

A reference-counted LTL formula.

7.99.2 Constructor & Destructor Documentation

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

7.99.2.2 spot::ltl::ref_formula::ref_formula () [protected]

7.99.3 Member Function Documentation

7.99.3.1 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](#).

7.99.3.2 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](#).

7.99.3.3 formula* spot::ltl::formula::clone () const [inherited]

clone this node

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

7.99.3.4 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\(\)](#).

7.99.3.5 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](#).

Referenced by [spot::ltl::formula_ptr_less_than::operator\(\)](#).

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

Return a hash key for the formula.

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

Referenced by `spot::ltl::formula_ptr_hash::operator()()`, and `spot::ltl::formula_ptr_less_than::operator()()`.

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

increment reference counter if any

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

7.99.3.8 unsigned spot::ltl::ref_formula::ref_count_ () [protected]

Number of references to this formula.

7.99.3.9 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](#).

7.99.4 Member Data Documentation

7.99.4.1 size_t spot::ltl::formula::count_ [protected, inherited]

The hash key of this formula.

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

7.99.4.2 unsigned spot::ltl::ref_formula::ref_counter_ [private]

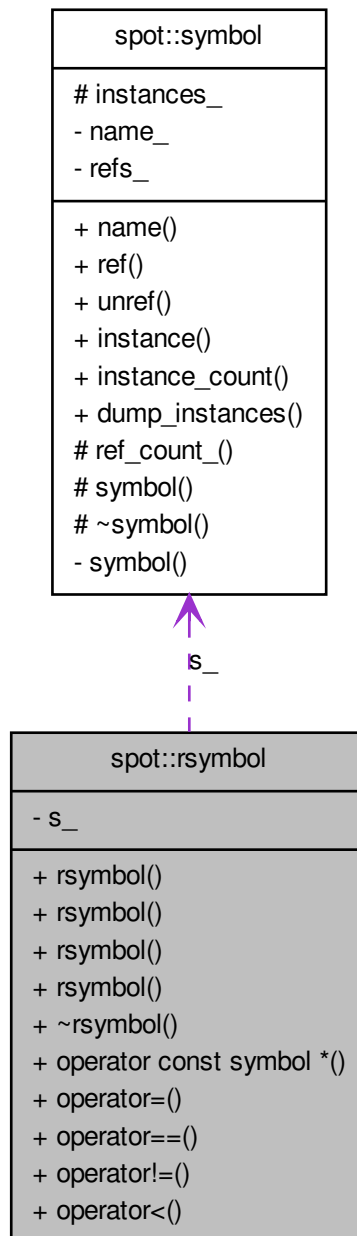
The documentation for this class was generated from the following file:

- [ltlast/refformula.hh](#)

7.100 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_`

7.100.1 Constructor & Destructor Documentation

7.100.1.1 `spot::rsymbol::rsymbol (const symbol * s) [inline]`

Referenced by `operator=()`.

7.100.1.2 `spot::rsymbol::rsymbol (const std::string & s) [inline]`

7.100.1.3 `spot::rsymbol::rsymbol (const char * s) [inline]`

7.100.1.4 `spot::rsymbol::rsymbol (const rsymbol & rs) [inline]`

References `spot::symbol::ref()`, and `s_`.

7.100.1.5 `spot::rsymbol::~~rsymbol () [inline]`

References `s_`, and `spot::symbol::unref()`.

Referenced by `operator=()`.

7.100.2 Member Function Documentation

7.100.2.1 `spot::rsymbol::operator const symbol * () const [inline]`

References `s_`.

7.100.2.2 `bool spot::rsymbol::operator!=(const rsymbol & rs) const [inline]`

References `s_`.

7.100.2.3 `bool spot::rsymbol::operator<(const rsymbol & rs) const [inline]`

References `s_`.

7.100.2.4 `const rsymbol& spot::rsymbol::operator=(const rsymbol & rs) [inline]`

References `rsymbol()`, and `~rsymbol()`.

7.100.2.5 `bool spot::rsymbol::operator==(const rsymbol & rs) const [inline]`

References `s_`.

7.100.3 Member Data Documentation

7.100.3.1 `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](#)

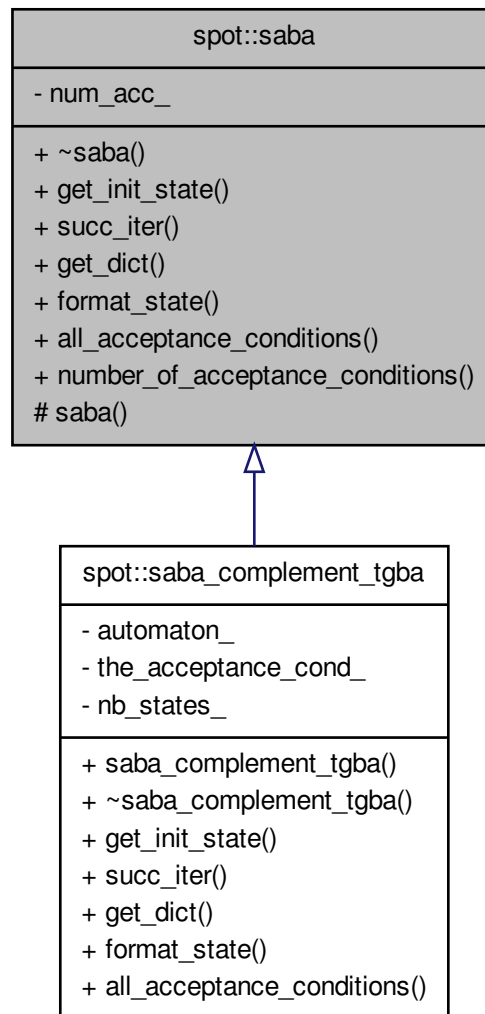
7.101 spot::saba Class Reference

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_](#)

7.101.1 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.

7.101.2 Constructor & Destructor Documentation

7.101.2.1 `spot::saba::saba ()` **[protected]**

7.101.2.2 `virtual spot::saba::~~saba ()` **[virtual]**

7.101.3 Member Function Documentation

7.101.3.1 `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](#).

7.101.3.2 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](#).

7.101.3.3 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](#).

7.101.3.4 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](#).

7.101.3.5 virtual unsigned int spot::saba::number_of_acceptance_conditions () const [virtual]

The number of acceptance conditions.

7.101.3.6 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](#).

7.101.4 Member Data Documentation

7.101.4.1 int spot::saba::num_acc_ [mutable, private]

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

- saba/[saba.hh](#)

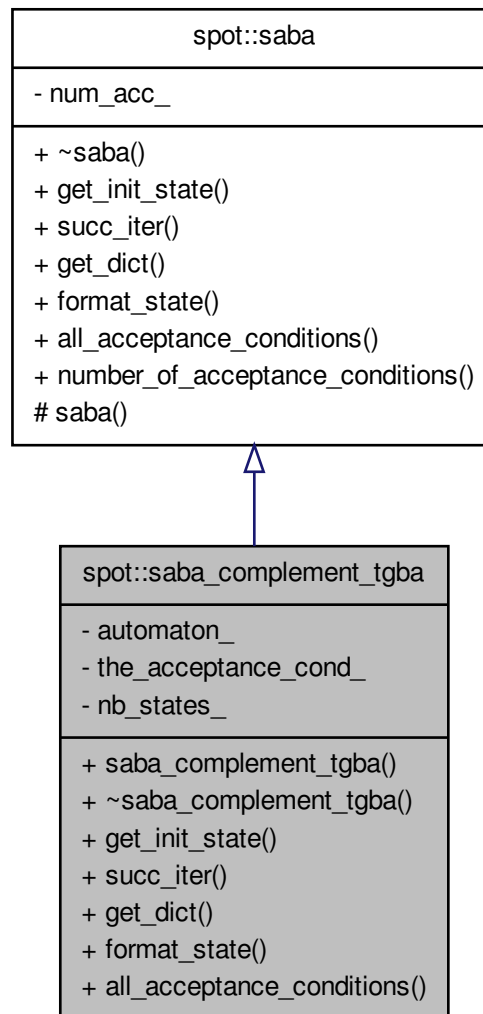
7.102 spot::saba_complement_tgba Class Reference

Complement a TGBA and produce a SABA.

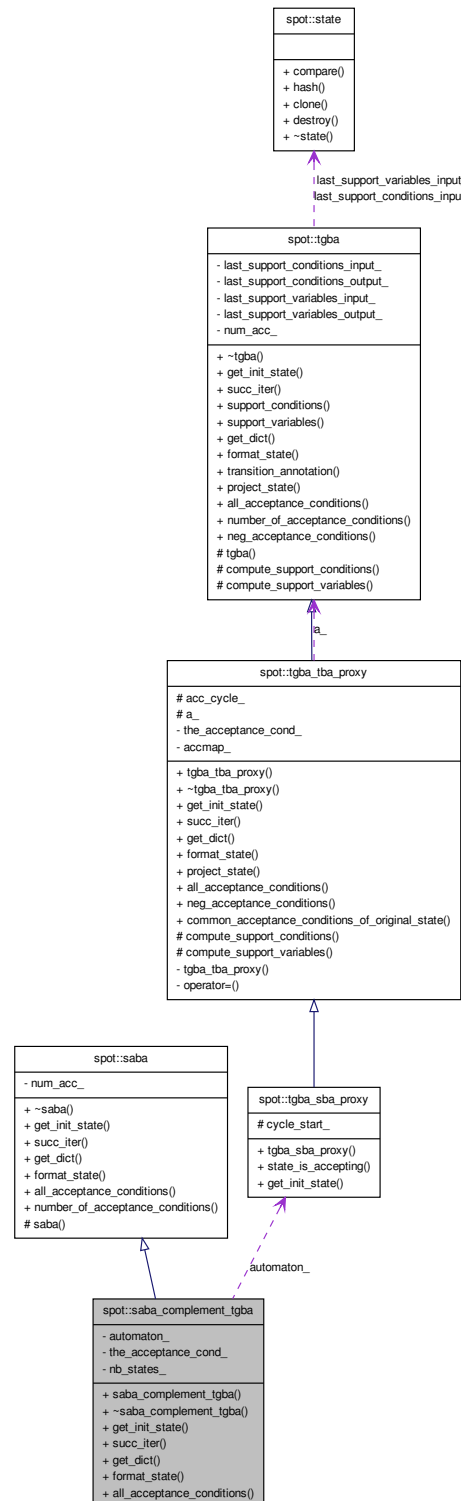
The original TGBA is transformed into a States-based Büchi Automaton.

```
#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_](#)

7.102.1 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`

7.102.2 Constructor & Destructor Documentation

7.102.2.1 `spot::saba_complement_tgba::saba_complement_tgba (const tgba * a)`

7.102.2.2 `virtual spot::saba_complement_tgba::~~saba_complement_tgba () [virtual]`

7.102.3 Member Function Documentation

7.102.3.1 `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](#).

7.102.3.2 `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](#).

7.102.3.3 `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](#).

7.102.3.4 `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](#).

7.102.3.5 `virtual unsigned int spot::saba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

7.102.3.6 `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](#).

7.102.4 Member Data Documentation

7.102.4.1 `const tgba_sba_proxy* spot::saba_complement_tgba::automaton_ [private]`

7.102.4.2 `unsigned spot::saba_complement_tgba::nb_states_ [private]`

7.102.4.3 `bdd spot::saba_complement_tgba::the_acceptance_cond_ [private]`

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

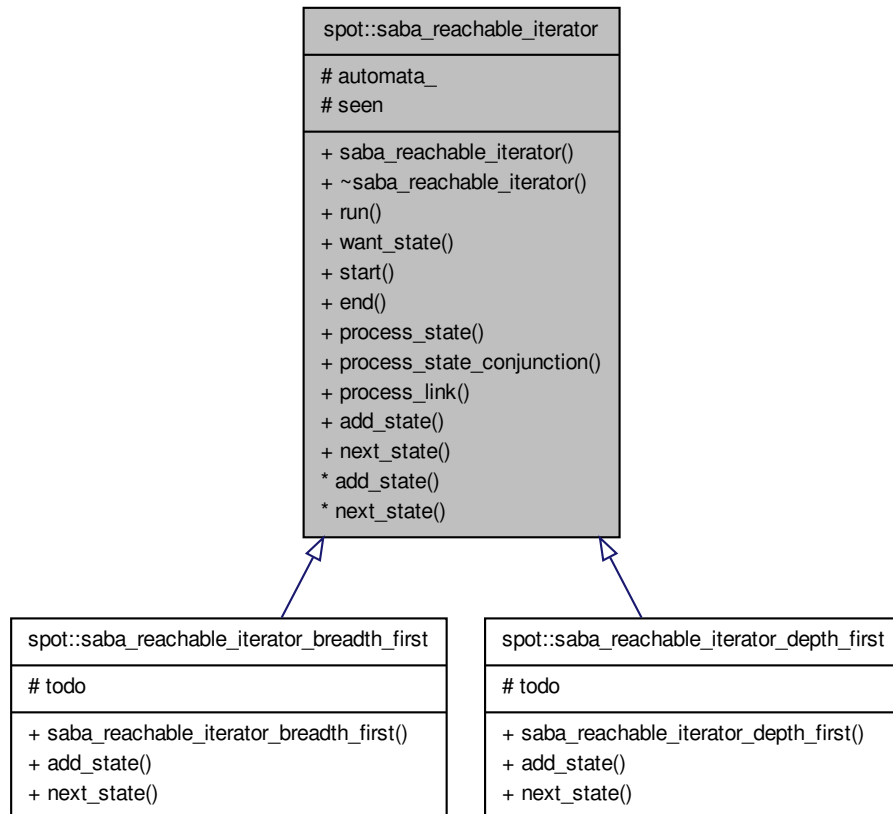
- [saba/sabacomplementtgba.hh](#)

7.103 spot::saba_reachable_iterator Class Reference

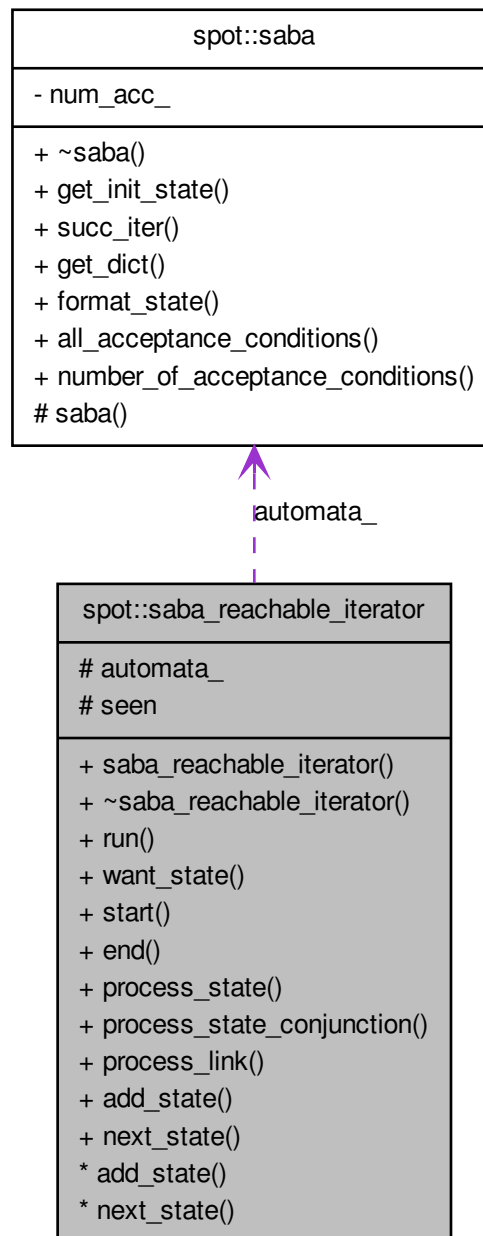
Iterate over all reachable states of a [spot::saba](#).

```
#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.

7.103.1 Detailed Description

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

7.103.2 Member Typedef Documentation

- 7.103.2.1** typedef `Sgi::hash_map`<const `saba_state`*, int, `saba_state_ptr_hash`, `saba_state_ptr_equal`> `spot::saba_reachable_iterator::seen_map` [protected]

7.103.3 Constructor & Destructor Documentation

7.103.3.1 `spot::saba_reachable_iterator::saba_reachable_iterator (const saba * a)`

7.103.3.2 `virtual spot::saba_reachable_iterator::~~saba_reachable_iterator () [virtual]`

7.103.4 Member Function Documentation

7.103.4.1 `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](#).

7.103.4.2 `virtual void spot::saba_reachable_iterator::end () [virtual]`

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

7.103.4.3 `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](#).

7.103.4.4 `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.

7.103.4.5 `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*.

7.103.4.6 `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.

7.103.4.7 `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.

7.103.4.8 `virtual void spot::saba_reachable_iterator::start () [virtual]`

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

7.103.4.9 `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.

7.103.5 Member Data Documentation

7.103.5.1 `const saba* spot::saba_reachable_iterator::automata_` `[protected]`

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

7.103.5.2 `seen_map spot::saba_reachable_iterator::seen` `[protected]`

States already seen.

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

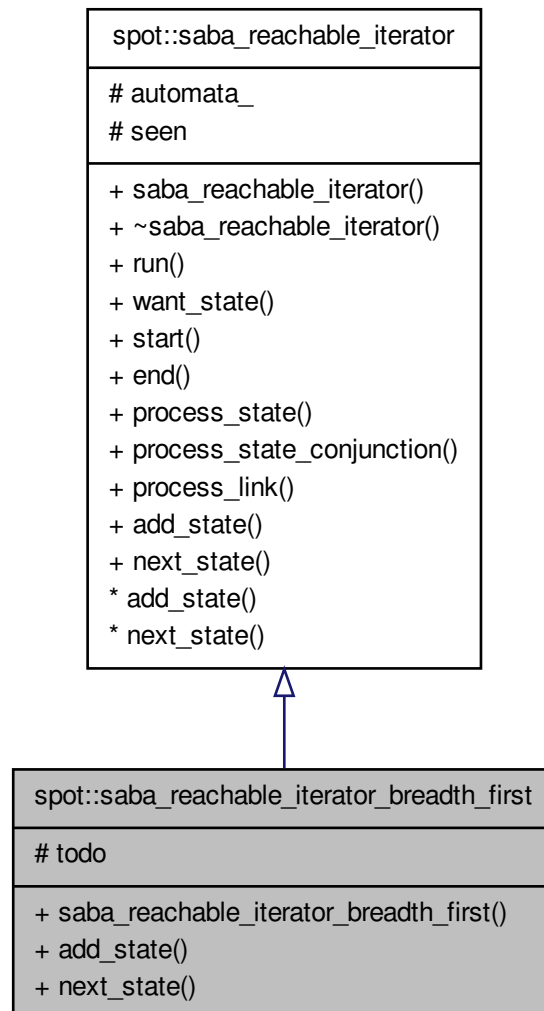
- `sabaalgos/sabareachiter.hh`

7.104 `spot::saba_reachable_iterator_breadth_first` Class Reference

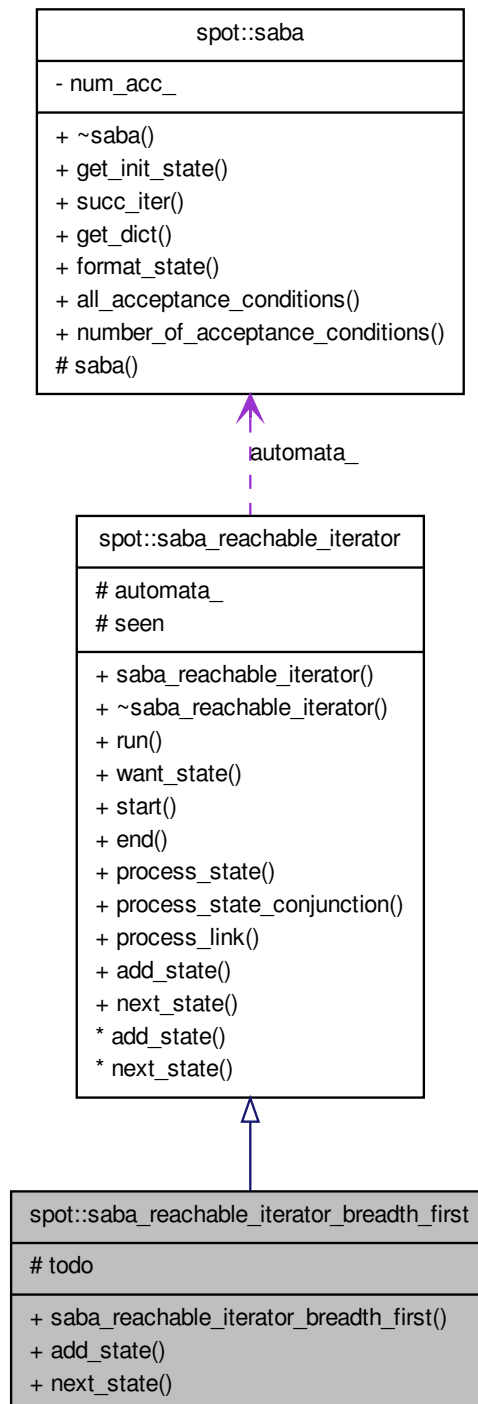
An implementation of [spot::saba_reachable_iterator](#) that browses states breadth first.

```
#include <sabaalgos/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.

7.104.1 Detailed Description

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

7.104.2 Member Typedef Documentation

- 7.104.2.1** typedef Sgi::hash_map<const [saba_state](#)*, int, [saba_state_ptr_hash](#), [saba_state_ptr_equal](#)> [spot::saba_reachable_iterator::seen_map](#) [[protected](#), [inherited](#)]

7.104.3 Constructor & Destructor Documentation

7.104.3.1 `spot::saba_reachable_iterator_breadth_first::saba_reachable_iterator_breadth_first (const saba * a)`

7.104.4 Member Function Documentation

7.104.4.1 `virtual void spot::saba_reachable_iterator_breadth_first::add_state (const saba_state * s) [virtual]`

Implements [spot::saba_reachable_iterator](#).

7.104.4.2 `virtual void spot::saba_reachable_iterator::end () [virtual, inherited]`

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

7.104.4.3 `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](#).

7.104.4.4 `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.

7.104.4.5 `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*.

7.104.4.6 `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.

7.104.4.7 `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.

7.104.4.8 `virtual void spot::saba_reachable_iterator::start () [virtual, inherited]`

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

7.104.4.9 `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.

7.104.5 Member Data Documentation

7.104.5.1 `const saba* spot::saba_reachable_iterator::automata_ [protected, inherited]`

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

7.104.5.2 seen_map spot::saba_reachable_iterator::seen [protected, inherited]

States already seen.

7.104.5.3 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:

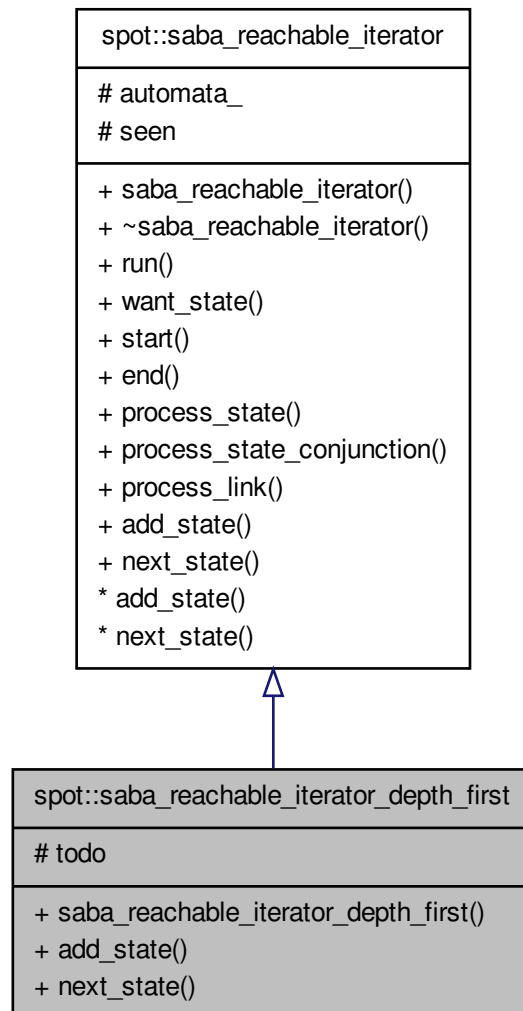
- sabaalgorithms/[sabareachiter.hh](#)

7.105 spot::saba_reachable_iterator_depth_first Class Reference

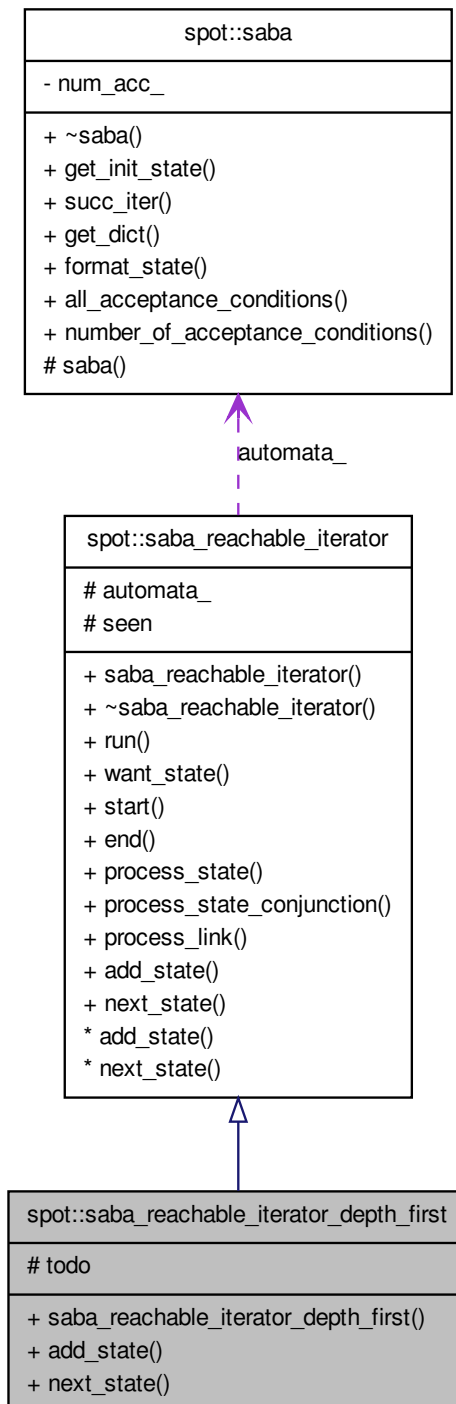
An implementation of [spot::saba_reachable_iterator](#) that browses states depth first.

```
#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.

7.105.1 Detailed Description

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

7.105.2 Member Typedef Documentation

- 7.105.2.1** typedef Sgi::hash_map<const [saba_state](#)*, int, [saba_state_ptr_hash](#), [saba_state_ptr_equal](#)> [spot::saba_reachable_iterator::seen_map](#) [[protected](#), [inherited](#)]

7.105.3 Constructor & Destructor Documentation

7.105.3.1 `spot::saba_reachable_iterator_depth_first::saba_reachable_iterator_depth_first (const saba * a)`

7.105.4 Member Function Documentation

7.105.4.1 `virtual void spot::saba_reachable_iterator_depth_first::add_state (const saba_state * s) [virtual]`

Implements [spot::saba_reachable_iterator](#).

7.105.4.2 `virtual void spot::saba_reachable_iterator::end () [virtual, inherited]`

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

7.105.4.3 `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](#).

7.105.4.4 `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.

7.105.4.5 `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*.

7.105.4.6 `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.

7.105.4.7 `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.

7.105.4.8 `virtual void spot::saba_reachable_iterator::start () [virtual, inherited]`

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

7.105.4.9 `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.

7.105.5 Member Data Documentation

7.105.5.1 `const saba* spot::saba_reachable_iterator::automata_ [protected, inherited]`

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

7.105.5.2 seen_map spot::saba_reachable_iterator::seen [protected, inherited]

States already seen.

7.105.5.3 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:

- sabaalgos/[sabareachiter.hh](#)

7.106 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](#) ()

7.106.1 Detailed Description

Abstract class for saba states.

7.106.2 Constructor & Destructor Documentation

7.106.2.1 virtual spot::saba_state::~~saba_state () [inline, virtual]

7.106.3 Member Function Documentation

7.106.3.1 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.

7.106.3.2 virtual saba_state* spot::saba_state::clone () const [pure virtual]

Duplicate a state.

7.106.3.3 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()()`.

7.106.3.4 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](#)

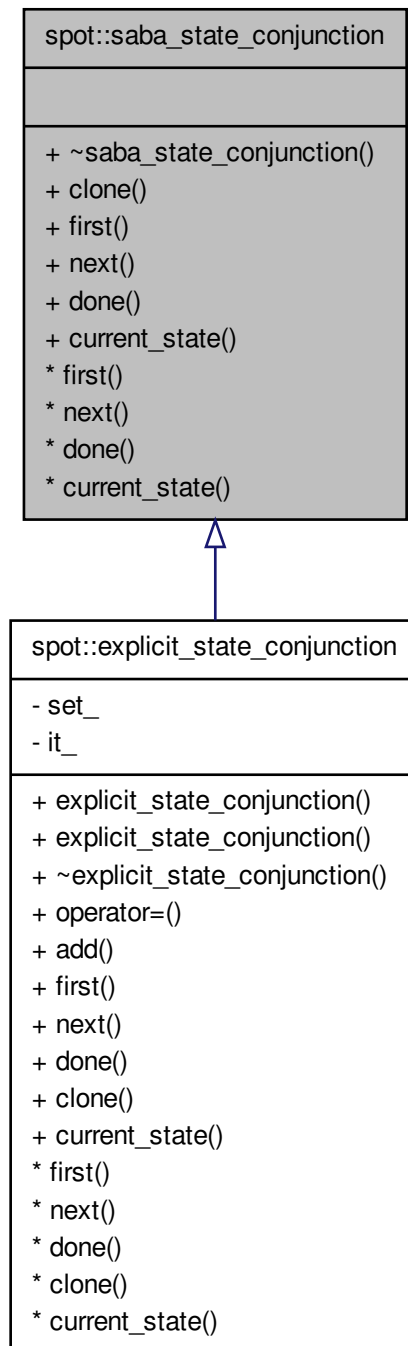
7.107 spot::saba_state_conjunction Class Reference

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.

7.107.1 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.

7.107.2 Constructor & Destructor Documentation

7.107.2.1 virtual spot::saba_state_conjunction::~~saba_state_conjunction () [inline, virtual]

7.107.3 Member Function Documentation

7.107.3.1 virtual saba_state_conjunction* spot::saba_state_conjunction::clone () const [pure virtual]

Duplicate a [saba_state](#) conjunction.

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

7.107.3.2 `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](#).

7.107.3.3 `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](#).

7.107.3.4 `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](#).

7.107.3.5 `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](#)

7.108 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

7.108.1 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;
```

7.108.2 Member Function Documentation

7.108.2.1 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](#)

7.109 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

7.109.1 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;
```

7.109.2 Member Function Documentation

7.109.2.1 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](#)

7.110 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

7.110.1 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;
```

7.110.2 Member Function Documentation

7.110.2.1 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](#)

7.111 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`

7.111.1 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;
```

7.111.2 Member Function Documentation

7.111.2.1 `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](#)

7.112 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`

7.112.1 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;
```

7.112.2 Member Function Documentation

7.112.2.1 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](#)

7.113 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

7.113.1 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;
```

7.113.2 Member Function Documentation

7.113.2.1 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](#)

7.114 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.

7.114.1 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),.

7.114.2 Constructor & Destructor Documentation

7.114.2.1 virtual spot::saba_succ_iterator::~~saba_succ_iterator () [inline, virtual]

7.114.3 Member Function Documentation

7.114.3.1 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.

7.114.3.2 `virtual saba_state_conjunction* spot::saba_succ_iterator::current_conjunction ()`
`const [pure virtual]`

Get current conjunction of successor states.

7.114.3.3 `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())
    ...
```

7.114.3.4 `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.

7.114.3.5 `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](#)

7.115 `spot::scc_map::scc` Struct Reference

```
#include <tgbalgorithms/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.

7.115.1 Constructor & Destructor Documentation

7.115.1.1 spot::scc_map::scc::scc (int *index*) [inline]

7.115.2 Member Data Documentation

7.115.2.1 bdd spot::scc_map::scc::acc

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

7.115.2.2 cond_set spot::scc_map::scc::conds

Set of conditions used in the SCC.

7.115.2.3 `int spot::scc_map::scc::index`

Index of the SCC.

7.115.2.4 `std::list<const state*> spot::scc_map::scc::states`

States of the component.

7.115.2.5 `succ_type spot::scc_map::scc::succ`

Successor SCC.

7.115.2.6 `bdd spot::scc_map::scc::supp`

Conjunction of atomic propositions used in the SCC.

7.115.2.7 `bdd spot::scc_map::scc::supp_rec`

Conjunction of atomic propositions used in the SCC.

7.115.2.8 `bool spot::scc_map::scc::trivial`

Trivial SCC have one state and no self-loops.

7.115.2.9 `bdd spot::scc_map::scc::useful_acc`

Useful acceptance conditions.

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

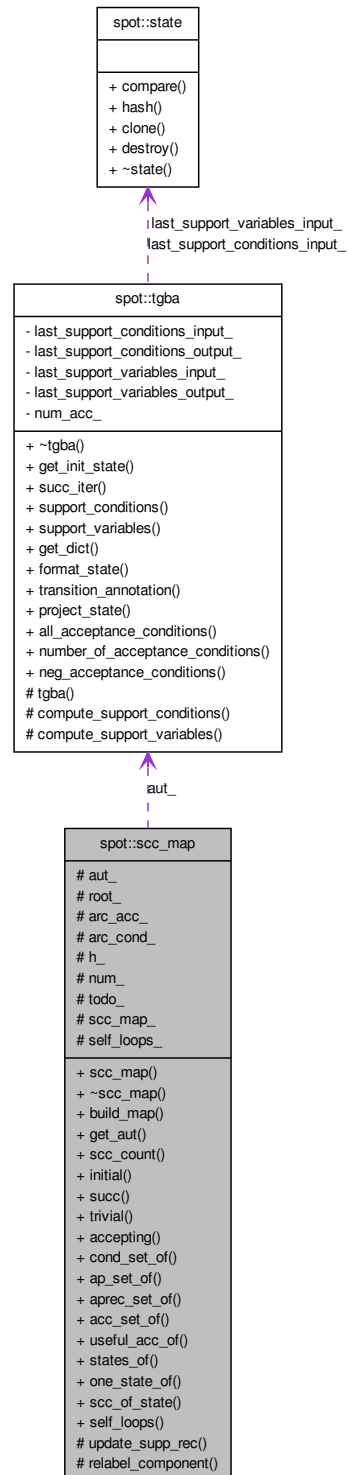
- [tgbaalgos/scc.hh](#)

7.116 `spot::scc_map` Class Reference

Build a map of Strongly Connected components in in a TGBA.

```
#include <tgbaalgos/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 [trivial](#) (unsigned n) const
Return whether an SCC is trivial.
- 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 of SCC `n`.

- `const std::list< const state * > & states_of (unsigned n) const`

Return the set of states of an SCC.

- `const state * one_state_of (unsigned n) const`

Return one state 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_`

7.116.1 Detailed Description

Build a map of Strongly Connected components in in a TGBA.

7.116.2 Member Typedef Documentation

7.116.2.1 `typedef std::set< bdd, bdd_less_than > spot::scc_map::cond_set`

7.116.2.2 `typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal>
spot::scc_map::hash_type [protected]`

7.116.2.3 `typedef std::pair<const spot::state*, tgba_succ_iterator*>
spot::scc_map::pair_state_iter [protected]`

7.116.2.4 `typedef std::vector<scc> spot::scc_map::scc_map_type [protected]`

7.116.2.5 `typedef std::list<scc> spot::scc_map::stack_type [protected]`

7.116.2.6 `typedef std::map<unsigned, bdd> spot::scc_map::succ_type`

7.116.3 Constructor & Destructor Documentation

7.116.3.1 `spot::scc_map::scc_map (const tgba * aut)`

Constructor.

This will not compute the map initially. You should call [build_map\(\)](#) to do so.

7.116.3.2 `spot::scc_map::~~scc_map ()`

7.116.4 Member Function Documentation

7.116.4.1 `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.

7.116.4.2 `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.

7.116.4.3 `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.

7.116.4.4 `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.

7.116.4.5 `void spot::scc_map::build_map ()`

Actually compute the graph of strongly connected components.

7.116.4.6 `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.

7.116.4.7 `const tgba* spot::scc_map::get_aut () const`

Get the automaton for which the map has been constructed.

7.116.4.8 `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.

7.116.4.9 `const state* spot::scc_map::one_state_of (unsigned n) const`

Return one state of an SCC.

The state in the returned list is still owned by the `scc_map` instance. It should NOT be destroyed by the client code.

Precondition

This should only be called once `build_map()` has run.

7.116.4.10 `int spot::scc_map::relabel_component () [protected]`**7.116.4.11** `unsigned spot::scc_map::scc_count () const`

Get the number of SCC in the automaton.

SCCs are labelled from 0 to `scc_count()-1`.

Precondition

This should only be called once `build_map()` has run.

7.116.4.12 `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.

7.116.4.13 `unsigned spot::scc_map::self_loops () const`

Return the number of self loops in the automaton.

7.116.4.14 `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 destroyed by the client code.

Precondition

This should only be called once `build_map()` has run.

7.116.4.15 `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.

7.116.4.16 `bool spot::scc_map::trivial (unsigned n) const`

Return whether an SCC is trivial.

Trivial SCCs have one state and no self-loop.

Precondition

This should only be called once `build_map()` has run.

7.116.4.17 `bdd spot::scc_map::update_supp_rec (unsigned state) [protected]`**7.116.4.18** `bdd spot::scc_map::useful_acc_of (unsigned n) const`

Return the set of useful acceptance conditions of SCC *n*.

Useless acceptance conditions are always implied by other acceptance conditions. This returns all the other acceptance conditions.

7.116.5 Member Data Documentation

7.116.5.1 `std::stack<bdd> spot::scc_map::arc_acc_` `[protected]`

7.116.5.2 `std::stack<bdd> spot::scc_map::arc_cond_` `[protected]`

7.116.5.3 `const tgba* spot::scc_map::aut_` `[protected]`

7.116.5.4 `hash_type spot::scc_map::h_` `[protected]`

7.116.5.5 `int spot::scc_map::num_` `[protected]`

7.116.5.6 `stack_type spot::scc_map::root_` `[protected]`

7.116.5.7 `scc_map_type spot::scc_map::scc_map_` `[protected]`

7.116.5.8 `unsigned spot::scc_map::self_loops_` `[protected]`

7.116.5.9 `std::stack<pair_state_iter> spot::scc_map::todo_` `[protected]`

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

- `tgbalgorithms/scc.hh`

7.117 `spot::scc_stack` Class Reference

```
#include <tgbalgorithms/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

7.117.1 Member Typedef Documentation

7.117.1.1 typedef std::list<connected_component> spot::scc_stack::stack_type

7.117.2 Member Function Documentation

7.117.2.1 unsigned spot::scc_stack::clear_rem ()

Purge all `rem` members.

Returns

the number of elements cleared.

7.117.2.2 `bool spot::scc_stack::empty () const`

Is the stack empty?

7.117.2.3 `void spot::scc_stack::pop ()`

Pop the top SCC.

7.117.2.4 `void spot::scc_stack::push (int index)`

Stack a new SCC with index *index*.

7.117.2.5 `std::list<const state*>& spot::scc_stack::rem ()`

The `rem` member of the top SCC.

7.117.2.6 `size_t spot::scc_stack::size () const`

How many SCC are in stack.

7.117.2.7 `const connected_component& spot::scc_stack::top () const`

Access the top SCC.

7.117.2.8 `connected_component& spot::scc_stack::top ()`

Access the top SCC.

7.117.3 Member Data Documentation**7.117.3.1 `stack_type spot::scc_stack::s`**

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

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

7.118 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](#)

7.118.1 Member Function Documentation

7.118.1.1 `std::ostream& spot::scc_stats::dump (std::ostream & out) const`

7.118.2 Member Data Documentation

7.118.2.1 `unsigned spot::scc_stats::acc_paths`

Number of maximal accepting paths.

A path is maximal and accepting if it ends in an accepting SCC that has only dead (i.e. non accepting) successors, or no successors at all.

7.118.2.2 `unsigned spot::scc_stats::acc_scc`

Total number of accepting SCC.

7.118.2.3 `unsigned spot::scc_stats::dead_paths`

Number of paths to a terminal dead SCC.

A terminal dead SCC is a dead SCC without successors.

7.118.2.4 `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.

7.118.2.5 `unsigned spot::scc_stats::scc_total`

Total number of SCCs.

7.118.2.6 `unsigned spot::scc_stats::self_loops`**7.118.2.7 `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).

7.118.2.8 `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](#)

7.119 `spot::sccs_set` Struct Reference

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

Public Attributes

- `std::set< unsigned >` [sccs](#)
- `unsigned` [size](#)

7.119.1 Member Data Documentation

7.119.1.1 `std::set<unsigned> spot::sccs_set::sccs`

7.119.1.2 `unsigned spot::sccs_set::size`

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

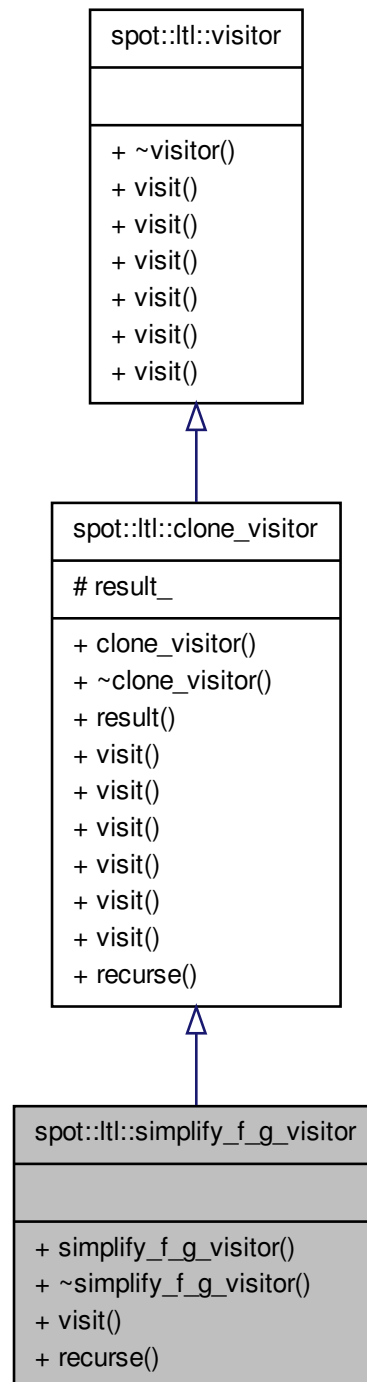
- `tgbaalgos/cutscc.hh`

7.120 `spot::ltl::simplify_f_g_visitor` Class Reference

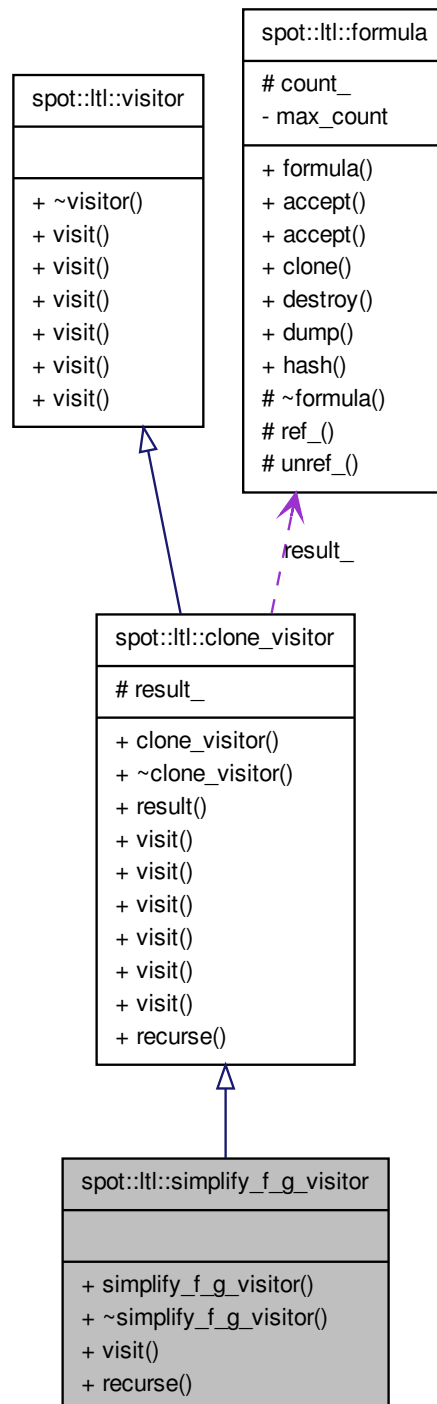
Replace `true` U `f` and `false` R `g` by `F f` and `G g`.

```
#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](#)

7.120.1 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`

7.120.2 Member Typedef Documentation

7.120.2.1 typedef `clone_visitor` `spot::ltl::simplify_f_g_visitor::super` [`private`]

7.120.3 Constructor & Destructor Documentation

7.120.3.1 `spot::ltl::simplify_f_g_visitor::simplify_f_g_visitor ()`

7.120.3.2 virtual `spot::ltl::simplify_f_g_visitor::~~simplify_f_g_visitor ()` [`virtual`]

7.120.4 Member Function Documentation

7.120.4.1 `virtual formula* spot::ltl::simplify_f_g_visitor::recurse (formula * f) [virtual]`

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

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

7.120.4.3 `void spot::ltl::clone_visitor::visit (multop * mo) [virtual, inherited]`

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

7.120.4.4 `void spot::ltl::clone_visitor::visit (automatop * mo) [virtual, inherited]`

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

7.120.4.5 `void spot::ltl::clone_visitor::visit (unop * uo) [virtual, inherited]`

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

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

7.120.4.6 `void spot::ltl::clone_visitor::visit (atomic_prop * ap) [virtual, inherited]`

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

7.120.4.7 `void spot::ltl::clone_visitor::visit (constant * c) [virtual, inherited]`

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

7.120.4.8 `void spot::ltl::simplify_f_g_visitor::visit (binop * bo) [virtual]`

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

7.120.5 Member Data Documentation

7.120.5.1 `formula* spot::ltdl::clone_visitor::result_` `[protected, inherited]`

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

- [ltdlvisit/simpfg.hh](#)

7.121 `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_](#)

7.121.1 Detailed Description

```
template<class T, class S = stack<T>> class eltlyy::slice< T, S >
```

Present a slice of the top of a stack.

7.121.2 Constructor & Destructor Documentation

7.121.2.1 `template<class T, class S = stack<T>> eltlyy::slice< T, S >::slice (const S & stack, unsigned int range)` `[inline]`

7.121.3 Member Function Documentation

7.121.3.1 `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_`.

7.121.4 Member Data Documentation

7.121.4.1 `template<class T , class S = stack<T>> unsigned int eltlyy::slice< T, S >::range_ [private]`

Referenced by `eltlyy::slice< T, S >::operator[]()`.

7.121.4.2 `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`

7.122 sautyy::slice< T, S > Class Template Reference

Present a slice of the top of a stack.

```
#include <sautparse/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_`

7.122.1 Detailed Description

```
template<class T, class S = stack<T>> class sautyy::slice< T, S >
```

Present a slice of the top of a stack.

7.122.2 Constructor & Destructor Documentation

7.122.2.1 `template<class T , class S = stack<T>> sautyy::slice< T, S >::slice (const S & stack, unsigned int range) [inline]`

7.122.3 Member Function Documentation

7.122.3.1 `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_.

7.122.4 Member Data Documentation

7.122.4.1 `template<class T , class S = stack<T>> unsigned int sautyy::slice< T, S >::range_ [private]`

Referenced by sautyy::slice< T, S >::operator[]().

7.122.4.2 `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](#)

7.123 neverclaimyy::slice< T, S > Class Template Reference

Present a slice of the top of a stack.

```
#include <neverparse/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_](#)

7.123.1 Detailed Description

`template<class T, class S = stack<T>> class neverclaimyy::slice< T, S >`

Present a slice of the top of a stack.

7.123.2 Constructor & Destructor Documentation

7.123.2.1 `template<class T , class S = stack<T>> neverclaimyy::slice< T, S >::slice (const S & stack, unsigned int range) [inline]`

7.123.3 Member Function Documentation

7.123.3.1 `template<class T , class S = stack<T>> const T& neverclaimyy::slice< T, S >::operator[] (unsigned int i) const [inline]`

References neverclaimyy::slice< T, S >::range_, and neverclaimyy::slice< T, S >::stack_.

7.123.4 Member Data Documentation

7.123.4.1 `template<class T , class S = stack<T>> unsigned int neverclaimyy::slice< T, S >::range_ [private]`

Referenced by neverclaimyy::slice< T, S >::operator[]().

7.123.4.2 `template<class T , class S = stack<T>> const S& neverclaimyy::slice< T, S >::stack_ [private]`

Referenced by neverclaimyy::slice< T, S >::operator[]().

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

- neverparse/[stack.hh](#)

7.124 Itlyy::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_](#)

7.124.1 Detailed Description

template<class T, class S = stack<T>> class ltlyy::slice< T, S >

Present a slice of the top of a stack.

7.124.2 Constructor & Destructor Documentation

7.124.2.1 template<class T , class S = stack<T>> ltlyy::slice< T, S >::slice (const S & *stack*, unsigned int *range*) [inline]

7.124.3 Member Function Documentation

7.124.3.1 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_.

7.124.4 Member Data Documentation

7.124.4.1 template<class T , class S = stack<T>> unsigned int ltlyy::slice< T, S >::range_ [private]

Referenced by ltlyy::slice< T, S >::operator[]().

7.124.4.2 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:

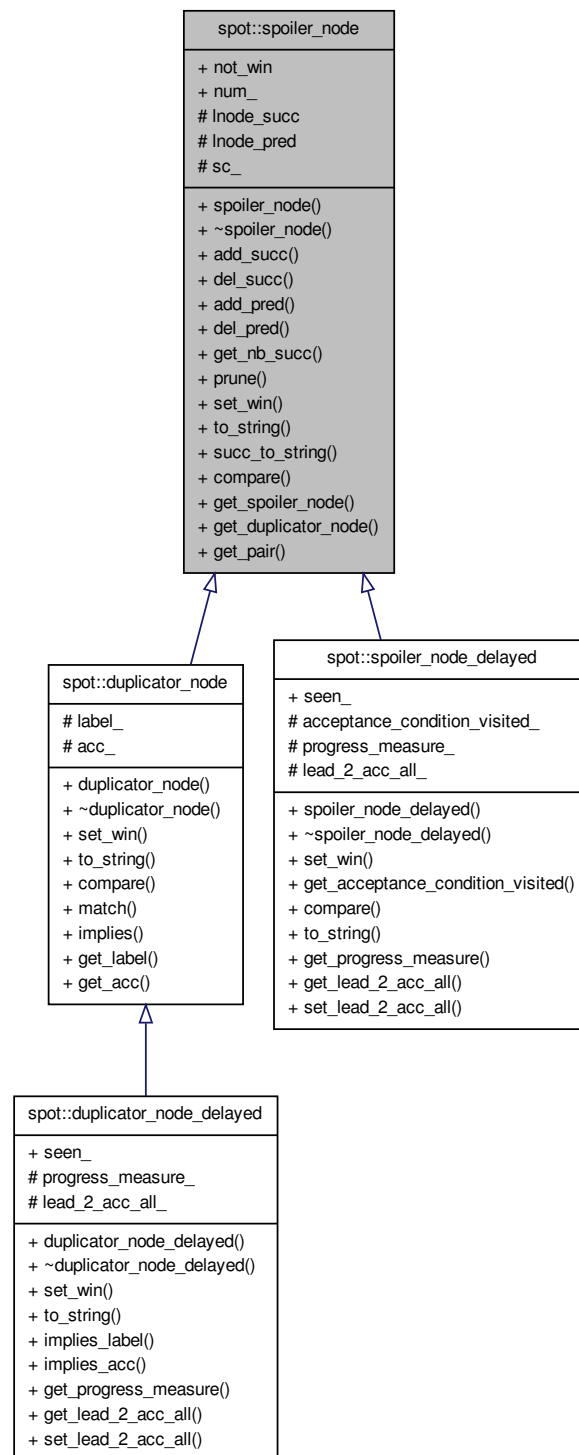
- [ltlparse/stack.hh](#)

7.125 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_`

7.125.1 Detailed Description

Spoiler node of parity game graph.

7.125.2 Constructor & Destructor Documentation

7.125.2.1 `spot::spoiler_node::spoiler_node (const state * d_node, const state * s_node, int num)`

7.125.2.2 `virtual spot::spoiler_node::~~spoiler_node ()` `[virtual]`

7.125.3 Member Function Documentation

7.125.3.1 `virtual void spot::spoiler_node::add_pred (spoiler_node * n) [virtual]`

7.125.3.2 `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.

7.125.3.3 `virtual bool spot::spoiler_node::compare (spoiler_node * n) [virtual]`

Reimplemented in [spot::duplicator_node](#), and [spot::spoiler_node_delayed](#).

7.125.3.4 `virtual void spot::spoiler_node::del_pred () [virtual]`

7.125.3.5 `void spot::spoiler_node::del_succ (spoiler_node * n)`

7.125.3.6 `const state* spot::spoiler_node::get_duplicator_node ()`

7.125.3.7 `int spot::spoiler_node::get_nb_succ ()`

7.125.3.8 `state_couple* spot::spoiler_node::get_pair ()`

7.125.3.9 `const state* spot::spoiler_node::get_spoiler_node ()`

7.125.3.10 `bool spot::spoiler_node::prune ()`

7.125.3.11 `virtual bool spot::spoiler_node::set_win () [virtual]`

Reimplemented in [spot::duplicator_node](#), [spot::spoiler_node_delayed](#), and [spot::duplicator_node_delayed](#).

7.125.3.12 `virtual std::string spot::spoiler_node::succ_to_string () [virtual]`**7.125.3.13** `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](#).

7.125.4 Member Data Documentation**7.125.4.1** `sn_v* spot::spoiler_node::lnode_pred [protected]`**7.125.4.2** `sn_v* spot::spoiler_node::lnode_succ [protected]`**7.125.4.3** `bool spot::spoiler_node::not_win`**7.125.4.4** `int spot::spoiler_node::num_`**7.125.4.5** `state_couple* spot::spoiler_node::sc_ [protected]`

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

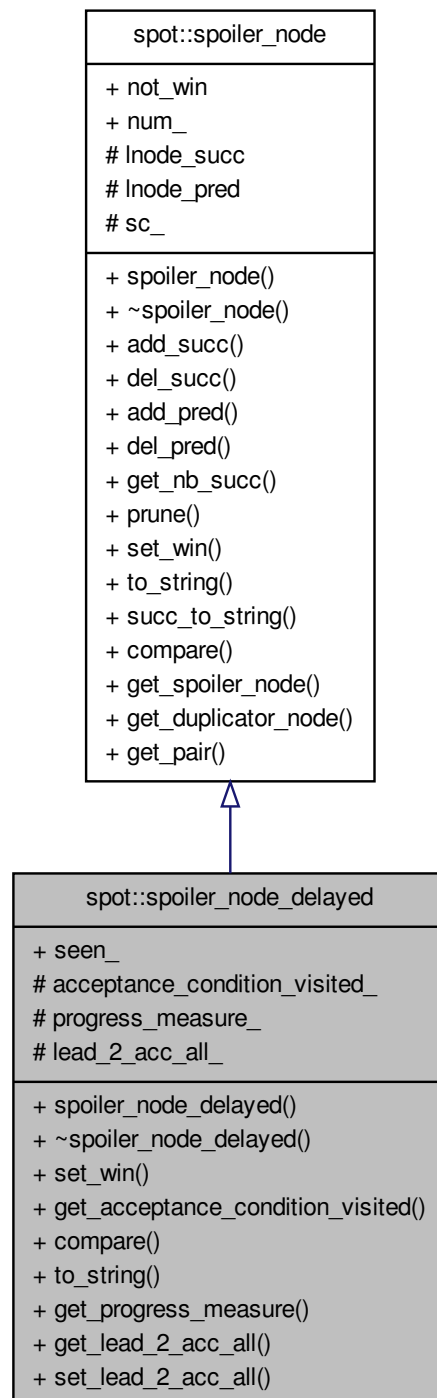
- [tgbaalgos/reductgba_sim.hh](#)

7.126 `spot::spoiler_node_delayed` Class Reference

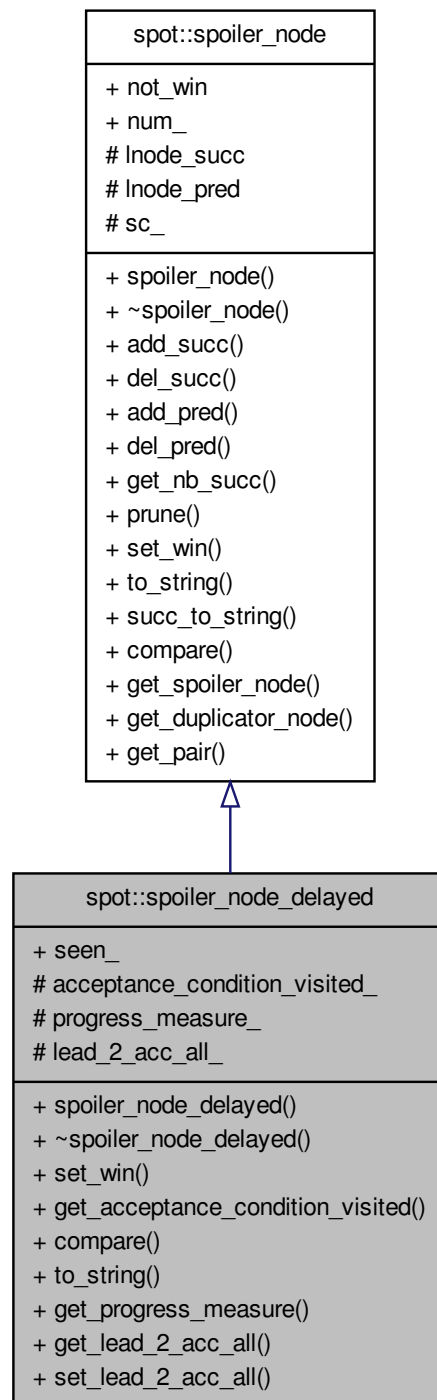
Spoiler node of parity game graph for delayed simulation.

```
#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_

7.126.1 Detailed Description

Spoiler node of parity game graph for delayed simulation.

7.126.2 Constructor & Destructor Documentation

- 7.126.2.1** spot::spoiler_node_delayed::spoiler_node_delayed (const state * d_node, const state * s_node, bdd a, int num)

7.126.2.2 `spot::spoiler_node_delayed::~~spoiler_node_delayed ()`

7.126.3 Member Function Documentation

7.126.3.1 `virtual void spot::spoiler_node::add_pred (spoiler_node * n) [virtual, inherited]`

7.126.3.2 `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.

7.126.3.3 `virtual bool spot::spoiler_node_delayed::compare (spoiler_node * n) [virtual]`

Reimplemented from [spot::spoiler_node](#).

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

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

7.126.3.6 `bdd spot::spoiler_node_delayed::get_acceptance_condition_visited () const`

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

7.126.3.8 `bool spot::spoiler_node_delayed::get_lead_2_acc_all ()`

7.126.3.9 `int spot::spoiler_node::get_nb_succ () [inherited]`

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

7.126.3.11 `int spot::spoiler_node_delayed::get_progress_measure () const`

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

7.126.3.13 `bool spot::spoiler_node::prune () [inherited]`

7.126.3.14 `bool spot::spoiler_node_delayed::set_lead_2_acc_all (bdd acc = bddfalse)`

7.126.3.15 `bool spot::spoiler_node_delayed::set_win () [virtual]`

Return true if the progress_measure has changed.

Reimplemented from [spot::spoiler_node](#).

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

7.126.3.17 `virtual std::string spot::spoiler_node_delayed::to_string (const tgba * a)
[virtual]`

Reimplemented from [spot::spoiler_node](#).

7.126.4 Member Data Documentation

7.126.4.1 `bdd spot::spoiler_node_delayed::acceptance_condition_visited_ [protected]`

a Bdd for retain all the acceptance condition that a node has visited.

7.126.4.2 `bool spot::spoiler_node_delayed::lead_2_acc_all_ [protected]`

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

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

7.126.4.5 `bool spot::spoiler_node::not_win` [inherited]

7.126.4.6 `int spot::spoiler_node::num_` [inherited]

7.126.4.7 `int spot::spoiler_node_delayed::progress_measure_` [protected]

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

7.126.4.9 `bool spot::spoiler_node_delayed::seen_`

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

- `tgbaalgos/reductgba_sim.hh`

7.127 neverclaimyy::stack< T, S > Class Template Reference

```
#include <neverparse/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 neverclaimyy::stack< T, S >

7.127.1 Member Typedef Documentation

**7.127.1.1 template<class T , class S = std::deque<T>> typedef S::const_reverse_iterator
neverclaimyy::stack< T, S >::const_iterator**

**7.127.1.2 template<class T , class S = std::deque<T>> typedef S::reverse_iterator
neverclaimyy::stack< T, S >::iterator**

7.127.2 Constructor & Destructor Documentation

**7.127.2.1 template<class T , class S = std::deque<T>> neverclaimyy::stack< T, S >::stack ()
[inline]**

**7.127.2.2 template<class T , class S = std::deque<T>> neverclaimyy::stack< T, S >::stack (
unsigned int *n*) [inline]**

7.127.3 Member Function Documentation

**7.127.3.1 template<class T , class S = std::deque<T>> const_iterator neverclaimyy::stack< T, S
>::begin () const [inline]**

References neverclaimyy::stack< T, S >::seq_.

**7.127.3.2 template<class T , class S = std::deque<T>> const_iterator neverclaimyy::stack< T, S
>::end () const [inline]**

References neverclaimyy::stack< T, S >::seq_.

7.127.3.3 `template<class T, class S = std::deque<T>> unsigned int neverclaimyy::stack< T, S >::height () const [inline]`

References neverclaimyy::stack< T, S >::seq_.

7.127.3.4 `template<class T, class S = std::deque<T>> const T& neverclaimyy::stack< T, S >::operator[] (unsigned int i) const [inline]`

References neverclaimyy::stack< T, S >::seq_.

7.127.3.5 `template<class T, class S = std::deque<T>> T& neverclaimyy::stack< T, S >::operator[] (unsigned int i) [inline]`

References neverclaimyy::stack< T, S >::seq_.

7.127.3.6 `template<class T, class S = std::deque<T>> void neverclaimyy::stack< T, S >::pop (unsigned int n = 1) [inline]`

References neverclaimyy::stack< T, S >::seq_.

7.127.3.7 `template<class T, class S = std::deque<T>> void neverclaimyy::stack< T, S >::push (const T & t) [inline]`

References neverclaimyy::stack< T, S >::seq_.

7.127.4 Member Data Documentation

7.127.4.1 `template<class T, class S = std::deque<T>> S neverclaimyy::stack< T, S >::seq_ [private]`

Referenced by neverclaimyy::stack< T, S >::begin(), neverclaimyy::stack< T, S >::end(), neverclaimyy::stack< T, S >::height(), neverclaimyy::stack< T, S >::operator[](), neverclaimyy::stack< T, S >::pop(), and neverclaimyy::stack< T, S >::push().

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

- [neverparse/stack.hh](#)

7.128 sautyy::stack< T, S > Class Template Reference

```
#include <sautyparse/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 >
```

7.128.1 Member Typedef Documentation

7.128.1.1 template<class T , class S = std::deque<T>> typedef S::const_reverse_iterator
 sautyy::stack< T, S >::const_iterator

7.128.1.2 template<class T , class S = std::deque<T>> typedef S::reverse_iterator
 sautyy::stack< T, S >::iterator

7.128.2 Constructor & Destructor Documentation

7.128.2.1 template<class T , class S = std::deque<T>> sautyy::stack< T, S >::stack ()
 [inline]

7.128.2.2 `template<class T , class S = std::deque<T>> sautyy::stack< T, S >::stack (unsigned int n) [inline]`

7.128.3 Member Function Documentation

7.128.3.1 `template<class T , class S = std::deque<T>> const_iterator sautyy::stack< T, S >::begin () const [inline]`

References sautyy::stack< T, S >::seq_.

7.128.3.2 `template<class T , class S = std::deque<T>> const_iterator sautyy::stack< T, S >::end () const [inline]`

References sautyy::stack< T, S >::seq_.

7.128.3.3 `template<class T , class S = std::deque<T>> unsigned int sautyy::stack< T, S >::height () const [inline]`

References sautyy::stack< T, S >::seq_.

7.128.3.4 `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_.

7.128.3.5 `template<class T , class S = std::deque<T>> T& sautyy::stack< T, S >::operator[] (unsigned int i) [inline]`

References sautyy::stack< T, S >::seq_.

7.128.3.6 `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_.

7.128.3.7 `template<class T , class S = std::deque<T>> void sautyy::stack< T, S >::push (const T & t) [inline]`

References `sautyy::stack< T, S >::seq_`.

7.128.4 Member Data Documentation

7.128.4.1 `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`

7.129 `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 >`

7.129.1 Member Typedef Documentation

7.129.1.1 `template<class T, class S = std::deque<T>> typedef S::const_reverse_iterator
eltlyy::stack< T, S >::const_iterator`

7.129.1.2 `template<class T, class S = std::deque<T>> typedef S::reverse_iterator eltlyy::stack<
T, S >::iterator`

7.129.2 Constructor & Destructor Documentation

7.129.2.1 `template<class T, class S = std::deque<T>> eltlyy::stack< T, S >::stack ()
[inline]`

7.129.2.2 `template<class T, class S = std::deque<T>> eltlyy::stack< T, S >::stack (unsigned
int n) [inline]`

7.129.3 Member Function Documentation

7.129.3.1 `template<class T, class S = std::deque<T>> const_iterator eltlyy::stack< T, S
>::begin () const [inline]`

References `eltlyy::stack< T, S >::seq_`.

7.129.3.2 `template<class T, class S = std::deque<T>> const_iterator eltlyy::stack< T, S >::end ()
const [inline]`

References `eltlyy::stack< T, S >::seq_`.

7.129.3.3 `template<class T, class S = std::deque<T>> unsigned int eltlyy::stack< T, S >::height
() const [inline]`

References `eltlyy::stack< T, S >::seq_`.

7.129.3.4 `template<class T , class S = std::deque<T>> const T& ltlly::stack< T, S >::operator[] (unsigned int i) const [inline]`

References `ltlly::stack< T, S >::seq_`.

7.129.3.5 `template<class T , class S = std::deque<T>> T& ltlly::stack< T, S >::operator[] (unsigned int i) [inline]`

References `ltlly::stack< T, S >::seq_`.

7.129.3.6 `template<class T , class S = std::deque<T>> void ltlly::stack< T, S >::pop (unsigned int n = 1) [inline]`

References `ltlly::stack< T, S >::seq_`.

7.129.3.7 `template<class T , class S = std::deque<T>> void ltlly::stack< T, S >::push (const T & t) [inline]`

References `ltlly::stack< T, S >::seq_`.

7.129.4 Member Data Documentation

7.129.4.1 `template<class T , class S = std::deque<T>> S ltlly::stack< T, S >::seq_ [private]`

Referenced by `ltlly::stack< T, S >::begin()`, `ltlly::stack< T, S >::end()`, `ltlly::stack< T, S >::height()`, `ltlly::stack< T, S >::operator[]()`, `ltlly::stack< T, S >::pop()`, and `ltlly::stack< T, S >::push()`.

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

- [ltlparse/stack.hh](#)

7.130 `ltlly::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 >

7.130.1 Member Typedef Documentation

7.130.1.1 **template<class T , class S = std::deque<T>> typedef S::const_reverse_iterator
ltlyy::stack< T, S >::const_iterator**

7.130.1.2 **template<class T , class S = std::deque<T>> typedef S::reverse_iterator ltlyy::stack<
T, S >::iterator**

7.130.2 Constructor & Destructor Documentation

7.130.2.1 **template<class T , class S = std::deque<T>> ltlyy::stack< T, S >::stack ()
[inline]**

7.130.2.2 **template<class T , class S = std::deque<T>> ltlyy::stack< T, S >::stack (unsigned int
n) [inline]**

7.130.3 Member Function Documentation

7.130.3.1 **template<class T , class S = std::deque<T>> const_iterator ltlyy::stack< T, S >::begin
() const [inline]**

References [ltlyy::stack< T, S >::seq_](#).

7.130.3.2 `template<class T, class S = std::deque<T>> const_iterator ltlyy::stack< T, S >::end () const [inline]`

References `ltlyy::stack< T, S >::seq_`.

7.130.3.3 `template<class T, class S = std::deque<T>> unsigned int ltlyy::stack< T, S >::height () const [inline]`

References `ltlyy::stack< T, S >::seq_`.

7.130.3.4 `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_`.

7.130.3.5 `template<class T, class S = std::deque<T>> T& ltlyy::stack< T, S >::operator[] (unsigned int i) [inline]`

References `ltlyy::stack< T, S >::seq_`.

7.130.3.6 `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_`.

7.130.3.7 `template<class T, class S = std::deque<T>> void ltlyy::stack< T, S >::push (const T & t) [inline]`

References `ltlyy::stack< T, S >::seq_`.

7.130.4 Member Data Documentation

7.130.4.1 `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](#)

7.131 spot::evtgba_explicit::state Struct Reference

```
#include <evtgba/explicit.hh>
```

Public Attributes

- [transition_list](#) in
- [transition_list](#) out

7.131.1 Member Data Documentation

7.131.1.1 transition_list spot::evtgba_explicit::state::in

7.131.1.2 transition_list spot::evtgba_explicit::state::out

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

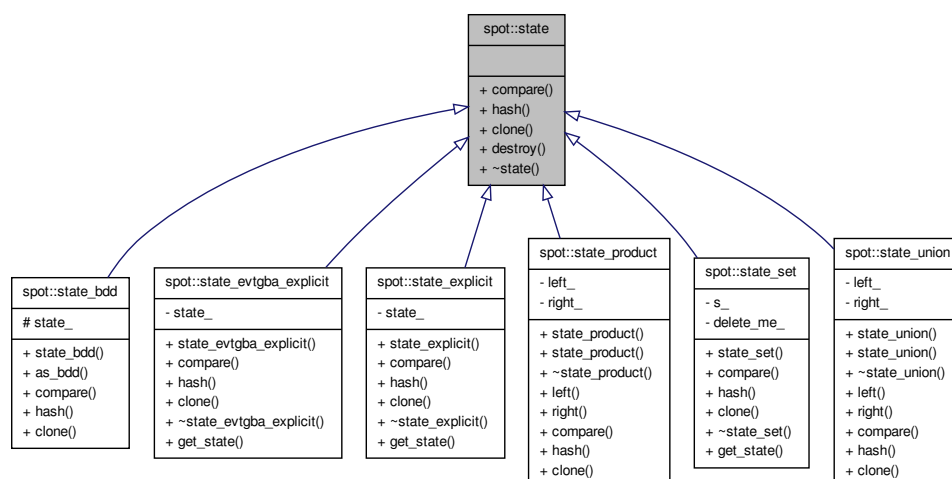
- [evtgba/explicit.hh](#)

7.132 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 void [destroy](#) () const
Release a state.
- virtual [~state](#) ()
Destructor.

7.132.1 Detailed Description

Abstract class for states.

7.132.2 Constructor & Destructor Documentation**7.132.2.1 virtual spot::state::~~state () [inline, virtual]**

Destructor.

Deprecated

Client code should now call `s->destroy()` ; instead of `delete s;`.

7.132.3 Member Function Documentation**7.132.3.1 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](#).

7.132.3.2 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\(\)](#).

7.132.3.3 `virtual void spot::state::destroy () const [inline, virtual]`

Release a state.

Methods from the `tgba` or [tgba_succ_iterator](#) always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine [destroy\(\)](#) to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by [spot::power_map::canonicalize\(\)](#), [spot::shared_state_deleter\(\)](#), and [spot::power_map::~~power_map\(\)](#).

7.132.3.4 `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\(\)](#).

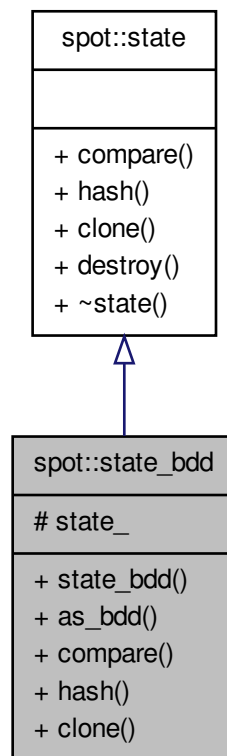
The documentation for this class was generated from the following file:

- [tgba/state.hh](#)

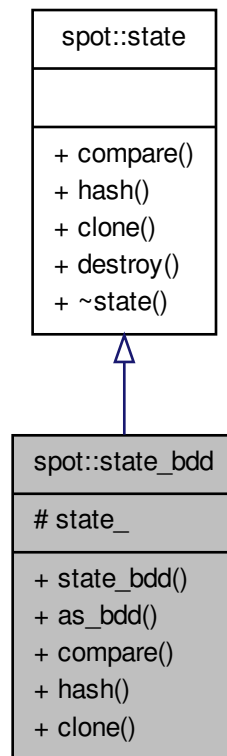
7.133 `spot::state_bdd` Class Reference

```
#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.
- virtual void `destroy` () const
Release a state.

Protected Attributes

- `bdd state_`
BDD representation of the state.

7.133.1 Detailed Description

A state whose representation is a BDD.

7.133.2 Constructor & Destructor Documentation

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

7.133.3 Member Function Documentation

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

Return the BDD part of the state.

References `state_`.

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

Duplicate a state.

Implements [spot::state](#).

7.133.3.3 `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](#).

7.133.3.4 `virtual void spot::state::destroy () const [inline, virtual, inherited]`

Release a state.

Methods from the `tgba` or `tgba_succ_iterator` always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine `destroy()` to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

7.133.3.5 `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`.

7.133.4 Member Data Documentation

7.133.4.1 `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:

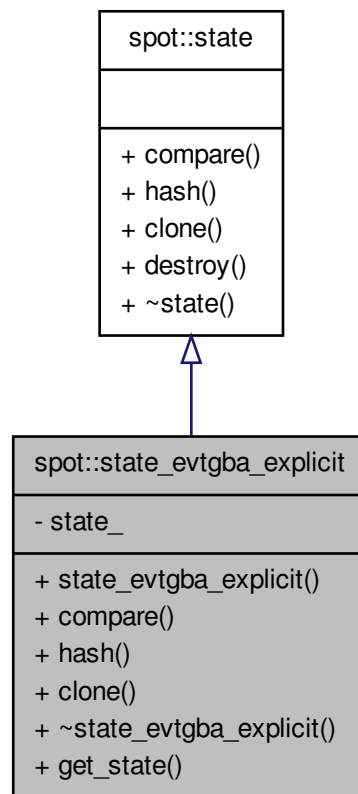
- `tgba/statebdd.hh`

7.134 `spot::state_evtgba_explicit` Class Reference

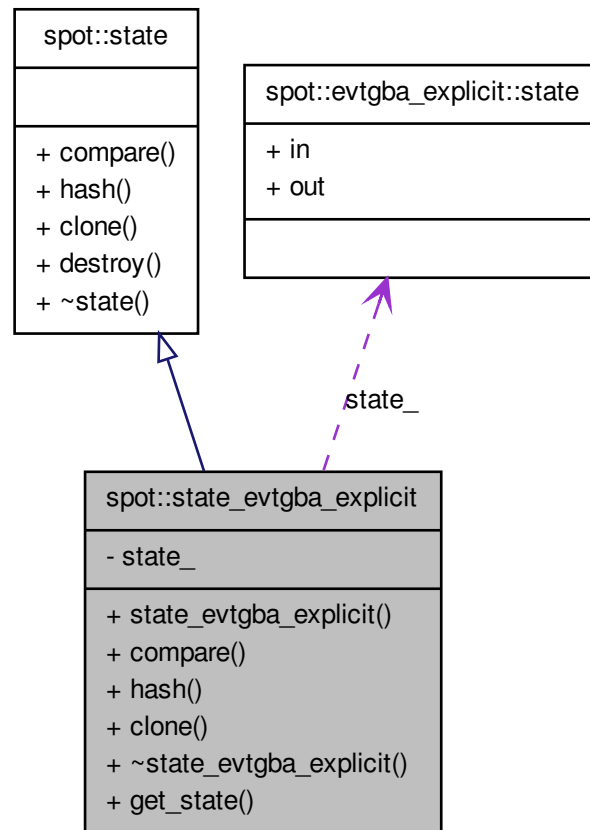
States used by `spot::tgba_evtgba_explicit`.

```
#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
- virtual void [destroy](#) () const

Release a state.

Private Attributes

- const [evtgba_explicit::state](#) * state_

7.134.1 Detailed Description

States used by `spot::tgba_evtgba_explicit`.

7.134.2 Constructor & Destructor Documentation

7.134.2.1 `spot::state_evtgba_explicit::state_evtgba_explicit (const evtgba_explicit::state * s)`
[inline]

7.134.2.2 `virtual spot::state_evtgba_explicit::~~state_evtgba_explicit ()` [inline, virtual]

7.134.3 Member Function Documentation

7.134.3.1 `virtual state_evtgba_explicit* spot::state_evtgba_explicit::clone () const` [virtual]

Duplicate a state.

Implements [spot::state](#).

7.134.3.2 `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](#).

7.134.3.3 `virtual void spot::state::destroy () const [inline, virtual, inherited]`

Release a state.

Methods from the `tgba` or `tgba_succ_iterator` always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine `destroy()` to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

7.134.3.4 `const evtgba_explicit::state* spot::state_evtgba_explicit::get_state () const`**7.134.3.5** `virtual size_t spot::state_evtgba_explicit::hash () const [virtual]`

Hash a state.

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

Note that the hash value is guaranteed to be unique for all equal states (in `compare()`'s sense) for only 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`.

7.134.4 Member Data Documentation**7.134.4.1** `const evtgba_explicit::state* spot::state_evtgba_explicit::state_ [private]`

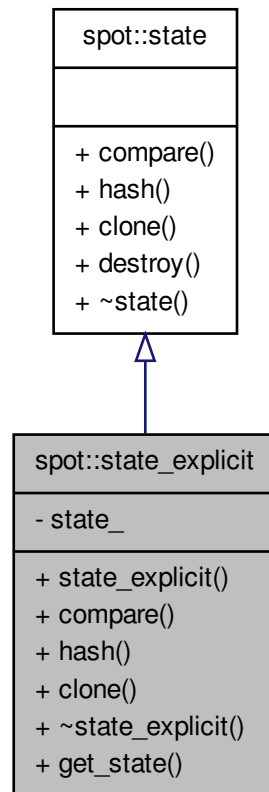
The documentation for this class was generated from the following file:

- [evtgba/explicit.hh](#)

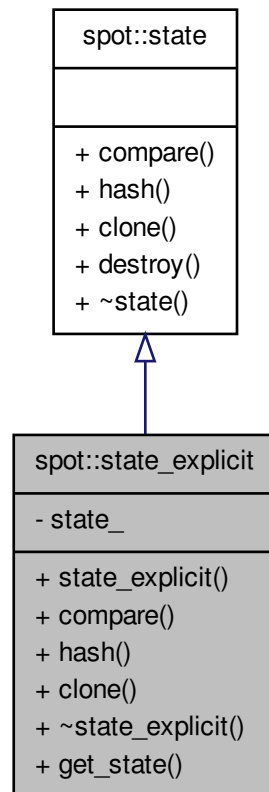
7.135 `spot::state_explicit` Class Reference

```
#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
- virtual void `destroy` () const
Release a state.

Private Attributes

- const [tgba_explicit::state](#) * `state_`

7.135.1 Detailed Description

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

7.135.2 Constructor & Destructor Documentation

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

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

7.135.3 Member Function Documentation

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

Duplicate a state.

Implements [spot::state](#).

7.135.3.2 `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](#).

7.135.3.3 `virtual void spot::state::destroy () const [inline, virtual, inherited]`

Release a state.

Methods from the `tgba` or [tgba_succ_iterator](#) always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update

your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine `destroy()` to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

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

Referenced by `spot::tgba_explicit_labelled< std::string, string_hash >::get_label()`.

7.135.3.5 `virtual size_t spot::state_explicit::hash () const [virtual]`

Hash a state.

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

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

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

Implements `spot::state`.

7.135.4 Member Data Documentation

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

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

- `tgba/tgbaexplicit.hh`

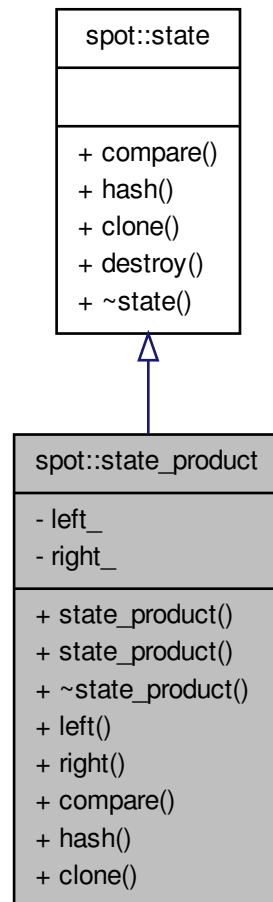
7.136 `spot::state_product` Class Reference

A state for `spot::tgba_product`.

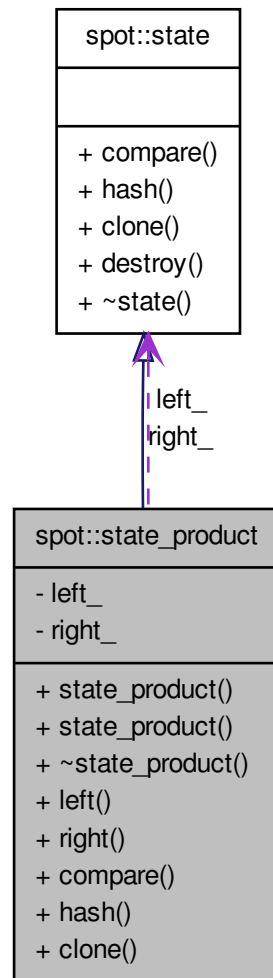
This state is in fact a pair of state: the state from the left automaton and that of the right.

```
#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.
- virtual void [destroy](#) () const
Release a state.

Private Attributes

- [state](#) * [left_](#)
State from the left automaton.
- [state](#) * [right_](#)
State from the right automaton.

7.136.1 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.

7.136.2 Constructor & Destructor Documentation

7.136.2.1 [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 destroyed on destruction.

7.136.2.2 [spot::state_product::state_product](#) (const [state_product](#) & *o*)

Copy constructor.

7.136.2.3 virtual [spot::state_product::~~state_product](#) () [virtual]

7.136.3 Member Function Documentation

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

Duplicate a state.

Implements [spot::state](#).

7.136.3.2 `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](#).

7.136.3.3 `virtual void spot::state::destroy () const [inline, virtual, inherited]`

Release a state.

Methods from the `tgba` or [tgba_succ_iterator](#) always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine [destroy\(\)](#) to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

7.136.3.4 `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 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](#).

7.136.3.5 `state* spot::state_product::left () const [inline]`

References `left_`.

7.136.3.6 `state* spot::state_product::right () const [inline]`

References `right_`.

7.136.4 Member Data Documentation

7.136.4.1 `state* spot::state_product::left_ [private]`

State from the left automaton.

Referenced by `left()`.

7.136.4.2 `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](#)

7.137 `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`

7.137.1 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;
```

7.137.2 Member Function Documentation

7.137.2.1 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](#)

7.138 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`

7.138.1 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;
```

7.138.2 Member Function Documentation

7.138.2.1 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](#)

7.139 `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`

7.139.1 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;
```

7.139.2 Member Function Documentation

7.139.2.1 `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:

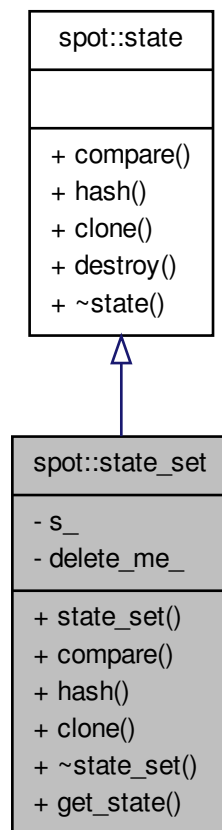
- [tgba/state.hh](#)

7.140 `spot::state_set` Class Reference

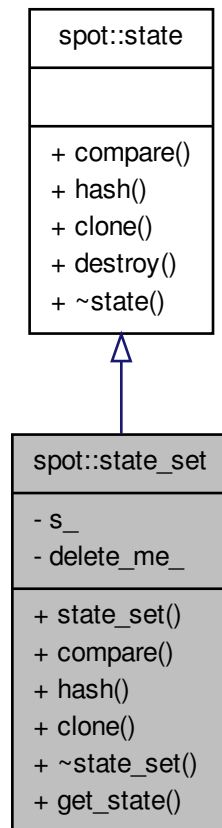
Set of states deriving from [spot::state](#).

```
#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
- virtual void `destroy` () const

Release a state.

Private Attributes

- const `taa_tgba::state_set` * `s_`
- bool `delete_me_`

7.140.1 Detailed Description

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

7.140.2 Constructor & Destructor Documentation

7.140.2.1 `spot::state_set::state_set (const taa_tgba::state_set * s, bool delete_me = false)`
[inline]

7.140.2.2 `virtual spot::state_set::~~state_set ()` [inline, virtual]

References `delete_me_`, and `s_`.

7.140.3 Member Function Documentation

7.140.3.1 `virtual state_set* spot::state_set::clone () const` [virtual]

Duplicate a state.

Implements `spot::state`.

7.140.3.2 `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`.

7.140.3.3 virtual void spot::state::destroy () const [inline, virtual, inherited]

Release a state.

Methods from the `tgba` or `tgba_succ_iterator` always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine `destroy()` to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

7.140.3.4 const taa_tgba::state_set* spot::state_set::get_state () const

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

7.140.3.5 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 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`.

7.140.4 Member Data Documentation

7.140.4.1 bool spot::state_set::delete_me_ [private]

Referenced by `~state_set()`.

7.140.4.2 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`

7.141 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`

7.141.1 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;
```

7.141.2 Member Function Documentation

7.141.2.1 `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](#)

7.142 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`

7.142.1 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;
```

7.142.2 Member Function Documentation

7.142.2.1 `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](#)

7.143 `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`

7.143.1 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;
```

7.143.2 Member Function Documentation

7.143.2.1 `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](#)

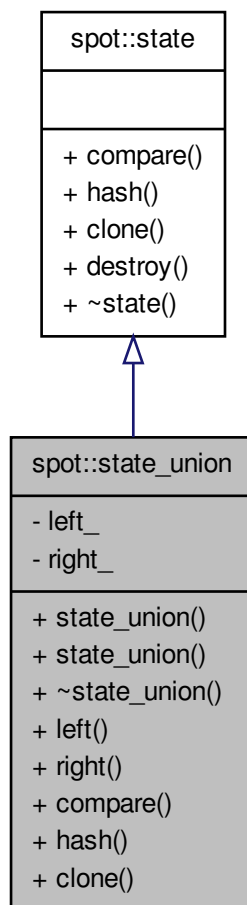
7.144 `spot::state_union` Class Reference

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

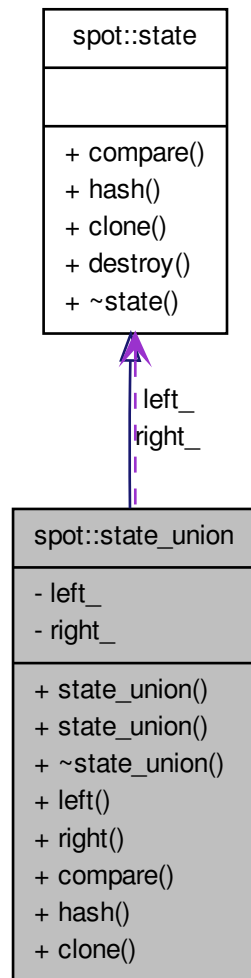
This state is in fact a pair. If the first member equals 0 and the second is different from 0, the state belongs to the left automaton. If the first member is different from 0 and the second is 0, the state belongs to the right automaton. If both members are 0, the state is the initial state.

```
#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.
- virtual void `destroy` () const
Release a state.

Private Attributes

- `state` * `left_`
- `state` * `right_`

7.144.1 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.

7.144.2 Constructor & Destructor Documentation

7.144.2.1 `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 destroyed on destruction.

7.144.2.2 `spot::state_union::state_union (const state_union & o)`

Copy constructor.

7.144.2.3 `virtual spot::state_union::~~state_union () [virtual]`

7.144.3 Member Function Documentation

7.144.3.1 virtual state_union* spot::state_union::clone () const [virtual]

Duplicate a state.

Implements [spot::state](#).

7.144.3.2 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](#).

7.144.3.3 virtual void spot::state::destroy () const [inline, virtual, inherited]

Release a state.

Methods from the `tgba` or [tgba_succ_iterator](#) always return a new state that you should deallocate with this function. Before Spot 0.7, you had to "delete" your state directly. Starting with Spot 0.7, you update your code to this function instead (which simply calls "delete"). In a future version, some subclasses will redefine [destroy\(\)](#) to allow better memory management (e.g. no memory allocation for explicit automata).

Referenced by `spot::power_map::canonicalize()`, `spot::shared_state_deleter()`, and `spot::power_map::~~power_map()`.

7.144.3.4 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 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](#).

7.144.3.5 state* spot::state_union::left () const [inline]

References left_.

7.144.3.6 state* spot::state_union::right () const [inline]

References right_.

7.144.4 Member Data Documentation

7.144.4.1 state* spot::state_union::left_ [private]

Does the state belongs to the left automaton ?

Referenced by left().

7.144.4.2 state* spot::state_union::right_ [private]

Does the state belongs to the right automaton ?

Referenced by right().

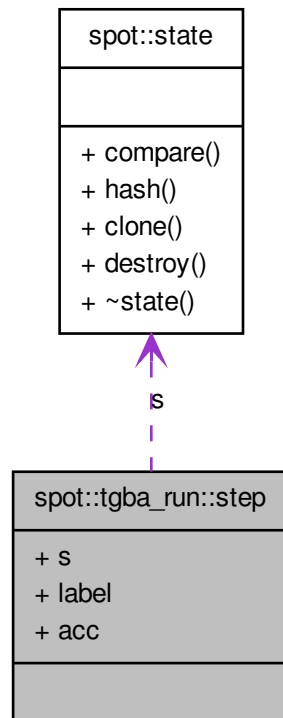
The documentation for this class was generated from the following file:

- [tgba/tgbaunion.hh](#)

7.145 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](#)

7.145.1 Member Data Documentation

7.145.1.1 bdd spot::tgba_run::step::acc

7.145.1.2 bdd spot::tgba_run::step::label

7.145.1.3 `const state* spot::tgba_run::step::s`

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

- [tgbaalgos/emptiness.hh](#)

7.146 `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`

7.146.1 Detailed Description

A hash function for strings.

7.146.2 Member Function Documentation

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

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

- [misc/hash.hh](#)

7.147 `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_`

7.147.1 Constructor & Destructor Documentation

7.147.1.1 spot::ltl::succ_iterator::succ_iterator (const nfa::state::const_iterator s) [inline]

7.147.2 Member Function Documentation

7.147.2.1 bool spot::ltl::succ_iterator::operator!= (const succ_iterator & rhs) const [inline]

References i_.

7.147.2.2 const nfa::transition* spot::ltl::succ_iterator::operator* () const [inline]

References i_.

7.147.2.3 void spot::ltl::succ_iterator::operator++ () [inline]

References i_.

7.147.3 Member Data Documentation

7.147.3.1 nfa::state::const_iterator spot::ltl::succ_iterator::i_ [private]

Referenced by operator!=(), operator*(), and operator++().

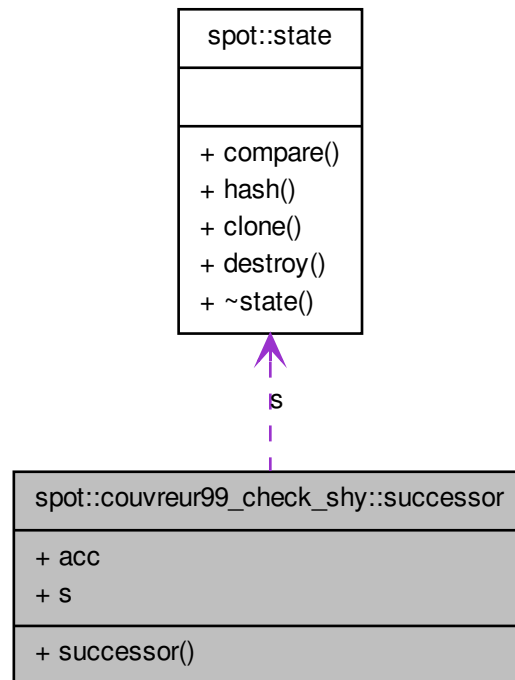
The documentation for this class was generated from the following file:

- ltlast/[nfa.hh](#)

7.148 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](#)

7.148.1 Constructor & Destructor Documentation

7.148.1.1 [spot::couvreur99_check_shy::successor::successor](#) (bdd *acc*, const [spot::state](#) * *s*)
[inline]

7.148.2 Member Data Documentation

7.148.2.1 bdd spot::couvreur99_check_shy::successor::acc

7.148.2.2 const spot::state* spot::couvreur99_check_shy::successor::s

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

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

7.149 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_](#)

7.149.1 Member Typedef Documentation

7.149.1.1 `typedef std::map<const std::string, const symbol*> spot::symbol::map`
`[protected]`

7.149.2 Constructor & Destructor Documentation

7.149.2.1 `spot::symbol::symbol (const std::string * name)` `[protected]`

7.149.2.2 `spot::symbol::~symbol ()` `[protected]`

7.149.2.3 `spot::symbol::symbol (const symbol &)` `[private]`

7.149.3 Member Function Documentation

7.149.3.1 `static std::ostream& spot::symbol::dump_instances (std::ostream & os)` `[static]`

List all instances of atomic propositions. For debugging.

7.149.3.2 `static const symbol* spot::symbol::instance (const std::string & name)` `[static]`

7.149.3.3 `static unsigned spot::symbol::instance_count ()` `[static]`

Number of instantiated atomic propositions. For debugging.

7.149.3.4 `const std::string& spot::symbol::name () const`

7.149.3.5 `void spot::symbol::ref () const`

Referenced by spot::rsymbol::rsymbol().

7.149.3.6 `int spot::symbol::ref_count_ () const` **[protected]**

7.149.3.7 `void spot::symbol::unref () const`

Referenced by spot::rsymbol::~rsymbol().

7.149.4 Member Data Documentation

7.149.4.1 `map spot::symbol::instances_` **[static, protected]**

7.149.4.2 `const std::string* spot::symbol::name_` **[private]**

Undefined.

7.149.4.3 `int spot::symbol::refs_` **[mutable, private]**

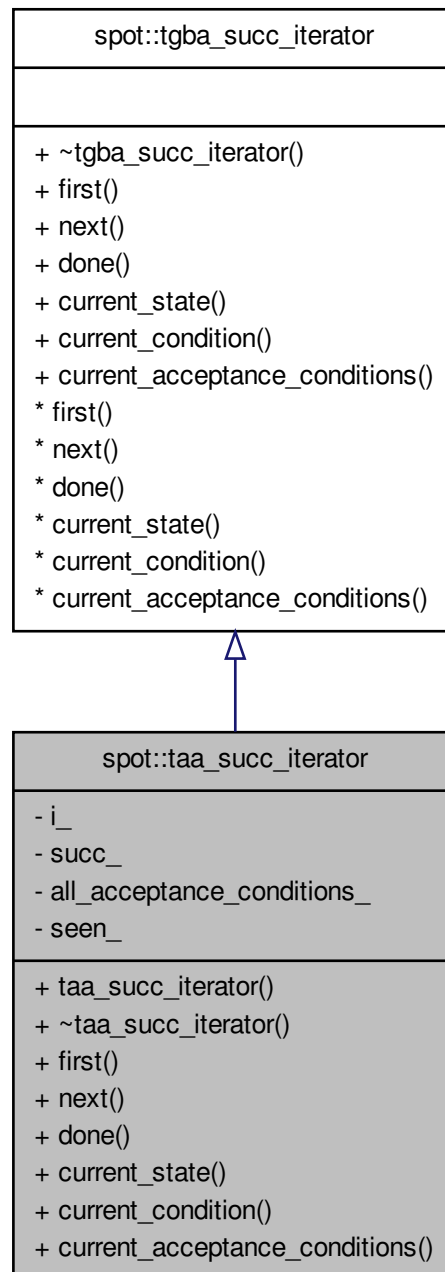
The documentation for this class was generated from the following file:

- [evtgba/symbol.hh](#)

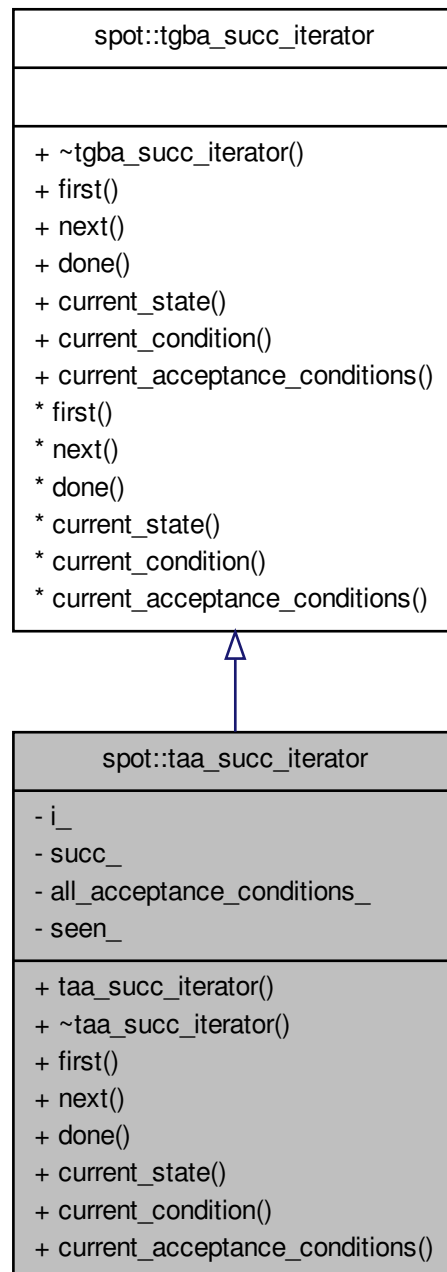
7.150 spot::taa_succ_iterator Class Reference

```
#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_](#)

7.150.1 Member Typedef Documentation

7.150.1.1 typedef std::vector<iterator_pair> spot::taa_succ_iterator::bounds_t [private]

7.150.1.2 `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.

7.150.1.3 `typedef std::pair<iterator, iterator> spot::taa_succ_iterator::iterator_pair [private]`

7.150.1.4 `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]`

7.150.2 Constructor & Destructor Documentation

7.150.2.1 `spot::taa_succ_iterator::taa_succ_iterator (const taa_tgba::state_set * s, bdd all_acc)`

7.150.2.2 `virtual spot::taa_succ_iterator::~~taa_succ_iterator () [virtual]`

7.150.3 Member Function Documentation

7.150.3.1 `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](#).

7.150.3.2 `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](#).

7.150.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

Implements [spot::tgba_succ_iterator](#).

7.150.3.4 `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](#).

7.150.3.5 `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](#).

7.150.3.6 `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](#).

7.150.4 Member Data Documentation

7.150.4.1 `bdd spot::taa_succ_iterator::all_acceptance_conditions_ [private]`

7.150.4.2 `std::vector<taa_tgba::transition*>::const_iterator spot::taa_succ_iterator::i_` `[private]`

7.150.4.3 `seen_map spot::taa_succ_iterator::seen_` `[private]`

7.150.4.4 `std::vector<taa_tgba::transition*> spot::taa_succ_iterator::succ_` `[private]`

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

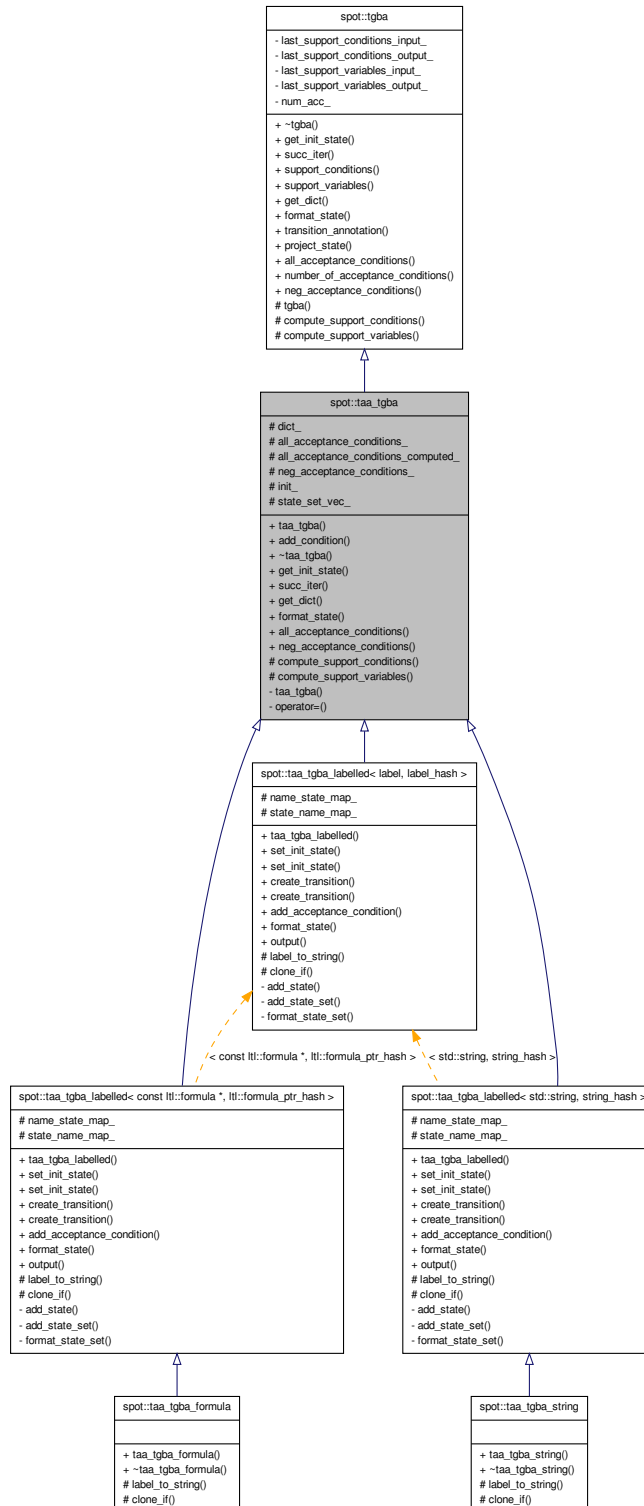
- [tgba/taatgba.hh](#)

7.151 spot::taa_tgba Class Reference

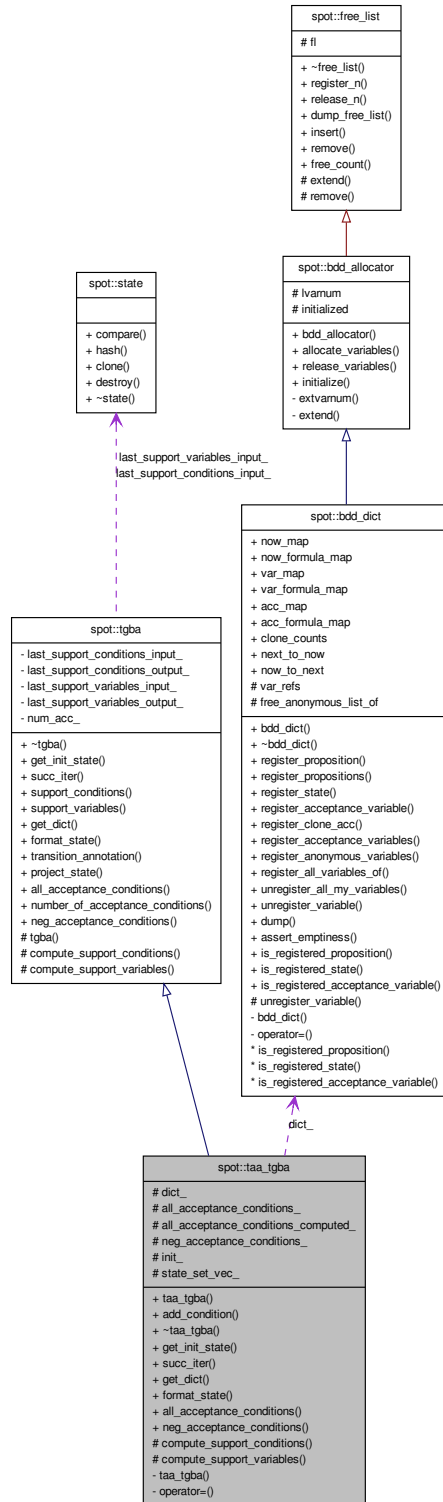
A self-loop Transition-based Alternating Automaton (TAA) which is seen as a TGBA (abstract class, see below).

```
#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)

7.151.1 Detailed Description

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

7.151.2 Member Typedef Documentation

7.151.2.1 typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected]

7.151.2.2 typedef std::list<transition*> spot::taa_tgba::state

7.151.2.3 typedef std::set<state*> spot::taa_tgba::state_set

7.151.3 Constructor & Destructor Documentation

7.151.3.1 `spot::taa_tgba::taa_tgba (bdd_dict * dict)`

7.151.3.2 `virtual spot::taa_tgba::~~taa_tgba () [virtual]`

TGBA interface.

7.151.3.3 `spot::taa_tgba::taa_tgba (const taa_tgba & other) [private]`

7.151.4 Member Function Documentation

7.151.4.1 `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f)`

7.151.4.2 `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](#).

7.151.4.3 `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](#).

7.151.4.4 `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](#).

7.151.4.5 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 >](#).

7.151.4.6 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](#).

7.151.4.7 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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.151.4.8 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](#).

7.151.4.9 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

7.151.4.10 `taa_tgba& spot::taa_tgba::operator= (const taa_tgba & other) [private]`

7.151.4.11 `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 destroyed by the caller.

Reimplemented in `spot::tgba_product`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.151.4.12 `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 destroy 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`.

7.151.4.13 `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.

7.151.4.14 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.

7.151.4.15 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`, and `spot::tgba_scc`.

7.151.5 Member Data Documentation**7.151.5.1 bdd spot::taa_tgba::all_acceptance_conditions_ [mutable, protected]****7.151.5.2 bool spot::taa_tgba::all_acceptance_conditions_computed_ [mutable, protected]**

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

7.151.5.3 bdd_dict* spot::taa_tgba::dict_ [protected]

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

7.151.5.4 taa_tgba::state_set* spot::taa_tgba::init_ [protected]

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

7.151.5.5 bdd spot::taa_tgba::neg_acceptance_conditions_ [protected]

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

7.151.5.6 ss_vec spot::taa_tgba::state_set_vec_ [protected]

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

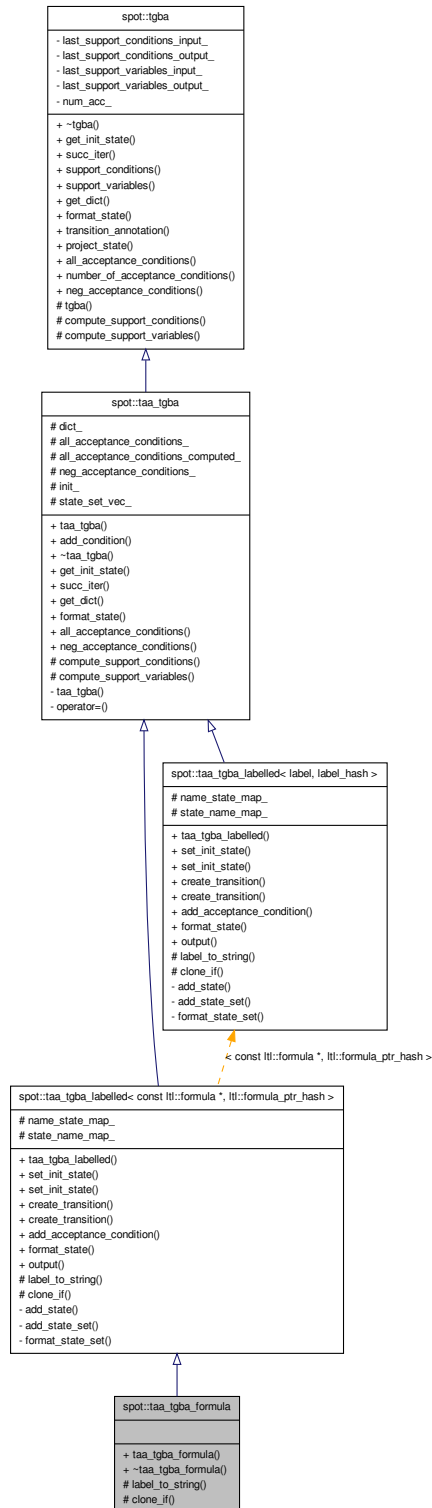
The documentation for this class was generated from the following file:

- [tgba/taatgba.hh](#)

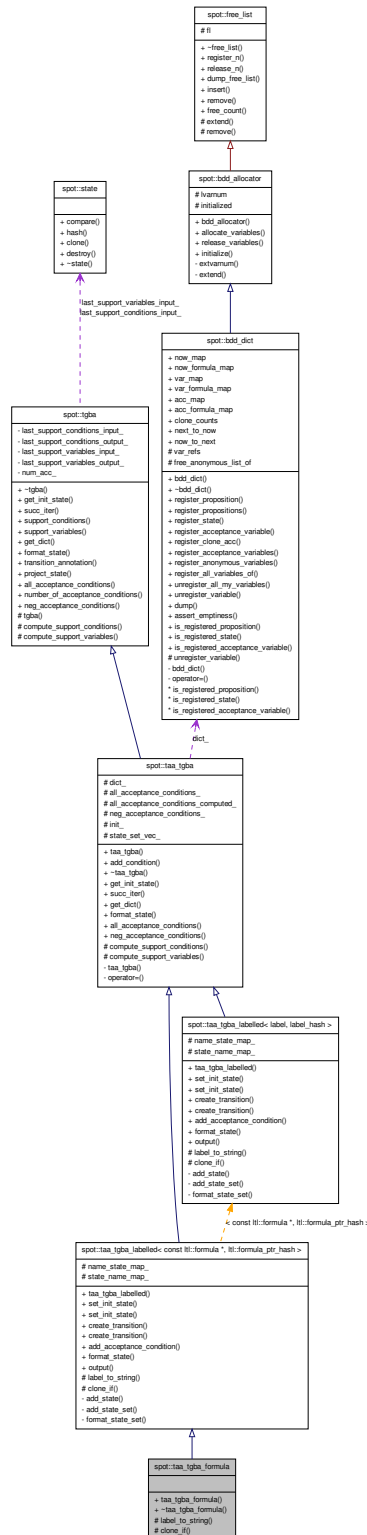
7.152 spot::taa_tgba_formula Class Reference

```
#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_](#)

7.152.1 Member Typedef Documentation

- 7.152.1.1** typedef const [ltl::formula](#) * [spot::taa_tgba_labelled](#)< const [ltl::formula](#) *, [ltl::formula_ptr_hash](#) >::[label_t](#) [[protected](#), [inherited](#)]

7.152.1.2 `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]`

7.152.1.3 `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]`

7.152.1.4 `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,
inherited]`

7.152.1.5 `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

7.152.1.6 `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

7.152.2 Constructor & Destructor Documentation

7.152.2.1 `spot::taa_tgba_formula::taa_tgba_formula (bdd_dict * dict) [inline]`

7.152.2.2 `spot::taa_tgba_formula::~~taa_tgba_formula ()`

7.152.3 Member Function Documentation

7.152.3.1 `void spot::taa_tgba_labelled< const ltl::formula * , ltl::formula_ptr_hash
>::add_acceptance_condition (transition * t, const ltl::formula * f) [inline,
inherited]`

7.152.3.2 `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f)
[inherited]`

7.152.3.3 `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](#).

7.152.3.4 `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 >](#).

7.152.3.5 `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](#).

7.152.3.6 `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](#).

7.152.3.7 `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]`

7.152.3.8 `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]`

7.152.3.9 `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](#).

7.152.3.10 `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](#).

7.152.3.11 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.152.3.12 `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 >](#).

7.152.3.13 `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](#).

7.152.3.14 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
[virtual, inherited]

The number of acceptance conditions.

7.152.3.15 `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.

7.152.3.16 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.152.3.17 `void spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash`
`>::set_init_state (const std::vector< const ltl::formula * > & s)` [inline,
inherited]

7.152.3.18 `void spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash`
`>::set_init_state (const const ltl::formula * & s)` [inline, inherited]

7.152.3.19 `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 destroy 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`.

7.152.3.20 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.

7.152.3.21 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.

7.152.3.22 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](#), and [spot::tgba_scc](#).

7.152.4 Member Data Documentation

7.152.4.1 `bdd spot::taa_tgba::all_acceptance_conditions_ [mutable, protected, inherited]`

7.152.4.2 `bool spot::taa_tgba::all_acceptance_conditions_computed_ [mutable, protected, inherited]`

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

7.152.4.3 `bdd_dict* spot::taa_tgba::dict_ [protected, inherited]`

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

7.152.4.4 `taa_tgba::state_set* spot::taa_tgba::init_ [protected, inherited]`

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

7.152.4.5 `ns_map spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::name_state_map_ [protected, inherited]`

7.152.4.6 `bdd spot::taa_tgba::neg_acceptance_conditions_ [protected, inherited]`

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

7.152.4.7 `sn_map spot::taa_tgba_labelled< const ltl::formula *, ltl::formula_ptr_hash >::state_name_map_ [protected, inherited]`

7.152.4.8 `ss_vec spot::taa_tgba::state_set_vec_` `[protected, inherited]`

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

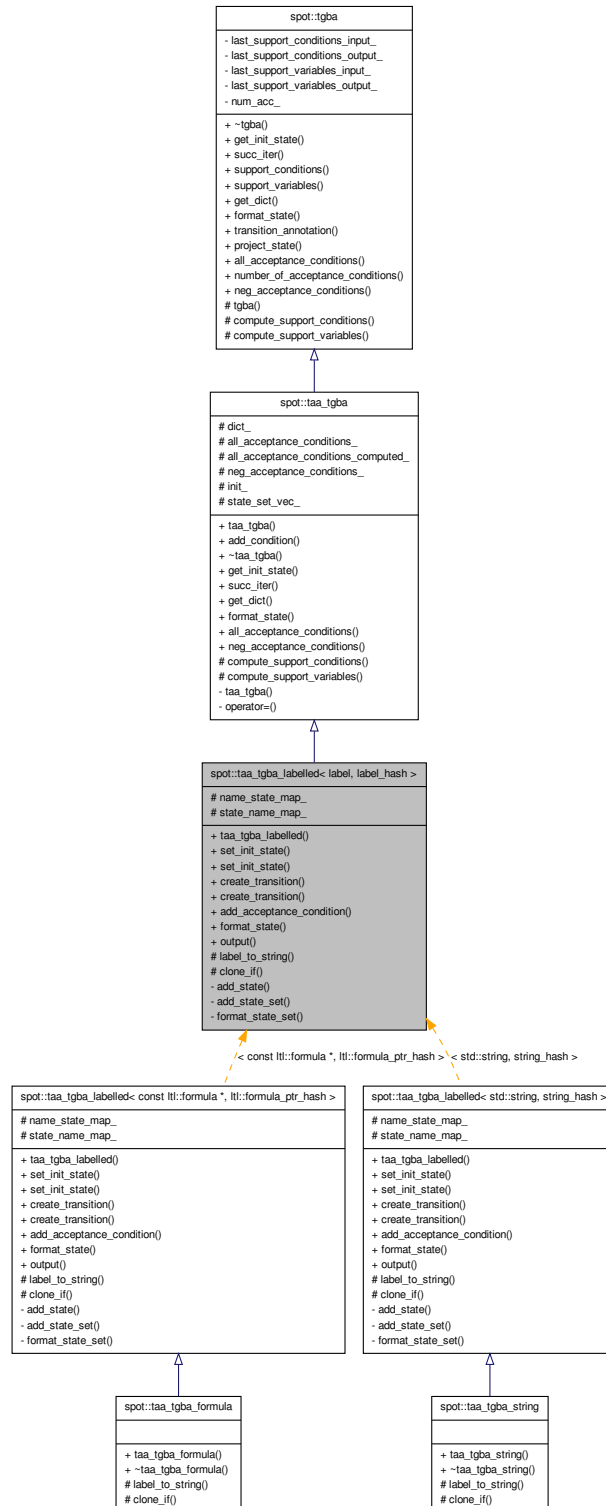
The documentation for this class was generated from the following file:

- `tgba/taatgba.hh`

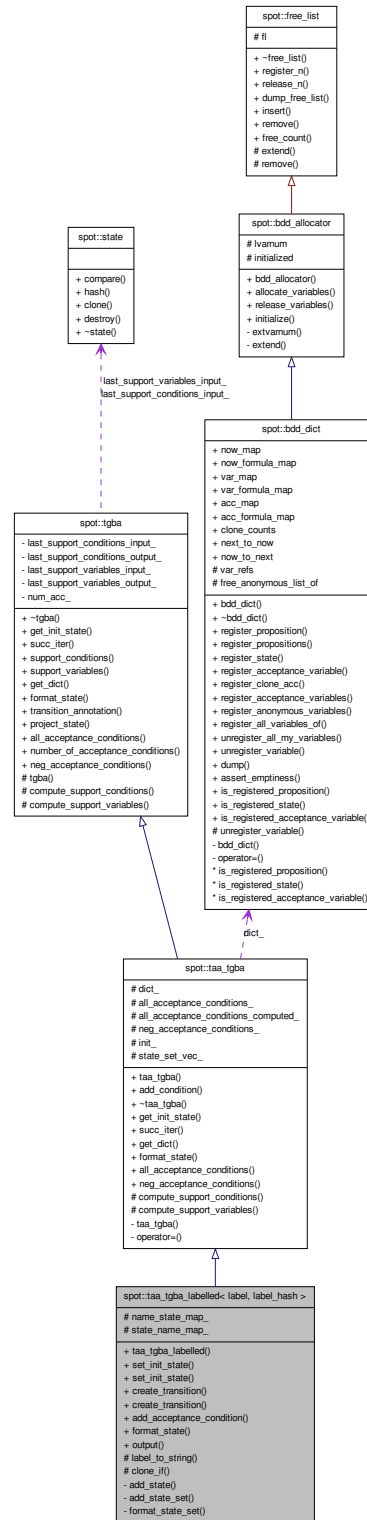
7.153 `spot::taa_tgba_labelled< label, label_hash >` Class Template Reference

```
#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

7.153.1 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.

7.153.2 Member Typedef Documentation

7.153.2.1 `template<typename label, typename label_hash> typedef label
spot::taa_tgba_labelled< label, label_hash >::label_t [protected]`

7.153.2.2 `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]`

7.153.2.3 `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]`

7.153.2.4 `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,
inherited]`

7.153.2.5 `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

7.153.2.6 `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

7.153.3 Constructor & Destructor Documentation

7.153.3.1 `template<typename label, typename label_hash> spot::taa_tgba_labelled< label,
label_hash >::taa_tgba_labelled (bdd_dict * dict) [inline]`

7.153.4 Member Function Documentation

7.153.4.1 `template<typename label, typename label_hash> void spot::taa_tgba_labelled< label,
label_hash >::add_acceptance_condition (transition * t, const ltl::formula * f)
[inline]`

7.153.4.2 `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f)`
`[inherited]`

7.153.4.3 `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()`.

7.153.4.4 `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()`.

7.153.4.5 `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`.

7.153.4.6 `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_formula`.

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

7.153.4.7 `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`.

7.153.4.8 `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`.

7.153.4.9 `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()`.

7.153.4.10 `template<typename label, typename label_hash> transition* spot::taa_tgba_labelled<
label, label_hash >::create_transition (const label & s, const label & d) [inline]`

7.153.4.11 `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`.

7.153.4.12 `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()`.

7.153.4.13 `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](#).

7.153.4.14 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.153.4.15 `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_formula](#).

Referenced by `spot::taa_tgba_labelled< std::string, string_hash >::format_state_set()`, and `spot::taa_tgba_labelled< std::string, string_hash >::output()`.

7.153.4.16 `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](#).

7.153.4.17 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const
[virtual, inherited]`

The number of acceptance conditions.

7.153.4.18 `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.

7.153.4.19 `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 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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.153.4.20 `template<typename label, typename label_hash> void spot::taa_tgba_labelled< label, label_hash >::set_init_state (const std::vector< label > & s) [inline]`

7.153.4.21 `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()`.

7.153.4.22 `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 destroy 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](#).

7.153.4.23 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.

7.153.4.24 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.

7.153.4.25 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](#), and [spot::tgba_scc](#).

7.153.5 Member Data Documentation

7.153.5.1 `bdd spot::taa_tgba::all_acceptance_conditions_` `[mutable, protected, inherited]`

7.153.5.2 `bool spot::taa_tgba::all_acceptance_conditions_computed_` `[mutable, protected, inherited]`

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

7.153.5.3 `bdd_dict* spot::taa_tgba::dict_` `[protected, inherited]`

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

7.153.5.4 `taa_tgba::state_set* spot::taa_tgba::init_` `[protected, inherited]`

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

7.153.5.5 `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()`.

7.153.5.6 `bdd spot::taa_tgba::neg_acceptance_conditions_` `[protected, inherited]`

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

7.153.5.7 `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()`.

7.153.5.8 `ss_vec spot::taa_tgba::state_set_vec_` `[protected, inherited]`

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

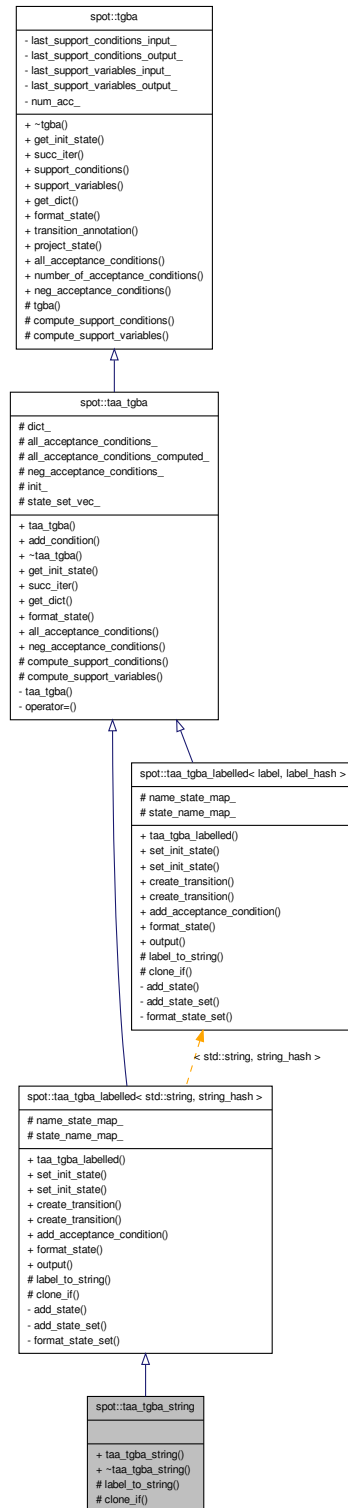
The documentation for this class was generated from the following file:

- tgba/[taatgba.hh](#)

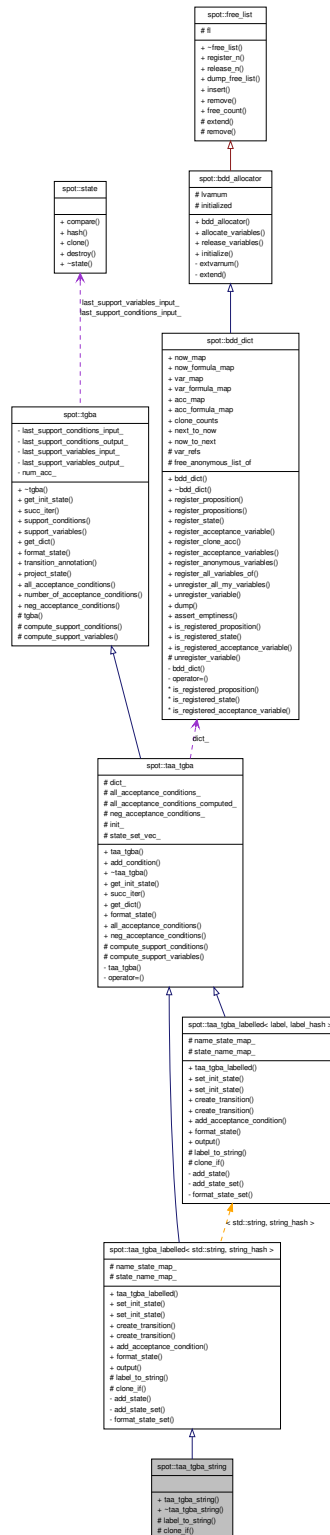
7.154 spot::taa_tgba_string Class Reference

```
#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
- virtual std::string [clone_if](#) (const std::string &label) const
- 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_](#)

7.154.1 Member Typedef Documentation

7.154.1.1 typedef std::string spot::taa_tgba_labelled< std::string , string_hash >::label_t
[protected, inherited]

7.154.1.2 typedef Sgi::hash_map< const std::string , taa_tgba::state*, string_hash >
spot::taa_tgba_labelled< std::string , string_hash >::ns_map [protected,
inherited]

7.154.1.3 `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]`

7.154.1.4 `typedef std::vector<taa_tgba::state_set*> spot::taa_tgba::ss_vec [protected,
inherited]`

7.154.1.5 `typedef std::list<transition*> spot::taa_tgba::state [inherited]`

7.154.1.6 `typedef std::set<state*> spot::taa_tgba::state_set [inherited]`

7.154.2 Constructor & Destructor Documentation

7.154.2.1 `spot::taa_tgba_string::taa_tgba_string (bdd_dict * dict) [inline]`

7.154.2.2 `spot::taa_tgba_string::~~taa_tgba_string ()`

7.154.3 Member Function Documentation

7.154.3.1 `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()`.

7.154.3.2 `void spot::taa_tgba::add_condition (transition * t, const ltl::formula * f) [inherited]`

7.154.3.3 `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](#).

7.154.3.4 `virtual label_t spot::taa_tgba_labelled< std::string, string_hash >::clone_if (const label_t & lbl) const [protected, pure virtual, inherited]`

Clone the label if necessary to assure it is owned by this, avoiding memory issues when label is a pointer.

7.154.3.5 `virtual std::string spot::taa_tgba_string::clone_if (const std::string & label) const [protected, virtual]`

7.154.3.6 `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](#).

7.154.3.7 `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](#).

7.154.3.8 `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](#).

7.154.3.9 `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()`.

7.154.3.10 `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()`.

7.154.3.11 `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`.

7.154.3.12 `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 `destroy` it when no longer needed.

Implements `spot::tgba`.

7.154.3.13 `virtual std::string spot::taa_tgba_labelled< std::string, string_hash >::label_to_string (const label_t & lbl) const [protected, pure virtual, inherited]`

Return a label as a string.

7.154.3.14 `virtual std::string spot::taa_tgba_string::label_to_string (const std::string & label) const [protected, virtual]`

7.154.3.15 `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](#).

7.154.3.16 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

7.154.3.17 `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_`.

7.154.3.18 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.154.3.19 `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_`.

7.154.3.20 `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()`.

7.154.3.21 `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 destroy 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`.

7.154.3.22 `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 `bdd true`, 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.

7.154.3.23 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.

7.154.3.24 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](#), and [spot::tgba_scc](#).

7.154.4 Member Data Documentation**7.154.4.1 bdd spot::taa_tgba::all_acceptance_conditions_ [mutable, protected, inherited]****7.154.4.2 bool spot::taa_tgba::all_acceptance_conditions_computed_ [mutable, protected, inherited]**

Referenced by [spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition\(\)](#).

7.154.4.3 bdd_dict* spot::taa_tgba::dict_ [protected, inherited]

Referenced by [spot::taa_tgba_labelled< std::string, string_hash >::add_acceptance_condition\(\)](#).

7.154.4.4 taa_tgba::state_set* spot::taa_tgba::init_ [protected, inherited]

Referenced by [spot::taa_tgba_labelled< std::string, string_hash >::set_init_state\(\)](#).

7.154.4.5 `ns_map spot::taa_tgba_labelled< std::string, string_hash >::name_state_map_`
`[protected, inherited]`

7.154.4.6 `bdd spot::taa_tgba::neg_acceptance_conditions_` `[protected, inherited]`

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

7.154.4.7 `sn_map spot::taa_tgba_labelled< std::string, string_hash >::state_name_map_`
`[protected, inherited]`

7.154.4.8 `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](#)

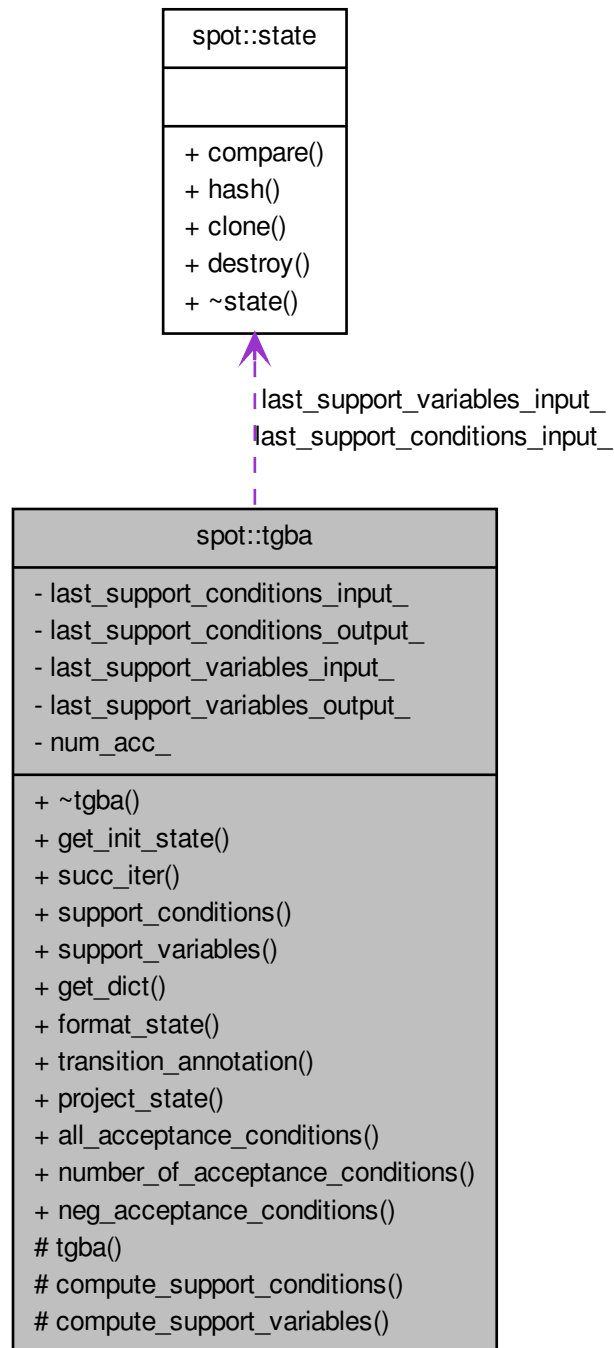
7.155 spot::tgba Class Reference

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>
```


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_](#)

7.155.1 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.

7.155.2 Constructor & Destructor Documentation**7.155.2.1 `spot::tgba::tgba ()` [protected]****7.155.2.2 `virtual spot::tgba::~~tgba ()` [virtual]****7.155.3 Member Function Documentation****7.155.3.1 `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](#).

7.155.3.2 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`.

7.155.3.3 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`.

7.155.3.4 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_explicit_number`, `spot::tgba_kv_complement`, `spot::tgba_product`, `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 >`.

7.155.3.5 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_safra_complement`, `spot::tgba_scc`, `spot::tgba_sgba_proxy`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.155.3.6 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 `destroy` 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_product_init](#), [spot::tgba_safra_complement](#), [spot::tgba_scc](#), [spot::tgba_sgba_proxy](#), [spot::tgba_tba_proxy](#), [spot::tgba_sba_proxy](#), and [spot::tgba_union](#).

7.155.3.7 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_safra_complement](#), [spot::tgba_scc](#), [spot::tgba_sgba_proxy](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.155.3.8 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual]

The number of acceptance conditions.

7.155.3.9 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 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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.155.3.10 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 destroy 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`.

7.155.3.11 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.

7.155.3.12 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.

7.155.3.13 virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const [virtual]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented in [spot::tgba_product](#), and [spot::tgba_scc](#).

7.155.4 Member Data Documentation

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

7.155.4.2 `bdd spot::tgba::last_support_conditions_output_` `[mutable, private]`

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

7.155.4.4 `bdd spot::tgba::last_support_variables_output_` `[mutable, private]`

7.155.4.5 `int spot::tgba::num_acc_` `[mutable, private]`

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

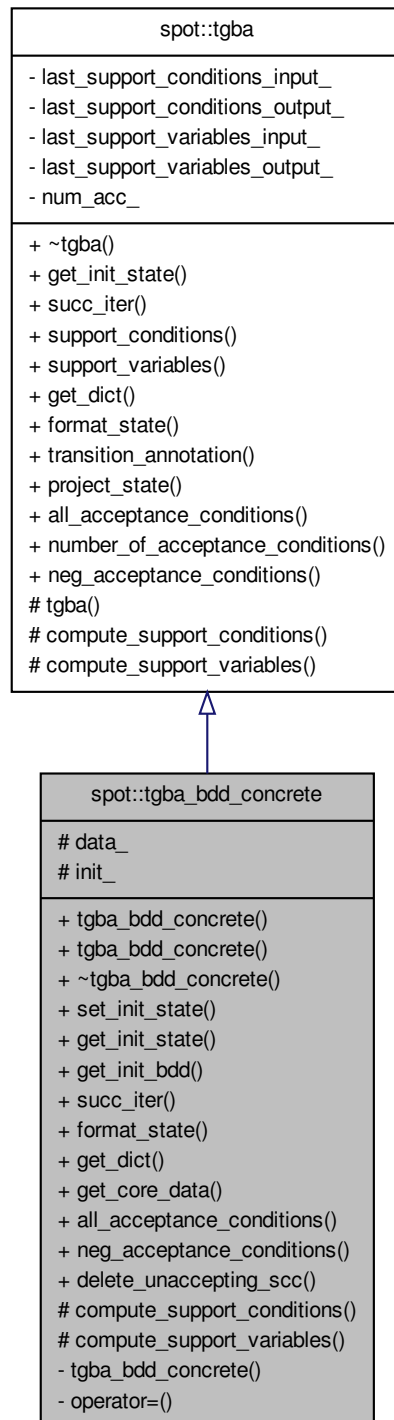
- [tgba/tgba.hh](#)

7.156 `spot::tgba_bdd_concrete` Class Reference

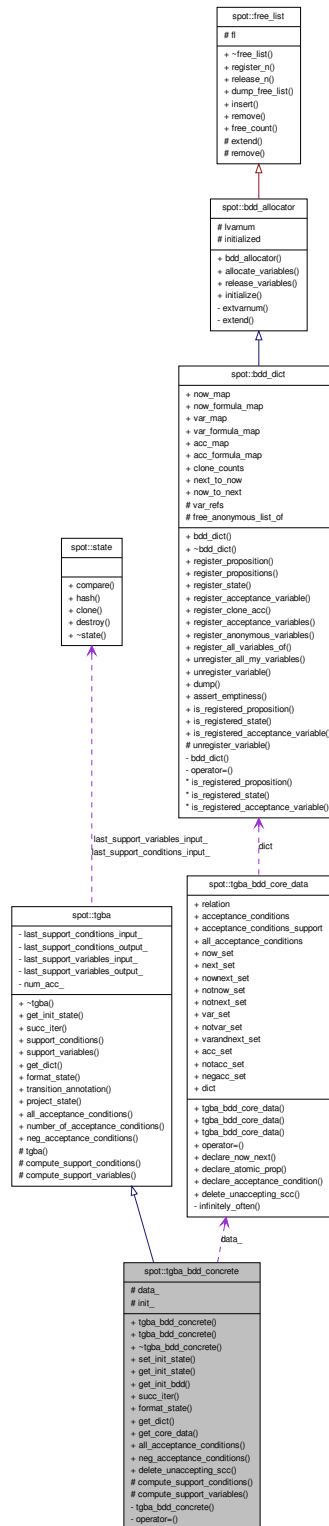
A concrete [spot::tgba](#) implemented using BDDs.

```
#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](#) &)

7.156.1 Detailed Description

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

7.156.2 Constructor & Destructor Documentation

7.156.2.1 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.

7.156.2.2 spot::tgba_bdd_concrete::tgba_bdd_concrete (const tgba_bdd_factory & fact, bdd init)

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

7.156.2.3 virtual spot::tgba_bdd_concrete::~~tgba_bdd_concrete () [virtual]

7.156.2.4 spot::tgba_bdd_concrete::tgba_bdd_concrete (const tgba_bdd_concrete &)
[private]

7.156.3 Member Function Documentation

7.156.3.1 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](#).

7.156.3.2 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](#).

7.156.3.3 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](#).

7.156.3.4 void spot::tgba_bdd_concrete::delete_unaccepting_scc ()

Delete SCCs (Strongly Connected Components) from the TGBA which cannot be accepting.

7.156.3.5 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](#).

7.156.3.6 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.

7.156.3.7 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](#).

7.156.3.8 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());
s->destroy();
```

7.156.3.9 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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.156.3.10 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](#).

7.156.3.11 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

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

7.156.3.13 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

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

Set the initial state.

7.156.3.15 `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 destroy 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](#).

7.156.3.16 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.

7.156.3.17 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.

7.156.3.18 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 is the empty string.

Parameters

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented in [spot::tgba_product](#), and [spot::tgba_scc](#).

7.156.4 Member Data Documentation

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

Core data associated to the automaton.

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

Initial state.

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

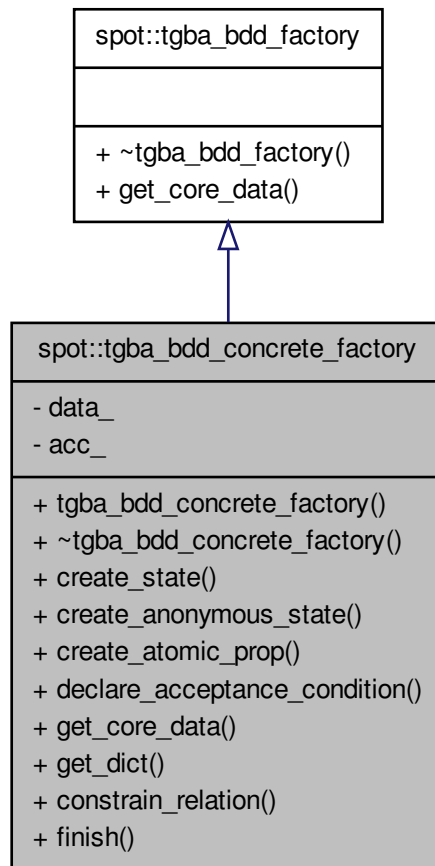
- [tgba/tgababddconcrete.hh](#)

7.157 `spot::tgba_bdd_concrete_factory` Class Reference

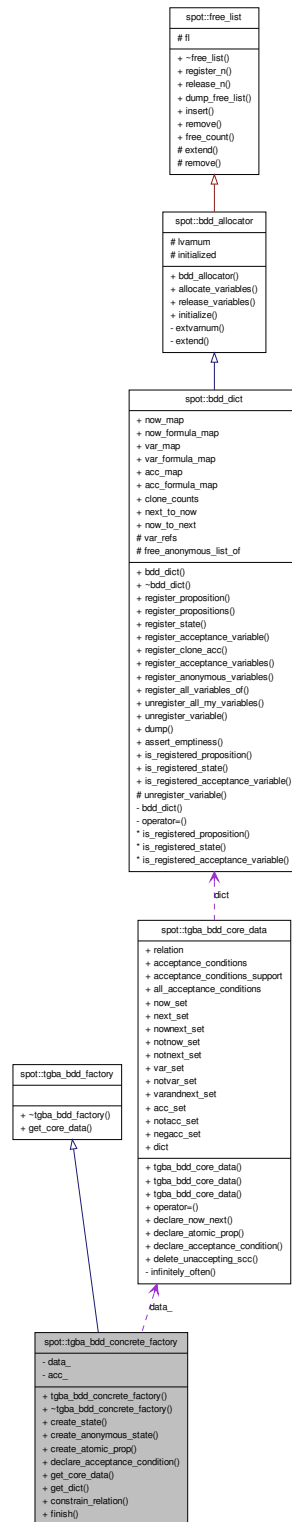
Helper class to build a [spot::tgba_bdd_concrete](#) object.

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

Inheritance diagram for spot::tgba_bdd_concrete_factory:



Collaboration diagram for spot::tgba_bdd_concrete_factory:



Public Member Functions

- [tgba_bdd_concrete_factory](#) (bdd_dict *dict)
- virtual [~tgba_bdd_concrete_factory](#) ()
- int [create_state](#) (const ltl::formula *f)
- int [create_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

Get the core data for the new automata.

- bdd_dict * [get_dict](#) () const
- void [constrain_relation](#) (bdd new_rel)

Add a new constraint to the relation.

- void [finish](#) ()

Perform final computations before the relation can be used.

Private Types

- typedef Sgi::hash_map< const ltl::formula *, bdd, ltl::formula_ptr_hash > [acc_map_](#)

Private Attributes

- [tgba_bdd_core_data](#) [data_](#)
Core data for the new automata.
- [acc_map_](#) [acc_](#)
BDD associated to each acceptance condition.

7.157.1 Detailed Description

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

7.157.2 Member Typedef Documentation

- 7.157.2.1** typedef Sgi::hash_map<const ltl::formula*, bdd, ltl::formula_ptr_hash>
spot::tgba_bdd_concrete_factory::acc_map_ [private]

7.157.3 Constructor & Destructor Documentation

- 7.157.3.1** spot::tgba_bdd_concrete_factory::tgba_bdd_concrete_factory (bdd_dict * dict)

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

7.157.4 Member Function Documentation

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

Add a new constraint to the relation.

7.157.4.2 `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 $v+1$.

7.157.4.3 `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()`.

7.157.4.4 `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()`.

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

Declare an acceptance condition.

Formula such as 'f U g' or 'F g' make the promise that 'g' will be fulfilled eventually. So once one of this formula has been translated into a BDD, we use [declare_acceptance_condition\(\)](#) to associate all other states to the acceptance set of 'g'.

Parameters

b a BDD indicating which variables are in the acceptance set

a the formula associated

7.157.4.6 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\(\)](#).

7.157.4.7 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](#).

7.157.4.8 bdd_dict* spot::tgba_bdd_concrete_factory::get_dict () const

7.157.5 Member Data Documentation

7.157.5.1 acc_map spot::tgba_bdd_concrete_factory::acc_ [private]

BDD associated to each acceptance condition.

7.157.5.2 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:

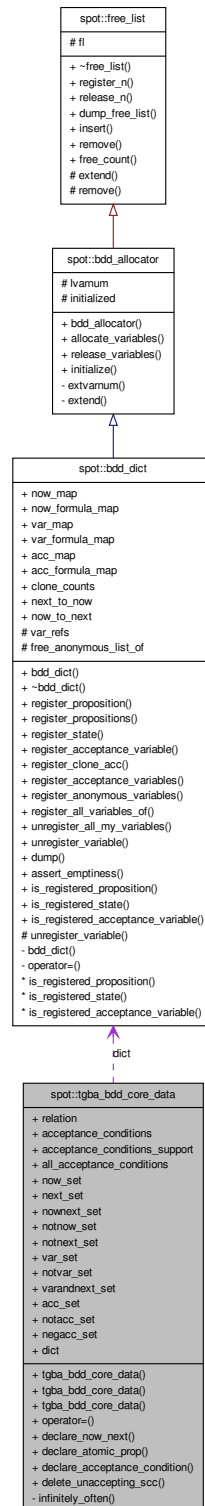
- [tgba/tgabddconcretefactory.hh](#)

7.158 spot::tgba_bdd_core_data Struct Reference

Core data for a TGBA encoded using BDDs.

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

Collaboration diagram for spot::tgba_bdd_core_data:



Public Member Functions

- [tgba_bdd_core_data](#) ([bdd_dict](#) *dict)
Default constructor.
- [tgba_bdd_core_data](#) (const [tgba_bdd_core_data](#) ©)
Copy constructor.
- [tgba_bdd_core_data](#) (const [tgba_bdd_core_data](#) &left, const [tgba_bdd_core_data](#) &right)
Merge two [tgba_bdd_core_data](#).
- const [tgba_bdd_core_data](#) & operator= (const [tgba_bdd_core_data](#) ©)
- void [declare_now_next](#) (bdd now, bdd next)
Update the variable sets to take a new pair of variables into account.
- void [declare_atomic_prop](#) (bdd var)
Update the variable sets to take a new atomic proposition into account.
- void [declare_acceptance_condition](#) (bdd prom)
Update the variable sets to take a new acceptance condition into account.
- 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 [acceptance_conditions_support](#)
The value of `bdd_support(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)

7.158.1 Detailed Description

Core data for a TGBA encoded using BDDs.

7.158.2 Constructor & Destructor Documentation

7.158.2.1 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.

7.158.2.2 spot::tgba_bdd_core_data::tgba_bdd_core_data (const tgba_bdd_core_data & copy)

Copy constructor.

7.158.2.3 `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.

7.158.3 Member Function Documentation

7.158.3.1 `void spot::tgba_bdd_core_data::declare_acceptance_condition (bdd prom)`

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

7.158.3.2 `void spot::tgba_bdd_core_data::declare_atomic_prop (bdd var)`

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

7.158.3.3 `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.

7.158.3.4 `void spot::tgba_bdd_core_data::delete_unaccepting_scc (bdd init)`

Delete SCCs (Strongly Connected Components) from the relation which cannot be accepting.

7.158.3.5 `bdd spot::tgba_bdd_core_data::infinitely_often (bdd s, bdd acc, bdd er)`
`[private]`

7.158.3.6 `const tgba_bdd_core_data& spot::tgba_bdd_core_data::operator= (const
tgba_bdd_core_data & copy)`

7.158.4 Member Data Documentation

7.158.4.1 `bdd spot::tgba_bdd_core_data::acc_set`

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

7.158.4.2 bdd spot::tgba_bdd_core_data::acceptance_conditions

encodes the acceptance conditions

$a \cup b$, or $F b$, both imply that b should be verified eventually. We encode this with generalized Büchi accepting conditions. An acceptance set, called $Acc[b]$, hold all the state that do not promise to verify b eventually. (I.e., all the states that contain b , or do not contain $a \cup b$, or $F b$.)

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

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

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

To summarize, `acceptance_conditions` contains three kinds of variables:

- "Next" variables, that encode the destination state,
- atomic propositions, which are things to verify before going on to the next state,
- "Acc" variables.

7.158.4.3 bdd spot::tgba_bdd_core_data::acceptance_conditions_support

The value of `bdd_support(acceptance_conditions)`.

7.158.4.4 bdd spot::tgba_bdd_core_data::all_acceptance_conditions

The set of all acceptance conditions used by the Automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these accepting conditions. I.e., the union of the accepting conditions of all transition in the SCC should be equal to the result of this function.

7.158.4.5 bdd_dict* spot::tgba_bdd_core_data::dict

The dictionary used by the automata.

7.158.4.6 bdd spot::tgba_bdd_core_data::negacc_set

The negative conjunction of all variables which are acceptance conditions.

7.158.4.7 bdd spot::tgba_bdd_core_data::next_set

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

7.158.4.8 bdd spot::tgba_bdd_core_data::notacc_set

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

7.158.4.9 bdd spot::tgba_bdd_core_data::notnext_set

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

7.158.4.10 bdd spot::tgba_bdd_core_data::notnow_set

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

7.158.4.11 bdd spot::tgba_bdd_core_data::notvar_set

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

7.158.4.12 bdd spot::tgba_bdd_core_data::now_set

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

7.158.4.13 bdd spot::tgba_bdd_core_data::nownext_set

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

7.158.4.14 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

7.158.4.15 `bdd spot::tgba_bdd_core_data::var_set`

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

7.158.4.16 `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:

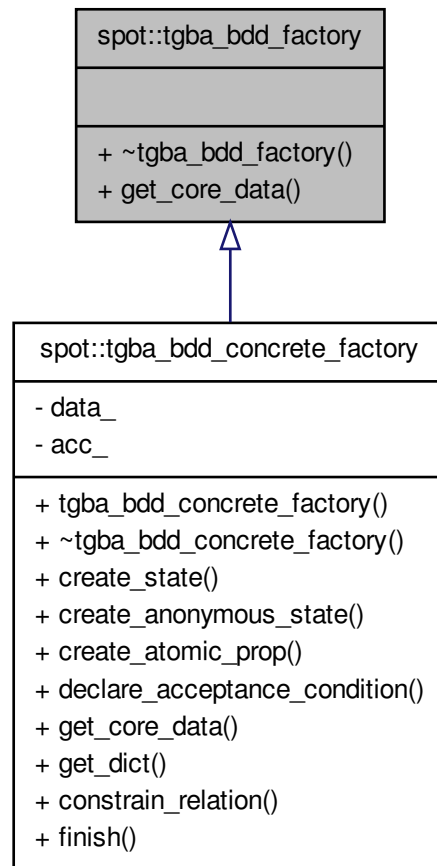
- [tgba/tgabddcoredata.hh](#)

7.159 `spot::tgba_bdd_factory` Class Reference

Abstract class for [spot::tgba_bdd_concrete](#) factories.

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


Inheritance diagram for spot::tgba_bdd_factory:



Public Member Functions

- virtual `~tgba_bdd_factory ()`
- virtual const `tgba_bdd_core_data & get_core_data ()` const =0

Get the core data for the new automata.

7.159.1 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.

7.159.2 Constructor & Destructor Documentation

7.159.2.1 virtual spot::tgba_bdd_factory::~~tgba_bdd_factory () [inline, virtual]

7.159.3 Member Function Documentation

7.159.3.1 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](#).

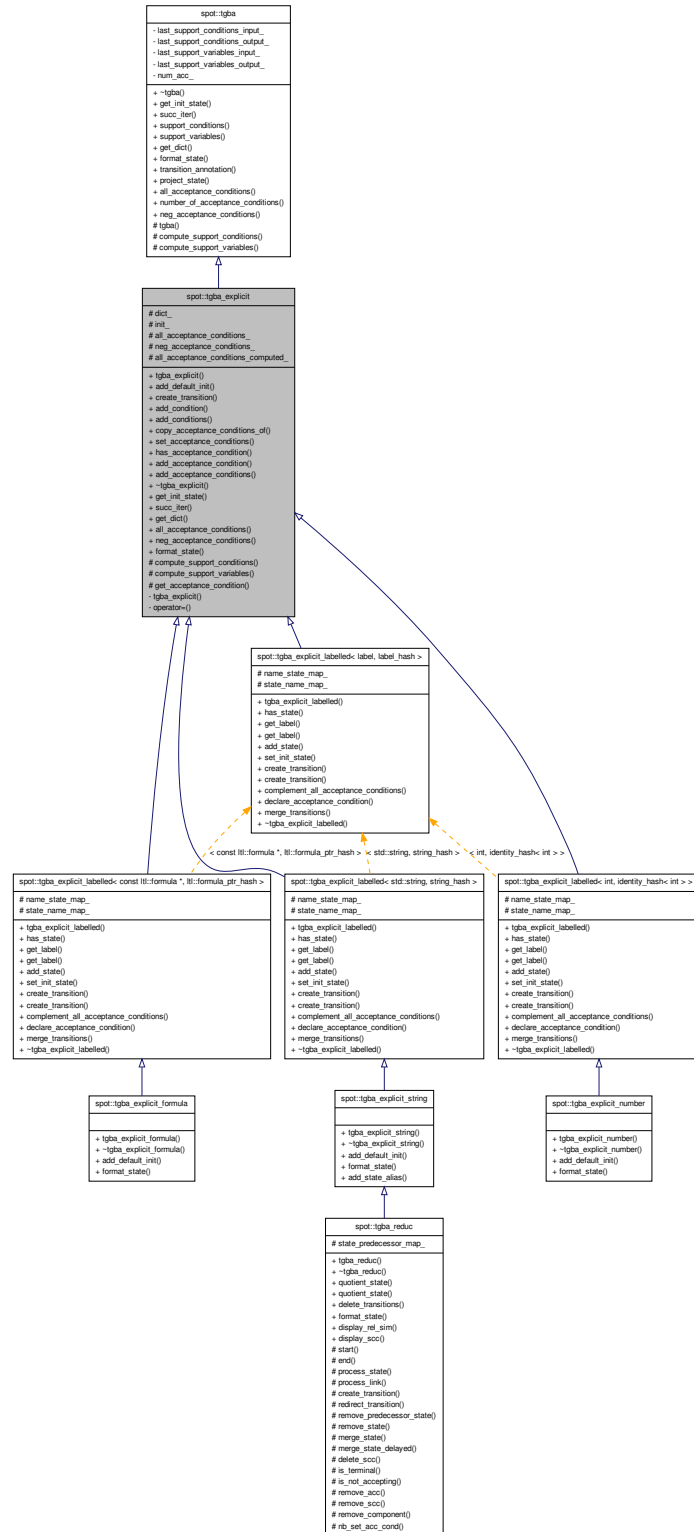
The documentation for this class was generated from the following file:

- [tgba/tgababddfactory.hh](#)

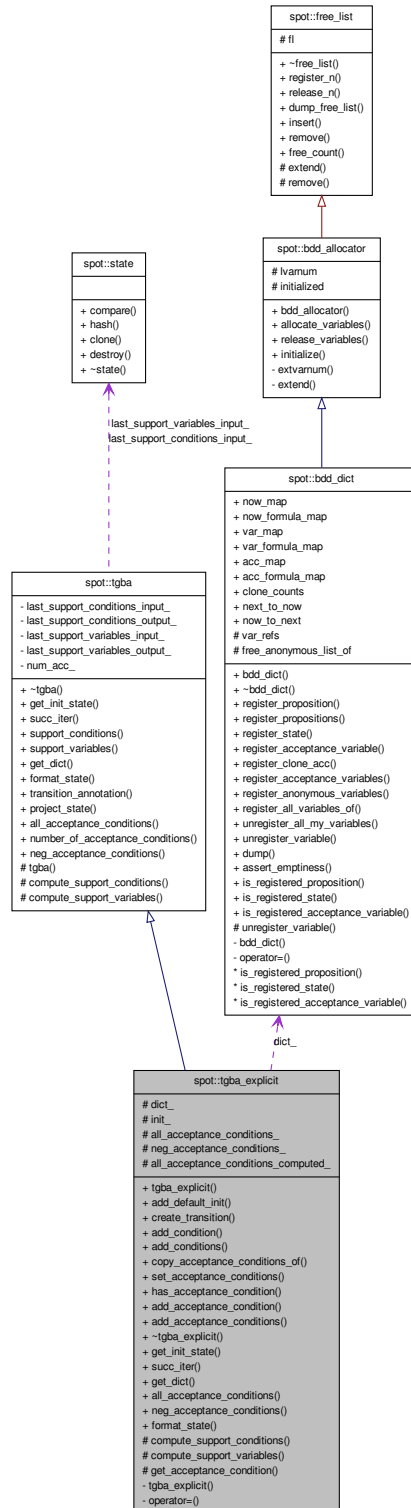
7.160 spot::tgba_explicit Class Reference

```
#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)

7.160.1 Detailed Description

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

7.160.2 Member Typedef Documentation

7.160.2.1 typedef std::list<transition*> spot::tgba_explicit::state

7.160.3 Constructor & Destructor Documentation

7.160.3.1 spot::tgba_explicit::tgba_explicit (bdd_dict * *dict*)

7.160.3.2 virtual spot::tgba_explicit::~~tgba_explicit () [virtual]

7.160.3.3 spot::tgba_explicit::tgba_explicit (const tgba_explicit & *other*) [private]

7.160.4 Member Function Documentation

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

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

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

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

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

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

7.160.4.5 virtual state* spot::tgba_explicit::add_default_init () [pure virtual]

Add a default initial state.

Implemented in [spot::tgba_explicit_string](#), [spot::tgba_explicit_formula](#), and [spot::tgba_explicit_number](#).

7.160.4.6 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()`.

7.160.4.7 `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](#).

7.160.4.8 `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](#).

7.160.4.9 `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.

7.160.4.10 `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 >](#), [spot::tgba_explicit_labelled< int, identity_hash< int > >](#), and [spot::tgba_explicit_labelled< std::string, string_hash >](#).

7.160.4.11 `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_explicit_number](#).

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

7.160.4.13 `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](#).

7.160.4.14 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

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

7.160.4.16 `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](#).

7.160.4.17 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

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

7.160.4.19 `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 destroyed by the caller.

Reimplemented in `spot::tgba_product`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.160.4.20 `void spot::tgba_explicit::set_acceptance_conditions (bdd acc)`

The the acceptance conditions.

7.160.4.21 `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 destroy 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`.

7.160.4.22 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.

7.160.4.23 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.

7.160.4.24 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`, and `spot::tgba_scc`.

7.160.5 Member Data Documentation**7.160.5.1 bdd spot::tgba_explicit::all_acceptance_conditions_ [mutable, protected]****7.160.5.2 bool spot::tgba_explicit::all_acceptance_conditions_computed_ [mutable, protected]**

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

7.160.5.3 bdd_dict* spot::tgba_explicit::dict_ [protected]

Referenced by spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition().

7.160.5.4 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().

7.160.5.5 bdd spot::tgba_explicit::neg_acceptance_conditions_ [protected]

Referenced by spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition().

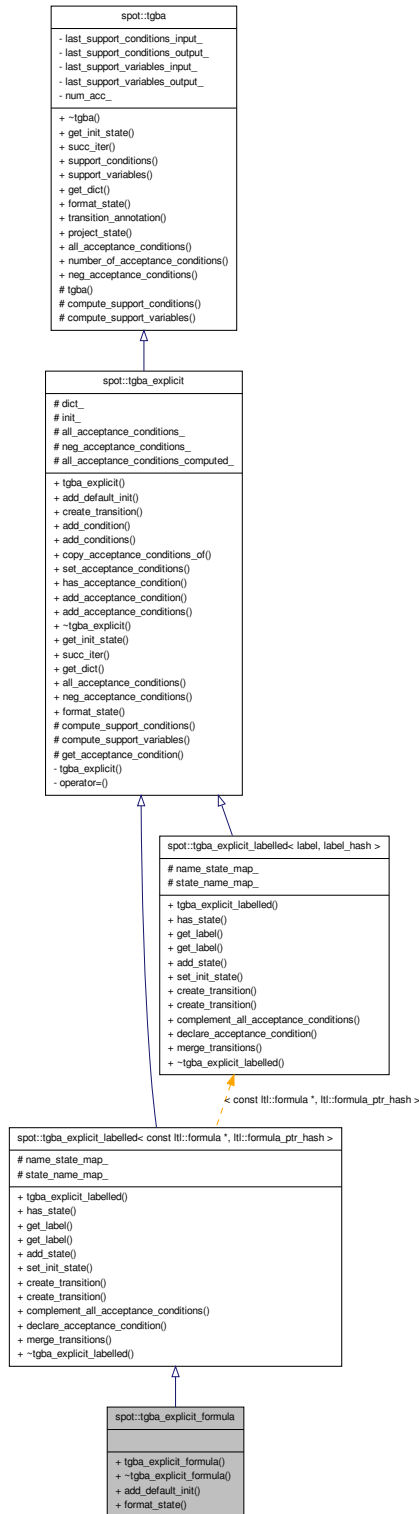
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

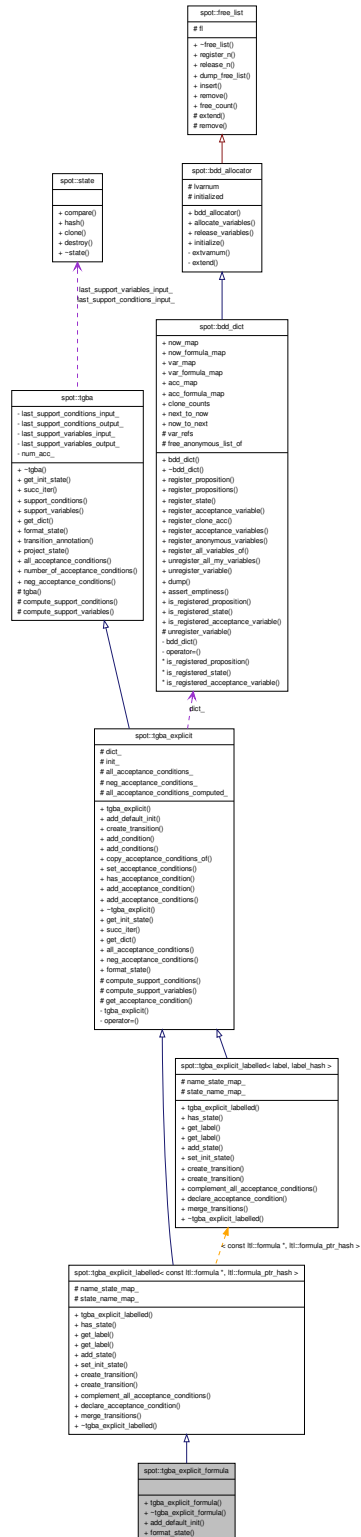
7.161 spot::tgba_explicit_formula Class Reference

```
#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_](#)

7.161.1 Member Typedef Documentation

- 7.161.1.1** `typedef const ltl::formula * spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::label_t [protected, inherited]`
- 7.161.1.2** `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]`
- 7.161.1.3** `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]`
- 7.161.1.4** `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

7.161.2 Constructor & Destructor Documentation

- 7.161.2.1** `spot::tgba_explicit_formula::tgba_explicit_formula (bdd_dict * dict) [inline]`
- 7.161.2.2** `virtual spot::tgba_explicit_formula::~~tgba_explicit_formula () [virtual]`

7.161.3 Member Function Documentation

- 7.161.3.1** `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`
- 7.161.3.2** `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]`

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

7.161.3.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*)
[inherited]

7.161.3.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*) [inherited]

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

7.161.3.5 virtual state* spot::tgba_explicit_formula::add_default_init () [virtual]

Add a default initial state.

Implements [spot::tgba_explicit](#).

7.161.3.6 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.

7.161.3.7 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().

7.161.3.8 void spot::tgba_explicit_labelled< const ltl::formula * , ltl::formula_ptr_hash
>::complement_all_acceptance_conditions () [inline, inherited]

7.161.3.9 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](#).

7.161.3.10 `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](#).

7.161.3.11 `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.

7.161.3.12 `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]`

7.161.3.13 `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](#).

7.161.3.14 `void spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::declare_acceptance_condition (const ltl::formula * f) [inline, inherited]`

7.161.3.15 `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](#).

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

7.161.3.17 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](#).

7.161.3.18 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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.161.3.19 const const ltl::formula * & spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::get_label (const spot::state * s) const [inline, inherited]**7.161.3.20 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]****7.161.3.21 bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const [inherited]****7.161.3.22 bool spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::has_state (const const ltl::formula * & name) [inline, inherited]****7.161.3.23 void spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::merge_transitions () [inline, inherited]**

7.161.3.24 `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](#).

7.161.3.25 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

7.161.3.26 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.161.3.27 `void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]`

The the acceptance conditions.

7.161.3.28 `state* spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash >::set_init_state (const const ltl::formula * & state) [inline, inherited]`

7.161.3.29 `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 destroy 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`.

7.161.3.30 `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.

7.161.3.31 `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.

7.161.3.32 `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](#), and [spot::tgba_scc](#).

7.161.4 Member Data Documentation

7.161.4.1 `bdd spot::tgba_explicit::all_acceptance_conditions_ [mutable, protected,
inherited]`

7.161.4.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_ [mutable,
protected, inherited]`

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

7.161.4.3 `bdd_dict* spot::tgba_explicit::dict_ [protected, inherited]`

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

7.161.4.4 `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()`.

7.161.4.5 `ns_map spot::tgba_explicit_labelled< const ltl::formula *, ltl::formula_ptr_hash
>::name_state_map_ [protected, inherited]`

7.161.4.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_ [protected, inherited]`

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

7.161.4.7 `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:

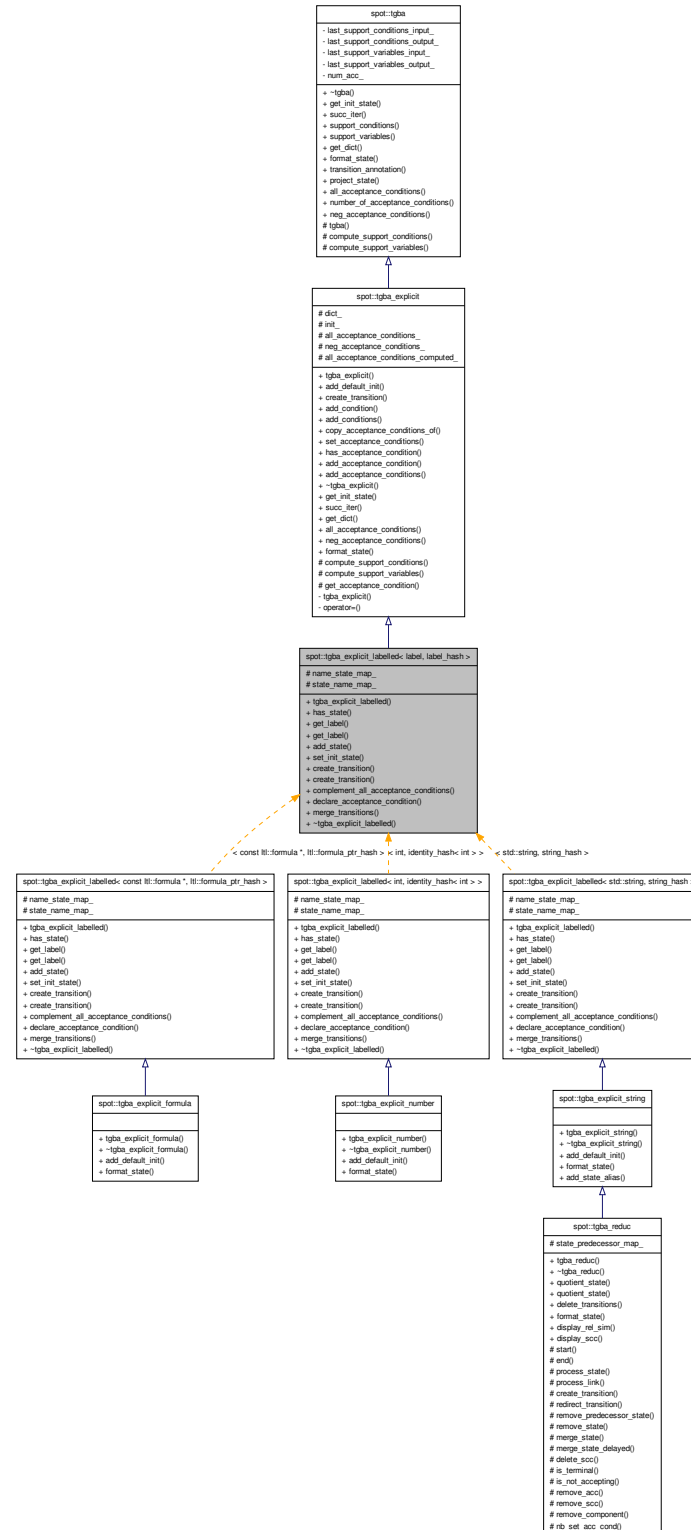
- [tgba/tgbaexplicit.hh](#)

7.162 spot::tgba_explicit_labelled< label, label_hash > Class Template Reference

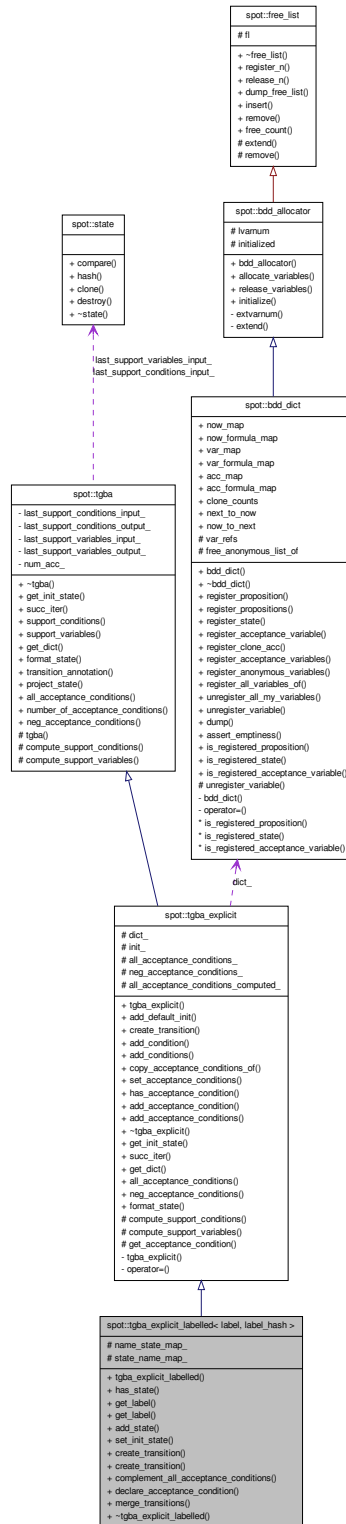
A [tgba_explicit](#) instance with states labeled by a given type.

```
#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_`

7.162.1 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.

7.162.2 Member Typedef Documentation

7.162.2.1 `template<typename label, typename label_hash> typedef label spot::tgba_explicit_labelled< label, label_hash >::label_t [protected]`

7.162.2.2 `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]`

7.162.2.3 `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]`

7.162.2.4 `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

7.162.3 Constructor & Destructor Documentation

7.162.3.1 `template<typename label, typename label_hash> spot::tgba_explicit_labelled< label, label_hash >::tgba_explicit_labelled (bdd_dict * dict) [inline]`

7.162.3.2 `template<typename label, typename label_hash> virtual spot::tgba_explicit_labelled< label, label_hash >::~~tgba_explicit_labelled () [inline, virtual]`

7.162.4 Member Function Documentation

7.162.4.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`

7.162.4.2 void spot::tgba_explicit::add_acceptance_conditions (transition * *t*, bdd *f*)
[*inherited*]

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

7.162.4.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*)
[*inherited*]

7.162.4.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*) [*inherited*]

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

7.162.4.5 virtual state* spot::tgba_explicit::add_default_init () [pure virtual,
inherited]

Add a default initial state.

Implemented in [spot::tgba_explicit_string](#), [spot::tgba_explicit_formula](#), and [spot::tgba_explicit_number](#).

7.162.4.6 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\(\)](#).

7.162.4.7 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\(\)](#).

7.162.4.8 `template<typename label, typename label_hash> void spot::tgba_explicit_labelled< label, label_hash >::complement_all_acceptance_conditions () [inline]`

7.162.4.9 `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](#).

7.162.4.10 `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](#).

7.162.4.11 `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.

7.162.4.12 `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\(\)](#).

7.162.4.13 `template<typename label, typename label_hash> transition* spot::tgba_explicit_labelled< label, label_hash >::create_transition (const label & source, const label & dest) [inline]`

7.162.4.14 `template<typename label, typename label_hash> void spot::tgba_explicit_labelled< label, label_hash >::declare_acceptance_condition (const ltl::formula * f) [inline]`

7.162.4.15 `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_explicit_number](#).

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

7.162.4.17 `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](#).

7.162.4.18 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.162.4.19 `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()`.

7.162.4.20 `template<typename label, typename label_hash> const label&`
`spot::tgba_explicit_labelled< label, label_hash >::get_label (const spot::state * s)`
`const` `[inline]`

7.162.4.21 `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const`
`[inherited]`

7.162.4.22 `template<typename label, typename label_hash> bool spot::tgba_explicit_labelled<`
`label, label_hash >::has_state (const label & name) [inline]`

7.162.4.23 `template<typename label, typename label_hash> void spot::tgba_explicit_labelled<`
`label, label_hash >::merge_transitions () [inline]`

7.162.4.24 `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](#).

7.162.4.25 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

7.162.4.26 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.162.4.27 void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]

The the acceptance conditions.

7.162.4.28 template<typename label, typename label_hash> state* spot::tgba_explicit_labelled< label, label_hash >::set_init_state (const label & state) [inline]**7.162.4.29 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 destroy 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`.

7.162.4.30 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.

7.162.4.31 `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.

7.162.4.32 `virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const [virtual, inherited]`

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters

t a non-done `tgba_succ_iterator` for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_scc`.

7.162.5 Member Data Documentation**7.162.5.1** `bdd spot::tgba_explicit::all_acceptance_conditions_ [mutable, protected, inherited]`**7.162.5.2** `bool spot::tgba_explicit::all_acceptance_conditions_computed_ [mutable, protected, inherited]`

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

7.162.5.3 `bdd_dict* spot::tgba_explicit::dict_ [protected, inherited]`

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

7.162.5.4 `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()`.

7.162.5.5 `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()`.

7.162.5.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_ [protected, inherited]`

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

7.162.5.7 `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()`.

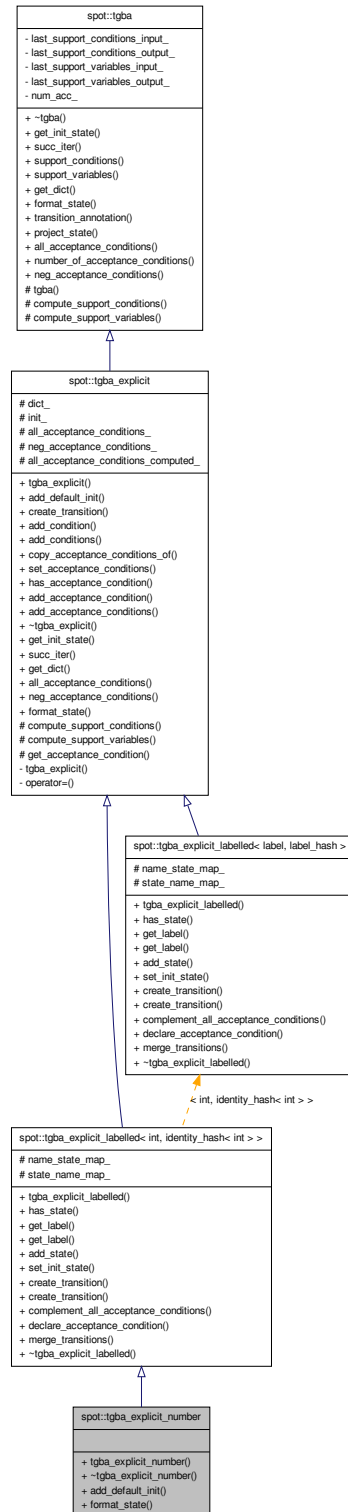
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

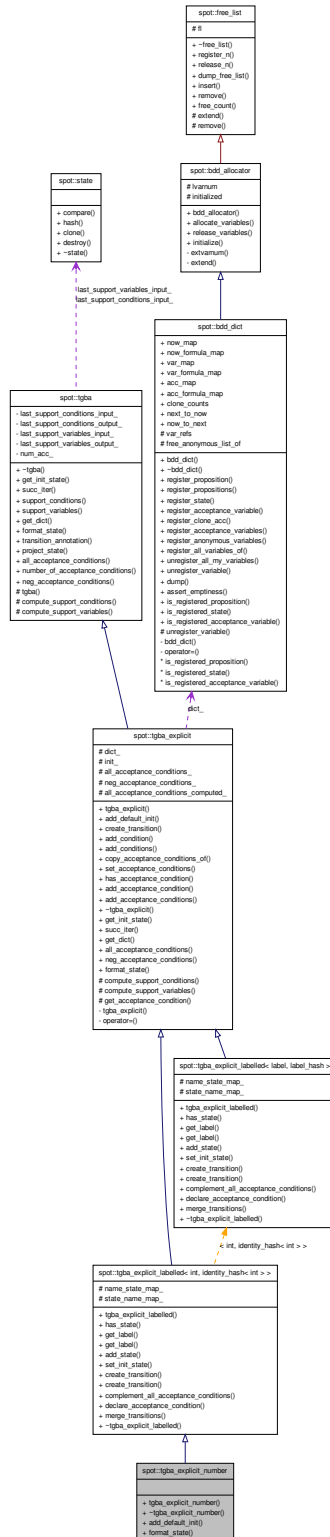
7.163 spot::tgba_explicit_number Class Reference

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

Inheritance diagram for spot::tgba_explicit_number:



Collaboration diagram for spot::tgba_explicit_number:



Public Types

- typedef std::list< [transition](#) * > [state](#)

Public Member Functions

- [tgba_explicit_number](#) (bdd_dict *dict)
- virtual [~tgba_explicit_number](#) ()
- 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 int &name)
- const int & [get_label](#) (const [tgba_explicit::state](#) *s) const
- const int & [get_label](#) (const [spot::state](#) *s) const
- [state](#) * [add_state](#) (const int &name)
- [state](#) * [set_init_state](#) (const int &state)
- [transition](#) * [create_transition](#) ([state](#) *source, const [state](#) *dest)
- [transition](#) * [create_transition](#) (const int &source, const int &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 int [label_t](#)
- typedef Sgi::hash_map< int, [tgba_explicit::state](#) *, [identity_hash](#)< int > > [ns_map](#)
- typedef Sgi::hash_map< const [tgba_explicit::state](#) *, int, [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_](#)

7.163.1 Member Typedef Documentation

7.163.1.1 `typedef int spot::tgba_explicit_labelled< int , identity_hash< int > >::label_t [protected, inherited]`

7.163.1.2 `typedef Sgi::hash_map<int , tgba_explicit::state*, identity_hash< int > > spot::tgba_explicit_labelled< int , identity_hash< int > >::ns_map [protected, inherited]`

7.163.1.3 `typedef Sgi::hash_map<const tgba_explicit::state*, int , ptr_hash<tgba_explicit::state> > spot::tgba_explicit_labelled< int , identity_hash< int > >::sn_map [protected, inherited]`

7.163.1.4 `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

7.163.2 Constructor & Destructor Documentation

7.163.2.1 `spot::tgba_explicit_number::tgba_explicit_number (bdd_dict * dict) [inline]`

7.163.2.2 `virtual spot::tgba_explicit_number::~~tgba_explicit_number () [virtual]`

7.163.3 Member Function Documentation

7.163.3.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`

7.163.3.2 `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]`

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

7.163.3.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*)
[*inherited*]

7.163.3.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*) [*inherited*]

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

7.163.3.5 virtual state* spot::tgba_explicit_number::add_default_init () [*virtual*]

Add a default initial state.

Implements [spot::tgba_explicit](#).

7.163.3.6 state* spot::tgba_explicit_labelled< int , identity_hash< int > >::add_state (const int
& *name*) [*inline*, *inherited*]

Return the tgba_explicit::state for *name*, creating the state if it does not exist.

7.163.3.7 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().

7.163.3.8 void spot::tgba_explicit_labelled< int , identity_hash< int > >::complement_all_acceptance_conditions () [*inline*, *inherited*]

7.163.3.9 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](#).

7.163.3.10 `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](#).

7.163.3.11 `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.

7.163.3.12 `transition* spot::tgba_explicit_labelled< int , identity_hash< int > >::create_transition (const int & source, const int & dest) [inline, inherited]`

7.163.3.13 `transition* spot::tgba_explicit_labelled< int , identity_hash< int > >::create_transition (state * source, const state * dest) [inline, inherited]`

Reimplemented from [spot::tgba_explicit](#).

7.163.3.14 `void spot::tgba_explicit_labelled< int , identity_hash< int > >::declare_acceptance_condition (const ltl::formula * f) [inline, inherited]`

7.163.3.15 `virtual std::string spot::tgba_explicit_number::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](#).

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

7.163.3.17 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](#).

7.163.3.18 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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.163.3.19 const int & spot::tgba_explicit_labelled< int , identity_hash< int > >::get_label (const spot::state * s) const [inline, inherited]**7.163.3.20 const int & spot::tgba_explicit_labelled< int , identity_hash< int > >::get_label (const tgba_explicit::state * s) const [inline, inherited]****7.163.3.21 bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const [inherited]****7.163.3.22 bool spot::tgba_explicit_labelled< int , identity_hash< int > >::has_state (const int & name) [inline, inherited]****7.163.3.23 void spot::tgba_explicit_labelled< int , identity_hash< int > >::merge_transitions () [inline, inherited]**

7.163.3.24 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](#).

7.163.3.25 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

7.163.3.26 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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.163.3.27 void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]

The the acceptance conditions.

7.163.3.28 state* spot::tgba_explicit_labelled< int , identity_hash< int > >::set_init_state (const int & state) [inline, inherited]

7.163.3.29 `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 destroy 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`.

7.163.3.30 `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.

7.163.3.31 `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.

7.163.3.32 `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](#), and [spot::tgba_scc](#).

7.163.4 Member Data Documentation

7.163.4.1 `bdd spot::tgba_explicit::all_acceptance_conditions_ [mutable, protected,
inherited]`

7.163.4.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_ [mutable,
protected, inherited]`

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

7.163.4.3 `bdd_dict* spot::tgba_explicit::dict_ [protected, inherited]`

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

7.163.4.4 `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()`.

7.163.4.5 `ns_map spot::tgba_explicit_labelled< int , identity_hash< int > >::name_state_map_
[protected, inherited]`

7.163.4.6 `bdd spot::tgba_explicit::neg_acceptance_conditions_ [protected, inherited]`

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

7.163.4.7 sn_map spot::tgba_explicit_labelled< int , identity_hash< int > >::state_name_map_[protected, inherited]

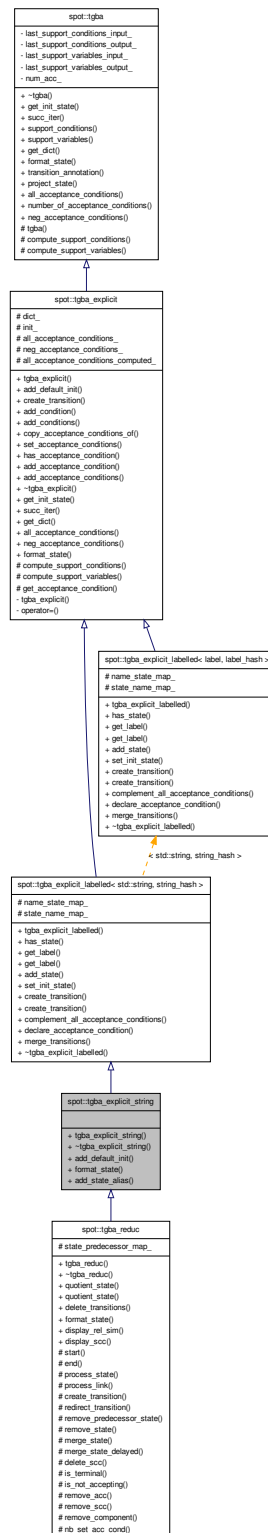
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

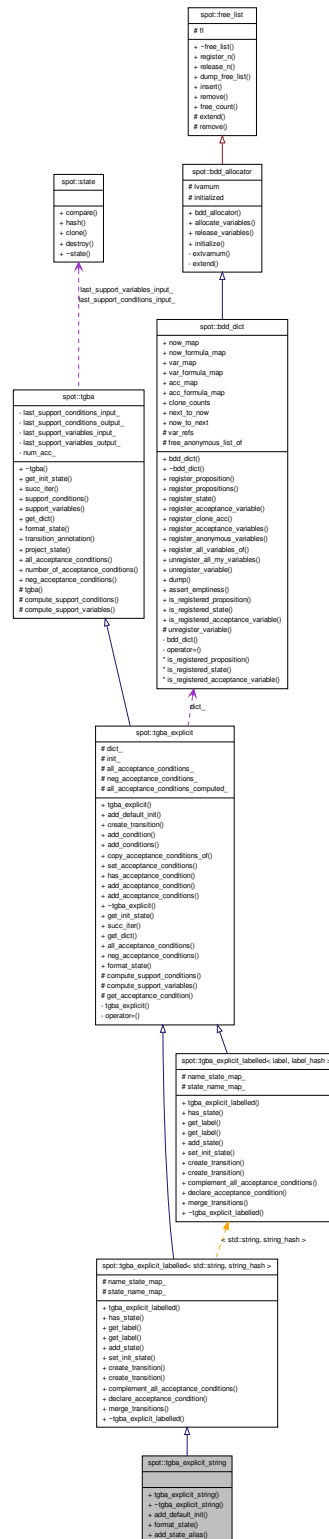
7.164 spot::tgba_explicit_string Class Reference

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


Inheritance diagram for spot::tgba_explicit_string:



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.
- virtual void [add_state_alias](#) (const std::string &alias_name, const std::string &real_name)
- 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_](#)

7.164.1 Member Typedef Documentation

7.164.1.1 `typedef std::string spot::tgba_explicit_labelled< std::string , string_hash >::label_t [protected, inherited]`

7.164.1.2 `typedef Sgi::hash_map<std::string , tgba_explicit::state*, string_hash > spot::tgba_explicit_labelled< std::string , string_hash >::ns_map [protected, inherited]`

7.164.1.3 `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]`

7.164.1.4 `typedef std::list<transition*> spot::tgba_explicit::state [inherited]`

7.164.2 Constructor & Destructor Documentation

7.164.2.1 `spot::tgba_explicit_string::tgba_explicit_string (bdd_dict * dict) [inline]`

7.164.2.2 `virtual spot::tgba_explicit_string::~~tgba_explicit_string () [virtual]`

7.164.3 Member Function Documentation

7.164.3.1 `void spot::tgba_explicit::add_acceptance_condition (transition * t, const ltl::formula * f) [inherited]`

7.164.3.2 `void spot::tgba_explicit::add_acceptance_conditions (transition * t, bdd f) [inherited]`

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

7.164.3.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*)
[*inherited*]

7.164.3.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*) [*inherited*]

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

7.164.3.5 virtual state* spot::tgba_explicit_string::add_default_init () [**virtual**]

Add a default initial state.

Implements [spot::tgba_explicit](#).

7.164.3.6 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_.

Referenced by add_state_alias().

7.164.3.7 virtual void spot::tgba_explicit_string::add_state_alias (const std::string & *alias_name*, const std::string & *real_name*) [**inline**, **virtual**]

Create an alias for a state. Any reference to *alias_name* will act as a reference to *real_name*.

References spot::tgba_explicit_labelled< std::string, string_hash >::add_state(), and spot::tgba_explicit_labelled< std::string, string_hash >::name_state_map_.

7.164.3.8 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().

7.164.3.9 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_.

7.164.3.10 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.

7.164.3.11 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.

7.164.3.12 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.

7.164.3.13 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().

7.164.3.14 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().

7.164.3.15 `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()`.

7.164.3.16 `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](#).

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

7.164.3.18 `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](#).

7.164.3.19 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.164.3.20 `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()`.

7.164.3.21 `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_`.

7.164.3.22 `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const [inherited]`

7.164.3.23 `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_`.

7.164.3.24 `void spot::tgba_explicit_labelled< std::string, string_hash >::merge_transitions () [inline, inherited]`

References `spot::tgba_explicit_labelled< label, label_hash >::name_state_map_`.

7.164.3.25 `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](#).

7.164.3.26 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]`

The number of acceptance conditions.

7.164.3.27 `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 destroyed by the caller.

Reimplemented in `spot::tgba_product`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.164.3.28 void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]

The the acceptance conditions.

7.164.3.29 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_`.

7.164.3.30 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 destroy 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`.

7.164.3.31 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.

7.164.3.32 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.

7.164.3.33 virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters

t a non-done `tgba_succ_iterator` for this automata

Reimplemented in `spot::tgba_product`, and `spot::tgba_scc`.

7.164.4 Member Data Documentation**7.164.4.1 bdd spot::tgba_explicit::all_acceptance_conditions_ [mutable, protected, inherited]****7.164.4.2 bool spot::tgba_explicit::all_acceptance_conditions_computed_ [mutable, protected, inherited]**

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

7.164.4.3 bdd_dict* spot::tgba_explicit::dict_ [protected, inherited]

Referenced by spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition().

7.164.4.4 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().

7.164.4.5 ns_map spot::tgba_explicit_labelled< std::string , string_hash >::name_state_map_ [protected, inherited]

Referenced by add_state_alias().

7.164.4.6 bdd spot::tgba_explicit::neg_acceptance_conditions_ [protected, inherited]

Referenced by spot::tgba_explicit_labelled< std::string, string_hash >::declare_acceptance_condition().

7.164.4.7 sn_map spot::tgba_explicit_labelled< std::string , string_hash >::state_name_map_ [protected, inherited]

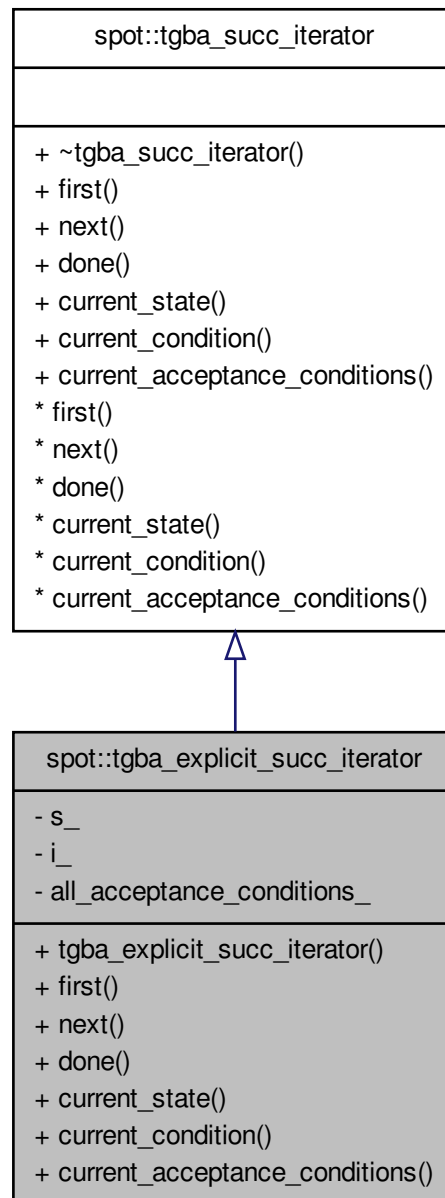
The documentation for this class was generated from the following file:

- [tgba/tgbaexplicit.hh](#)

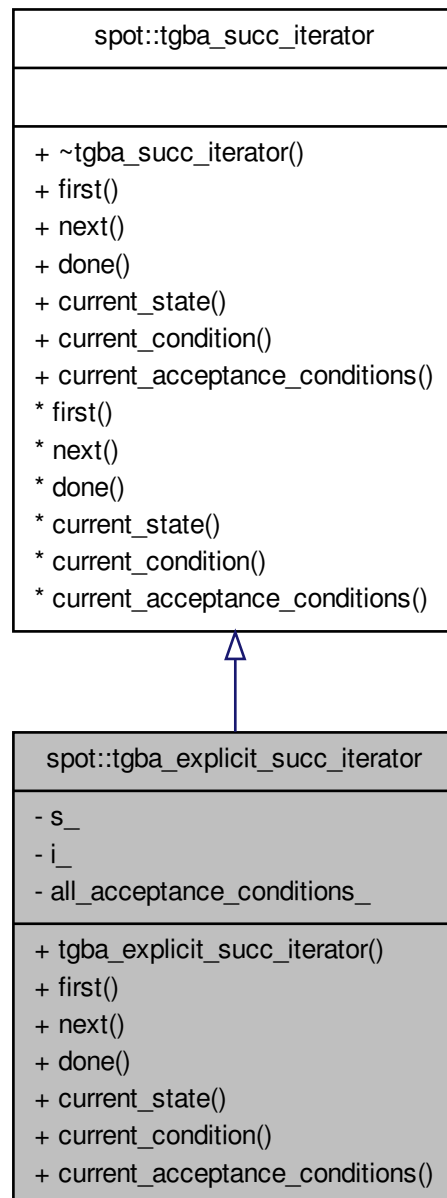
7.165 spot::tgba_explicit_succ_iterator Class Reference

```
#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_](#)

7.165.1 Detailed Description

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

7.165.2 Constructor & Destructor Documentation

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

7.165.3 Member Function Documentation

- 7.165.3.1** 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](#).

- 7.165.3.2** 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](#).

7.165.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

Implements [spot::tgba_succ_iterator](#).

7.165.3.4 `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](#).

7.165.3.5 `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](#).

7.165.3.6 `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`.

7.165.4 Member Data Documentation

7.165.4.1 `bdd spot::tgba_explicit_succ_iterator::all_acceptance_conditions_` `[private]`

7.165.4.2 `tgba_explicit::state::const_iterator spot::tgba_explicit_succ_iterator::i_` `[private]`

7.165.4.3 `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](#)

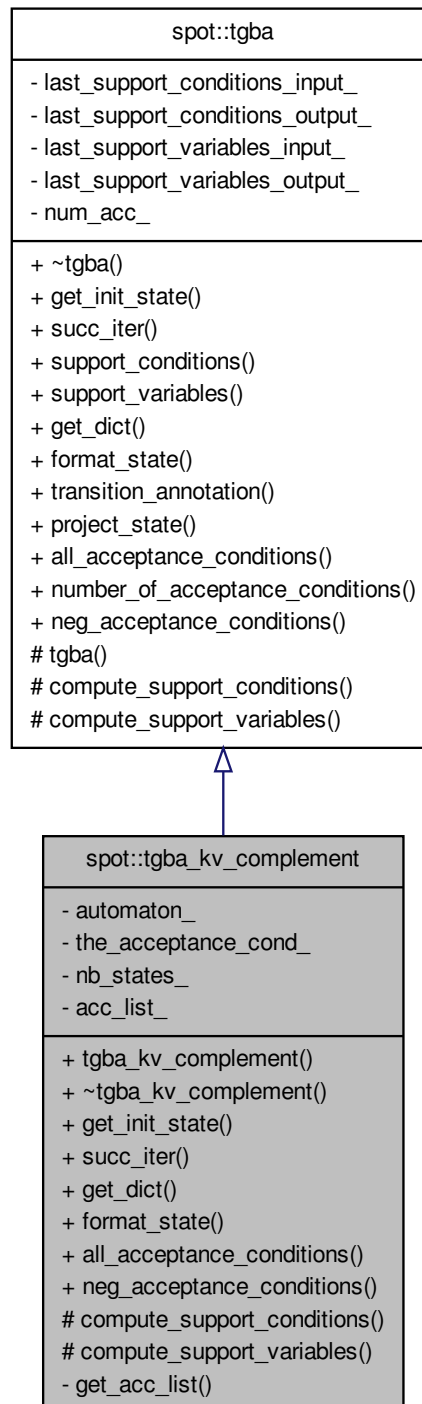
7.166 `spot::tgba_kv_complement` Class Reference

Build a complemented automaton.

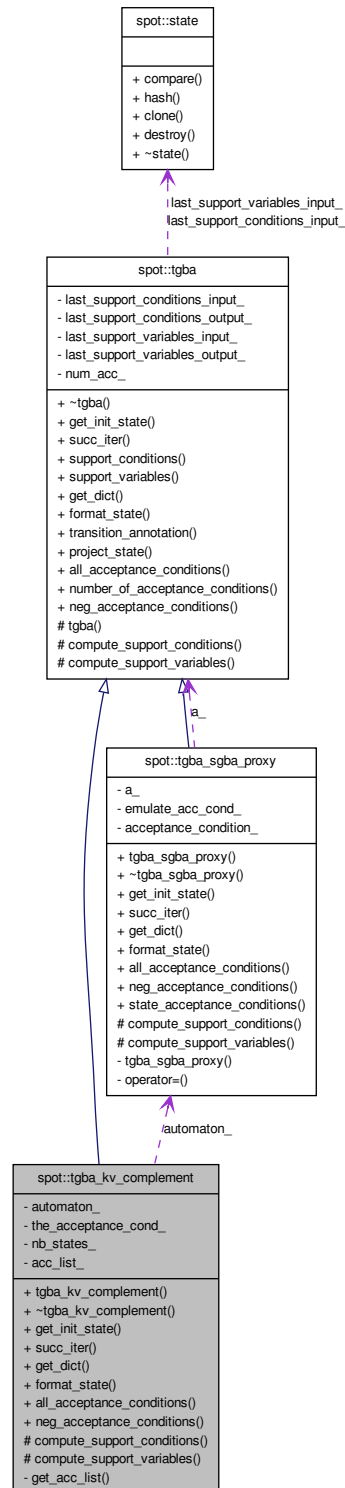
The construction comes from:

```
#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_](#)

7.166.1 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`

7.166.2 Constructor & Destructor Documentation

7.166.2.1 `spot::tgba_kv_complement::tgba_kv_complement (const tgba * a)`

7.166.2.2 `virtual spot::tgba_kv_complement::~~tgba_kv_complement () [virtual]`

7.166.3 Member Function Documentation

7.166.3.1 `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](#).

7.166.3.2 `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](#).

7.166.3.3 `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](#).

7.166.3.4 `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](#).

7.166.3.5 `void spot::tgba_kv_complement::get_acc_list () [private]`

Retrieve all the atomic acceptance conditions of the automaton. They are inserted into *acc_list_*.

7.166.3.6 `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](#).

7.166.3.7 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.166.3.8 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](#).

7.166.3.9 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 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 destroyed 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_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 destroy 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](#).

7.166.3.12 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](#), and [spot::tgba_scc](#).

7.166.4 Member Data Documentation

7.166.4.1 `acc_list_t spot::tgba_kv_complement::acc_list_` `[private]`

7.166.4.2 `const tgba_sgba_proxy* spot::tgba_kv_complement::automaton_` `[private]`

7.166.4.3 `unsigned spot::tgba_kv_complement::nb_states_` `[private]`

7.166.4.4 `bdd spot::tgba_kv_complement::the_acceptance_cond_` `[private]`

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

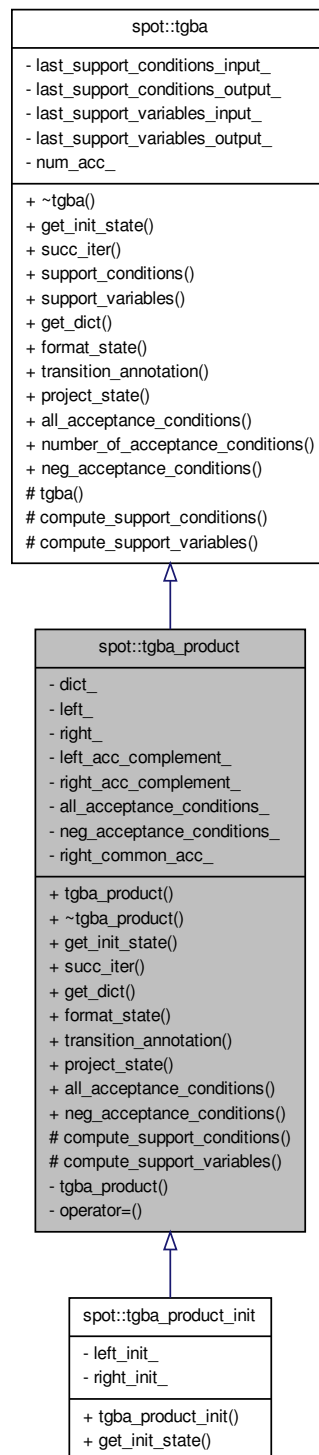
- [tgba/tgbakvcomplement.hh](#)

7.167 `spot::tgba_product` Class Reference

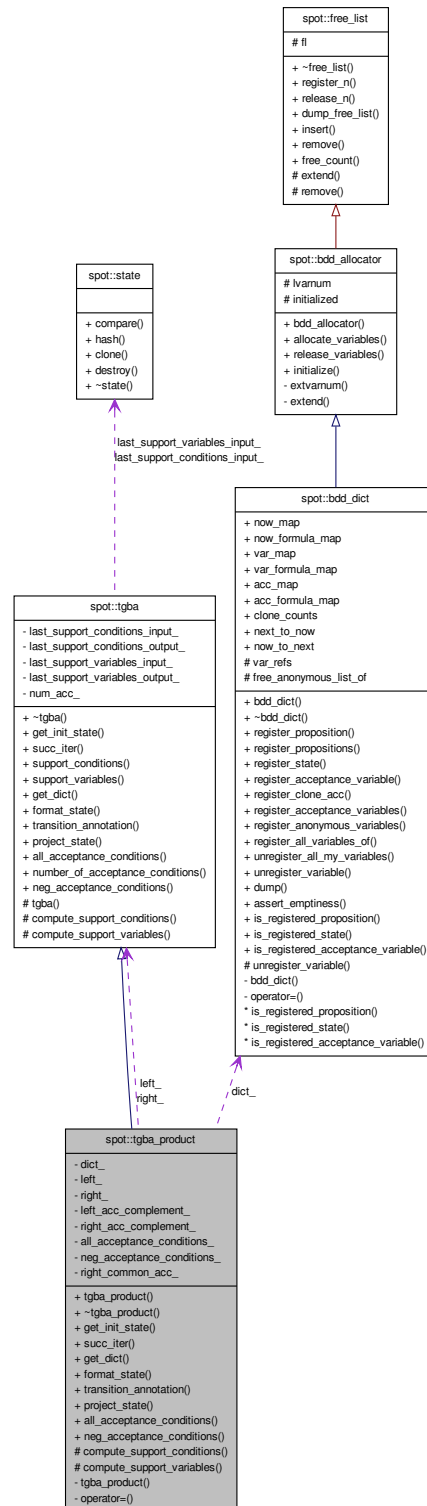
A lazy product. (States are computed on the fly.).

```
#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_](#)

7.167.1 Detailed Description

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

7.167.2 Constructor & Destructor Documentation

7.167.2.1 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.

7.167.2.2 virtual spot::tgba_product::~~tgba_product () [**virtual**]

7.167.2.3 spot::tgba_product::tgba_product (const [tgba_product](#) &) [**private**]

7.167.3 Member Function Documentation

7.167.3.1 virtual [bdd](#) spot::tgba_product::all_acceptance_conditions () const [**virtual**]

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

7.167.3.2 `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](#).

7.167.3.3 `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](#).

7.167.3.4 `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](#).

7.167.3.5 `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](#).

7.167.3.6 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

Reimplemented in [spot::tgba_product_init](#).

7.167.3.7 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](#).

7.167.3.8 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

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

7.167.3.10 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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.167.3.11 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 destroy 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`.

7.167.3.12 `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.

7.167.3.13 `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.

7.167.3.14 `virtual std::string spot::tgba_product::transition_annotation (const tgba_succ_iterator * t) const [virtual]`

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation it the empty string.

Parameters

t a non-done `tgba_succ_iterator` for this automata

Reimplemented from [spot::tgba](#).

7.167.4 Member Data Documentation

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

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

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

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

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

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

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

7.167.4.8 `bddPair* spot::tgba_product::right_common_acc_` `[private]`

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

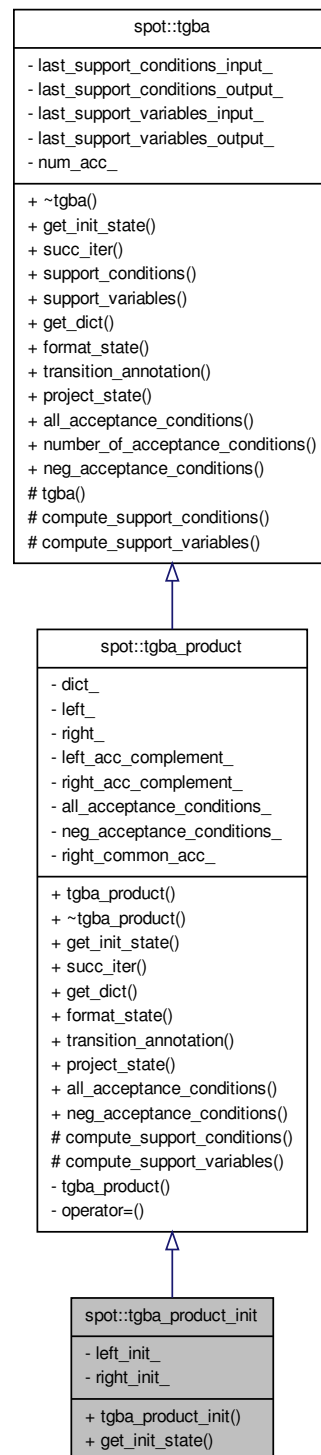
- [tgba/tgbaproduct.hh](#)

7.168 `spot::tgba_product_init` Class Reference

A lazy product with different initial states.

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

Inheritance diagram for spot::tgba_product_init:



Collaboration diagram for spot::tgba_product_init:



Public Member Functions

- [tgba_product_init](#) (const [tgba](#) *left, const [tgba](#) *right, const [state](#) *left_init, const [state](#) *right_init)
- 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 Attributes

- const [state](#) * [left_init_](#)
- const [state](#) * [right_init_](#)

7.168.1 Detailed Description

A lazy product with different initial states.

7.168.2 Constructor & Destructor Documentation

7.168.2.1 `spot::tgba_product_init::tgba_product_init (const tgba * left, const tgba * right, const state * left_init, const state * right_init)`

7.168.3 Member Function Documentation

7.168.3.1 `virtual bdd spot::tgba_product::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](#).

7.168.3.2 `virtual bdd spot::tgba_product::compute_support_conditions (const state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support_conditions\(\)](#).

Implements [spot::tgba](#).

7.168.3.3 `virtual bdd spot::tgba_product::compute_support_variables (const state * state) const [protected, virtual, inherited]`

Do the actual computation of [tgba::support_variables\(\)](#).

Implements [spot::tgba](#).

7.168.3.4 `virtual std::string spot::tgba_product::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](#).

7.168.3.5 virtual bdd_dict* spot::tgba_product::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](#).

7.168.3.6 virtual state* spot::tgba_product_init::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 `destroy` it when no longer needed.

Reimplemented from [spot::tgba_product](#).

7.168.3.7 virtual bdd spot::tgba_product::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](#).

7.168.3.8 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_product::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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.168.3.10 `virtual tgba_succ_iterator_product* spot::tgba_product::succ_iter (const state * local_state, const state * global_state = 0, const tgba * global_automaton = 0) const [virtual, 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 destroy 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](#).

7.168.3.11 `bdd spot::tgba::support_conditions (const state * state) const [inherited]`

Get a formula that must hold whatever successor is taken.

Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by [succ_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute_support_conditions\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

7.168.3.12 `bdd spot::tgba::support_variables (const state * state) const [inherited]`

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some `succ_iter()` to reduce the number of successor to compute in a product.

Sub classes should implement `compute_support_variables()`, this function is just a wrapper that will cache the last return value for efficiency.

7.168.3.13 `virtual std::string spot::tgba_product::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`.

7.168.4 Member Data Documentation

7.168.4.1 `const state* spot::tgba_product_init::left_init_` `[private]`

7.168.4.2 `const state* spot::tgba_product_init::right_init_` `[private]`

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

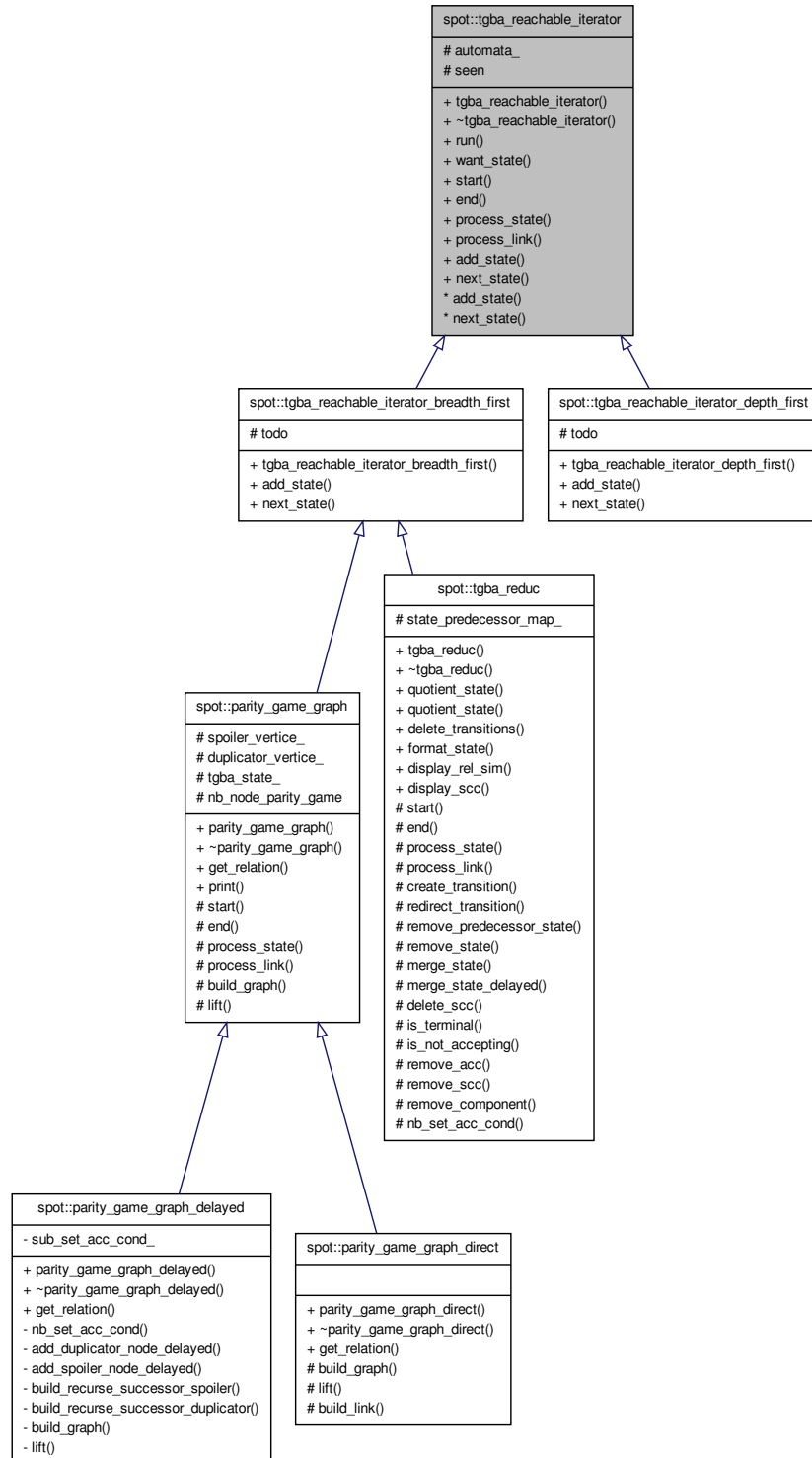
- `tgba/tgbaproduct.hh`

7.169 `spot::tgba_reachable_iterator` Class Reference

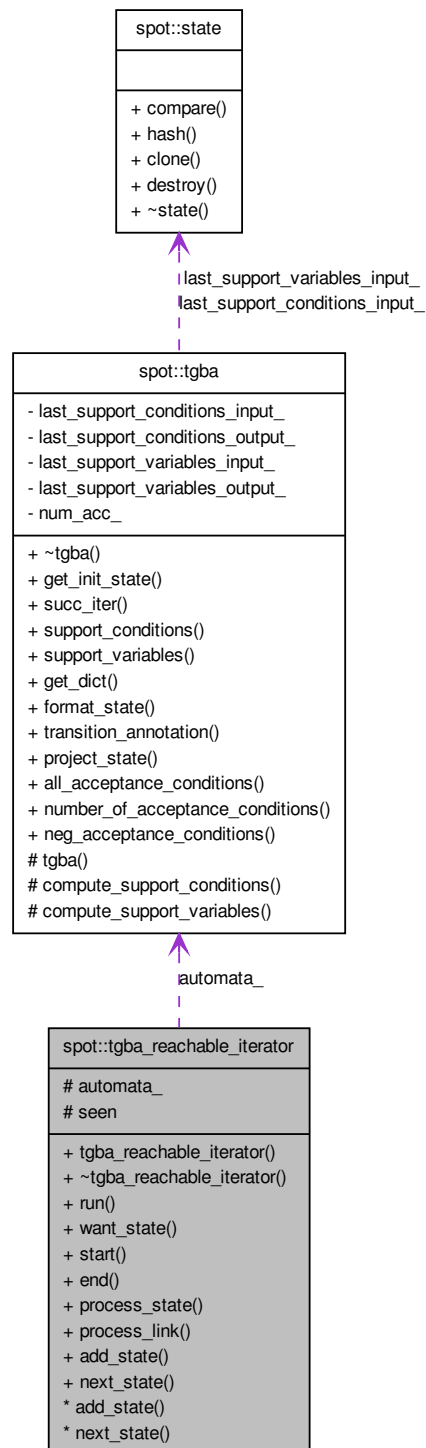
Iterate over all reachable states of a `spot::tgba`.

```
#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.

7.169.1 Detailed Description

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

7.169.2 Member Typedef Documentation

- 7.169.2.1** typedef Sgi::hash_map<const state*, int, state_ptr_hash, state_ptr_equal>
spot::tgba_reachable_iterator::seen_map [protected]

7.169.3 Constructor & Destructor Documentation

7.169.3.1 `spot::tgba_reachable_iterator::tgba_reachable_iterator (const tgba * a)`

7.169.3.2 `virtual spot::tgba_reachable_iterator::~~tgba_reachable_iterator () [virtual]`

7.169.4 Member Function Documentation

7.169.4.1 `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](#).

7.169.4.2 `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](#).

7.169.4.3 `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](#).

7.169.4.4 `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.

7.169.4.5 `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](#).

7.169.4.6 `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.

7.169.4.7 `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](#).

7.169.4.8 `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.

7.169.5 Member Data Documentation

7.169.5.1 `const tgba* spot::tgba_reachable_iterator::automata_ [protected]`

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

7.169.5.2 `seen_map spot::tgba_reachable_iterator::seen [protected]`

States already seen.

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

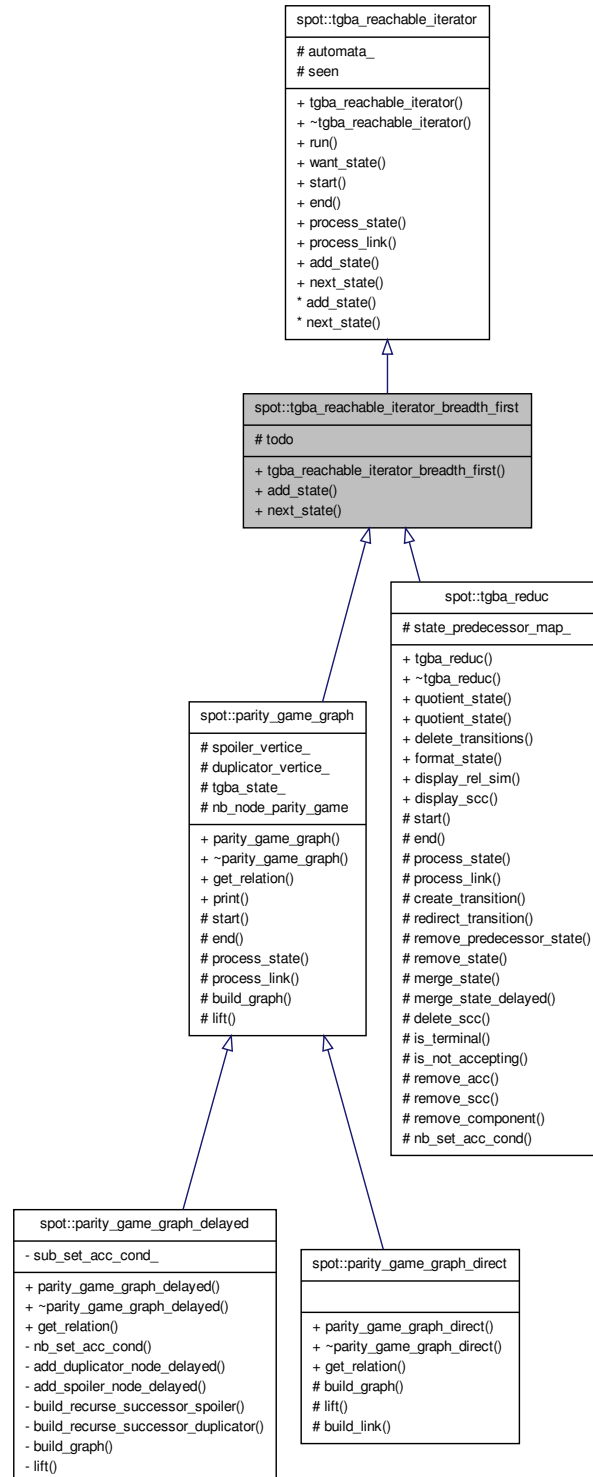
- [tgbaalgos/reachiter.hh](#)

7.170 spot::tgba_reachable_iterator_breadth_first Class Reference

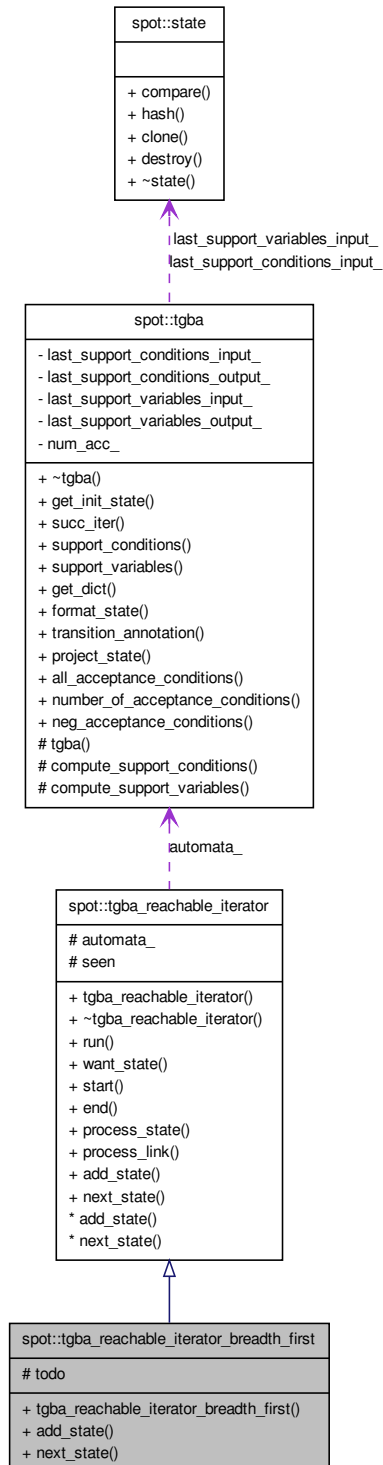
An implementation of [spot::tgba_reachable_iterator](#) that browses states breadth first.

```
#include <tgbaalgs/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.

7.170.1 Detailed Description

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

7.170.2 Member Typedef Documentation

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

7.170.3 Constructor & Destructor Documentation

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

7.170.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

7.170.4.2 `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](#).

7.170.4.3 `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](#).

7.170.4.4 `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](#) positioned 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.

7.170.4.5 `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](#).

7.170.4.6 `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.

7.170.4.7 `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](#).

7.170.4.8 `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.

7.170.5 Member Data Documentation

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

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

7.170.5.2 `seen_map spot::tgba_reachable_iterator::seen [protected, inherited]`

States already seen.

7.170.5.3 `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:

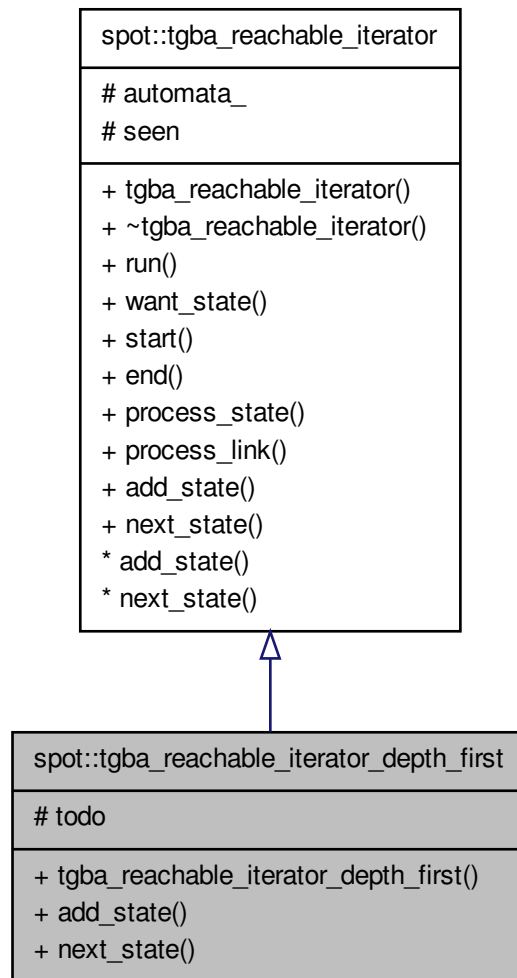
- `tgbaalgos/reachiter.hh`

7.171 `spot::tgba_reachable_iterator_depth_first` Class Reference

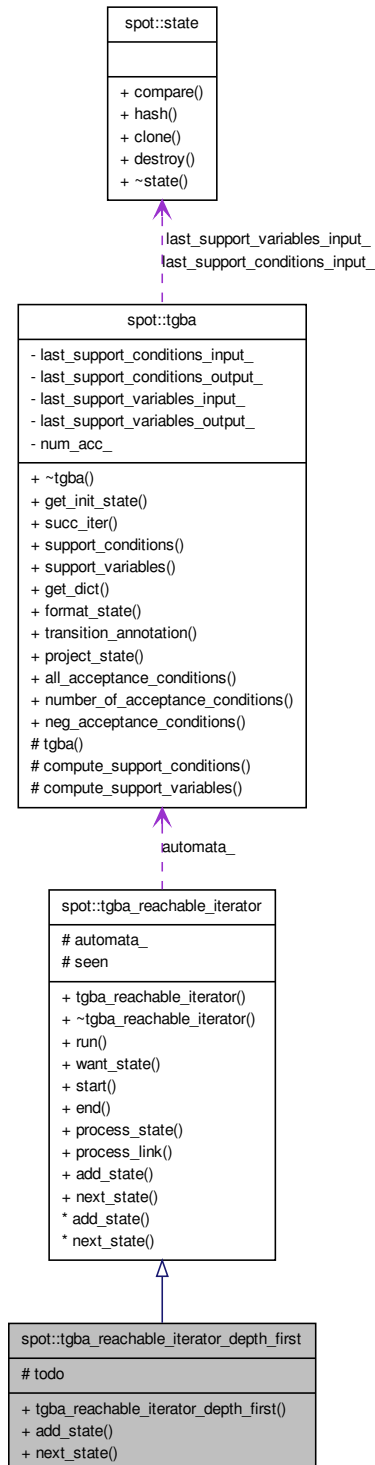
An implementation of `spot::tgba_reachable_iterator` that browses states depth first.

```
#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.

7.171.1 Detailed Description

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

7.171.2 Member Typedef Documentation

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

7.171.3 Constructor & Destructor Documentation

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

7.171.4 Member Function Documentation

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

Implements [spot::tgba_reachable_iterator](#).

7.171.4.2 `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](#).

7.171.4.3 `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](#).

7.171.4.4 `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.

7.171.4.5 `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](#).

7.171.4.6 `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.

7.171.4.7 `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](#).

7.171.4.8 `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.

7.171.5 Member Data Documentation

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

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

7.171.5.2 `seen_map spot::tgba_reachable_iterator::seen [protected, inherited]`

States already seen.

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

A stack of states yet to explore.

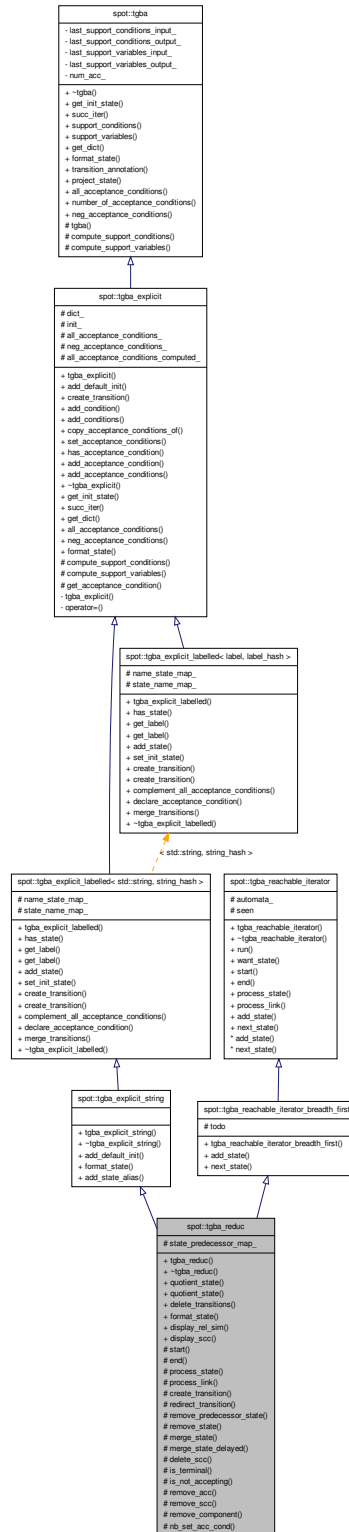
The documentation for this class was generated from the following file:

- tgbaalgos/[reachiter.hh](#)

7.172 spot::tgba_reduc Class Reference

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

Inheritance diagram for spot::tgba_reduc:



[illegible]

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.
- virtual std::string [format_state](#) (const [spot::state](#) *s) const
Format the state as a string for printing.
- virtual void [add_state_alias](#) (const std::string &alias_name, const std::string &real_name)
- 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.

7.172.1 Detailed Description

Explicit automata used in reductions.

7.172.2 Member Typedef Documentation

7.172.2.1 typedef std::string spot::tgba_explicit_labelled< std::string , string_hash >::label_t
[protected, inherited]

7.172.2.2 typedef Sgi::hash_map<std::string , tgba_explicit::state*, string_hash >
spot::tgba_explicit_labelled< std::string , string_hash >::ns_map [protected,
inherited]

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

7.172.2.4 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]

7.172.2.5 typedef Sgi::hash_map<const tgba_explicit::state*, std::list<state*>*,
ptr_hash<tgba_explicit::state> > spot::tgba_reduc::sp_map [protected]

7.172.2.6 typedef std::list<transition*> spot::tgba_explicit::state [inherited]

7.172.3 Constructor & Destructor Documentation

7.172.3.1 spot::tgba_reduc::tgba_reduc (const tgba * *a*)

7.172.3.2 spot::tgba_reduc::~~tgba_reduc ()

7.172.4 Member Function Documentation

7.172.4.1 void spot::tgba_explicit::add_acceptance_condition (transition * *t*, const ltl::formula * *f*) [inherited]

7.172.4.2 void spot::tgba_explicit::add_acceptance_conditions (transition * *t*, bdd *f*) [inherited]

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

7.172.4.3 void spot::tgba_explicit::add_condition (transition * *t*, const ltl::formula * *f*) [inherited]

7.172.4.4 void spot::tgba_explicit::add_conditions (transition * *t*, bdd *f*) [inherited]

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

7.172.4.5 virtual state* spot::tgba_explicit_string::add_default_init () [virtual, inherited]

Add a default initial state.

Implements [spot::tgba_explicit](#).

7.172.4.6 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_](#).

Referenced by `spot::tgba_explicit_string::add_state_alias()`.

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

Implements [spot::tgba_reachable_iterator](#).

7.172.4.8 `virtual void spot::tgba_explicit_string::add_state_alias (const std::string &
alias_name, const std::string & real_name) [inline, virtual, inherited]`

Create an alias for a state. Any reference to *alias_name* will act as a reference to *real_name*.

References `spot::tgba_explicit_labelled< std::string, string_hash >::add_state()`, and `spot::tgba_explicit_labelled< std::string, string_hash >::name_state_map_`.

7.172.4.9 `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()`.

7.172.4.10 `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_`.

7.172.4.11 `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](#).

7.172.4.12 `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](#).

7.172.4.13 `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.

7.172.4.14 `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\(\)](#).

7.172.4.15 `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\(\)](#).

7.172.4.16 `transition* spot::tgba_reduc::create_transition (const spot::state * source, const
spot::state * dest) [protected]`

Create a transition using two state of a TGBA.

7.172.4.17 `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\(\)](#).

7.172.4.18 `void spot::tgba_reduc::delete_scc () [protected]`

Remove all the scc which are terminal and doesn't contains all the acceptance conditions.

7.172.4.19 `void spot::tgba_reduc::delete_transitions (simulation_relation * rel)`

Delete some transitions with help of a simulation relation.

7.172.4.20 `void spot::tgba_reduc::display_rel_sim (simulation_relation * rel, std::ostream & os)`

7.172.4.21 `void spot::tgba_reduc::display_scc (std::ostream & os)`

7.172.4.22 `void spot::tgba_reduc::end () [protected, virtual]`

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.172.4.23 `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*.

7.172.4.24 `virtual std::string spot::tgba_explicit_string::format_state (const spot::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_explicit](#).

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

7.172.4.26 `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](#).

7.172.4.27 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.172.4.28 `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()`.

7.172.4.29 `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_`.

7.172.4.30 `bool spot::tgba_explicit::has_acceptance_condition (const ltl::formula * f) const [inherited]`

7.172.4.31 `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_`.

7.172.4.32 `bool spot::tgba_reduc::is_not_accepting (const spot::state * s, int n = -1) [protected]`

7.172.4.33 `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}
/// }
///

```

7.172.4.34 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.

7.172.4.35 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.

7.172.4.36 void spot::tgba_explicit_labelled< std::string , string_hash >::merge_transitions () [inline, inherited]

References spot::tgba_explicit_labelled< label, label_hash >::name_state_map_.

7.172.4.37 int spot::tgba_reduc::nb_set_acc_cond () const [protected]

7.172.4.38 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](#).

7.172.4.39 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](#).

7.172.4.40 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

7.172.4.41 `void spot::tgba_reduc::process_link (int in, int out, const tgba_succ_iterator * si)`
`[protected]`

7.172.4.42 `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.

7.172.4.43 `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](#).

7.172.4.44 `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 destroyed by the caller.

Reimplemented in `spot::tgba_product`, `spot::tgba_scc`, `spot::tgba_tba_proxy`, and `spot::tgba_union`.

7.172.4.45 `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.

7.172.4.46 `void spot::tgba_reduc::quotient_state (delayed_simulation_relation * rel)`

Build the quotient automata. Call this method when use to a delayed simulation relation.

7.172.4.47 `void spot::tgba_reduc::redirect_transition (const spot::state * s, const spot::state * simul) [protected]`

Remove all the transition from the state q , predecessor of both s and $simul$, which can be removed.

7.172.4.48 `void spot::tgba_reduc::remove_acc (const spot::state * s) [protected]`

If a scc maximal do not contains all the acceptance conditions we can remove all the acceptance conditions in this scc.

7.172.4.49 `void spot::tgba_reduc::remove_component (const spot::state * from) [protected]`

Same as `remove_scc` but more efficient.

For `compute_scc`.

7.172.4.50 `void spot::tgba_reduc::remove_predecessor_state (const state * s, const state * p) [protected]`

Remove p of the predecessor of s .

7.172.4.51 `void spot::tgba_reduc::remove_scc (spot::state * s) [protected]`

Remove all the state which belong to the same scc that s .

7.172.4.52 void spot::tgba_reduc::remove_state (const spot::state * s) [protected]

Remove all the transition leading to s. s is then unreachable and can be consider as remove.

7.172.4.53 void spot::tgba_reachable_iterator::run () [inherited]

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

This is a template method that will call [add_state\(\)](#), [next_state\(\)](#), [start\(\)](#), [end\(\)](#), [process_state\(\)](#), and [process_link\(\)](#), while it iterates over states.

7.172.4.54 void spot::tgba_explicit::set_acceptance_conditions (bdd acc) [inherited]

The the acceptance conditions.

7.172.4.55 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_](#).

7.172.4.56 void spot::tgba_reduc::start () [protected, virtual]

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

Reimplemented from [spot::tgba_reachable_iterator](#).

7.172.4.57 virtual tgba_succ_iterator* spot::tgba_explicit::succ_iter (const spot::state * local_state, const spot::state * global_state = 0, const tgba * global_automaton = 0) const [virtual, inherited]

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 destroy 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](#).

7.172.4.58 bdd spot::tgba::support_conditions (const state * state) const [inherited]

Get a formula that must hold whatever successor is taken.

Returns

A formula which must be verified for all successors of *state*.

This can be as simple as `bddtrue`, or more completely the disjunction of the condition of all successors. This is used as an hint by [succ_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute_support_conditions\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

7.172.4.59 bdd spot::tgba::support_variables (const state * state) const [inherited]

Get the conjunctions of variables tested by the outgoing transitions of *state*.

All variables tested by outgoing transitions must be returned. This is mandatory.

This is used as an hint by some [succ_iter\(\)](#) to reduce the number of successor to compute in a product.

Sub classes should implement [compute_support_variables\(\)](#), this function is just a wrapper that will cache the last return value for efficiency.

7.172.4.60 virtual std::string spot::tgba::transition_annotation (const tgba_succ_iterator * t) const [virtual, inherited]

Return a possible annotation for the transition pointed to by the iterator.

Implementing this function is optional; the default annotation is the empty string.

Parameters

t a non-done [tgba_succ_iterator](#) for this automata

Reimplemented in [spot::tgba_product](#), and [spot::tgba_scc](#).

7.172.4.61 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`.

7.172.5 Member Data Documentation

7.172.5.1 `bdd spot::tgba_explicit::all_acceptance_conditions_` `[mutable, protected, inherited]`

7.172.5.2 `bool spot::tgba_explicit::all_acceptance_conditions_computed_` `[mutable, protected, inherited]`

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

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

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

7.172.5.4 `bdd_dict* spot::tgba_explicit::dict_` `[protected, inherited]`

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

7.172.5.5 `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()`.

7.172.5.6 `ns_map spot::tgba_explicit_labelled< std::string, string_hash >::name_state_map_` `[protected, inherited]`

Referenced by `spot::tgba_explicit_string::add_state_alias()`.

7.172.5.7 `bdd spot::tgba_explicit::neg_acceptance_conditions_` `[protected, inherited]`

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

7.172.5.8 `seen_map spot::tgba_reachable_iterator::seen` `[protected, inherited]`

States already seen.

7.172.5.9 `sn_map spot::tgba_explicit_labelled< std::string , string_hash >::state_name_map_`
`[protected, inherited]`

7.172.5.10 `sp_map spot::tgba_reduc::state_predecessor_map_` `[protected]`

7.172.5.11 `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](#)

7.173 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

7.173.1 Detailed Description

An accepted run, for a tgba.

7.173.2 Member Typedef Documentation

7.173.2.1 `typedef std::list<step> spot::tgba_run::steps`

7.173.3 Constructor & Destructor Documentation

7.173.3.1 `spot::tgba_run::~~tgba_run ()`

7.173.3.2 `spot::tgba_run::tgba_run () [inline]`

7.173.3.3 `spot::tgba_run::tgba_run (const tgba_run & run)`

7.173.4 Member Function Documentation

7.173.4.1 `tgba_run& spot::tgba_run::operator= (const tgba_run & run)`

7.173.5 Member Data Documentation

7.173.5.1 `steps spot::tgba_run::cycle`

7.173.5.2 `steps spot::tgba_run::prefix`

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

- [tgbaalgos/emptiness.hh](#)

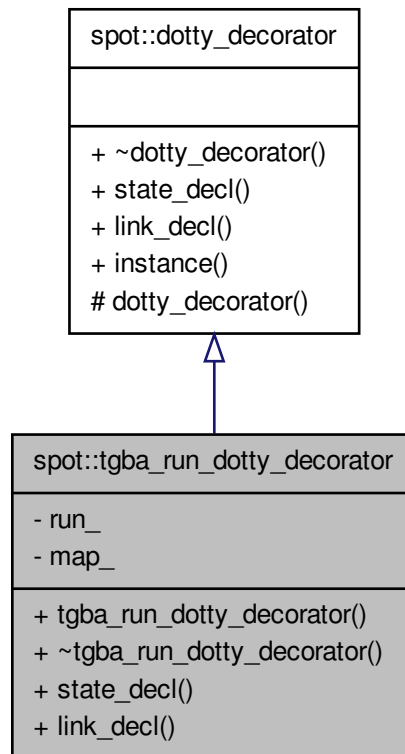
7.174 `spot::tgba_run_dotty_decorator` Class Reference

Highlight a [spot::tgba_run](#) on a [spot::tgba](#).

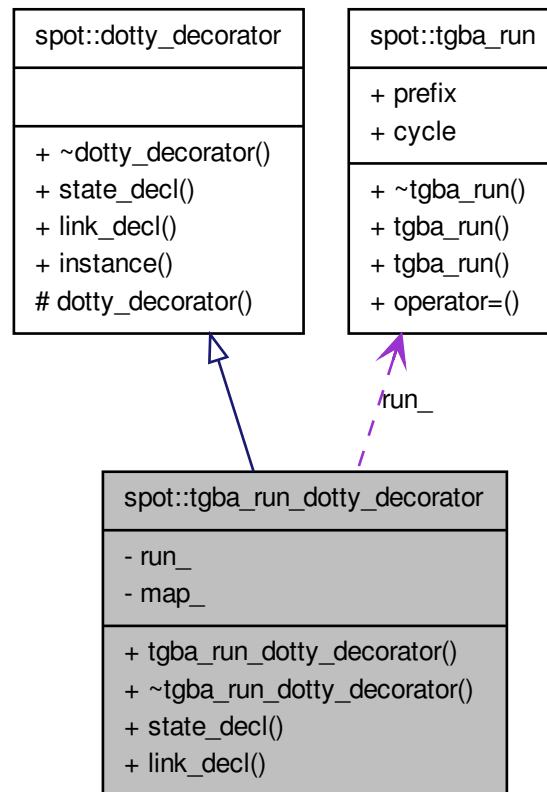
An instance of this class can be passed to [spot::dotty_reachable](#).

```
#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_](#)

7.174.1 Detailed Description

Highlight a [spot::tgba_run](#) on a [spot::tgba](#).

An instance of this class can be passed to [spot::dotty_reachable](#).

7.174.2 Member Typedef Documentation

7.174.2.1 typedef std::map<const state*, std::pair<step_set, step_set>, spot::state_ptr_less_than> spot::tgba_run_dotty_decorator::step_map [private]

7.174.2.2 typedef std::pair<tgba_run::steps::const_iterator, int> spot::tgba_run_dotty_decorator::step_num [private]

7.174.2.3 typedef std::list<step_num> spot::tgba_run_dotty_decorator::step_set [private]

7.174.3 Constructor & Destructor Documentation

7.174.3.1 spot::tgba_run_dotty_decorator::tgba_run_dotty_decorator (const tgba_run * *run*)

7.174.3.2 virtual spot::tgba_run_dotty_decorator::~~tgba_run_dotty_decorator () [virtual]

7.174.4 Member Function Documentation

7.174.4.1 static dotty_decorator* spot::dotty_decorator::instance() [static, inherited]

Get the unique instance of the default [dotty_decorator](#).

7.174.4.2 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](#).

7.174.4.3 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](#).

7.174.5 Member Data Documentation

7.174.5.1 `step_map` `spot::tgba_run_dotty_decorator::map_` [`private`]

7.174.5.2 `const tgba_run*` `spot::tgba_run_dotty_decorator::run_` [`private`]

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

- `tgbaalgos/rundotdec.hh`

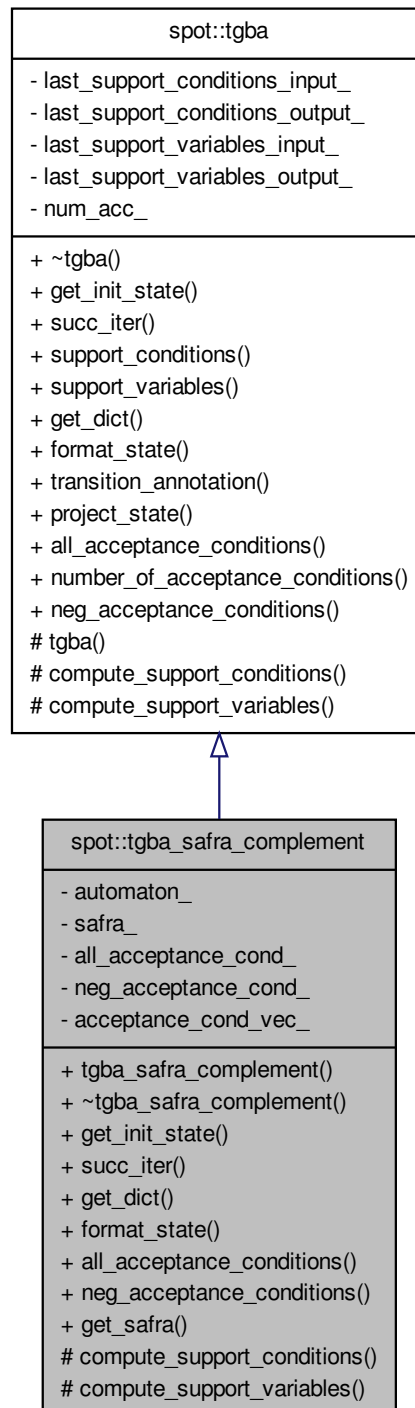
7.175 `spot::tgba_safra_complement` Class Reference

Build a complemented automaton.

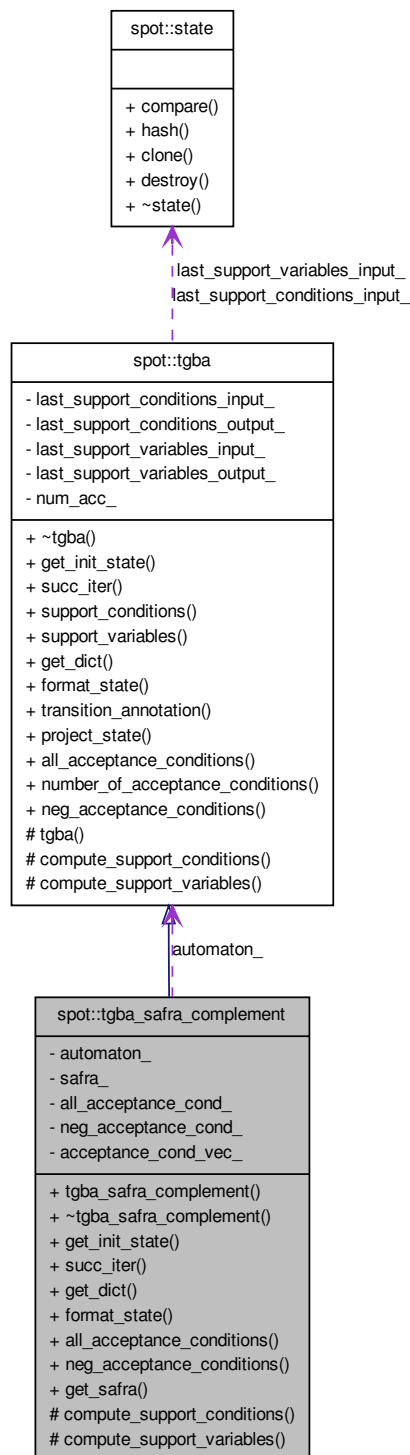
It creates an automaton that recognizes the negated language of *aut*.

```
#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](#) ()
- 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.
- void * [get_safra](#) () const
- [bdd](#) [support_conditions](#) (const [state](#) *state) const
Get a formula that must hold whatever successor is taken.
- [bdd](#) [support_variables](#) (const [state](#) *state) const
Get the conjunctions of variables tested by the outgoing transitions of state.
- 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_](#)
- void * [safra_](#)
- bdd [all_acceptance_cond_](#)
- bdd [neg_acceptance_cond_](#)
- std::vector< int > [acceptance_cond_vec_](#)

7.175.1 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](#).

7.175.2 Constructor & Destructor Documentation

7.175.2.1 `spot::tgba_safra_complement::tgba_safra_complement (const tgba * a)`

7.175.2.2 `virtual spot::tgba_safra_complement::~~tgba_safra_complement () [virtual]`

7.175.3 Member Function Documentation

7.175.3.1 `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](#).

7.175.3.2 `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](#).

7.175.3.3 `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](#).

7.175.3.4 `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](#).

7.175.3.5 `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](#).

7.175.3.6 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.175.3.7 `void* spot::tgba_safra_complement::get_safra () const [inline]`

References `safra_`.

7.175.3.8 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](#).

7.175.3.9 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

7.175.3.10 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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.175.3.11 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 destroy 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](#).

7.175.3.12 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.

7.175.3.13 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.

7.175.3.14 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](#), and [spot::tgba_scc](#).

7.175.4 Member Data Documentation

7.175.4.1 `std::vector<int> spot::tgba_safra_complement::acceptance_cond_vec_` `[private]`

7.175.4.2 `bdd spot::tgba_safra_complement::all_acceptance_cond_` `[private]`

7.175.4.3 `const tgba* spot::tgba_safra_complement::automaton_` `[private]`

7.175.4.4 `bdd spot::tgba_safra_complement::neg_acceptance_cond_` `[private]`

7.175.4.5 `void* spot::tgba_safra_complement::safra_` `[private]`

Referenced by `get_safra()`.

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

- [tgba/tgbasafracomplement.hh](#)

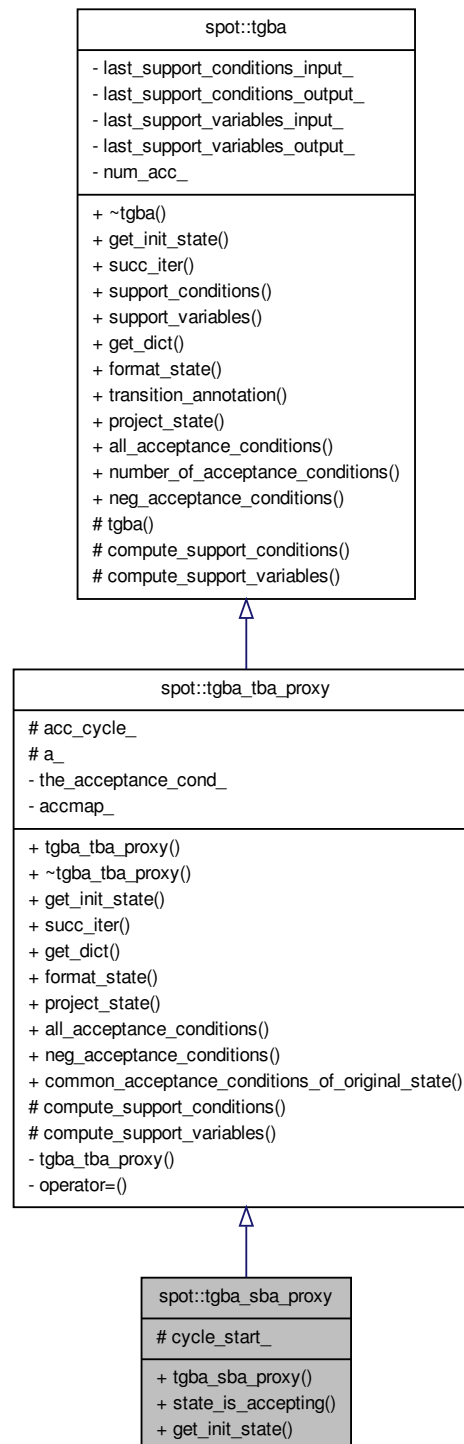
7.176 `spot::tgba_sba_proxy` Class Reference

Degeneralize a [spot::tgba](#) on the fly, producing an SBA.

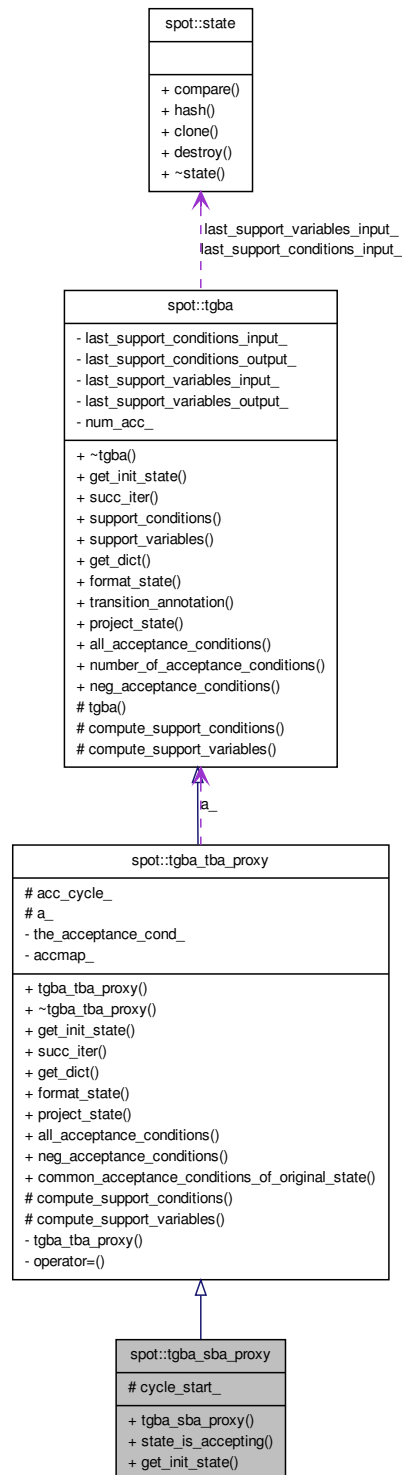
This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly.

```
#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 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 [common_acceptance_conditions_of_original_state](#) (const [state](#) *ostate) const
Return the acceptance conditions common to all outgoing transitions of state ostate in the original 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 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\(\)](#).

Protected Attributes

- [cycle_list::iterator](#) [cycle_start_](#)
- [cycle_list](#) [acc_cycle_](#)
- const [tgba](#) * [a_](#)

7.176.1 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](#).)

7.176.2 Member Typedef Documentation

7.176.2.1 `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list [inherited]`

7.176.3 Constructor & Destructor Documentation

7.176.3.1 `spot::tgba_sba_proxy::tgba_sba_proxy (const tgba * a)`

7.176.4 Member Function Documentation

7.176.4.1 `virtual bdd spot::tgba_tba_proxy::all_acceptance_conditions () const [virtual, inherited]`

Return the set of all acceptance conditions used by this automaton.

The goal of the emptiness check is to ensure that a strongly connected component walks through each of these acceptance conditions. I.e., the union of the acceptance conditions of all transition in the SCC should be equal to the result of this function.

Implements [spot::tgba](#).

7.176.4.2 `bdd spot::tgba_tba_proxy::common_acceptance_conditions_of_original_state (const state * ostate) const [inherited]`

Return the acceptance conditions common to all outgoing transitions of state *ostate* in the original automaton.

This internal function is only meant to be used to implement the iterator returned by `succ_iter`.

The result of this function is computed the first time, and then cached.

7.176.4.3 `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](#).

7.176.4.4 `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](#).

7.176.4.5 `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](#).

7.176.4.6 `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](#).

7.176.4.7 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 `destroy` it when no longer needed.

Reimplemented from [spot::tgba_tba_proxy](#).

7.176.4.8 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](#).

7.176.4.9 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

7.176.4.10 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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.176.4.11 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.

7.176.4.12 `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 destroy 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](#).

7.176.4.13 `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.

7.176.4.14 `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.

7.176.4.15 `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`, and `spot::tgba_scc`.

7.176.5 Member Data Documentation

7.176.5.1 `const tgba* spot::tgba_tba_proxy::a_ [protected, inherited]`

7.176.5.2 `cycle_list spot::tgba_tba_proxy::acc_cycle_ [protected, inherited]`

7.176.5.3 `cycle_list::iterator spot::tgba_sba_proxy::cycle_start_ [protected]`

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

- [tgba/tgbatba.hh](#)

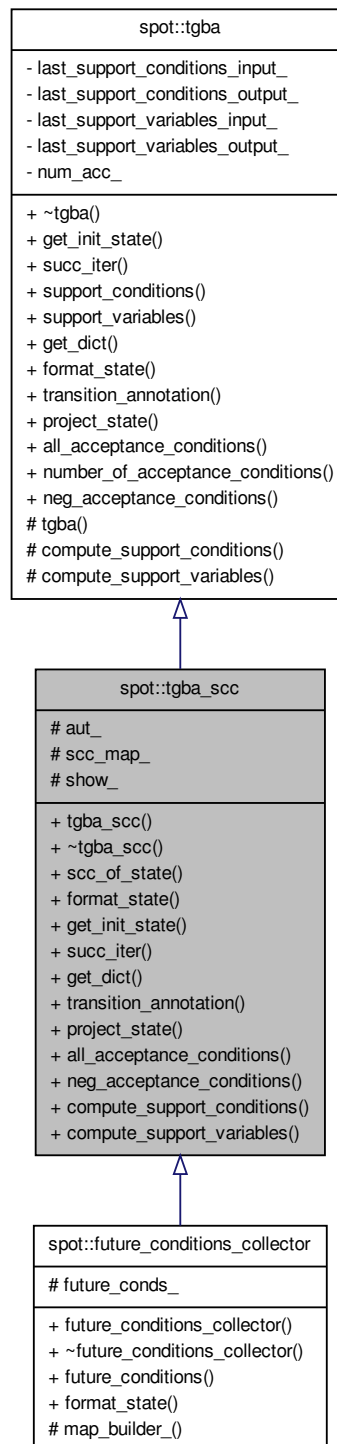
7.177 `spot::tgba_scc` Class Reference

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

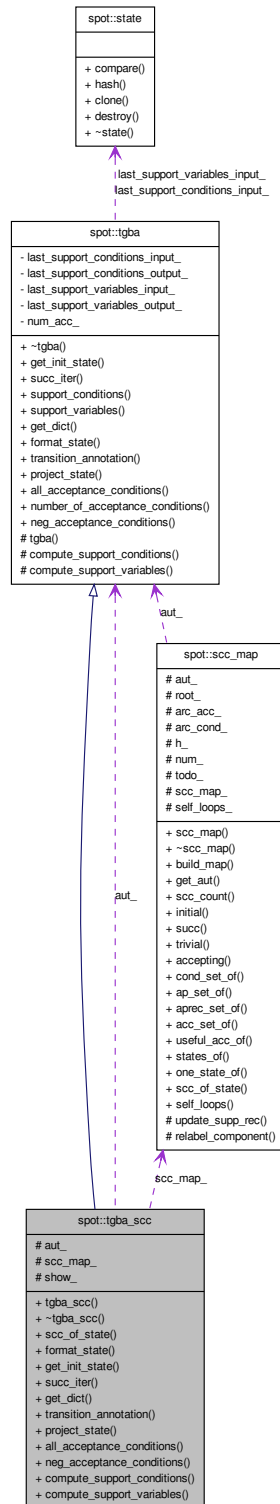
This class is a `spot::tgba` wrapper that simply add a new method `scc_of_state()` to retrieve the number of a SCC a state belongs to.

```
#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_](#)

7.177.1 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.

7.177.2 Constructor & Destructor Documentation

7.177.2.1 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.

7.177.2.2 virtual spot::tgba_scc::~tgba_scc () [virtual]

7.177.3 Member Function Documentation

7.177.3.1 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](#).

7.177.3.2 virtual bdd spot::tgba_scc::compute_support_conditions (const state * *state*) const [virtual]

Do the actual computation of [tgba::support_conditions\(\)](#).

Implements [spot::tgba](#).

7.177.3.3 `virtual bdd spot::tgba_scc::compute_support_variables (const state * state) const`
[virtual]

Do the actual computation of [tgba::support_variables\(\)](#).

Implements [spot::tgba](#).

7.177.3.4 `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](#).

7.177.3.5 `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](#).

7.177.3.6 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.177.3.7 `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](#).

7.177.3.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
[virtual, inherited]

The number of acceptance conditions.

7.177.3.9 `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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.177.3.10 `unsigned spot::tgba_scc::scc_of_state (const spot::state * s) const`

Returns the number of the SCC *s* belongs to.

7.177.3.11 `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 destroy 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](#).

7.177.3.12 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.

7.177.3.13 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.

7.177.3.14 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`.

7.177.4 Member Data Documentation**7.177.4.1 const tgba* spot::tgba_scc::aut_ [protected]****7.177.4.2 scc_map spot::tgba_scc::scc_map_ [protected]**

7.177.4.3 `bool spot::tgba_scc::show_` [protected]

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

- [tgba/tgbascc.hh](#)

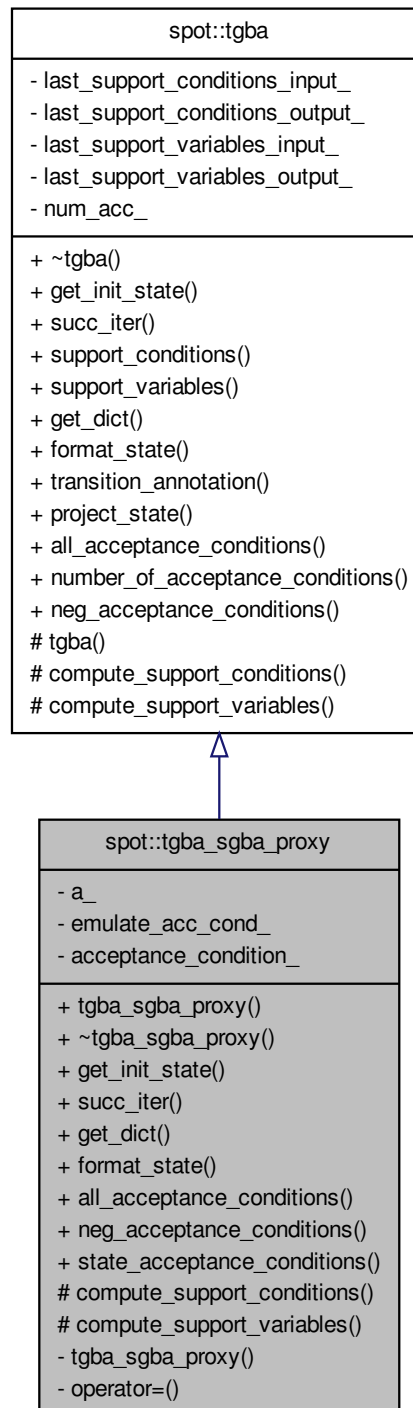
7.178 `spot::tgba_sgba_proxy` Class Reference

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

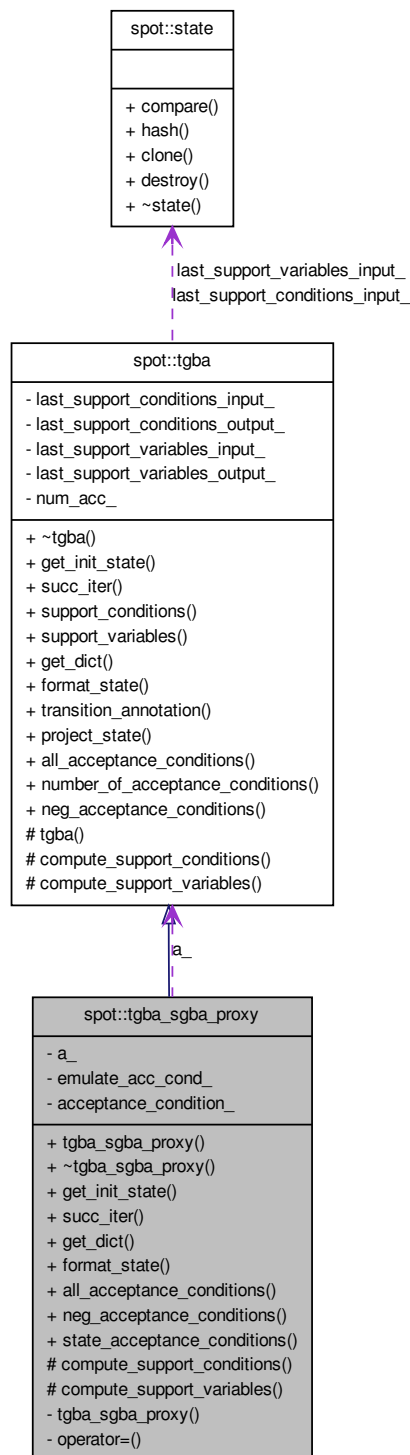
This class acts as a proxy in front of a [spot::tgba](#), that should label on states on-the-fly. The result is still a [spot::tgba](#), but acceptances conditions are also on states.

```
#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_`

7.178.1 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 acceptance conditions are also on states.

7.178.2 Constructor & Destructor Documentation

7.178.2.1 `spot::tgba_sgba_proxy::tgba_sgba_proxy (const tgba * a, bool no_zero_acc = true)`

7.178.2.2 `virtual spot::tgba_sgba_proxy::~~tgba_sgba_proxy ()` **[virtual]**

7.178.2.3 `spot::tgba_sgba_proxy::tgba_sgba_proxy (const tgba_sgba_proxy &)` **[private]**

7.178.3 Member Function Documentation

7.178.3.1 `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`.

7.178.3.2 `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](#).

7.178.3.3 `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](#).

7.178.3.4 `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](#).

7.178.3.5 `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](#).

7.178.3.6 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.178.3.7 `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](#).

7.178.3.8 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

7.178.3.9 `tgba_sgba_proxy& spot::tgba_sgba_proxy::operator= (const tgba_sgba_proxy &)`
`[private]`

7.178.3.10 `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 destroyed by the caller.

Reimplemented in [spot::tgba_product](#), [spot::tgba_scc](#), [spot::tgba_tba_proxy](#), and [spot::tgba_union](#).

7.178.3.11 `bdd spot::tgba_sgba_proxy::state_acceptance_conditions (const state * state) const`

Retrieve the acceptance condition of a state.

7.178.3.12 `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 destroy 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](#).

7.178.3.13 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.

7.178.3.14 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.

7.178.3.15 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](#), and [spot::tgba_scc](#).

7.178.4 Member Data Documentation

7.178.4.1 `const tgba* spot::tgba_sgba_proxy::a_` `[private]`

7.178.4.2 `bdd spot::tgba_sgba_proxy::acceptance_condition_` `[private]`

7.178.4.3 `bool spot::tgba_sgba_proxy::emulate_acc_cond_` `[private]`

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

- [tgba/tgbasgba.hh](#)

7.179 spot::tgba_statistics Struct Reference

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

Public Member Functions

- `std::ostream & dump` (`std::ostream &out`) `const`

Public Attributes

- unsigned [transitions](#)
- unsigned [states](#)

7.179.1 Member Function Documentation

7.179.1.1 `std::ostream& spot::tgba_statistics::dump (std::ostream & out) const`

7.179.2 Member Data Documentation

7.179.2.1 `unsigned spot::tgba_statistics::states`

7.179.2.2 `unsigned spot::tgba_statistics::transitions`

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

- [tgbaalgos/stats.hh](#)

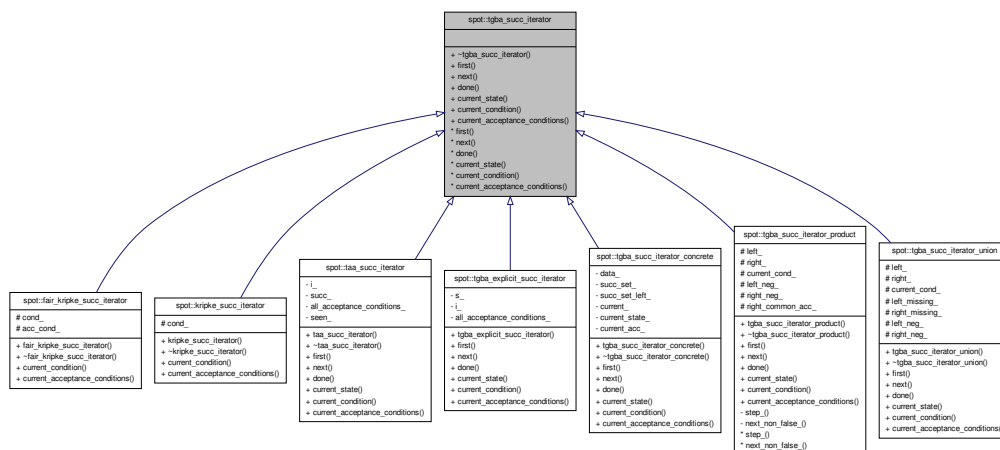
7.180 spot::tgba_succ_iterator Class Reference

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.

7.180.1 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.

7.180.2 Constructor & Destructor Documentation

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

7.180.3 Member Function Documentation

7.180.3.1 `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::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](#).

7.180.3.2 `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::fair_kripke_succ_iterator](#), [spot::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](#).

7.180.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

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](#).

7.180.3.4 `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`.

7.180.3.5 `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`.

7.180.3.6 `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`.

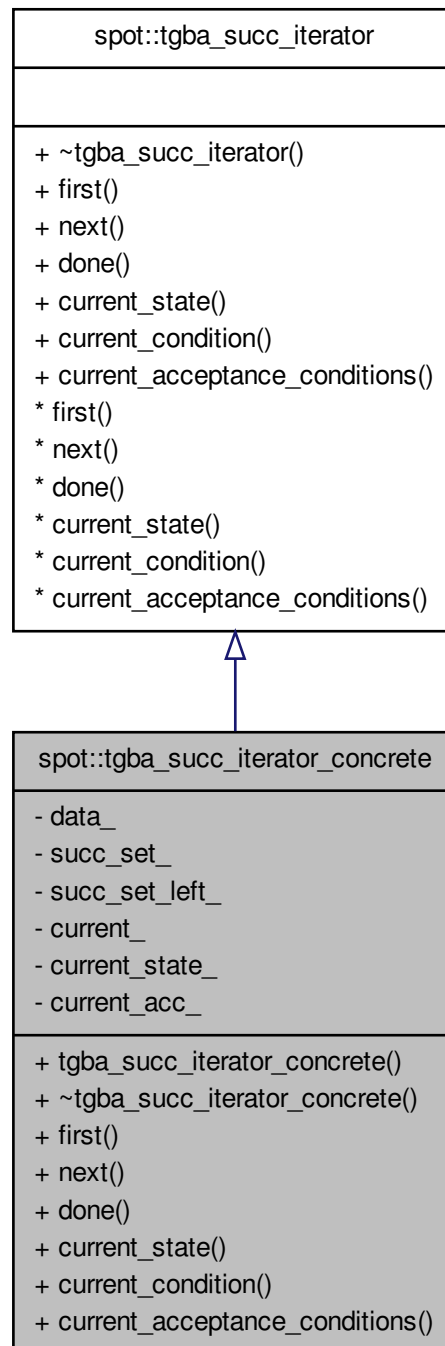
The documentation for this class was generated from the following file:

- `tgba/succiter.hh`

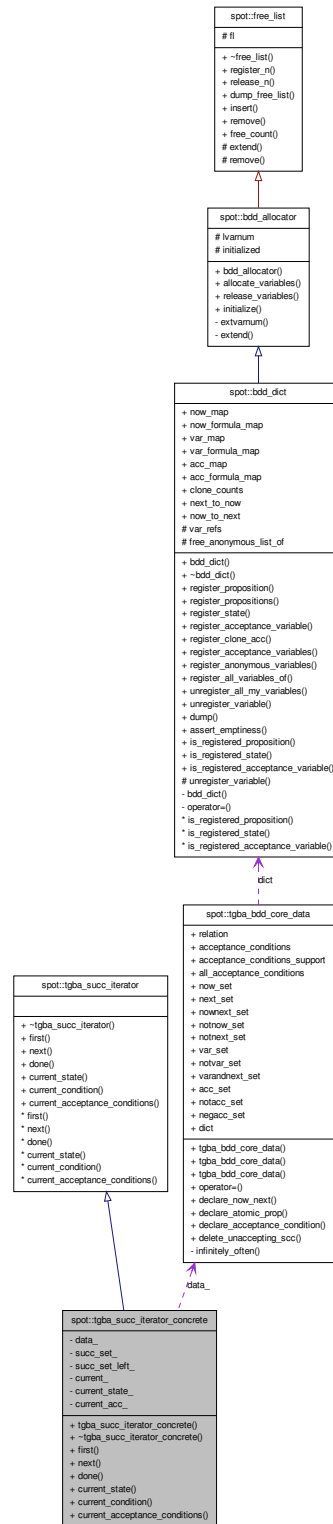
7.181 `spot::tgba_succ_iterator_concrete` Class Reference

```
#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.

7.181.1 Detailed Description

A concrete iterator over successors of a TGBA state.

7.181.2 Constructor & Destructor Documentation

7.181.2.1 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.

7.181.2.2 virtual spot::tgba_succ_iterator_concrete::~~tgba_succ_iterator_concrete () const [virtual]

7.181.3 Member Function Documentation

7.181.3.1 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](#).

7.181.3.2 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](#).

7.181.3.3 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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

Implements [spot::tgba_succ_iterator](#).

7.181.3.4 `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](#).

7.181.3.5 `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](#).

7.181.3.6 `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](#).

7.181.4 Member Data Documentation

7.181.4.1 `bdd spot::tgba_succ_iterator_concrete::current_ [private]`

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

7.181.4.2 `bdd spot::tgba_succ_iterator_concrete::current_acc_ [private]`

Acceptance conditions for the current transition.

7.181.4.3 `bdd spot::tgba_succ_iterator_concrete::current_state_ [private]`

Current successor, as a conjunction of Now variables.

7.181.4.4 `const tgba_bdd_core_data& spot::tgba_succ_iterator_concrete::data_ [private]`

Core data of the automaton.

7.181.4.5 `bdd spot::tgba_succ_iterator_concrete::succ_set_ [private]`

The set of successors.

7.181.4.6 `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:

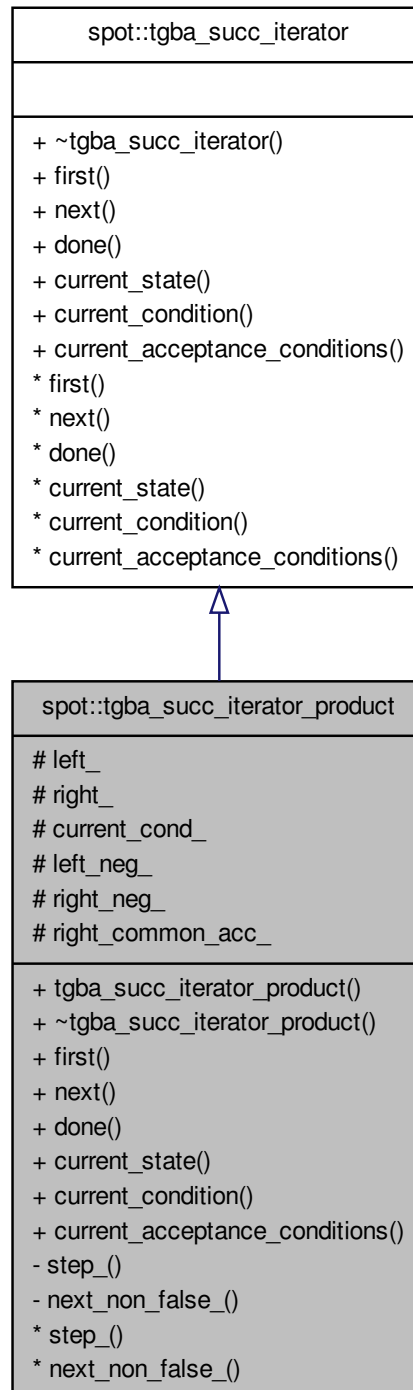
- [tgba/succiterconcrete.hh](#)

7.182 `spot::tgba_succ_iterator_product` Class Reference

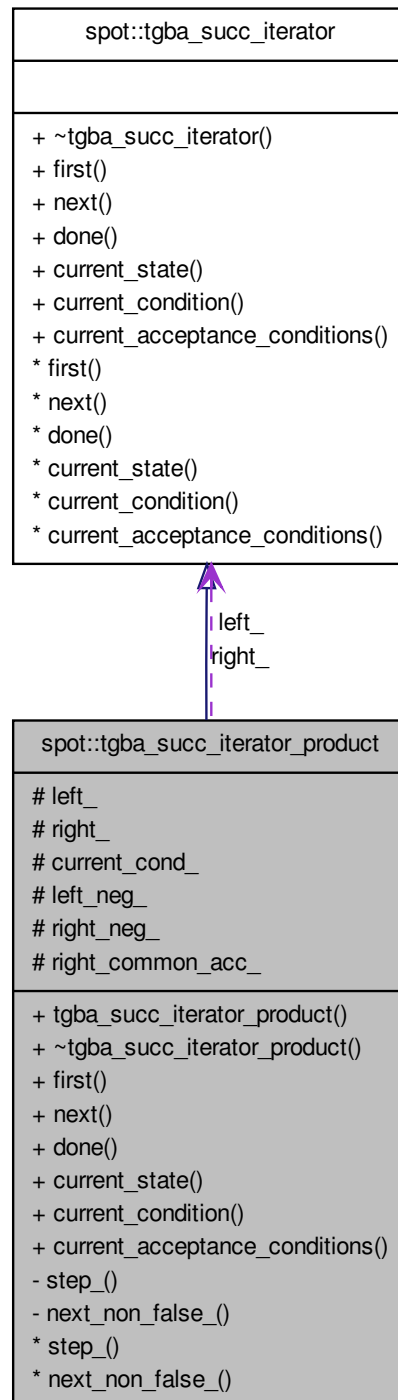
Iterate over the successors of a product computed on the fly.

```
#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](#)

7.182.1 Detailed Description

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

7.182.2 Constructor & Destructor Documentation

7.182.2.1 `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)`

7.182.2.2 `virtual spot::tgba_succ_iterator_product::~tgba_succ_iterator_product ()`
[**virtual**]

7.182.3 Member Function Documentation

7.182.3.1 `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](#).

7.182.3.2 `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](#).

7.182.3.3 `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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

Implements [spot::tgba_succ_iterator](#).

7.182.3.4 `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`.

7.182.3.5 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`.

7.182.3.6 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`.

7.182.3.7 void spot::tgba_succ_iterator_product::next_non_false_() [private]

7.182.3.8 void spot::tgba_succ_iterator_product::step_() [private]

Internal routines to advance to the next successor.

7.182.4 Friends And Related Function Documentation

7.182.4.1 friend class tgba_product [friend]

7.182.5 Member Data Documentation

7.182.5.1 bdd spot::tgba_succ_iterator_product::current_cond_ [protected]

7.182.5.2 tgba_succ_iterator* spot::tgba_succ_iterator_product::left_ [protected]

7.182.5.3 bdd spot::tgba_succ_iterator_product::left_neg_ [protected]

7.182.5.4 tgba_succ_iterator* spot::tgba_succ_iterator_product::right_ [protected]

7.182.5.5 bddPair* spot::tgba_succ_iterator_product::right_common_acc_ [protected]

7.182.5.6 bdd spot::tgba_succ_iterator_product::right_neg_ [protected]

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

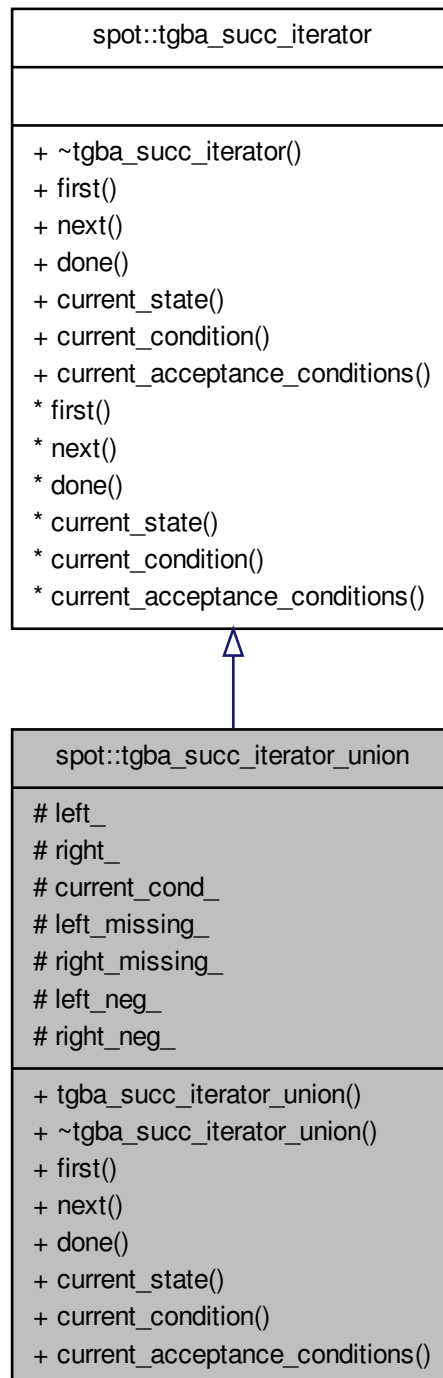
- [tgba/tgbaproduct.hh](#)

7.183 spot::tgba_succ_iterator_union Class Reference

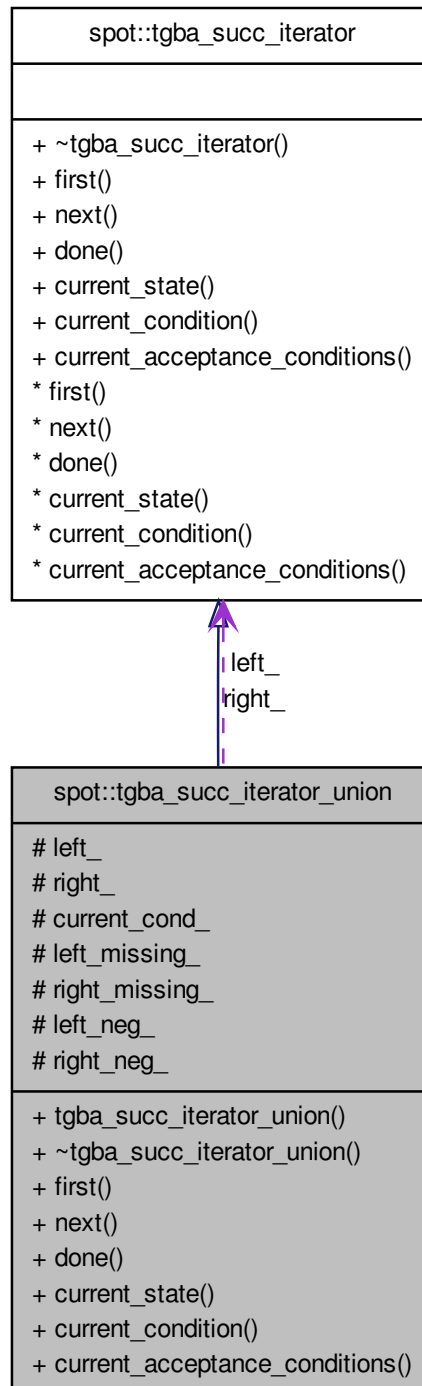
Iterate over the successors of an union computed on the fly.

```
#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](#)

7.183.1 Detailed Description

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

7.183.2 Constructor & Destructor Documentation

- 7.183.2.1** [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)

7.183.2.2 virtual spot::tgba_succ_iterator_union::~~tgba_succ_iterator_union () [virtual]

7.183.3 Member Function Documentation

7.183.3.1 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](#).

7.183.3.2 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](#).

7.183.3.3 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.

The returned state should be destroyed (see [state::destroy](#)) by the caller after it is no longer used.

Implements [spot::tgba_succ_iterator](#).

7.183.3.4 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](#).

7.183.3.5 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`.

7.183.3.6 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`.

7.183.4 Friends And Related Function Documentation

7.183.4.1 friend class tgba_union [friend]

7.183.5 Member Data Documentation

7.183.5.1 bdd spot::tgba_succ_iterator_union::current_cond_ [protected]

7.183.5.2 tgba_succ_iterator* spot::tgba_succ_iterator_union::left_ [protected]

7.183.5.3 bdd spot::tgba_succ_iterator_union::left_missing_ [protected]

7.183.5.4 bdd spot::tgba_succ_iterator_union::left_neg_ [protected]

7.183.5.5 `tgba_succ_iterator*` `spot::tgba_succ_iterator_union::right_` [protected]

7.183.5.6 `bdd` `spot::tgba_succ_iterator_union::right_missing_` [protected]

7.183.5.7 `bdd` `spot::tgba_succ_iterator_union::right_neg_` [protected]

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

- [tgba/tgbaunion.hh](#)

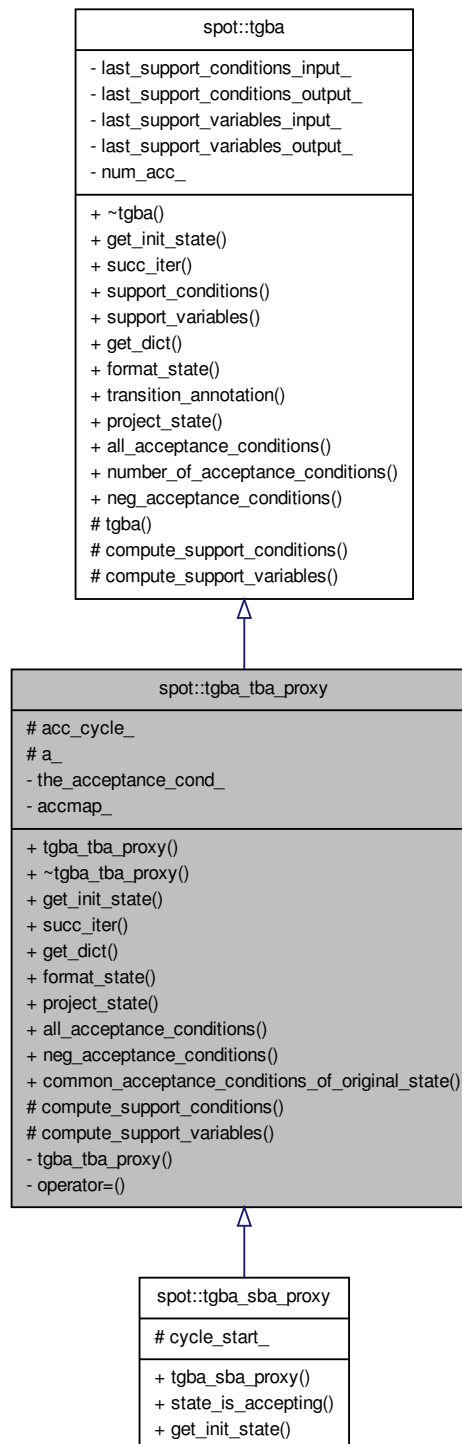
7.184 `spot::tgba_tba_proxy` Class Reference

Degeneralize a [spot::tgba](#) on the fly, producing a TBA.

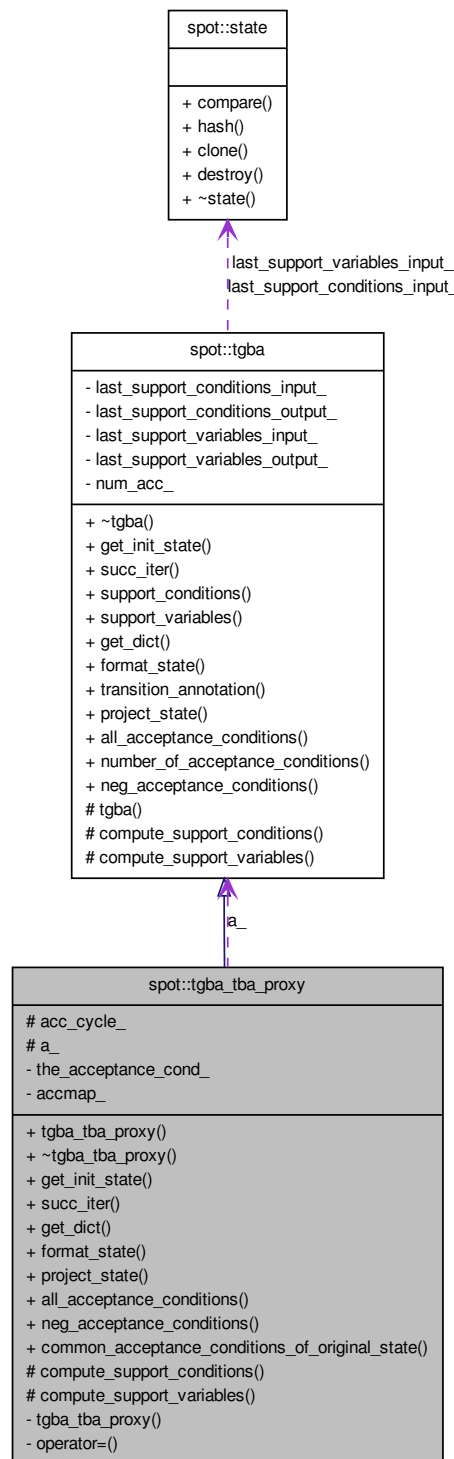
This class acts as a proxy in front of a [spot::tgba](#), that should be degeneralized on the fly. The result is still a [spot::tgba](#), but it will always have exactly one acceptance condition so it could be called TBA (without the G).

```
#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 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 [common_acceptance_conditions_of_original_state](#) (const [state](#) *ostate) const
Return the acceptance conditions common to all outgoing transitions of state ostate in the original 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 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()`.

Protected Attributes

- `cycle_list acc_cycle_`
- const `tgba * a_`

Private Types

- typedef `Sgi::hash_map< const state *, bdd, state_ptr_hash, state_ptr_equal > accmap_t`

Private Member Functions

- `tgba_tba_proxy` (const `tgba_tba_proxy &`)
- `tgba_tba_proxy & operator=` (const `tgba_tba_proxy &`)

Private Attributes

- bdd `the_acceptance_cond_`
- `accmap_t accmap_`

7.184.1 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`](#)

7.184.2 Member Typedef Documentation

- 7.184.2.1** `typedef Sgi::hash_map<const state*, bdd, state_ptr_hash, state_ptr_equal> spot::tgba_tba_proxy::accmap_t [private]`

7.184.2.2 `typedef std::list<bdd> spot::tgba_tba_proxy::cycle_list`

7.184.3 Constructor & Destructor Documentation

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

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

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

7.184.4 Member Function Documentation

7.184.4.1 `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](#).

7.184.4.2 `bdd spot::tgba_tba_proxy::common_acceptance_conditions_of_original_state (const state * ostate) const`

Return the acceptance conditions common to all outgoing transitions of state *ostate* in the original automaton.

This internal function is only meant to be used to implement the iterator returned by `succ_iter`.

The result of this function is computed the first time, and then cached.

7.184.4.3 `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](#).

7.184.4.4 `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](#).

7.184.4.5 `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](#).

7.184.4.6 `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](#).

7.184.4.7 `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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

Reimplemented in [spot::tgba_sba_proxy](#).

7.184.4.8 `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](#).

7.184.4.9 `virtual unsigned int spot::tgba::number_of_acceptance_conditions () const`
`[virtual, inherited]`

The number of acceptance conditions.

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

7.184.4.11 `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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.184.4.12 `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 destroy 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](#).

7.184.4.13 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.

7.184.4.14 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.

7.184.4.15 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`, and `spot::tgba_scc`.

7.184.5 Member Data Documentation**7.184.5.1 const tgba* spot::tgba_tba_proxy::a_ [protected]****7.184.5.2 cycle_list spot::tgba_tba_proxy::acc_cycle_ [protected]**

7.184.5.3 accmap_t spot::tgba_tba_proxy::accmap_ [mutable, private]

7.184.5.4 bdd spot::tgba_tba_proxy::the_acceptance_cond_ [private]

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

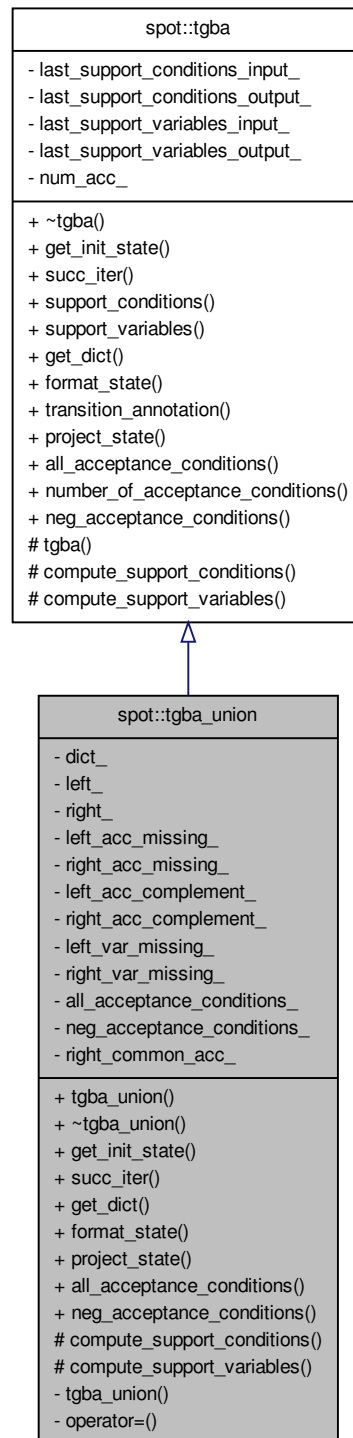
- [tgba/tgbatba.hh](#)

7.185 spot::tgba_union Class Reference

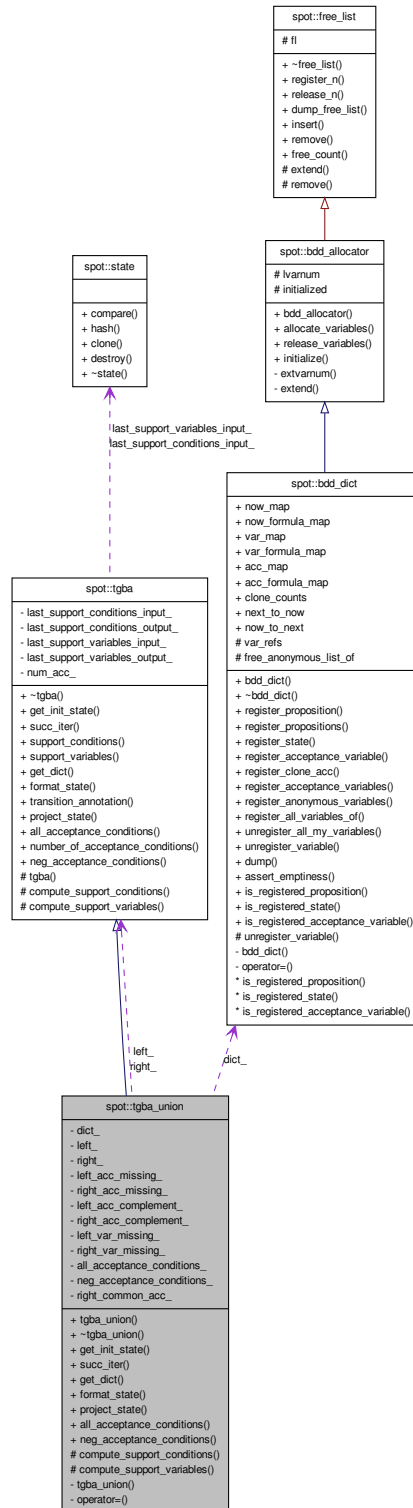
A lazy union. (States are computed on the fly.).

```
#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_](#)

7.185.1 Detailed Description

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

7.185.2 Constructor & Destructor Documentation

7.185.2.1 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.

7.185.2.2 virtual spot::tgba_union::~tgba_union () [virtual]

7.185.2.3 spot::tgba_union::tgba_union (const [tgba_union](#) &) [private]

7.185.3 Member Function Documentation

7.185.3.1 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](#).

7.185.3.2 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](#).

7.185.3.3 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](#).

7.185.3.4 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](#).

7.185.3.5 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](#).

7.185.3.6 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 `destroy` it when no longer needed.

Implements [spot::tgba](#).

7.185.3.7 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](#).

7.185.3.8 virtual unsigned int spot::tgba::number_of_acceptance_conditions () const [virtual, inherited]

The number of acceptance conditions.

7.185.3.9 tgba_union& spot::tgba_union::operator= (const tgba_union &) [private]**7.185.3.10 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 destroyed by the caller.

Reimplemented from [spot::tgba](#).

7.185.3.11 `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 destroy 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`.

7.185.3.12 `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.

7.185.3.13 `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.

7.185.3.14 `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](#), and [spot::tgba_scc](#).

7.185.4 Member Data Documentation

7.185.4.1 `bdd spot::tgba_union::all_acceptance_conditions_ [private]`

7.185.4.2 `bdd_dict* spot::tgba_union::dict_ [private]`

7.185.4.3 `const tgba* spot::tgba_union::left_ [private]`

7.185.4.4 `bdd spot::tgba_union::left_acc_complement_ [private]`

7.185.4.5 `bdd spot::tgba_union::left_acc_missing_ [private]`

7.185.4.6 `bdd spot::tgba_union::left_var_missing_ [private]`

7.185.4.7 `bdd spot::tgba_union::neg_acceptance_conditions_ [private]`

7.185.4.8 `const tgba* spot::tgba_union::right_ [private]`

7.185.4.9 `bdd spot::tgba_union::right_acc_complement_ [private]`

7.185.4.10 `bdd spot::tgba_union::right_acc_missing_ [private]`

7.185.4.11 `bddPair* spot::tgba_union::right_common_acc_ [private]`

7.185.4.12 `bdd spot::tgba_union::right_var_missing_ [private]`

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

- [tgba/tgbaunion.hh](#)

7.186 `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](#)

7.186.1 Detailed Description

A structure to record elapsed time in clock ticks.

7.186.2 Constructor & Destructor Documentation

7.186.2.1 `spot::time_info::time_info() [inline]`

7.186.3 Member Data Documentation

7.186.3.1 `clock_t spot::time_info::stime`

Referenced by `spot::timer::start()`, `spot::timer::stime()`, and `spot::timer::stop()`.

7.186.3.2 `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:

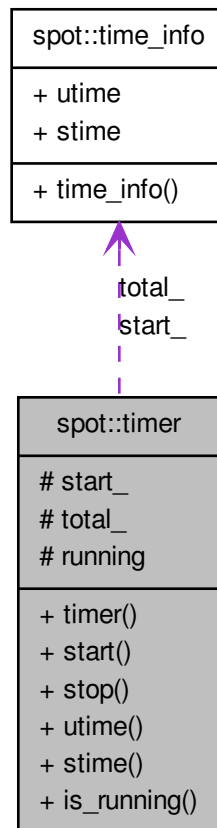
- [misc/timer.hh](#)

7.187 `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`

7.187.1 Detailed Description

A timekeeper that accumulate interval of time.

7.187.2 Constructor & Destructor Documentation

7.187.2.1 `spot::timer::timer ()` [inline]

7.187.3 Member Function Documentation

7.187.3.1 `bool spot::timer::is_running () const` [inline]

Whether the timer is running.

References `running`.

7.187.3.2 `void spot::timer::start ()` [inline]

Start a time interval.

References `running`, `start_`, `spot::time_info::stime`, and `spot::time_info::utime`.

7.187.3.3 `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_`.

7.187.3.4 `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`.

7.187.3.5 `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`.

7.187.4 Member Data Documentation

7.187.4.1 `bool spot::timer::running [protected]`

Referenced by `is_running()`, `start()`, and `stop()`.

7.187.4.2 `time_info spot::timer::start_ [protected]`

Referenced by `start()`, and `stop()`.

7.187.4.3 `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](#)

7.188 `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`

7.188.1 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.

7.188.2 Member Typedef Documentation

7.188.2.1 `typedef std::pair<spot::timer, int> spot::timer_map::item_type [protected]`

7.188.2.2 `typedef std::map<std::string, item_type> spot::timer_map::tm_type [protected]`

7.188.3 Member Function Documentation

7.188.3.1 `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`.

7.188.3.2 `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.

7.188.3.3 `std::ostream& spot::timer_map::print (std::ostream & os) const`

Format information about all timers in a table.

7.188.3.4 `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.

7.188.3.5 `void spot::timer_map::stop (const std::string & name) [inline]`

Stop timer *name*.

The timer must have been previously started with `start()`.

References tm.

7.188.3.6 `spot::timer& spot::timer_map::timer (const std::string & name) [inline]`

Return the timer *name*.

References tm.

7.188.3.7 `const spot::timer& spot::timer_map::timer (const std::string & name) const [inline]`

Return the timer *name*.

References tm.

7.188.4 Member Data Documentation

7.188.4.1 tm_type spot::timer_map::tm [protected]

Referenced by cancel(), empty(), start(), stop(), and timer().

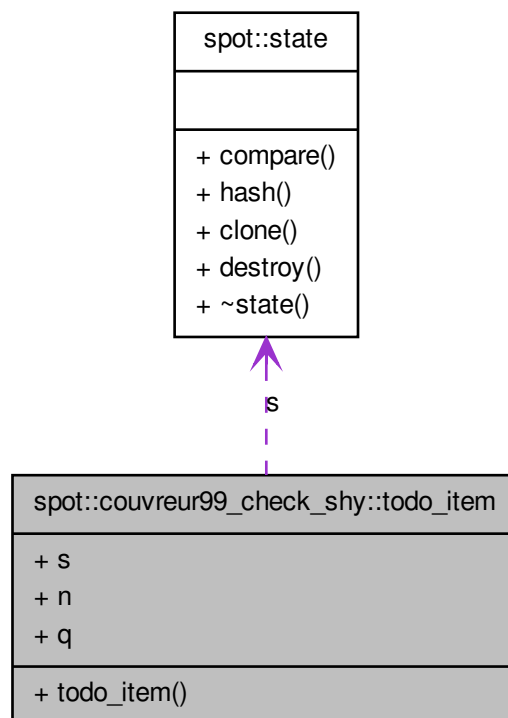
The documentation for this class was generated from the following file:

- misc/timer.hh

7.189 spot::couvreur99_check_shy::todo_item Struct Reference

```
#include <tgbalgorithms/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](#)

7.189.1 Constructor & Destructor Documentation

7.189.1.1 spot::couvreur99_check_shy::todo_item::todo_item (const state * *s*, int *n*, couvreur99_check_shy * *shy*)

7.189.2 Member Data Documentation

7.189.2.1 int spot::couvreur99_check_shy::todo_item::n

7.189.2.2 succ_queue spot::couvreur99_check_shy::todo_item::q

7.189.2.3 const state* spot::couvreur99_check_shy::todo_item::s

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

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

7.190 spot::taa_tgba::transition Struct Reference

Explicit transitions.

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

Public Attributes

- bdd [condition](#)
- bdd [acceptance_conditions](#)
- const [state_set](#) * [dst](#)

7.190.1 Detailed Description

Explicit transitions.

7.190.2 Member Data Documentation

7.190.2.1 `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()`.

7.190.2.2 `bdd spot::taa_tgba::transition::condition`

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

7.190.2.3 `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](#)

7.191 `spot::ltl::nfa::transition` Struct Reference

Explicit transitions.

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

Public Attributes

- [label lbl](#)
- `const state * dst`

7.191.1 Detailed Description

Explicit transitions.

7.191.2 Member Data Documentation

7.191.2.1 `const state* spot::ltl::nfa::transition::dst`

7.191.2.2 `label spot::ltl::nfa::transition::lbl`

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

- [ltlast/nfa.hh](#)

7.192 spot::tgba_explicit::transition Struct Reference

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

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

Public Attributes

- bdd [condition](#)
- bdd [acceptance_conditions](#)
- const [state](#) * [dest](#)

7.192.1 Detailed Description

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

7.192.2 Member Data Documentation

7.192.2.1 bdd spot::tgba_explicit::transition::acceptance_conditions

7.192.2.2 bdd spot::tgba_explicit::transition::condition

7.192.2.3 const state* spot::tgba_explicit::transition::dest

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

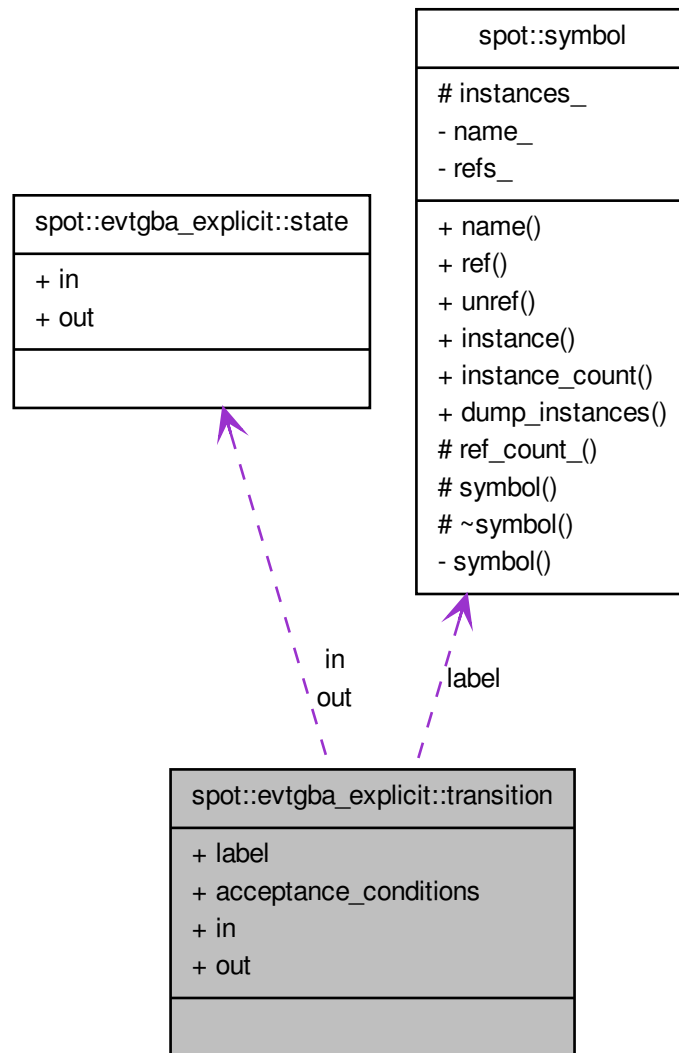
- [tgba/tgbaexplicit.hh](#)

7.193 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`

7.193.1 Detailed Description

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

7.193.2 Member Data Documentation

7.193.2.1 `symbol_set spot::evtgba_explicit::transition::acceptance_conditions`

7.193.2.2 `state* spot::evtgba_explicit::transition::in`

7.193.2.3 `const symbol* spot::evtgba_explicit::transition::label`

7.193.2.4 `state* spot::evtgba_explicit::transition::out`

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

- [evtgba/explicit.hh](#)

7.194 `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`

7.194.1 Detailed Description

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

7.194.2 Member Function Documentation

7.194.2.1 `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](#)

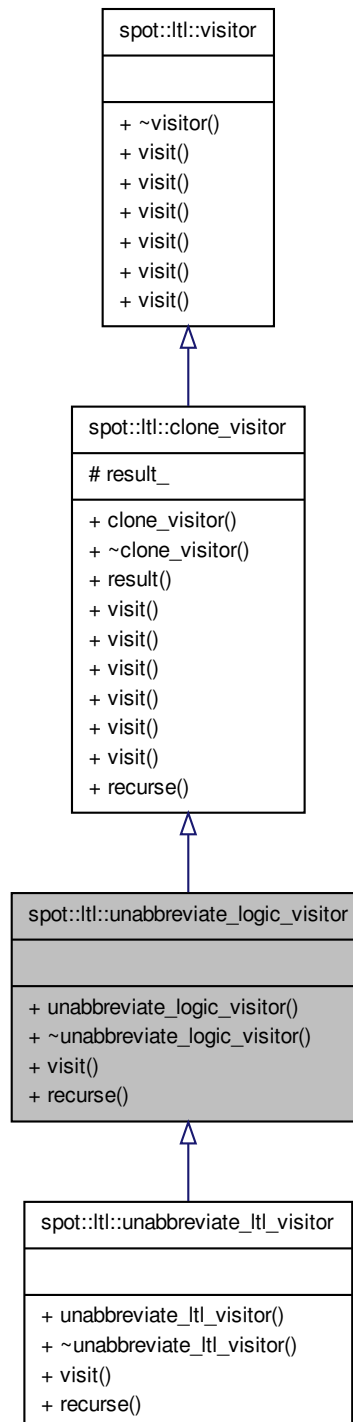
7.195 spot::ltl::unabbreviate_logic_visitor Class Reference

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

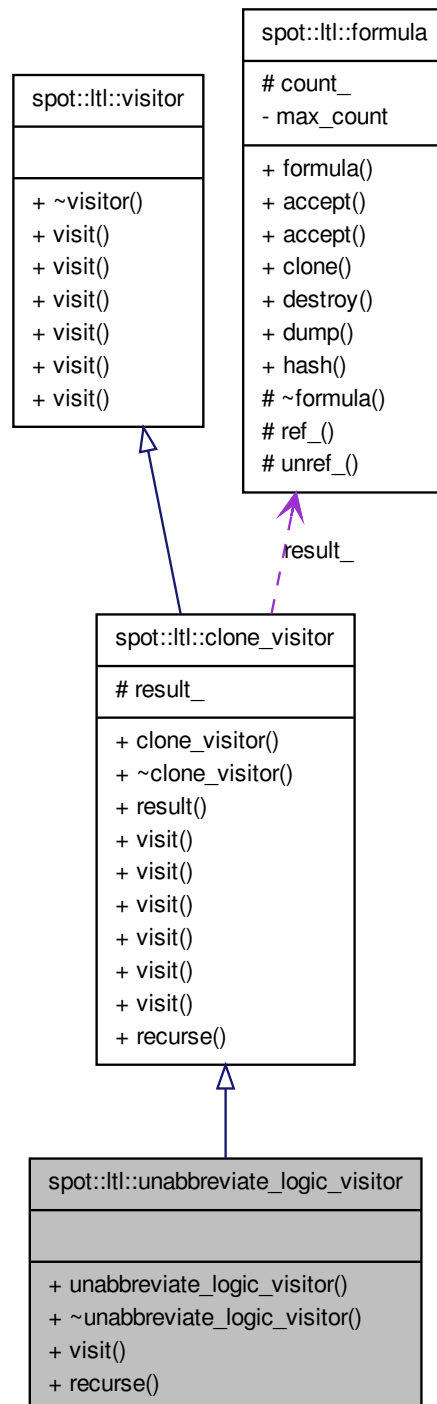
This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#).

```
#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](#)

7.195.1 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.

7.195.2 Member Typedef Documentation

7.195.2.1 typedef clone_visitor spot::ltl::unabbreviate_logic_visitor::super [private]

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

7.195.3 Constructor & Destructor Documentation

7.195.3.1 spot::ltl::unabbreviate_logic_visitor::unabbreviate_logic_visitor ()

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

7.195.4 Member Function Documentation

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

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

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

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

7.195.4.3 void spot::ltl::clone_visitor::visit (multop * *mo*) [virtual, inherited]

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

7.195.4.4 void spot::ltl::clone_visitor::visit (automatop * *mo*) [virtual, inherited]

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

7.195.4.5 void spot::ltl::clone_visitor::visit (unop * *uo*) [virtual, inherited]

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

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

7.195.4.6 void spot::ltl::clone_visitor::visit (atomic_prop * *ap*) [virtual, inherited]

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

7.195.4.7 void spot::ltl::clone_visitor::visit (constant * *c*) [virtual, inherited]

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

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

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

7.195.5 Member Data Documentation

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

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

- `ltlvisit/tunabbrev.hh`

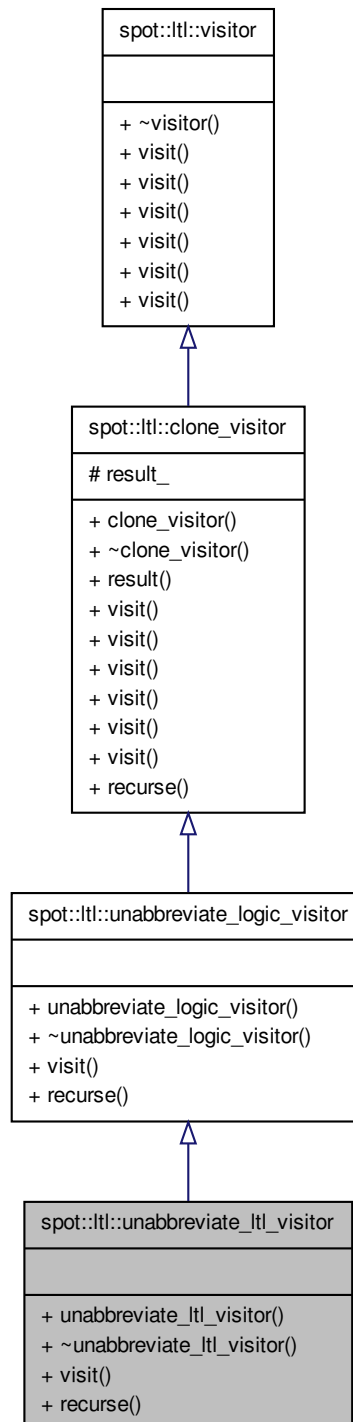
7.196 `spot::ltl::unabbreviate_ltl_visitor` Class Reference

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

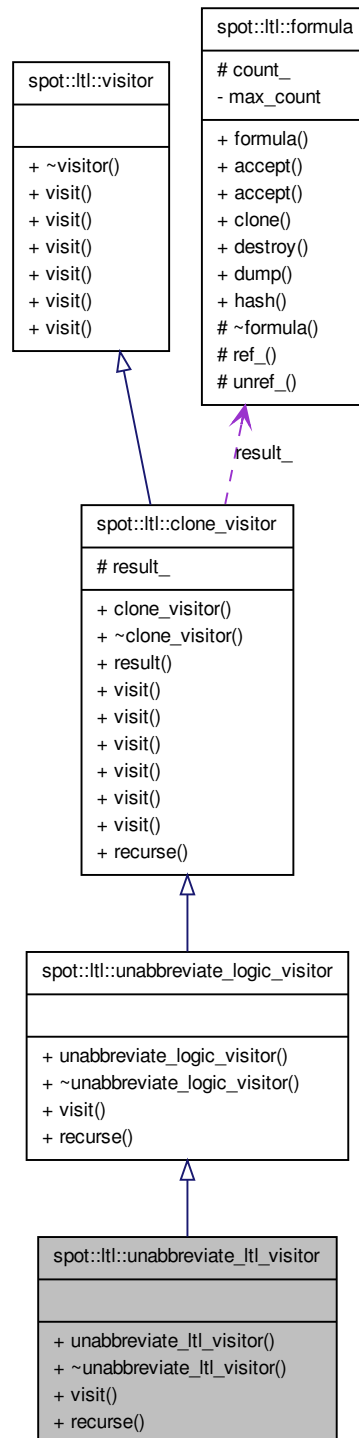
The rewriting performed on logical operator is the same as the one done by `spot::ltl::unabbreviate_logic_visitor`.

```
#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](#)

7.196.1 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.

7.196.2 Member Typedef Documentation

7.196.2.1 typedef [unabbreviate_logic_visitor](#) [spot::ltl::unabbreviate_ltl_visitor::super](#) [**private**]

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

7.196.3 Constructor & Destructor Documentation

7.196.3.1 [spot::ltl::unabbreviate_ltl_visitor::unabbreviate_ltl_visitor](#) ()

7.196.3.2 virtual [spot::ltl::unabbreviate_ltl_visitor::~unabbreviate_ltl_visitor](#) () [**virtual**]

7.196.4 Member Function Documentation

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

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

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

7.196.4.3 void spot::ltl::clone_visitor::visit (constant * *c*) [virtual, inherited]

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

7.196.4.4 void spot::ltl::clone_visitor::visit (multop * *mo*) [virtual, inherited]

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

7.196.4.5 void spot::ltl::clone_visitor::visit (automatop * *mo*) [virtual, inherited]

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

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

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

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

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

7.196.4.8 void spot::ltl::clone_visitor::visit (atomic_prop * *ap*) [virtual, inherited]

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

7.196.5 Member Data Documentation

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

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

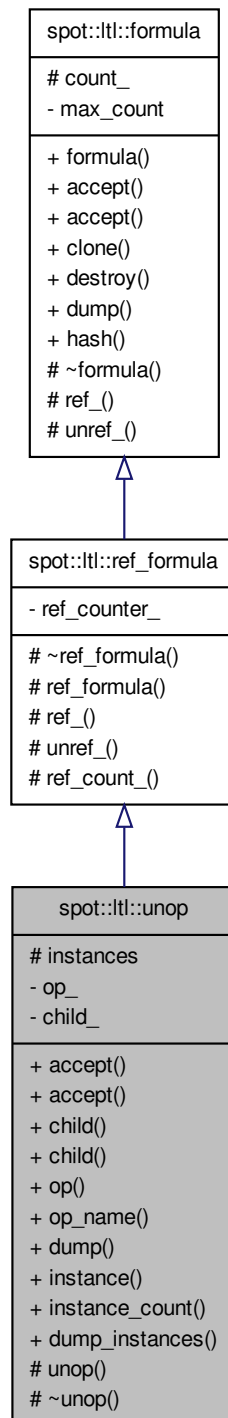
- `ltlvisit/tunabbrev.hh`

7.197 `spot::ltl::unop` Class Reference

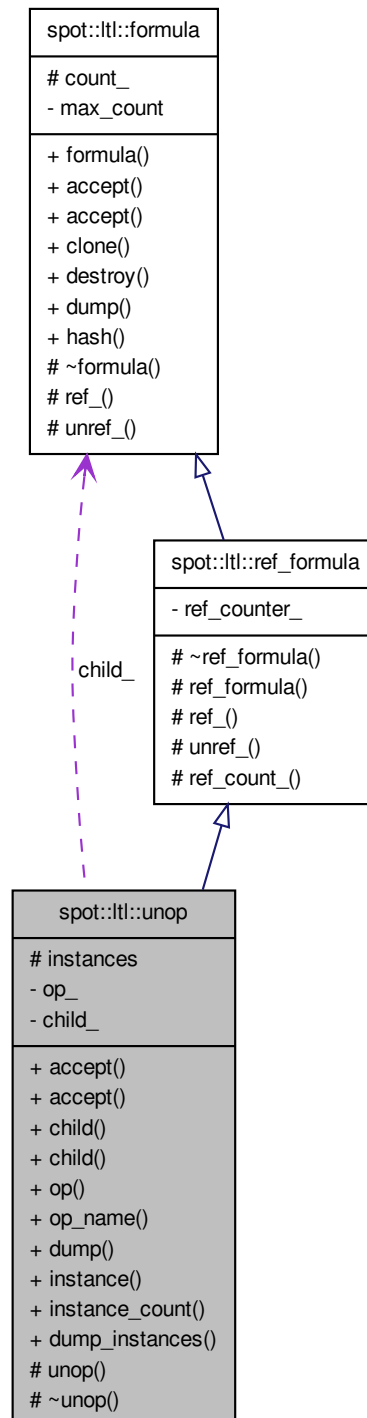
Unary operators.

```
#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_

7.197.1 Detailed Description

Unary operators.

7.197.2 Member Typedef Documentation

7.197.2.1 typedef std::map<pair, unop*> spot::lfl::unop::map [protected]

7.197.2.2 typedef std::pair<type, formula*> spot::lfl::unop::pair [protected]

7.197.3 Member Enumeration Documentation

7.197.3.1 enum spot::ltl::unop::type

Enumerator:

Not
X
F
G
Finish

7.197.4 Constructor & Destructor Documentation

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

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

7.197.5 Member Function Documentation

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

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

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

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

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

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

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

Get the sole operand of this operator.

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

Get the sole operand of this operator.

7.197.5.5 formula* spot::ltl::formula::clone () const [inherited]

clone this node

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

7.197.5.6 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()`.

7.197.5.7 virtual std::string spot::ltl::unop::dump () const [virtual]

Return a canonic representation of the atomic proposition.

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

7.197.5.8 static std::ostream& spot::ltl::unop::dump_instances (std::ostream & os) [static]

Dump all instances. For debugging.

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

Return a hash key for the formula.

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

Referenced by `spot::ltl::formula_ptr_hash::operator()()`, and `spot::ltl::formula_ptr_less_than::operator()()`.

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

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

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

Number of instantiated unary operators. For debugging.

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

Get the type of this operator.

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

Get the type of this operator, as a string.

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

increment reference counter if any

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

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

Number of references to this formula.

7.197.5.16 `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](#).

7.197.6 Member Data Documentation**7.197.6.1** `formula* spot::ltl::unop::child_ [private]`**7.197.6.2** `size_t spot::ltl::formula::count_ [protected, inherited]`

The hash key of this formula.

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

7.197.6.3 `map spot::ltl::unop::instances [static, protected]`

7.197.6.4 type spot::ltl::unop::op_ [private]

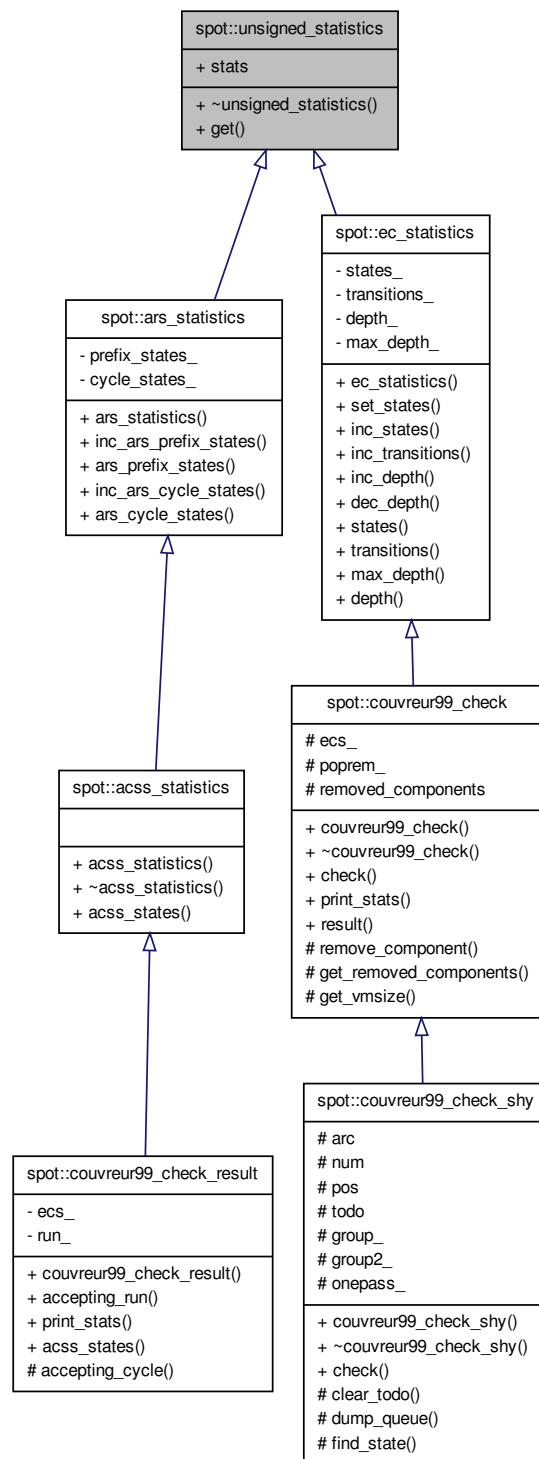
The documentation for this class was generated from the following file:

- [ltlast/unop.hh](#)

7.198 spot::unsigned_statistics Struct Reference

```
#include <tgbaaalgos/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

7.198.1 Member Typedef Documentation

7.198.1.1 typedef std::map<const char*, unsigned_fun, char_ptr_less_than>
spot::unsigned_statistics::stats_map

7.198.1.2 typedef unsigned(unsigned_statistics::* spot::unsigned_statistics::unsigned_fun)() const

7.198.2 Constructor & Destructor Documentation

7.198.2.1 virtual spot::unsigned_statistics::~~unsigned_statistics () [[inline](#), [virtual](#)]

7.198.3 Member Function Documentation

7.198.3.1 unsigned spot::unsigned_statistics::get (const char * *str*) const [[inline](#)]

References stats.

7.198.4 Member Data Documentation

7.198.4.1 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](#)

7.199 `spot::unsigned_statistics_copy` Class Reference

comparable statistics

```
#include <tgbaaalgos/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`

7.199.1 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.)

7.199.2 Member Typedef Documentation

7.199.2.1 `typedef std::map<const char*, unsigned, char_ptr_less_than>
spot::unsigned_statistics_copy::stats_map`

7.199.3 Constructor & Destructor Documentation

7.199.3.1 `spot::unsigned_statistics_copy::unsigned_statistics_copy () [inline]`

7.199.3.2 `spot::unsigned_statistics_copy::unsigned_statistics_copy (const unsigned_statistics & o
) [inline]`

References [seteq\(\)](#).

7.199.4 Member Function Documentation

7.199.4.1 `bool spot::unsigned_statistics_copy::operator!=(const unsigned_statistics_copy & o)
const [inline]`

7.199.4.2 `bool spot::unsigned_statistics_copy::operator==(const unsigned_statistics_copy & o)
const [inline]`

References stats.

7.199.4.3 `bool spot::unsigned_statistics_copy::seteq (const unsigned_statistics & o)
[inline]`

References stats, and spot::unsigned_statistics::stats.

Referenced by unsigned_statistics_copy().

7.199.5 Member Data Documentation

7.199.5.1 `bool spot::unsigned_statistics_copy::set`

7.199.5.2 `stats_map spot::unsigned_statistics_copy::stats`

Referenced by operator==(), and seteq().

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

- [tgbaalgorithms/emptiness_stats.hh](#)

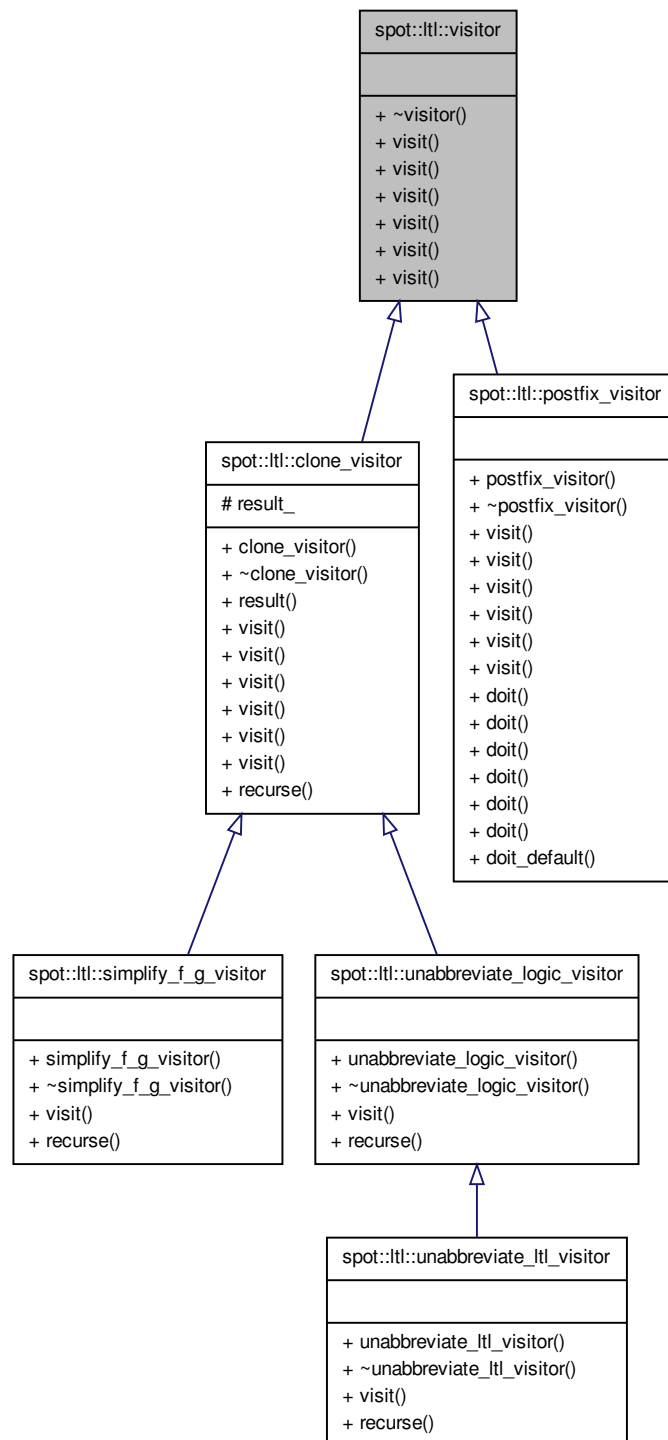
7.200 spot::ltl::visitor Struct Reference

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

7.200.1 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.

7.200.2 Constructor & Destructor Documentation

7.200.2.1 virtual [spot::ltl::visitor::~~visitor](#) () [[inline](#), [virtual](#)]

7.200.3 Member Function Documentation

7.200.3.1 virtual void [spot::ltl::visitor::visit](#) ([atomic_prop](#) * *node*) [[pure](#) [virtual](#)]

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

7.200.3.2 virtual void [spot::ltl::visitor::visit](#) ([automatop](#) * *node*) [[pure](#) [virtual](#)]

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

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

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

7.200.3.4 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](#).

7.200.3.5 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](#).

7.200.3.6 virtual void spot::ltl::visitor::visit (constant * 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](#)

7.201 spot::weight Class Reference

Manage for a given automaton a vector of counter indexed by its acceptance condition.

```
#include <tgbalgorithms/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)

7.201.1 Detailed Description

Manage for a given automaton a vector of counter indexed by its acceptance condition.

7.201.2 Member Typedef Documentation

7.201.2.1 `typedef std::map<int, int> spot::weight::weight_vector [private]`

7.201.3 Constructor & Destructor Documentation

7.201.3.1 `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\(\)](#)).

7.201.4 Member Function Documentation

7.201.4.1 `static void spot::weight::dec_weight_handler (char * varset, int size) [static, private]`

7.201.4.2 `static void spot::weight::inc_weight_handler (char * varset, int size) [static, private]`

7.201.4.3 `weight& spot::weight::operator+= (const bdd & acc)`

Increment by one the counters of each acceptance condition in *acc*.

7.201.4.4 `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*.

7.201.4.5 `weight& spot::weight::operator-= (const bdd & acc)`

Decrement by one the counters of each acceptance condition in *acc*.

7.201.5 Friends And Related Function Documentation

7.201.5.1 `std::ostream& operator<< (std::ostream & os, const weight & w) [friend]`

7.201.6 Member Data Documentation

7.201.6.1 `weight_vector spot::weight::m [private]`

7.201.6.2 `bdd spot::weight::neg_all_acc [private]`

7.201.6.3 `weight_vector* spot::weight::pm [static, private]`

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

- [tgbaalgos/weight.hh](#)

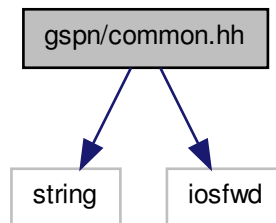
8 File Documentation

8.1 `mainpage.dox` File Reference

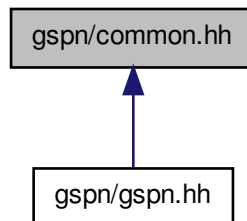
8.2 `gspn/common.hh` File Reference

```
#include <string>
#include <iosfwd>
```

Include dependency graph for common.hh:



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



Classes

- class [spot::gspn_exception](#)
An exception used to forward GSPN errors.

Namespaces

- namespace [spot](#)

Functions

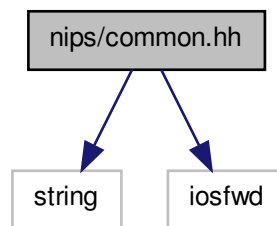
- `std::ostream & spot::operator<< (std::ostream &os, const gspn_exception &e)`

8.3 nips/common.hh File Reference

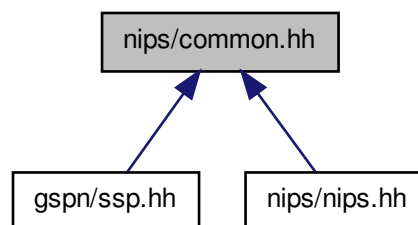
```
#include <string>
```

```
#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](#)

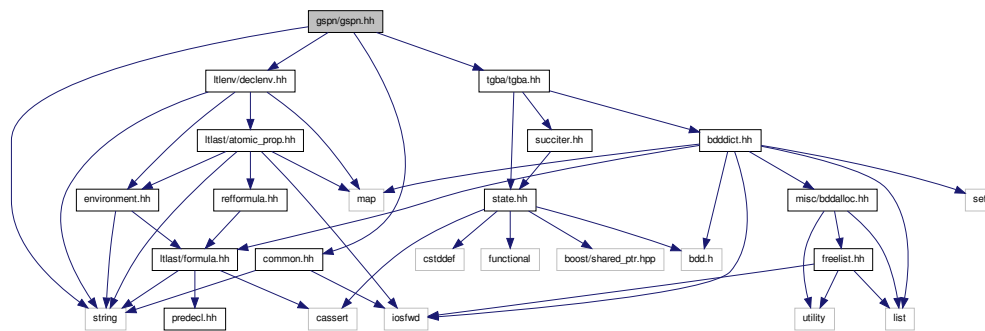
Functions

- `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 "ltlenv/declenv.hh"
```

Include dependency graph for gspn.hh:



Classes

- class [spot::gspn_interface](#)

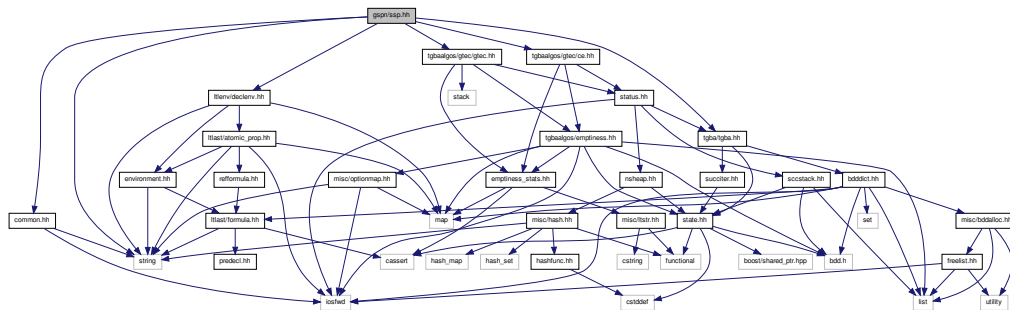
Namespaces

- namespace [spot](#)

8.5 gspn/spp.hh File Reference

```
#include <string>
#include "tgba/tgba.hh"
#include "common.hh"
#include "tgbaalgos/gtec/gtec.hh"
#include "tgbaalgos/gtec/ce.hh"
#include "ltlenv/declenv.hh"
```

Include dependency graph for ssp.hh:



Classes

- class `spot::gspn_ssp_interface`

Namespaces

- namespace **spot**

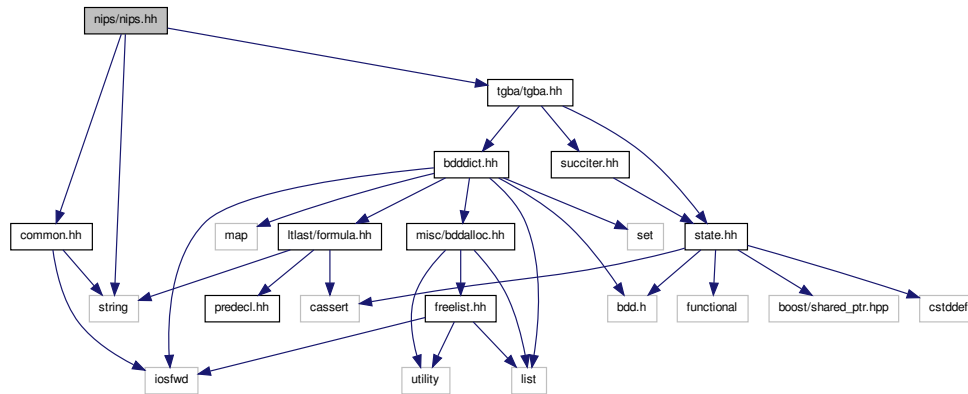
Functions

- `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 dependency graph for nips.hh:



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

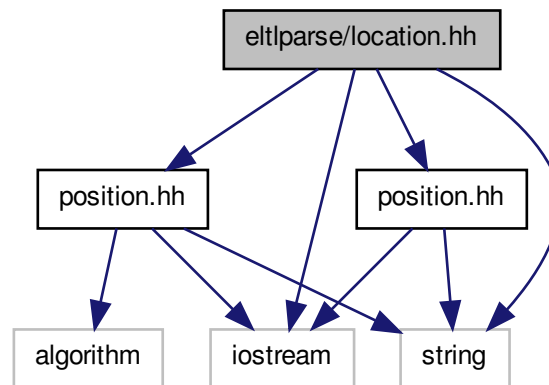
8.7 eltlparse/location.hh File Reference

```
#include <iostream>
#include <string>
```

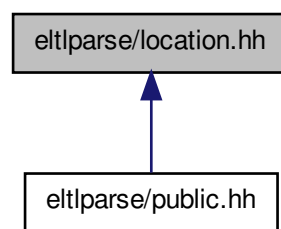
```
#include "position.hh"
```

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

Include dependency graph for location.hh:



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



Classes

- class `eltly::location`
Abstract a location.

Namespaces

- namespace `eltly`

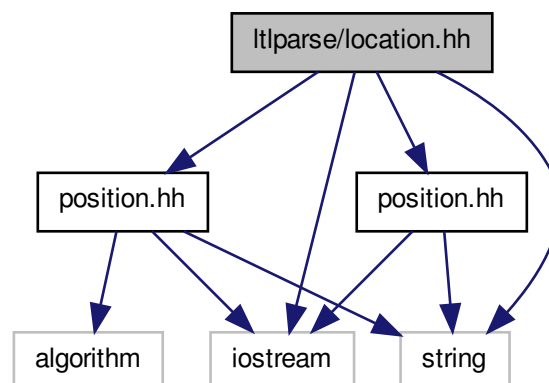
Functions

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

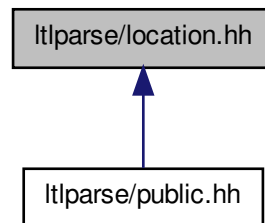
8.8 Itlparse/location.hh File Reference

```
#include <iostream>
#include <string>
#include "position.hh"
#include "position.hh"
```

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](#)

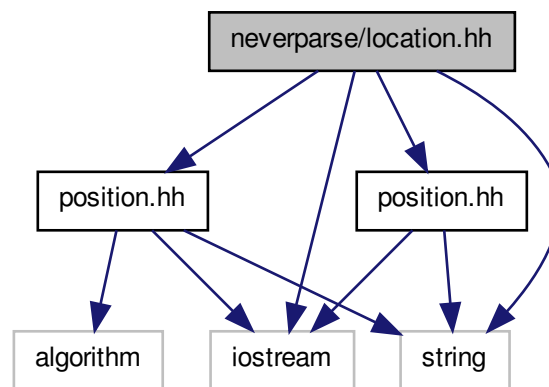
Functions

- 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 [ltlty::operator!=](#) (const location &loc1, const location &loc2)
Compare two location objects.
- std::ostream & [ltlty::operator<<](#) (std::ostream &ostr, const location &loc)
Intercept output stream redirection.

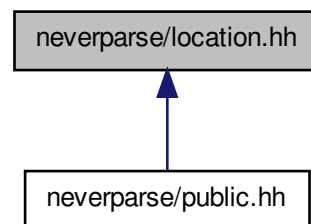
8.9 neverparse/location.hh File Reference

```
#include <iostream>
#include <string>
#include "position.hh"
#include "position.hh"
```

Include dependency graph for location.hh:



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



Classes

- class `neverclaimyy::location`
Abstract a location.

Namespaces

- namespace [neverclaimyy](#)

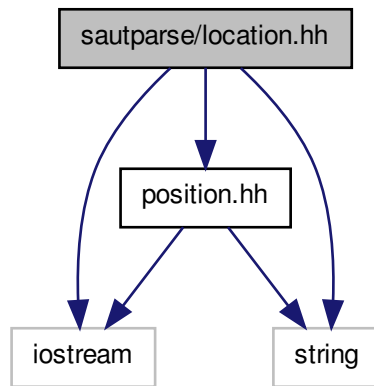
Functions

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

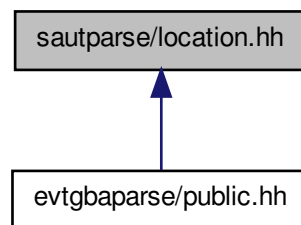
8.10 sautparse/location.hh File Reference

```
#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`

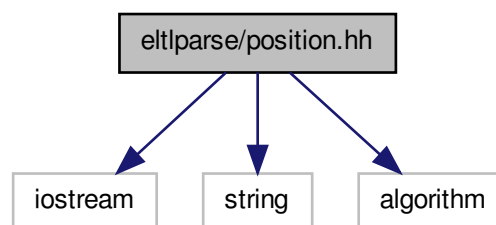
Functions

- 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 & [sauty::operator+=](#) (location &res, unsigned int width)
Add and assign a location.
- std::ostream & [sauty::operator<<](#) (std::ostream &ostr, const location &loc)
Intercept output stream redirection.

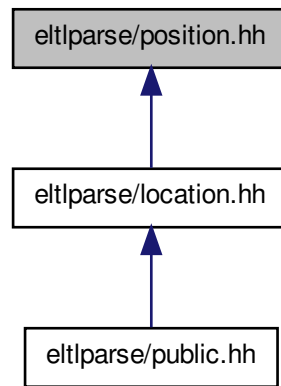
8.11 eltlparse/position.hh File Reference

```
#include <iostream>
#include <string>
#include <algorithm>
```

Include dependency graph for position.hh:



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



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

Compare two position objects.

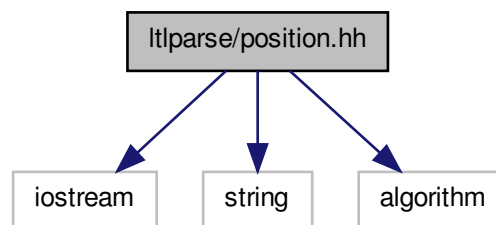
- `std::ostream & eltly::operator<< (std::ostream &ostr, const position &pos)`

Intercept output stream redirection.

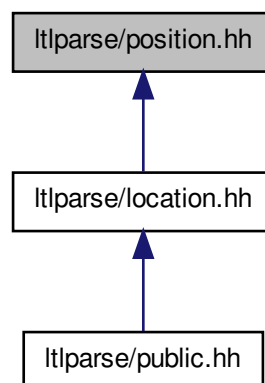
8.12 Itlparse/position.hh File Reference

```
#include <iostream>
#include <string>
#include <algorithm>
```

Include dependency graph for position.hh:



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



Classes

- class `ltlty::position`
Abstract a position.

Namespaces

- namespace `ltlty`

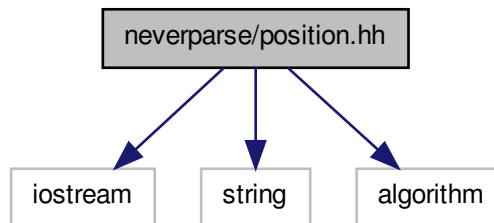
Functions

- `const position & ltlty::operator+=` (`position &res`, `const int width`)
Add and assign a position.
- `const position ltlty::operator+` (`const position &begin`, `const int width`)
Add two position objects.
- `const position & ltlty::operator-=` (`position &res`, `const int width`)
Add and assign a position.
- `const position ltlty::operator-` (`const position &begin`, `const int width`)
Add two position objects.
- `bool ltlty::operator==` (`const position &pos1`, `const position &pos2`)
Compare two position objects.
- `bool ltlty::operator!=` (`const position &pos1`, `const position &pos2`)
Compare two position objects.
- `std::ostream & ltlty::operator<<` (`std::ostream &ostr`, `const position &pos`)
Intercept output stream redirection.

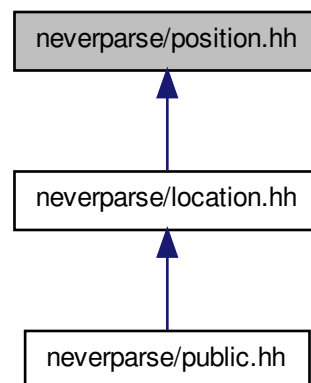
8.13 neverparse/position.hh File Reference

```
#include <iostream>
#include <string>
#include <algorithm>
```


Include dependency graph for position.hh:



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



Classes

- class `neverclaimyy::position`
Abstract a position.

Namespaces

- namespace `neverclaimyy`

Functions

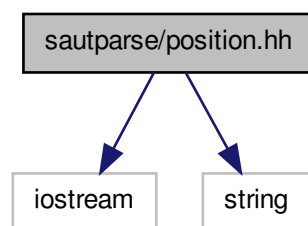
- `const position & neverclaimyy::operator+= (position &res, const int width)`
Add and assign a position.
- `const position neverclaimyy::operator+ (const position &begin, const int width)`
Add two position objects.
- `const position & neverclaimyy::operator-= (position &res, const int width)`
Add and assign a position.
- `const position neverclaimyy::operator- (const position &begin, const int width)`
Add two position objects.
- `bool neverclaimyy::operator== (const position &pos1, const position &pos2)`
Compare two position objects.
- `bool neverclaimyy::operator!= (const position &pos1, const position &pos2)`
Compare two position objects.
- `std::ostream & neverclaimyy::operator<< (std::ostream &ostr, const position &pos)`
Intercept output stream redirection.

8.14 sautparse/position.hh File Reference

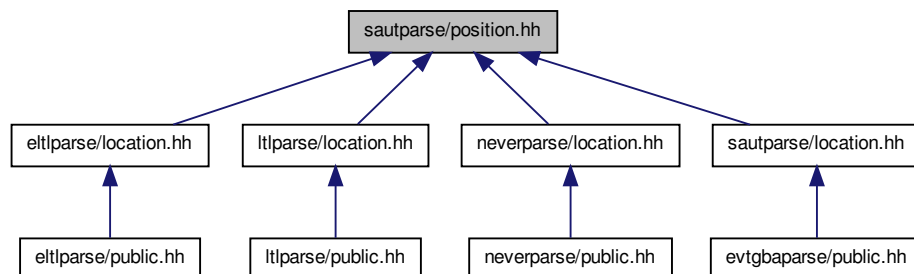
```
#include <iostream>
```

```
#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

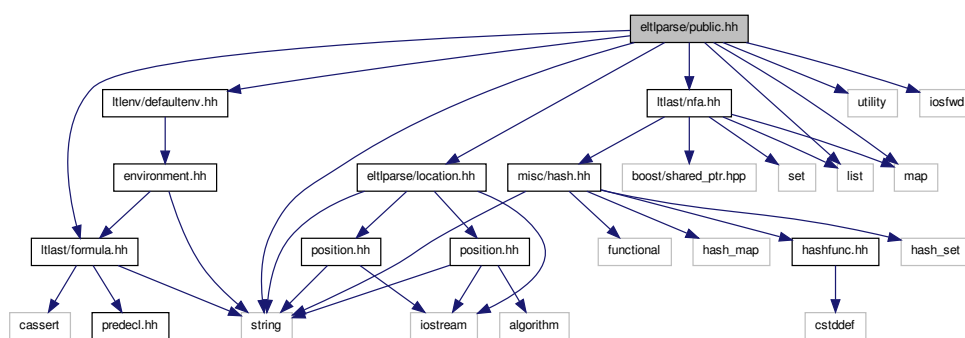
- `const position & sauty::operator+= (position &res, const int width)`
Add and assign a position.
- `const position sauty::operator+ (const position &begin, const int width)`
Add two position objects.
- `const position & sauty::operator-= (position &res, const int width)`
Add and assign a position.
- `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.

8.15 eltlparse/public.hh File Reference

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

```
#include "ltlast/nfa.hh"
#include "eltlparse/location.hh"
#include <string>
#include <list>
#include <map>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



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 ECTL string.

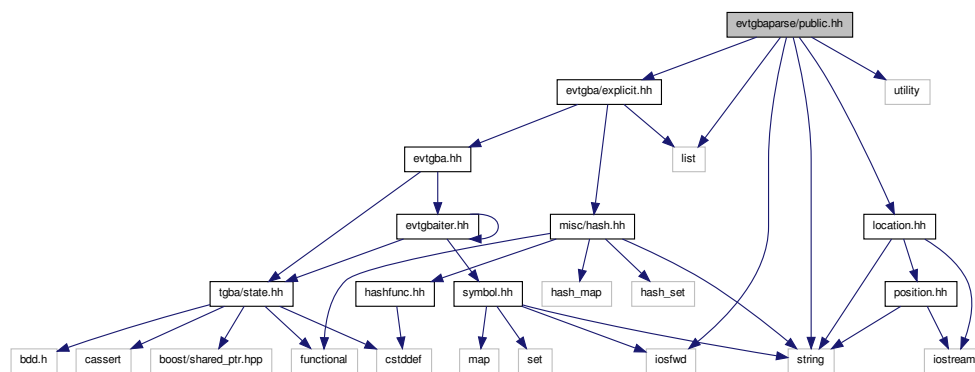
- bool `spot::eltl::format_parse_errors` (std::ostream &os, parse_error_list &error_list)

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

8.16 evtgbaparse/public.hh File Reference

```
#include "evtgba/explicit.hh"
#include "location.hh"
#include <string>
#include <list>
#include <utility>
#include <iosfwd>
```

Include dependency graph for public.hh:



Namespaces

- namespace `spot`

Typedefs

- typedef std::pair< evtgbayy::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.

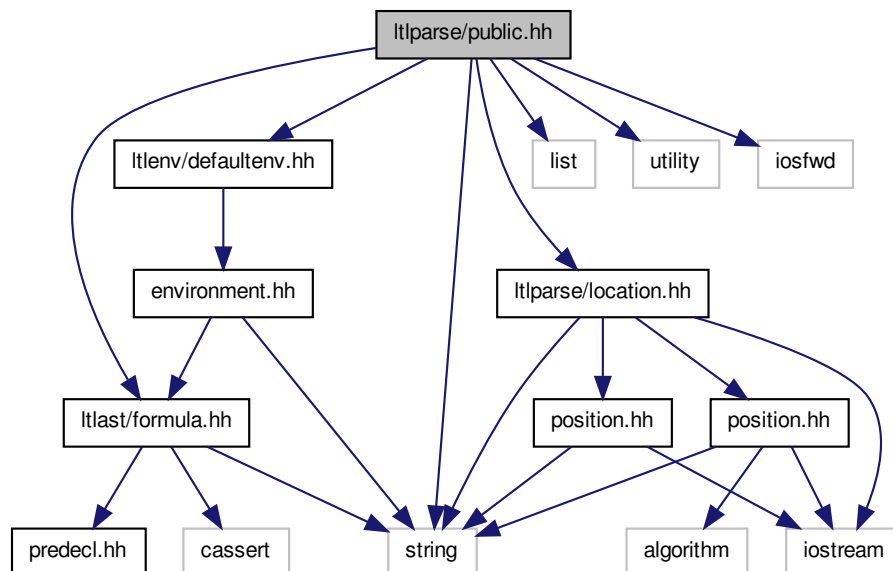
- `bool spot::format_evtgba_parse_errors` (std::ostream &os, const std::string &filename, evtgba_parse_error_list &error_list)

Format diagnostics produced by `spot::evtgba_parse`.

8.17 Itlparse/public.hh File Reference

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

Include dependency graph for public.hh:



Namespaces

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

Typedefs

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

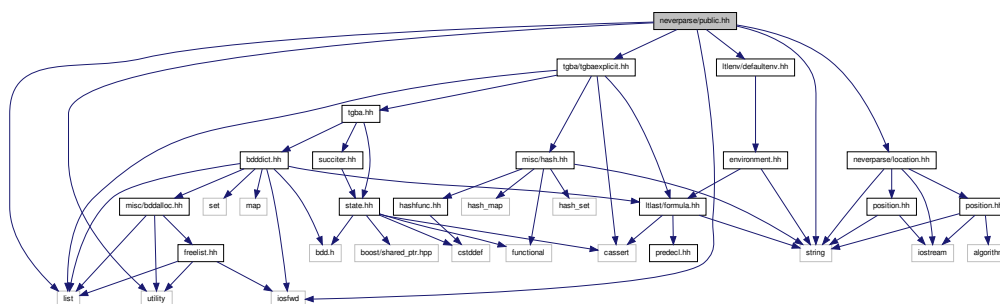
Functions

- formula * [spot::ltl::parse](#) (const std::string <l_string, parse_error_list &error_list, environment &env=default_environment::instance(), bool debug=false)
Build a formula from an LTL string.
- bool [spot::ltl::format_parse_errors](#) (std::ostream &os, const std::string <l_string, parse_error_list &error_list)
Format diagnostics produced by spot::ltl::parse.

8.18 neverparse/public.hh File Reference

```
#include "tgba/tgbaexplicit.hh"
#include "neverparse/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< [neverclaimyy::location](#), std::string > [spot::neverclaim_parse_error](#)
A parse diagnostic with its location.
- typedef std::list< neverclaim_parse_error > [spot::neverclaim_parse_error_list](#)
A list of parser diagnostics, as filled by parse.

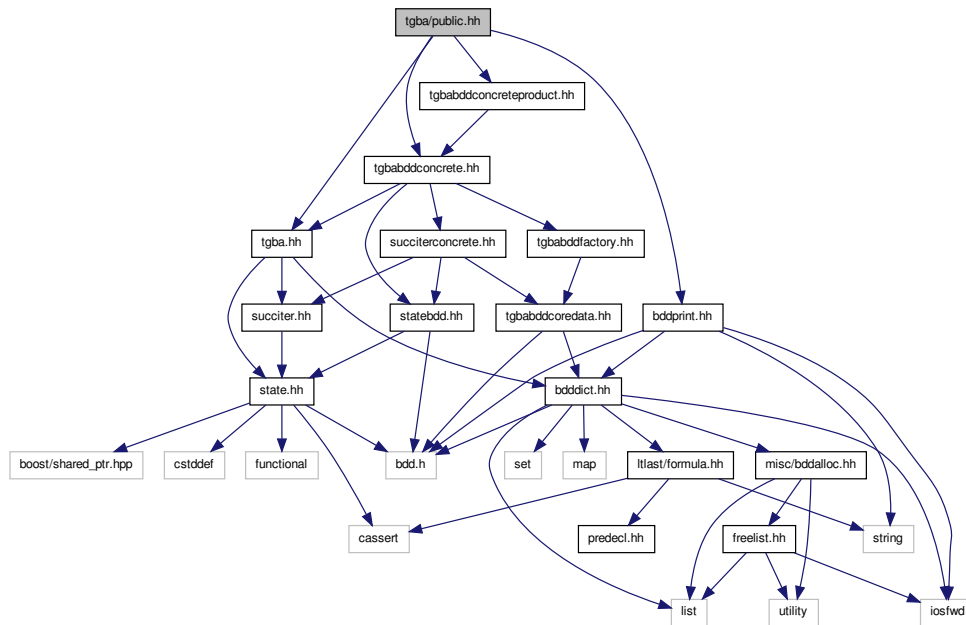
Functions

- tgba_explicit_string * [spot::neverclaim_parse](#) (const std::string &filename, neverclaim_parse_error_list &error_list, bdd_dict *dict, ltl::environment &env=ltl::default_environment::instance(), bool debug=false)
Build a [spot::tgba_explicit](#) from a Spin never claim file.
- bool [spot::format_neverclaim_parse_errors](#) (std::ostream &os, const std::string &filename, neverclaim_parse_error_list &error_list)
Format diagnostics produced by [spot::neverclaim_parse](#).

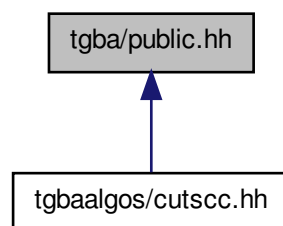
8.19 tgba/public.hh File Reference

```
#include "tgba.hh"
#include "tgbabddconcrete.hh"
#include "tgbabddconcretetproduct.hh"
#include "bddprint.hh"
```


Include dependency graph for public.hh:



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



8.20 tgbaparse/public.hh File Reference

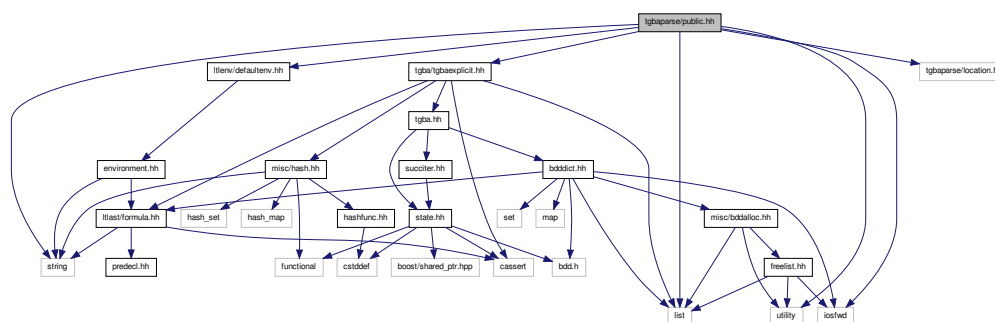
```

#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< tgba::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.

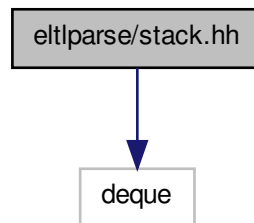
Functions

- 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](#).

8.21 eltlparse/stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



Classes

- class [eltlpy::stack< T, S >](#)
- class [eltlpy::slice< T, S >](#)
Present a slice of the top of a stack.

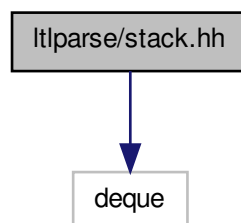
Namespaces

- namespace [eltlpy](#)

8.22 Itlparse/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.

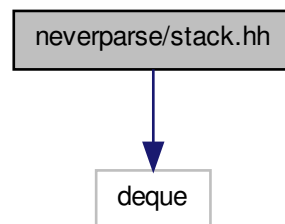
Namespaces

- namespace [ltlyy](#)

8.23 neverparse/stack.hh File Reference

```
#include <deque>
```

Include dependency graph for stack.hh:



Classes

- class [neverclaimyy::stack< T, S >](#)
- class [neverclaimyy::slice< T, S >](#)

Present a slice of the top of a stack.

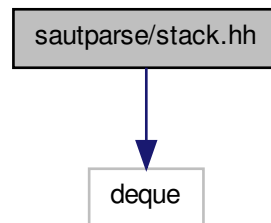
Namespaces

- namespace [neverclaimyy](#)

8.24 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

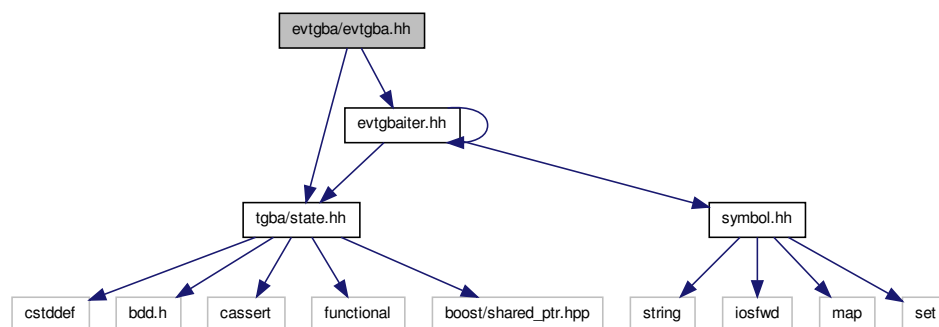
- namespace [sauty](#)

8.25 evtgba/evtgba.hh File Reference

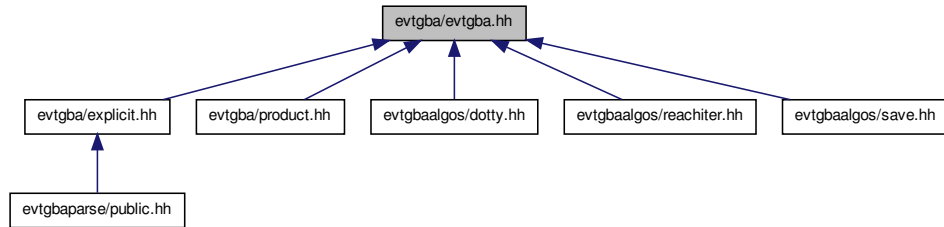
```
#include "tgba/state.hh"
```

```
#include "evtgbaiter.hh"
```

Include dependency graph for evtgba.hh:



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



Classes

- class `spot::evtgba`

Namespaces

- namespace `spot`

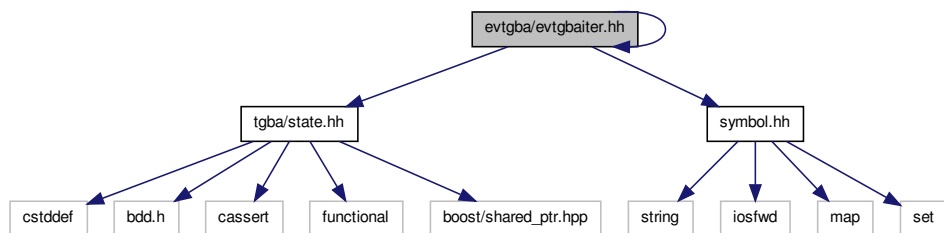
8.26 evtgba/evtgbaiter.hh File Reference

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

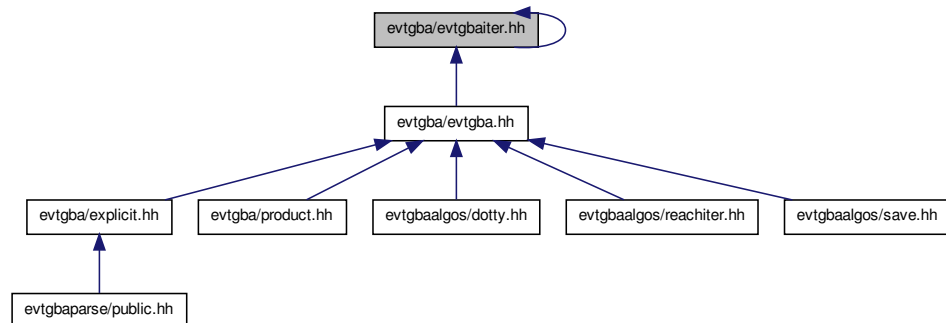
```
#include "symbol.hh"
```

```
#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`

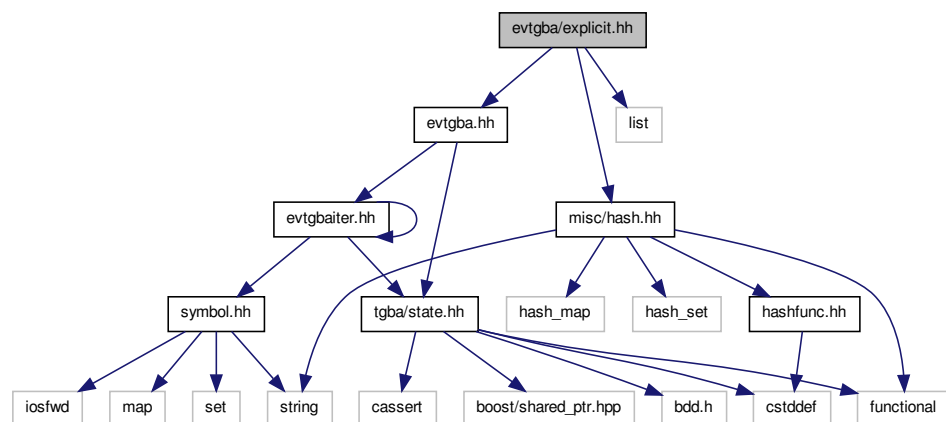
Namespaces

- namespace `spot`

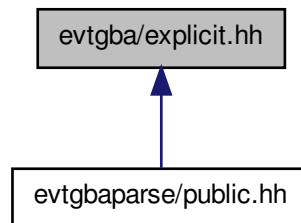
8.27 evtgba/explicit.hh File Reference

```
#include "evtgba.hh"
#include <list>
#include "misc/hash.hh"
```

Include dependency graph for `explicit.hh`:



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



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](#).

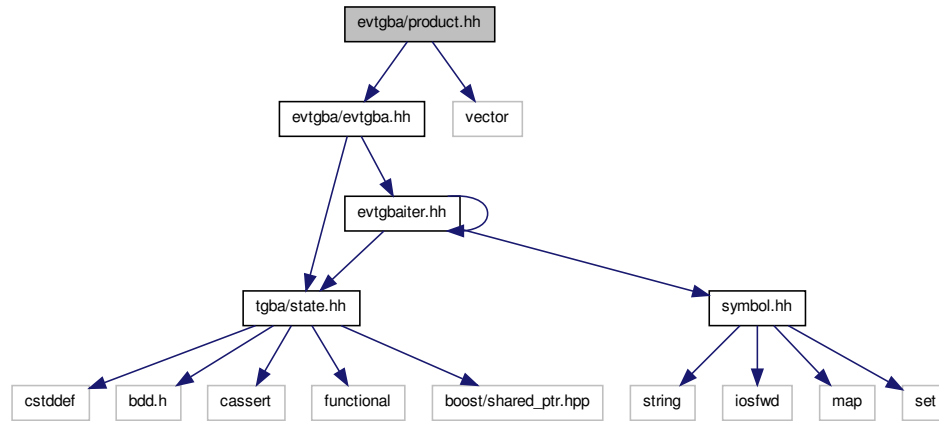
Namespaces

- namespace [spot](#)

8.28 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`

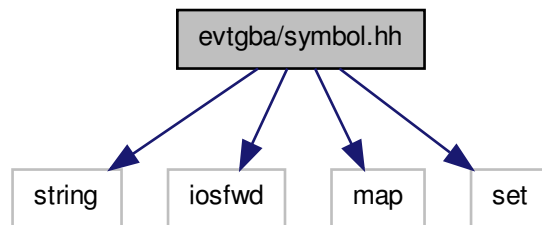
8.29 evtgba/symbol.hh File Reference

```

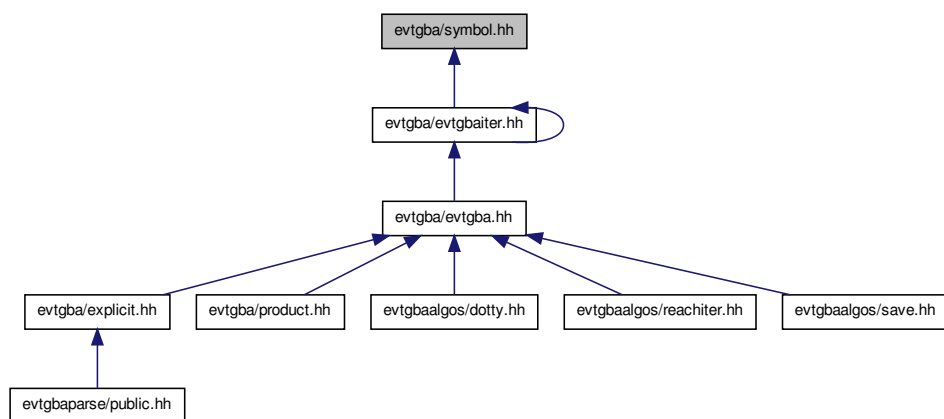
#include <string>
#include <iosfwd>
#include <map>
#include <set>

```

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

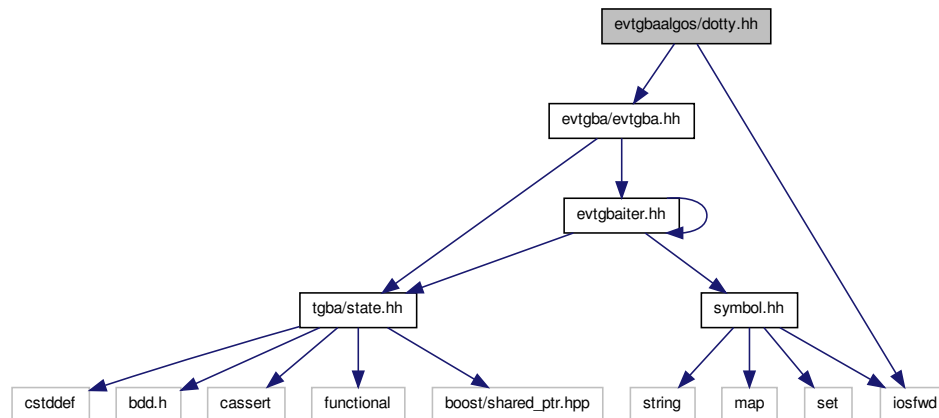
- typedef `std::set< const symbol * >` [spot::symbol_set](#)
- typedef `std::set< rsymbol >` [spot::rsymbol_set](#)

8.30 evtgbaalgos/dotty.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:



Namespaces

- namespace [spot](#)

Functions

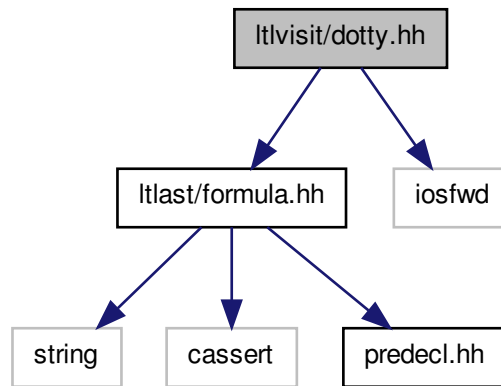
- `std::ostream & spot::dotty_reachable (std::ostream &os, const evtgba *g)`
Print reachable states in dot format.

8.31 Itlvisit/dotty.hh File Reference

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

```
#include <iosfwd>
```

Include dependency graph for dotty.hh:



Namespaces

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

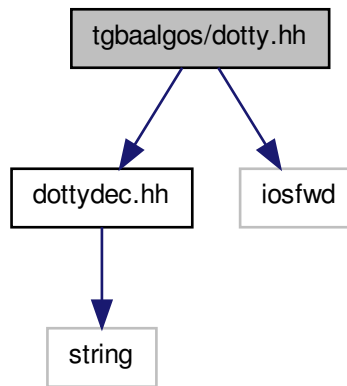
Functions

- `std::ostream & spot::ltl::dotty (std::ostream &os, const formula *f)`
Write a formula tree using dot's syntax.

8.32 tgbaalgos/dotty.hh File Reference

```
#include "dottydec.hh"
#include <iosfwd>
```

Include dependency graph for `dotty.hh`:



Namespaces

- namespace `spot`

Functions

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

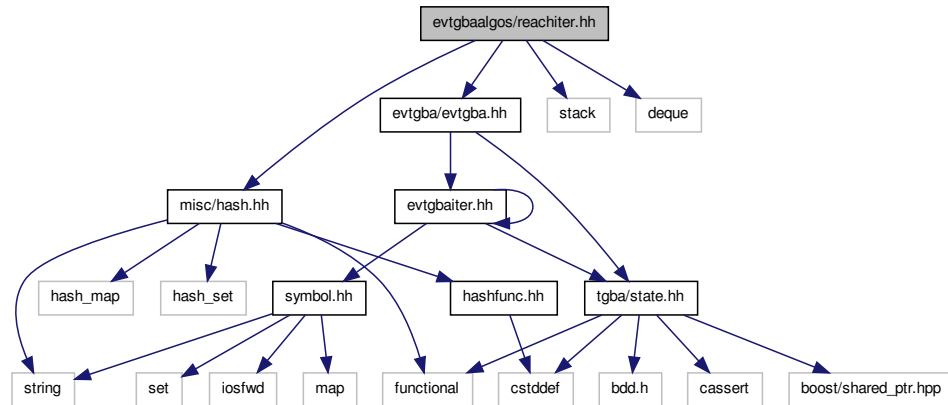
Print reachable states in dot format.

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

8.33 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](#)

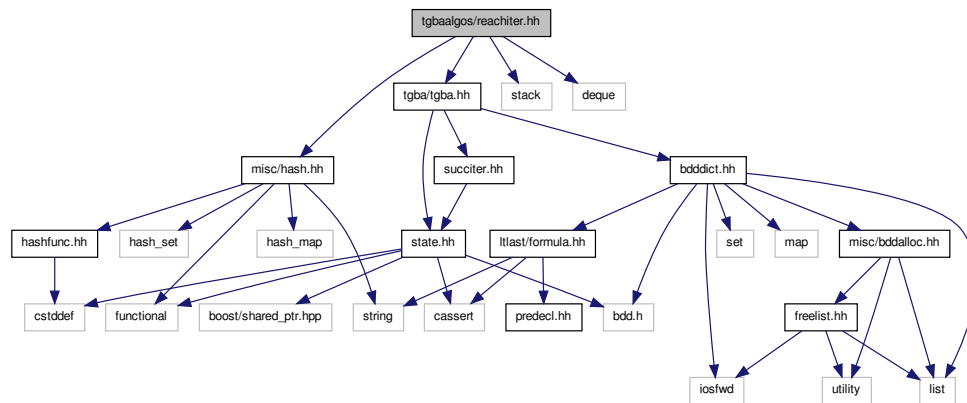
8.34 tgbaalgorithms/reachiter.hh File Reference

```

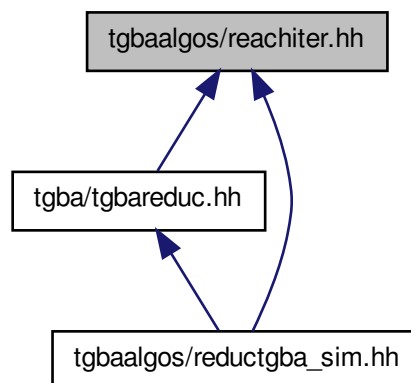
#include "misc/hash.hh"
#include "tgba/tgba.hh"
#include <stack>
#include <deque>

```

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

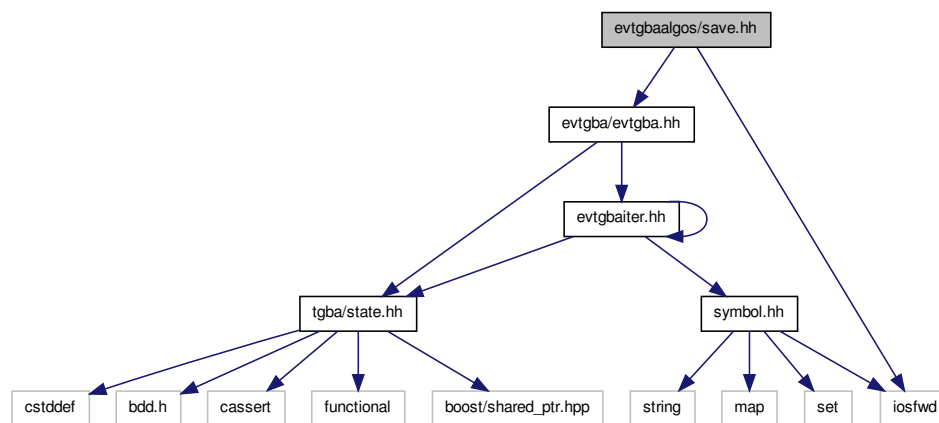
- namespace [spot](#)

8.35 evtgbaalgos/save.hh File Reference

```
#include "evtgba/evtgba.hh"
```

```
#include <iosfwd>
```

Include dependency graph for save.hh:



Namespaces

- namespace [spot](#)

Functions

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

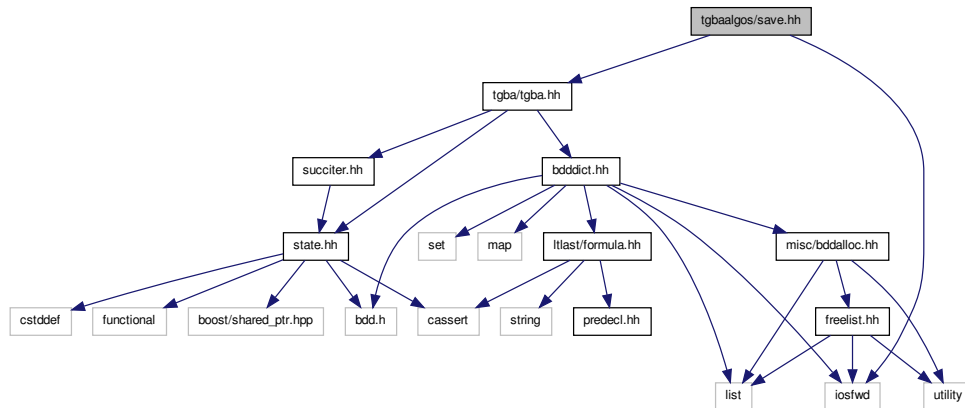
Save reachable states in text format.

8.36 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.

8.37 evtgbaalgos/tgba2evtgba.hh File Reference

Namespaces

- namespace [spot](#)

Functions

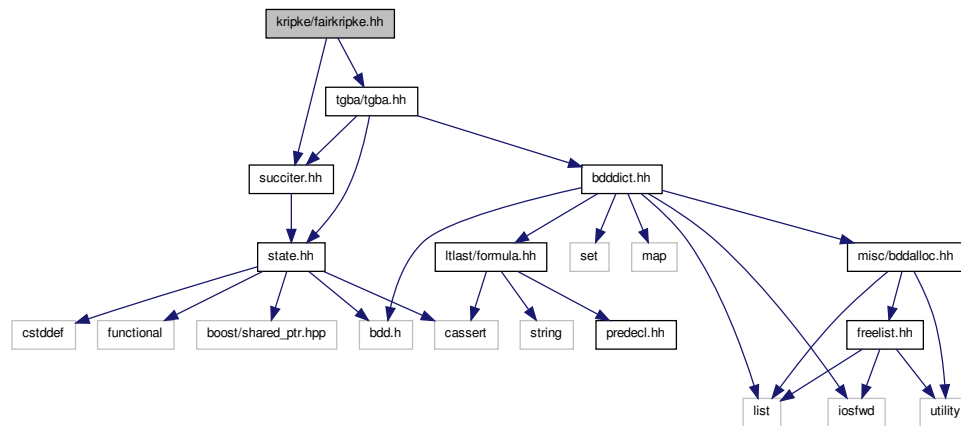
- `evtgba_explicit * spot::tgba_to_evtgba (const tgba *a)`
Convert a tgba into an evtgba.

8.38 kripke/fairkripke.hh File Reference

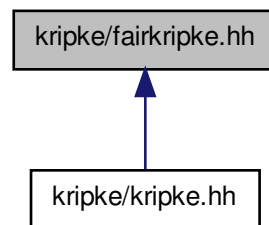
```
#include "tgba/tgba.hh"
```

```
#include "tgba/succiter.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](#)

Iterator code for a Fair Kripke structure.

This iterator can be used to simplify the writing of an iterator on a Fair Kripke structure (or lookalike).

- class [spot::fair_kripke](#)

Interface for a Fair Kripke structure.

A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition, and a set of acceptance conditions.

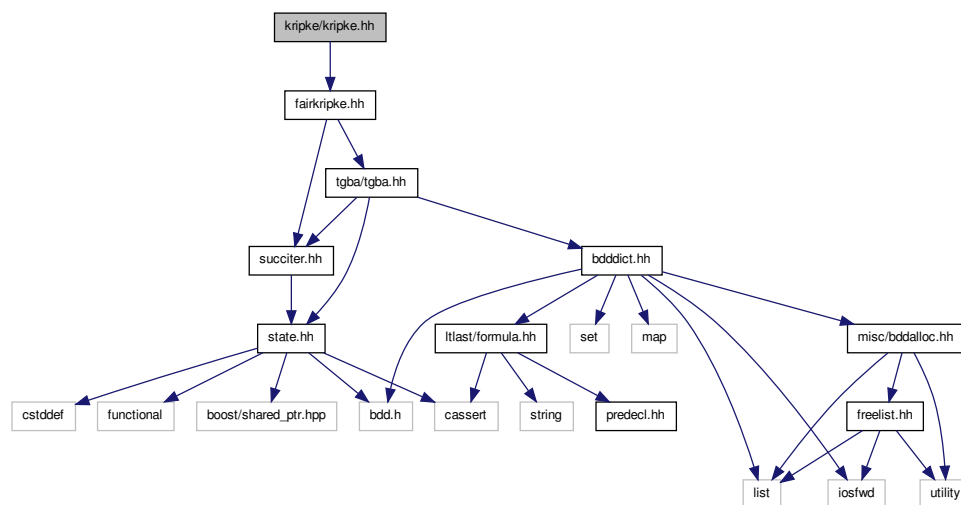
Namespaces

- namespace [spot](#)

8.39 kripke/kripke.hh File Reference

```
#include "fairkripke.hh"
```

Include dependency graph for kripke.hh:



Classes

- class [spot::kripke_succ_iterator](#)
Iterator code for Kripke structure
This iterator can be used to simplify the writing of an iterator on a Kripke structure (or lookalike).
- class [spot::kripke](#)
Interface for a Kripke structure
A Kripke structure is a graph in which each node (=state) is labeled by a conjunction of atomic proposition.

Namespaces

- namespace [spot](#)

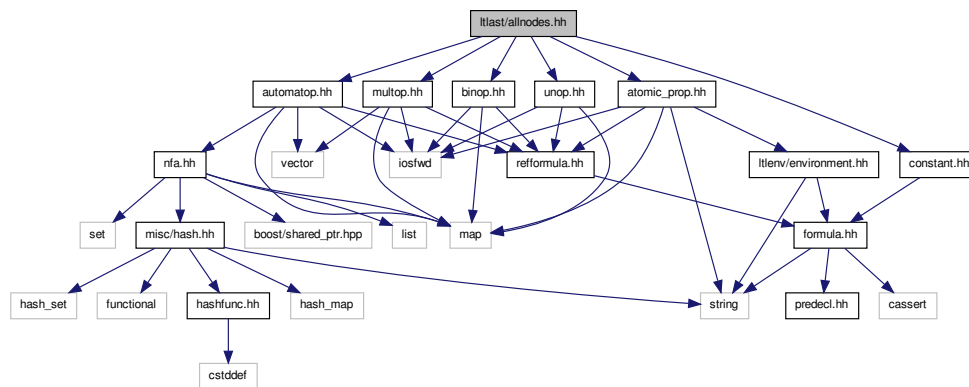
8.40 ltlast/allnodes.hh File Reference

Define all LTL node types.

```
#include "binop.hh"
```

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

Include dependency graph for allnodes.hh:



8.40.1 Detailed Description

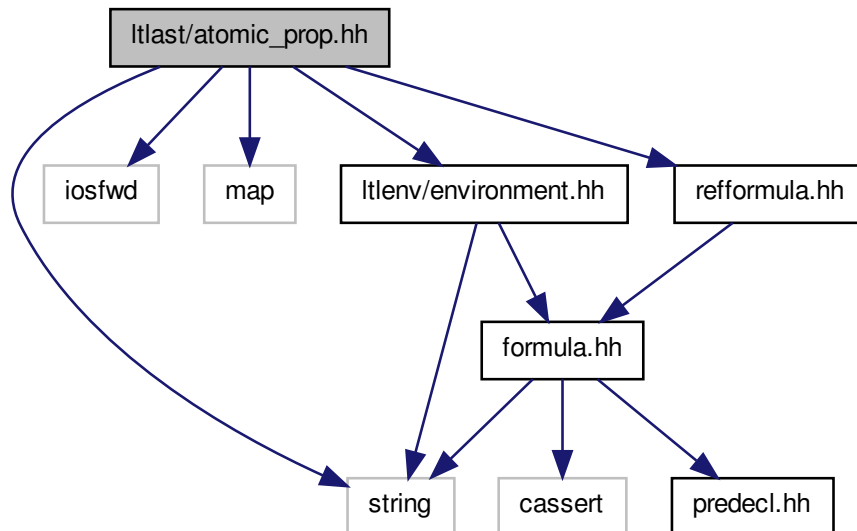
Define all LTL node types. This file is usually needed when **defining** a visitor. Prefer [ltlast/predecl.hh](#) when only **declaring** methods and functions over LTL nodes.

8.41 ltlast/atomic_prop.hh File Reference

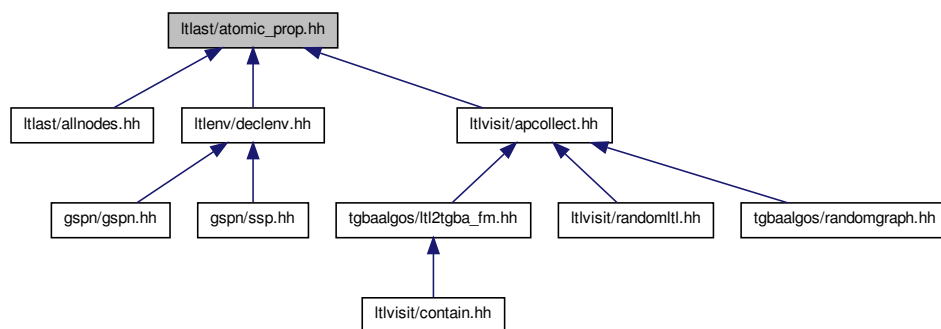
LTL atomic propositions.

```
#include <string>
#include <iosfwd>
#include <map>
#include "refformula.hh"
#include "ltlenv/environment.hh"
```

Include dependency graph for atomic_prop.hh:



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



Classes

- class `spot::Itl::atomic_prop`
Atomic propositions.

Namespaces

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

8.41.1 Detailed Description

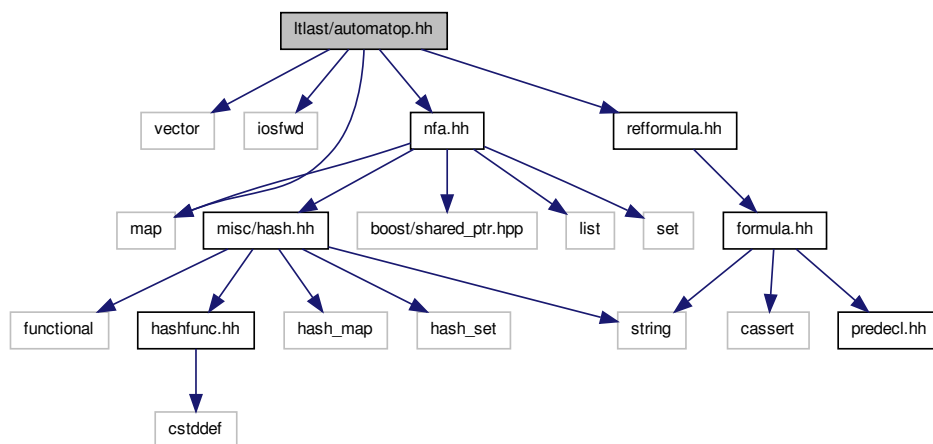
LTL atomic propositions.

8.42 Itlast/automatop.hh File Reference

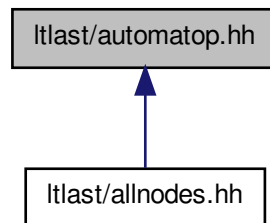
ELTL automaton operators.

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

Include dependency graph for automatop.hh:



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



Classes

- class [spot::ltl::automatop](#)
Automaton operators.
- struct [spot::ltl::automatop::tripletemp](#)
Comparison functor used internally by [ltl::automatop](#).

Namespaces

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

8.42.1 Detailed Description

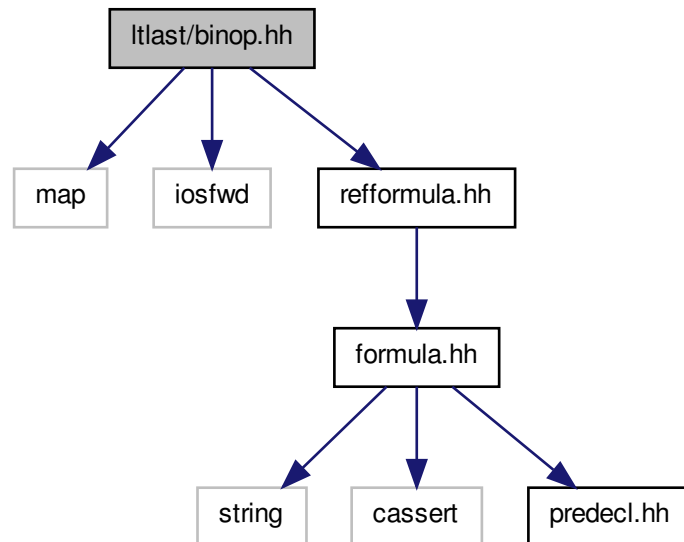
ELTL automaton operators.

8.43 Itlast/binop.hh File Reference

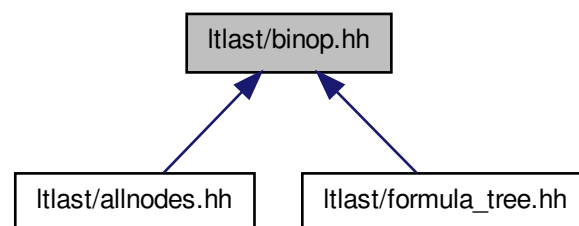
LTL binary operators.

```
#include <map>
#include <iosfwd>
#include "reformula.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](#)

8.43.1 Detailed Description

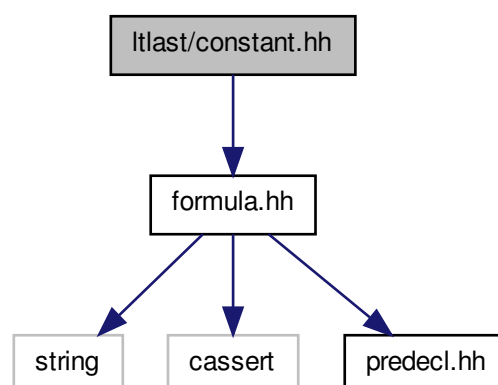
LTL binary operators. This does not include AND and OR operators. These are considered to be multi-operand operators (see [spot::ltl::multop](#)).

8.44 Itlast/constant.hh File Reference

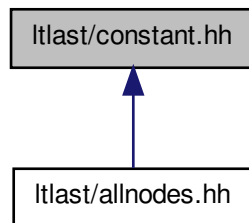
LTL constants.

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

Include dependency graph for constant.hh:



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



Classes

- class `spot::ltl::constant`
A constant (True or False).

Namespaces

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

8.44.1 Detailed Description

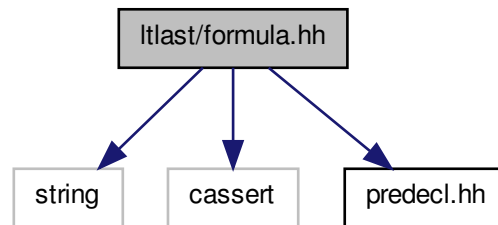
LTL constants.

8.45 Itlast/formula.hh File Reference

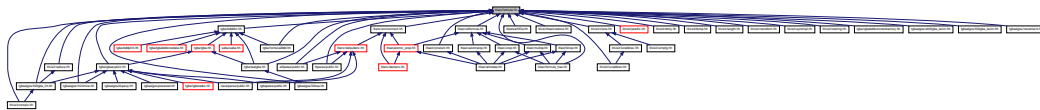
LTL formula interface.

```
#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](#)

8.45.1 Detailed Description

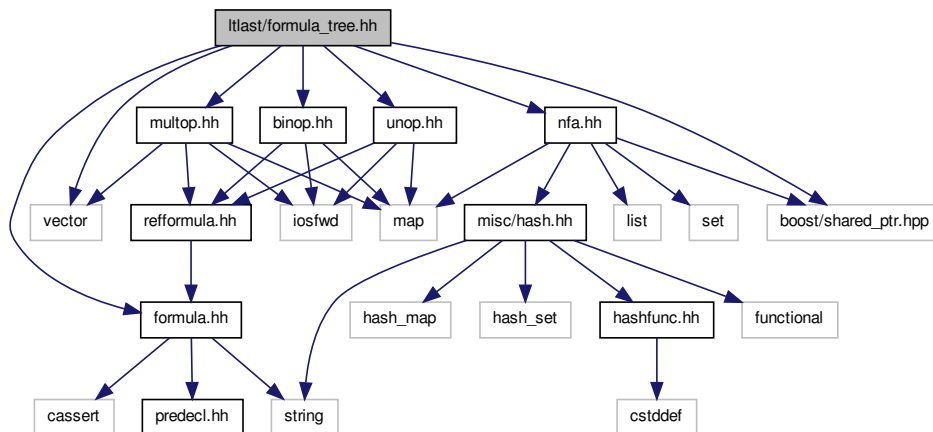
LTL formula interface.

8.46 ltlast/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::instanciate](#) (const node_ptr np, const std::vector< formula * > &v)
- size_t [spot::ltl::formula_tree::arity](#) (const node_ptr np)

Get the arity.

8.46.1 Detailed Description

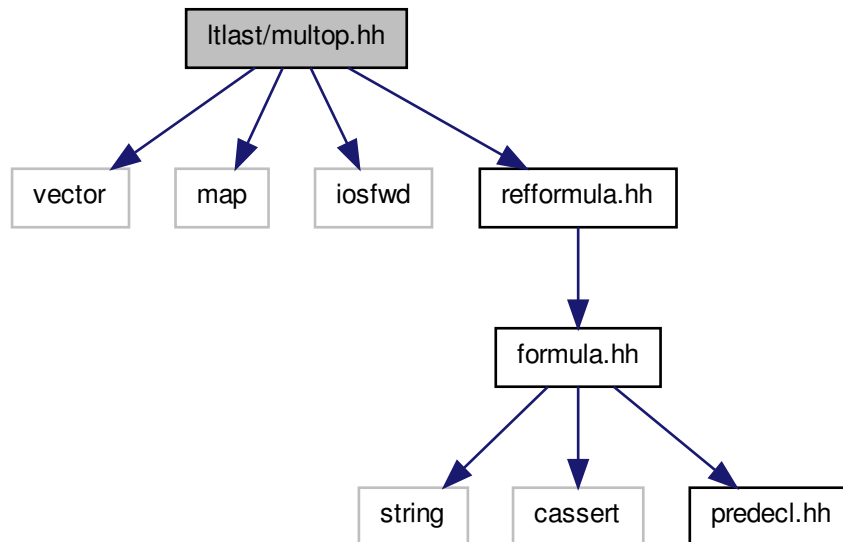
Trees representing formulae where atomic propositions are unknown.

8.47 Itlast/multop.hh File Reference

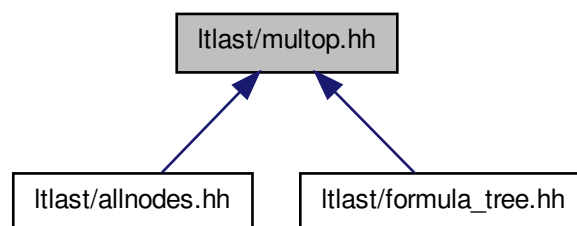
LTL multi-operand operators.

```
#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::pairemp`

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

Namespaces

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

8.47.1 Detailed Description

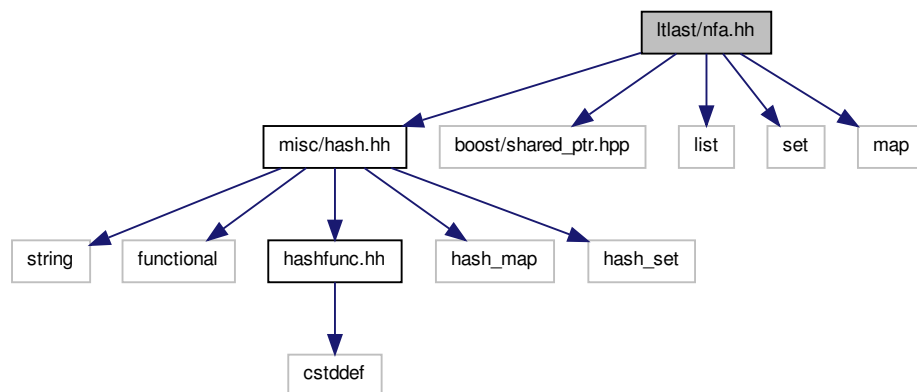
LTL multi-operand operators.

8.48 Itlast/nfa.hh File Reference

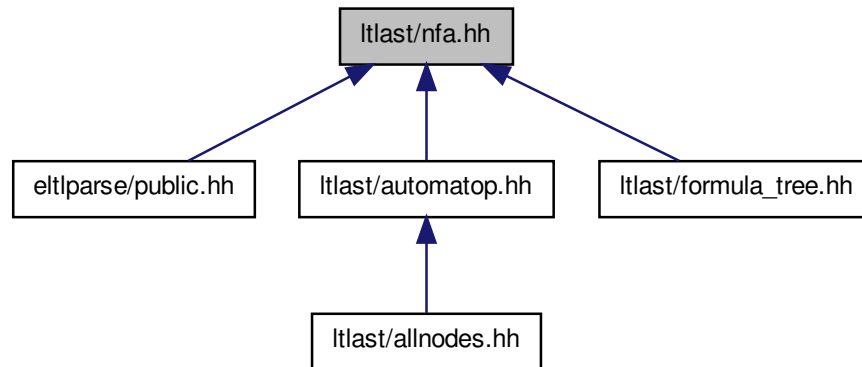
NFA interface used by `automatop`.

```
#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.

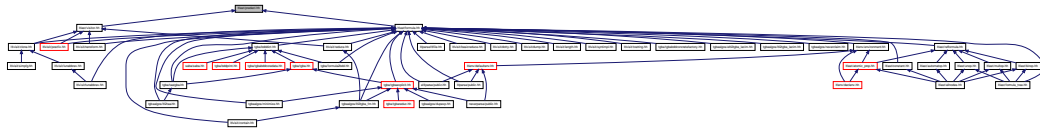
8.48.1 Detailed Description

NFA interface used by automatop.

8.49 Itlast/predecl.hh File Reference

Predeclare all LTL node types.

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



Namespaces

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

8.49.1 Detailed Description

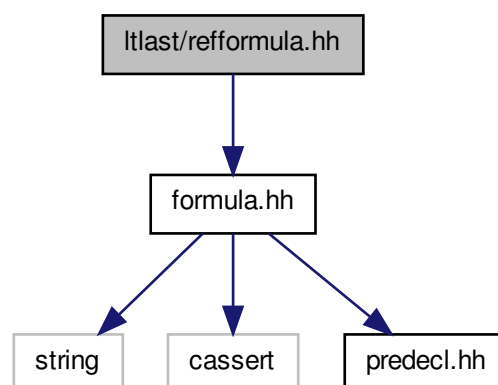
Predeclare all LTL node types. This file is usually used when **declaring** methods and functions over LTL nodes. Use [ltlast/allnodes.hh](#) or an individual header when the definition of the node is actually needed.

8.50 Itlast/reformula.hh File Reference

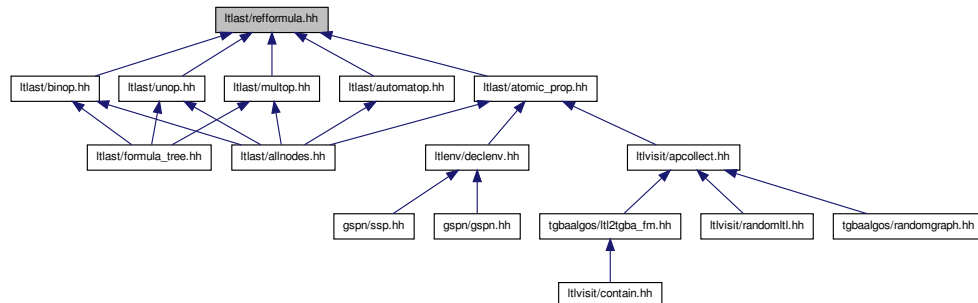
Reference-counted LTL formulae.

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

Include dependency graph for reformula.hh:



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



Classes

- class [spot::ltl::ref_formula](#)
A reference-counted LTL formula.

Namespaces

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

8.50.1 Detailed Description

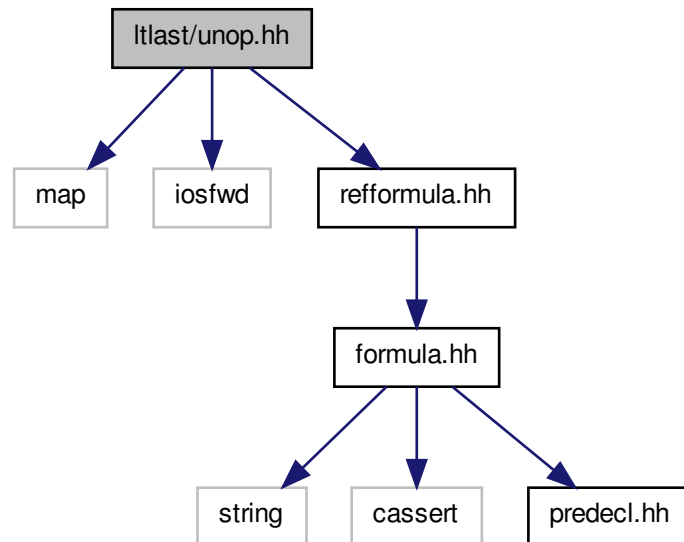
Reference-counted LTL formulae.

8.51 Itlast/unop.hh File Reference

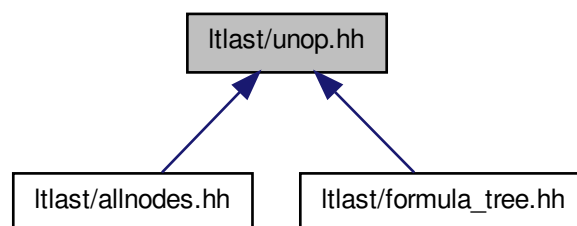
LTL unary operators.

```
#include <map>
#include <iosfwd>
#include "reformula.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](#)

8.51.1 Detailed Description

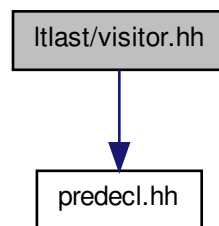
LTL unary operators.

8.52 Itlast/visitor.hh File Reference

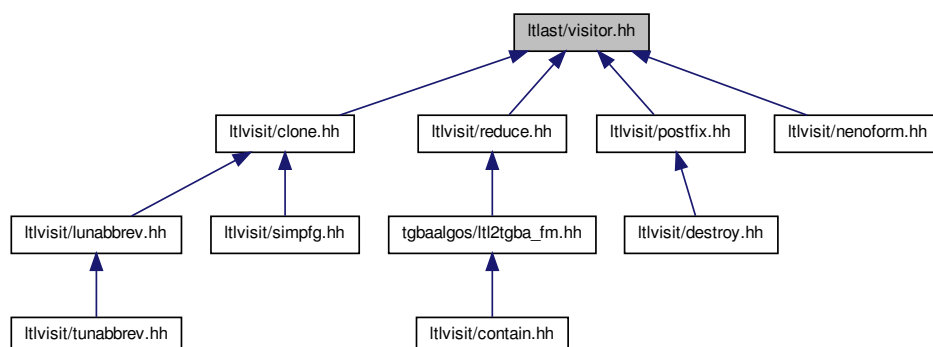
LTL visitor interface.

```
#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`

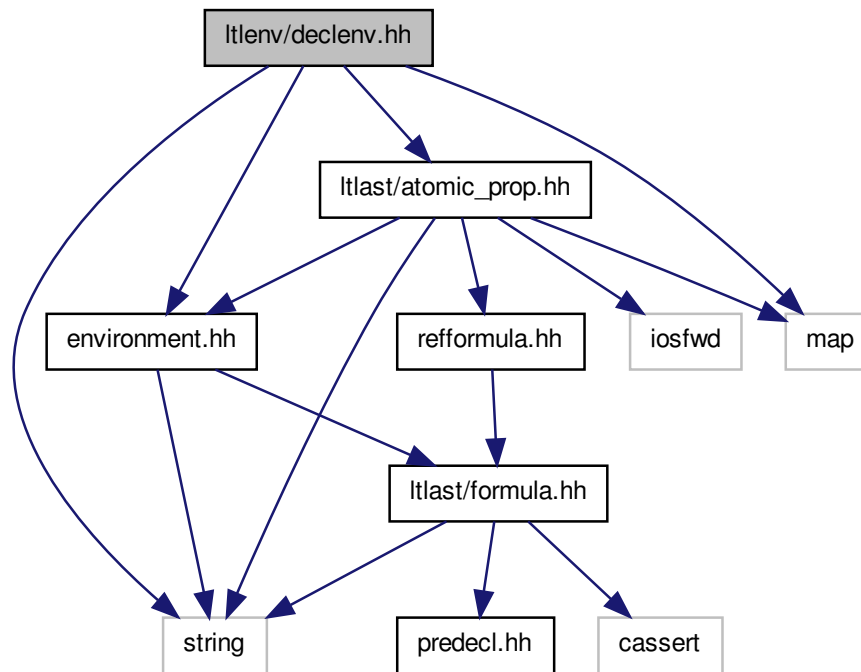
8.52.1 Detailed Description

LTL visitor interface.

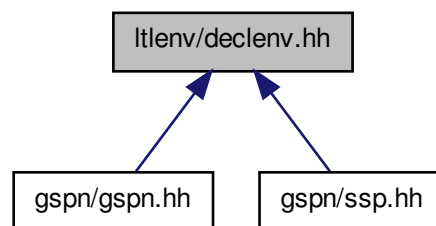
8.53 Itlenv/declenv.hh File Reference

```
#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::Itl::declarative_environment](#)

A declarative environment.

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

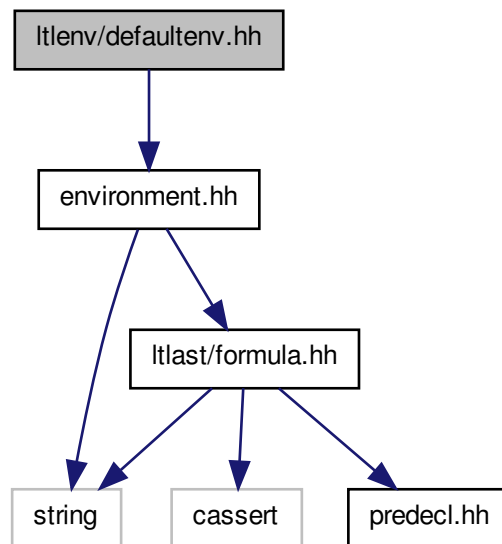
Namespaces

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

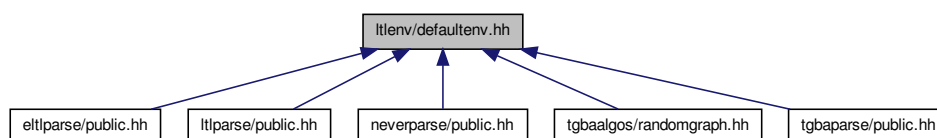
8.54 Itlenv/defaultenv.hh File Reference

```
#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

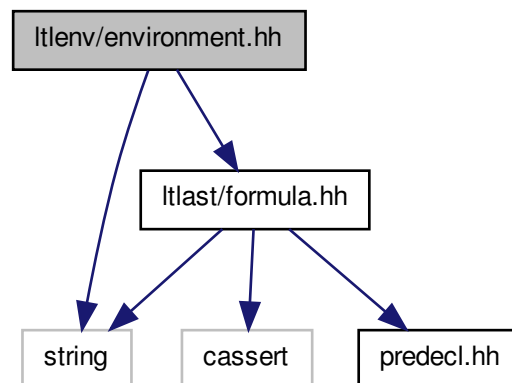
- namespace [spot](#)
- namespace [spot::ltl](#)

8.55 Itlenv/environment.hh File Reference

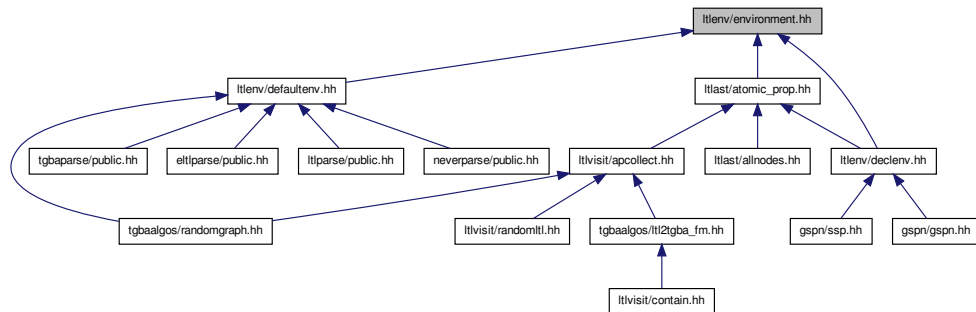
```
#include "ltlast/formula.hh"
```

```
#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.

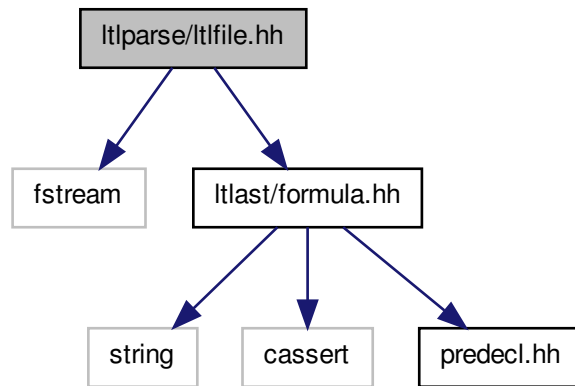
Namespaces

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

8.56 Itlparse/ltlfile.hh File Reference

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

Include dependency graph for Itlfile.hh:



Classes

- class [spot::ltl::ltl_file](#)
Read LTL formulae from a file, one by one.

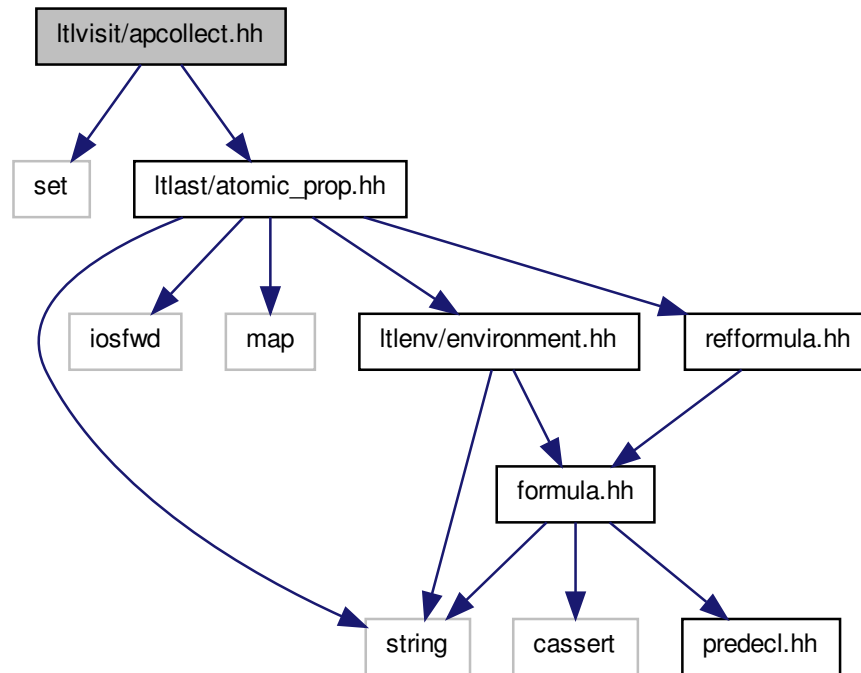
Namespaces

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

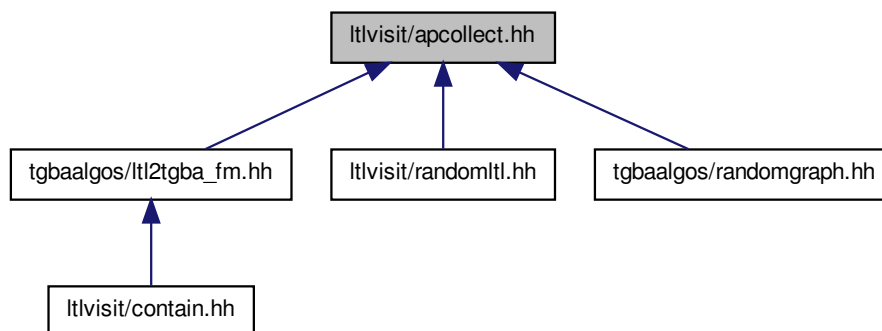
8.57 Itlvisit/apcollect.hh File Reference

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

Include dependency graph for apcollect.hh:



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



Namespaces

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

Typedefs

- typedef std::set< atomic_prop *, formula_ptr_less_than > [spot::ltl::atomic_prop_set](#)
Set of atomic propositions.

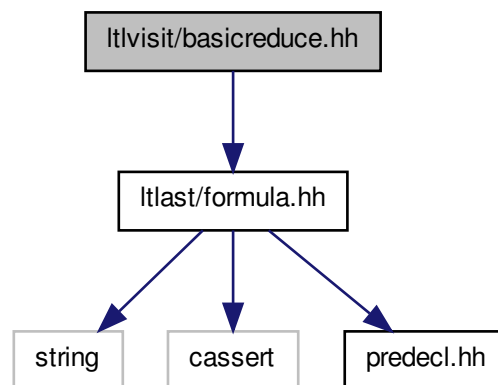
Functions

- 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.

8.58 Itlvisit/basicreduce.hh File Reference

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

Include dependency graph for basicreduce.hh:



Namespaces

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

Functions

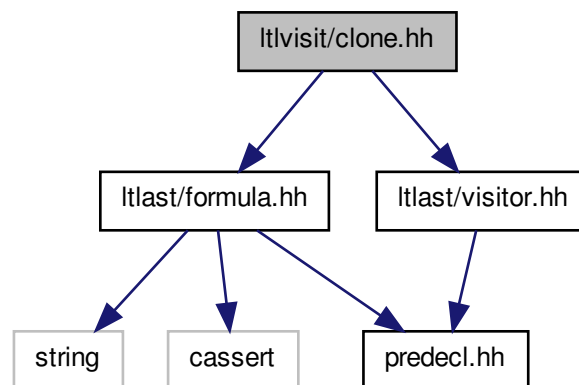
- formula * [spot::ltl::basic_reduce](#) (const formula *f)
Basic rewritings.
- bool [spot::ltl::is_GF](#) (const formula *f)
Whether a formula starts with GF.
- bool [spot::ltl::is_FG](#) (const formula *f)
Whether a formula starts with FG.

8.59 Itlvisit/clone.hh File Reference

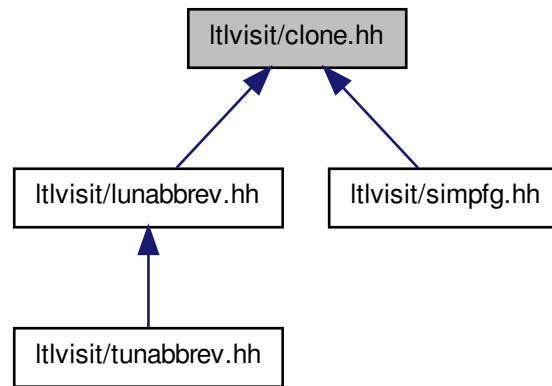
```
#include "ltlast/formula.hh"
```

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

Include dependency graph for clone.hh:



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



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

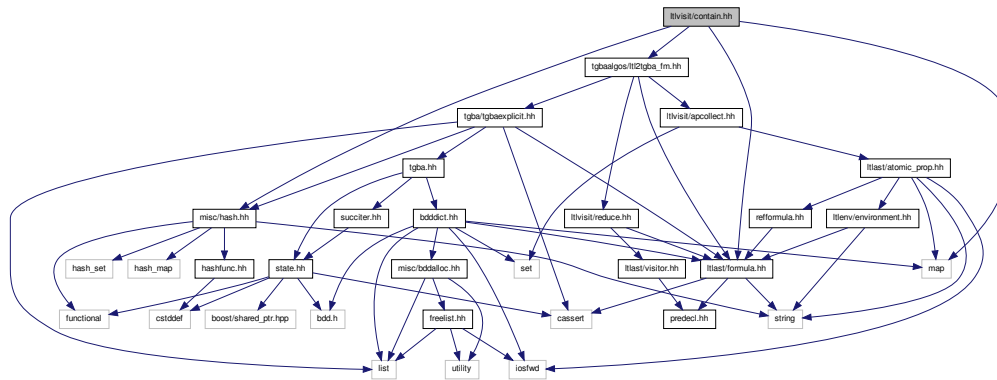
- formula * [spot::ltl::clone](#) (const formula *f)

Clone a formula.

8.60 Itlvisit/contain.hh File Reference

```
#include "ltlast/formula.hh"
#include "tgbaalgorithms/ltl2tgba_fm.hh"
#include "misc/hash.hh"
#include <map>
```

Include dependency graph for contain.hh:



Classes

- class `spot::ltl::language_containment_checker`
- struct `spot::ltl::language_containment_checker::record_`

Namespaces

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

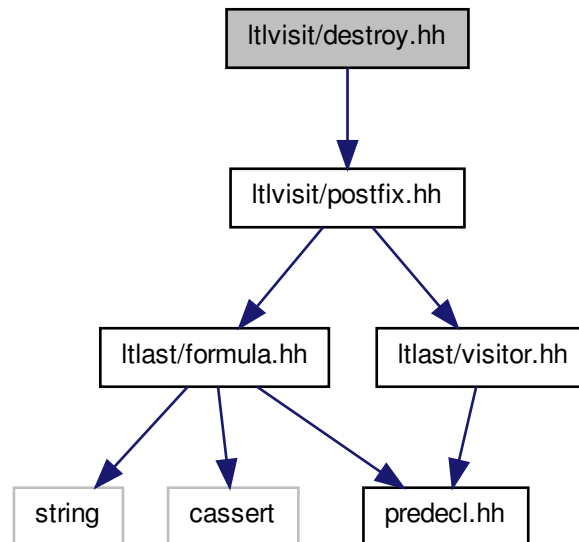
Functions

- `formula * spot::ltl::reduce_tau03` (const formula *f, bool stronger=true)
Reduce a formula using language containment relationships.

8.61 ltlvisit/destroy.hh File Reference

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

Include dependency graph for destroy.hh:



Namespaces

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

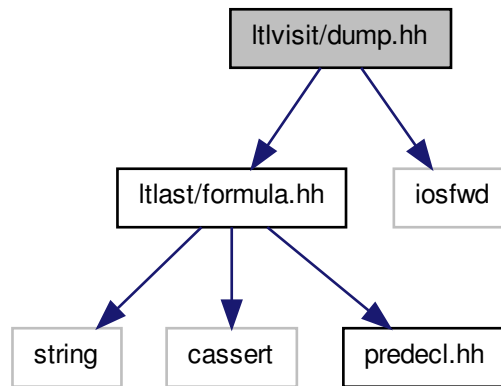
Functions

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

8.62 Itlvisit/dump.hh File Reference

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


Include dependency graph for dump.hh:



Namespaces

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

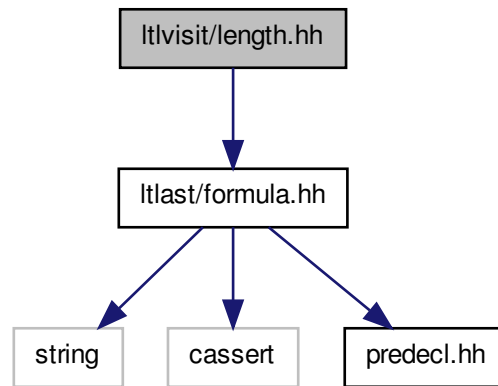
Functions

- `std::ostream & spot::ltl::dump (std::ostream &os, const formula *f)`
Dump a formula tree.

8.63 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)`

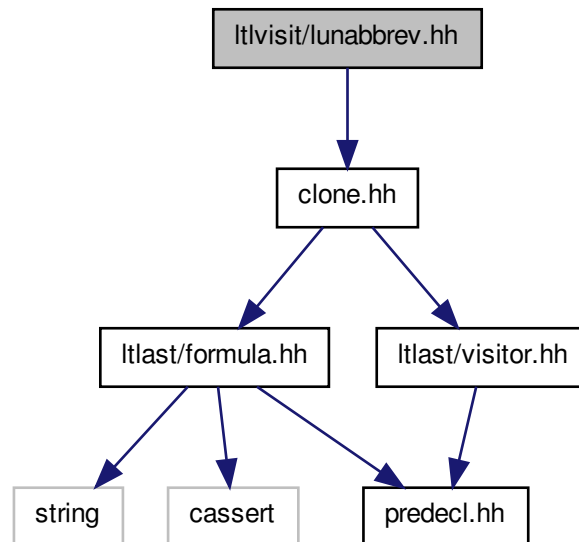
Compute the length of a formula.

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).

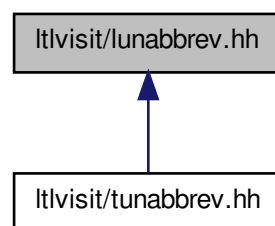
8.64 Itlvisit/lunabbrev.hh File Reference

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

Include dependency graph for lunabbrev.hh:



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



Classes

- class [spot::Itl::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

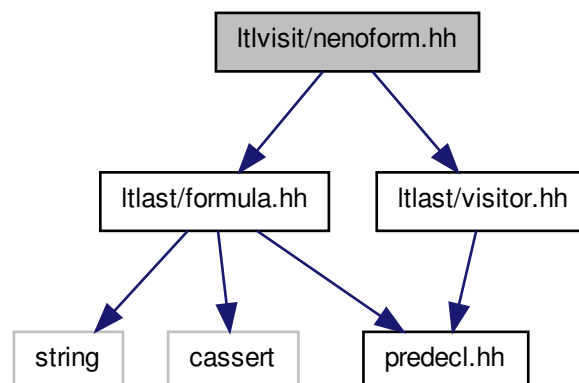
- formula * [spot::ltl::unabbreviate_logic](#) (const formula *f)
*Clone and rewrite a formula to remove most of the abbreviated logical operators.
This will rewrite binary operators such as [binop::Implies](#), [binop::Equals](#), and [binop::Xor](#), using only [unop::Not](#), [multop::Or](#), and [multop::And](#).*

8.65 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

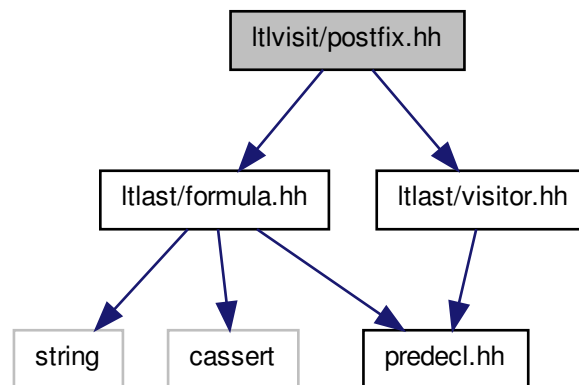
- formula * [spot::ltl::negative_normal_form](#) (const formula *f, bool negated=false)
*Build the negative normal form of f.
All negations of the formula are pushed in front of the atomic propositions.*

8.66 Itlvisit/postfix.hh File Reference

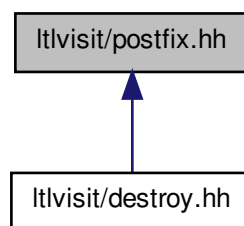
```
#include "ltlast/formula.hh"
```

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

Include dependency graph for postfix.hh:



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



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.

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

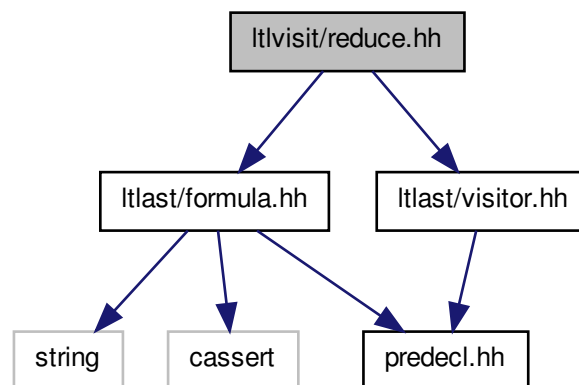
- namespace `spot`
- namespace `spot::ltl`

8.68 Itlvisit/reduce.hh File Reference

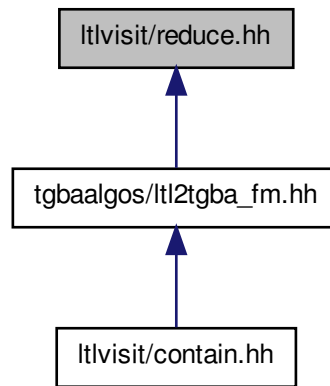
```
#include "ltlast/formula.hh"
```

```
#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`.

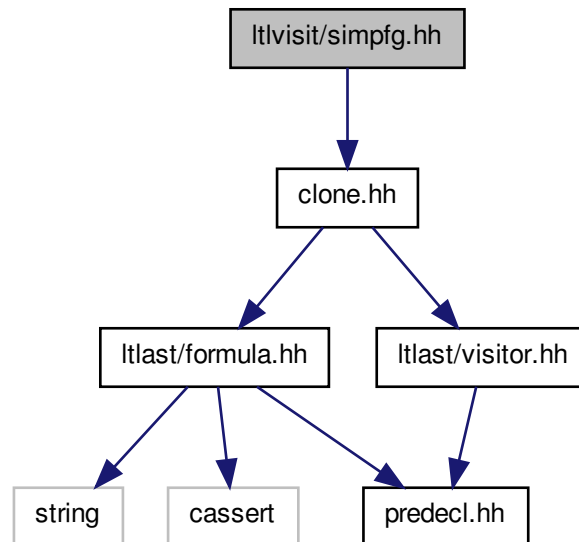
Functions

- 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.*

8.69 Itlvisit/simpfg.hh File Reference

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

Include dependency graph for simpfg.hh:



Classes

- class [spot::ltl::simplify_f_g_visitor](#)
Replace $true \cup f$ and $false \cap g$ by $F f$ and $G g$.

Namespaces

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

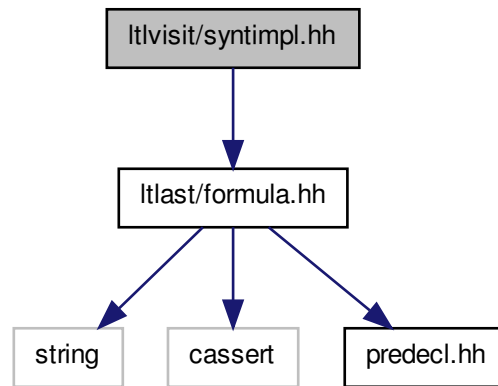
Functions

- formula * [spot::ltl::simplify_f_g](#) (const formula *f)
Replace $true \cup f$ and $false \cap g$ by $F f$ and $G g$.

8.70 Itlvisit/syntimpl.hh File Reference

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

Include dependency graph for syntimpl.hh:



Namespaces

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

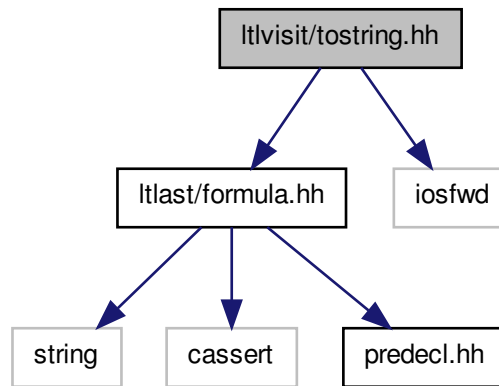
Functions

- 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.*

8.71 Itlvisit/tostring.hh File Reference

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

Include dependency graph for tostring.hh:



Namespaces

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

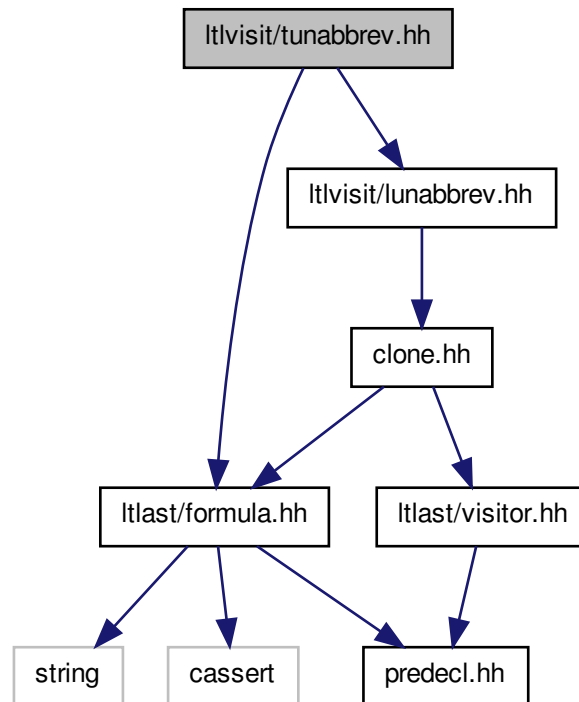
Functions

- `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, bool full_parent=false)
Output a formula as a (parsable by Spin) string.
- `std::string spot::ltl::to_spin_string` (const formula *f, bool full_parent=false)
Convert a formula into a (parsable by Spin) string.

8.72 ltlvisit/tunabbrev.hh File Reference

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

Include dependency graph for tunabbrev.hh:



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

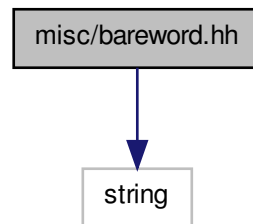
- formula * [spot::ltl::unabbreviate_ltl](#) (const formula *f)

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

8.73 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.

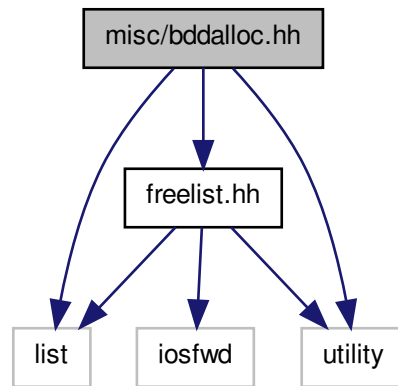
8.74 misc/bddalloc.hh File Reference

```
#include "freelist.hh"
```

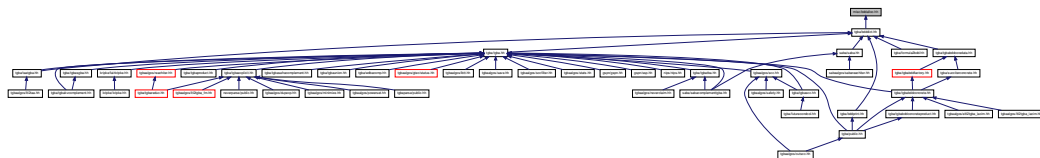
```
#include <list>
```

```
#include <utility>
```

Include dependency graph for bddalloc.hh:



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



Classes

- class [spot::bdd_allocator](#)
Manage ranges of variables.

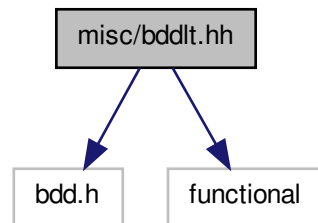
Namespaces

- namespace [spot](#)

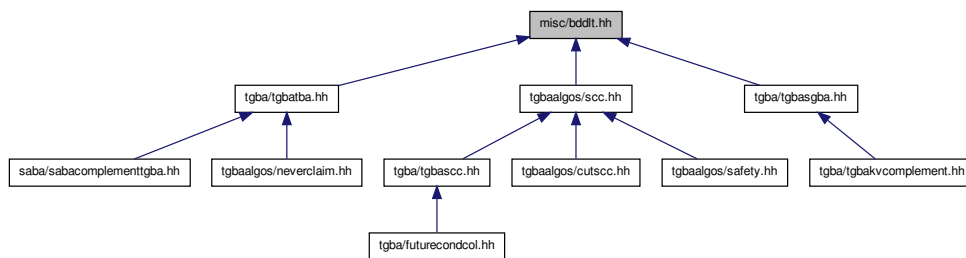
8.75 misc/bddlt.hh File Reference

```
#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.

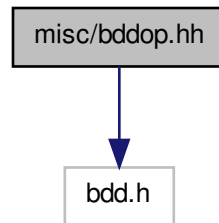
Namespaces

- namespace [spot](#)

8.76 misc/bddop.hh File Reference

```
#include "bdd.h"
```

Include dependency graph for bddop.hh:



Namespaces

- namespace [spot](#)

Functions

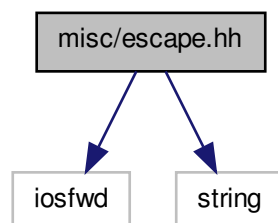
- bdd [spot::compute_all_acceptance_conditions](#) (bdd neg_acceptance_conditions)
Compute all acceptance conditions from all neg acceptance conditions.
- bdd [spot::compute_neg_acceptance_conditions](#) (bdd all_acceptance_conditions)
Compute neg acceptance conditions from all acceptance conditions.

8.77 misc/escape.hh File Reference

```
#include <iosfwd>
```

```
#include <string>
```

Include dependency graph for escape.hh:



Namespaces

- namespace [spot](#)

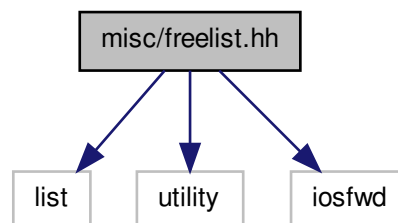
Functions

- `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.

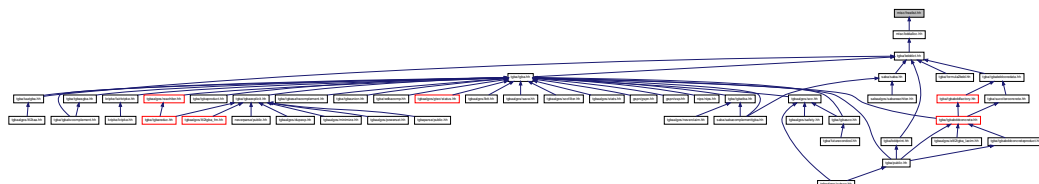
8.78 misc/freelist.hh File Reference

```
#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.

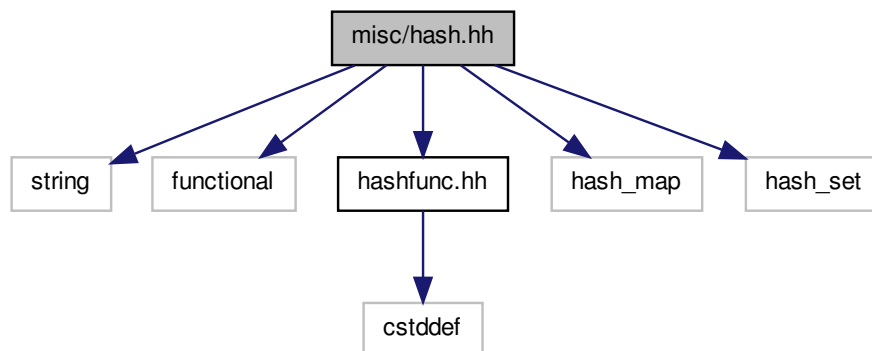
Namespaces

- namespace [spot](#)

8.79 misc/hash.hh File Reference

```
#include <string>
#include <functional>
#include "hashfunc.hh"
#include <hash_map>
#include <hash_set>
```

Include dependency graph for hash.hh:



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



Classes

- struct [spot::ptr_hash< T >](#)
A hash function for pointers.

- struct [spot::string_hash](#)
A hash function for strings.
- struct [spot::identity_hash< T >](#)
A hash function that returns identity.

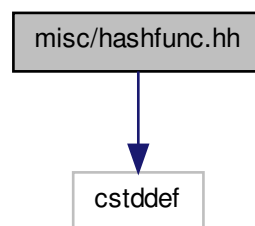
Namespaces

- namespace [spot](#)

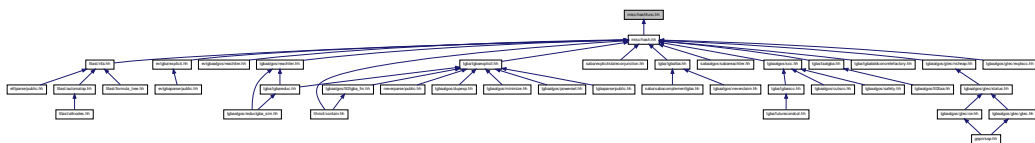
8.80 misc/hashfunc.hh File Reference

```
#include <cstdint>
```

Include dependency graph for hashfunc.hh:



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



Namespaces

- namespace [spot](#)

Functions

- size_t [spot::wang32_hash](#) (size_t key)
Thomas Wang's 32 bit hash function.

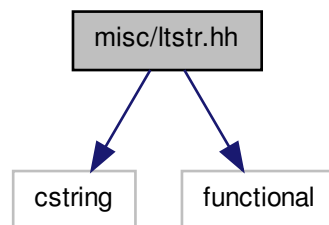
- `size_t` [spot::knuth32_hash](#) (`size_t` key)
Knuth's Multiplicative hash function.

8.81 misc/ltstr.hh File Reference

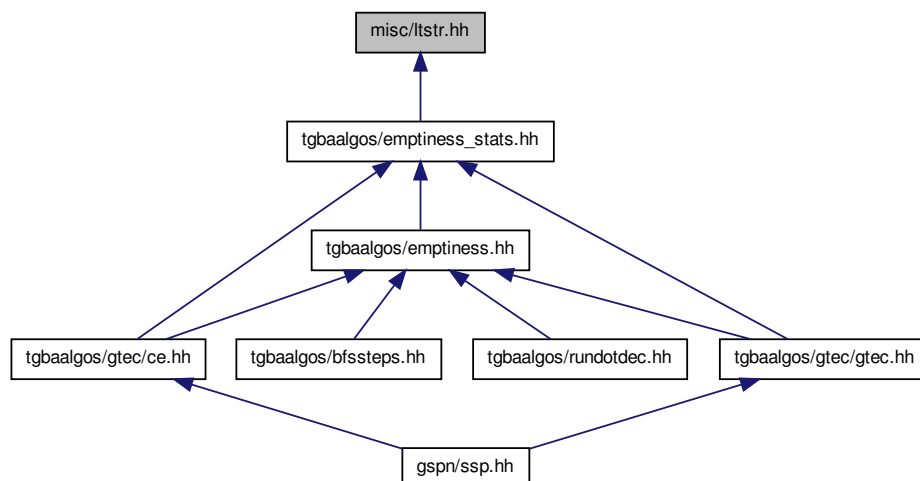
```
#include <cstring>
```

```
#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](#)

8.82 misc/memusage.hh File Reference

Namespaces

- namespace [spot](#)

Functions

- int [spot::memusage](#) ()

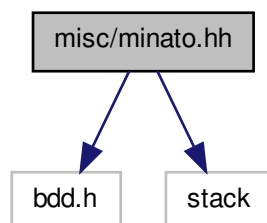
Total number of pages in use by the program.

8.83 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](#)

8.84 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](#)

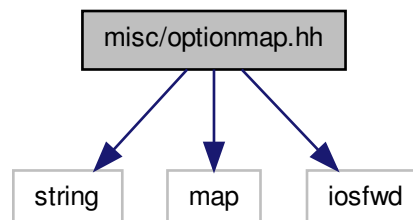
8.85 misc/optionmap.hh File Reference

```
#include <string>
```

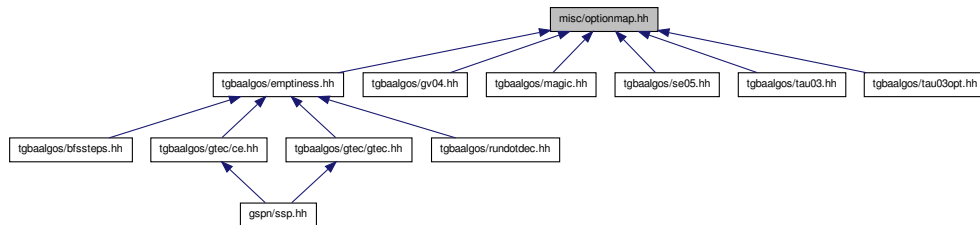
```
#include <map>
```

```
#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.*

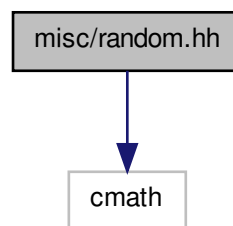
Namespaces

- namespace [spot](#)

8.86 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](#)

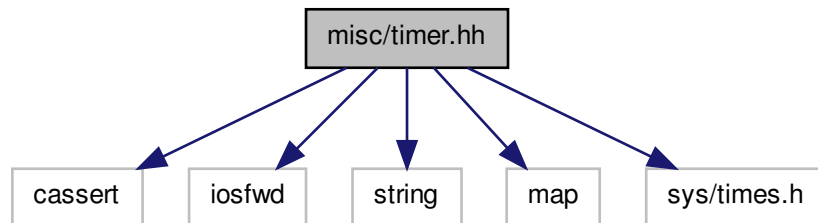
Functions

- 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::mrnd](#) (int max)
Compute a pseudo-random integer value between 0 and max-1 included.
- double [spot::drand](#) ()
Compute a pseudo-random double value between 0.0 and 1.0 (1.0 excluded).
- double [spot::nrnd](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Odeh & Evans).
- double [spot::bmrand](#) ()
Compute a pseudo-random double value following a standard normal distribution. (Box-Muller).
- int [spot::prand](#) (double p)
Return a pseudo-random positive integer value following a Poisson distribution with parameter p.

8.87 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`

8.88 misc/version.hh File Reference

Namespaces

- namespace `spot`

Functions

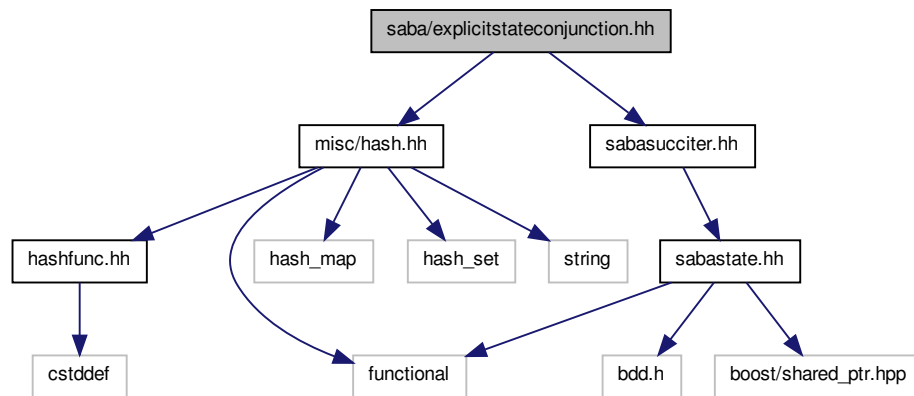
- `const char * spot::version ()`
Return Spot's version.

8.89 saba/explicitstateconjunction.hh File Reference

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

```
#include "sabasucciter.hh"
```

Include dependency graph for explicitstateconjunction.hh:



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](#)

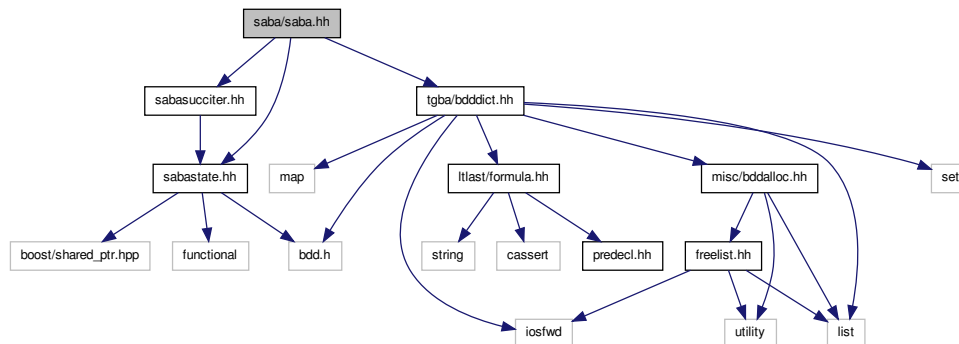
8.90 saba/saba.hh File Reference

```
#include "sabastate.hh"
```

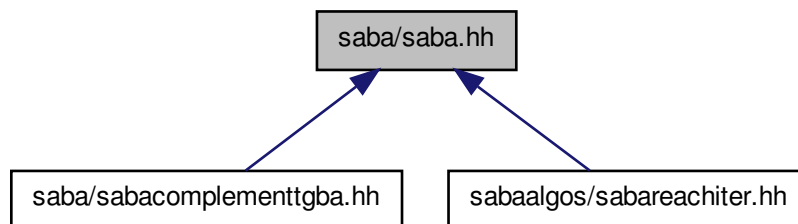
```
#include "sabasucciter.hh"
```

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

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](#)

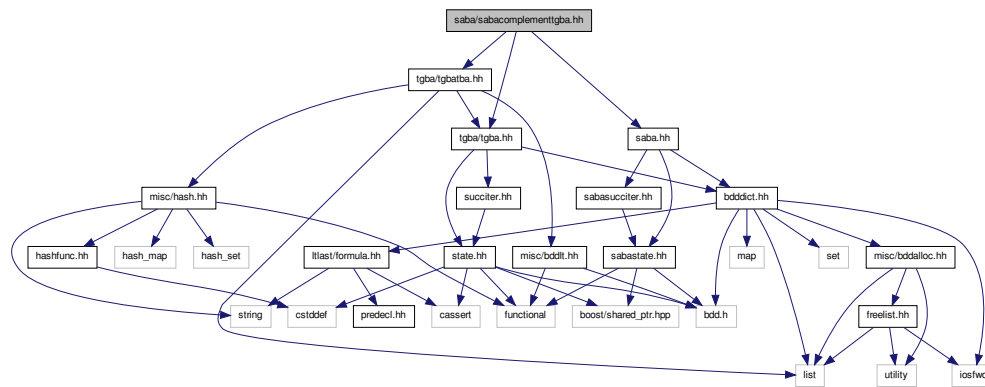
8.91 saba/sabacomplementtgba.hh File Reference

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

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

```
#include "saba.hh"
```

Include dependency graph for sabacomplementtgba.hh:



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](#)

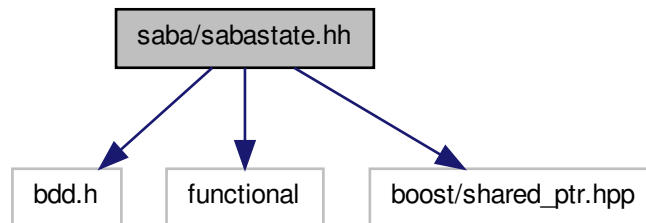
8.92 saba/sabastate.hh File Reference

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

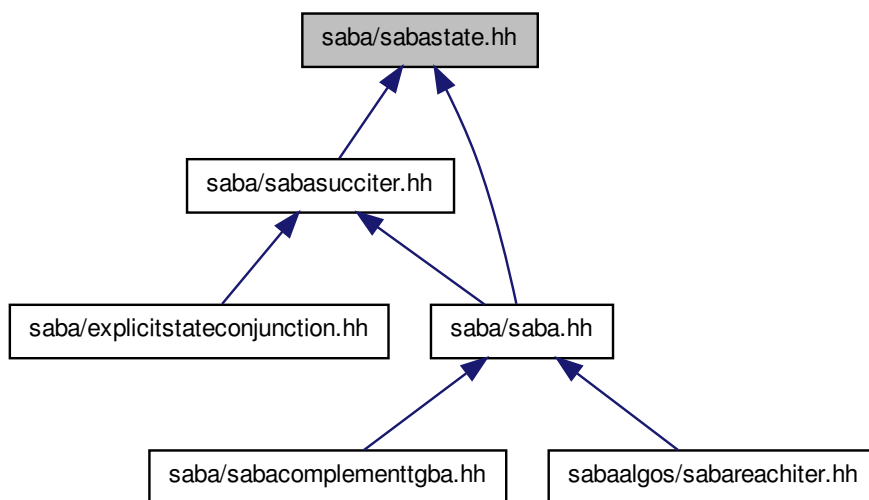
```
#include <functional>
```

```
#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](#)

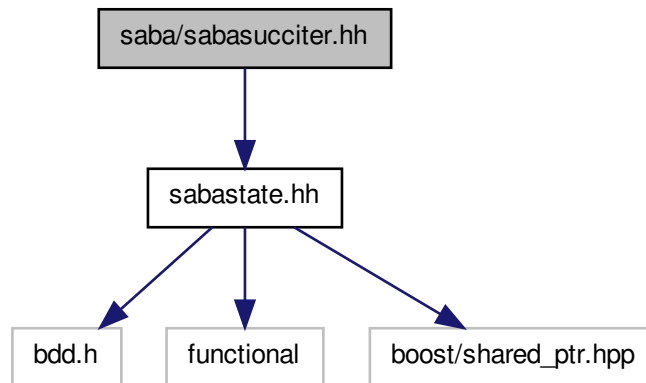
Typedefs

- typedef boost::shared_ptr< const saba_state > [spot::shared_saba_state](#)

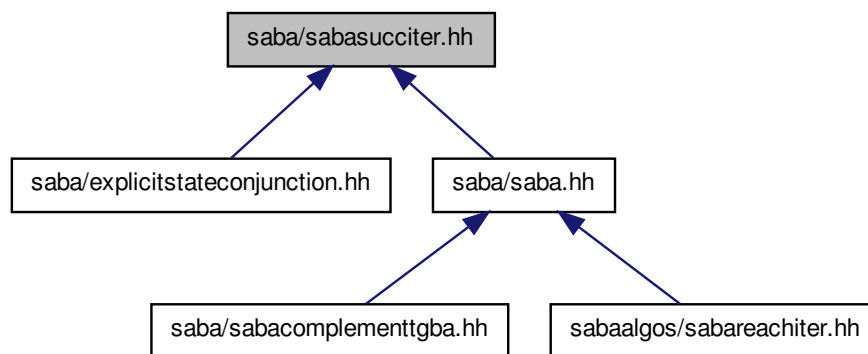
8.93 saba/sabasucciter.hh File Reference

```
#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),.

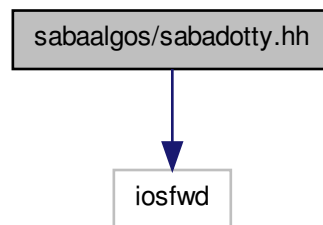
Namespaces

- namespace [spot](#)

8.94 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.

8.95 sabaalgos/sabareachiter.hh File Reference

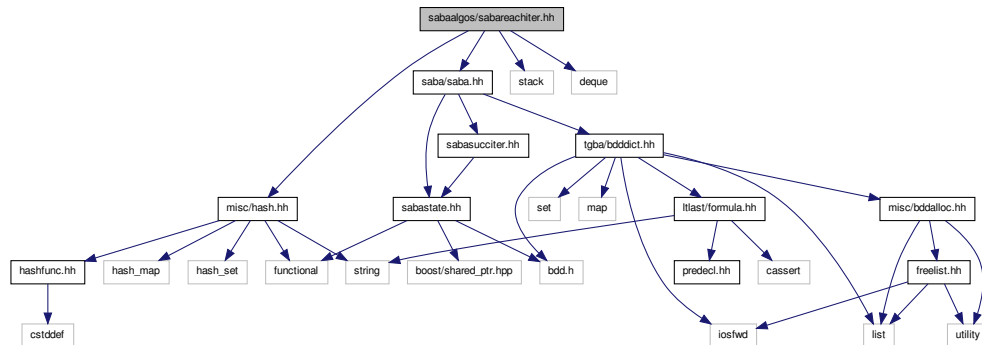
```
#include "misc/hash.hh"
```

```
#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](#)

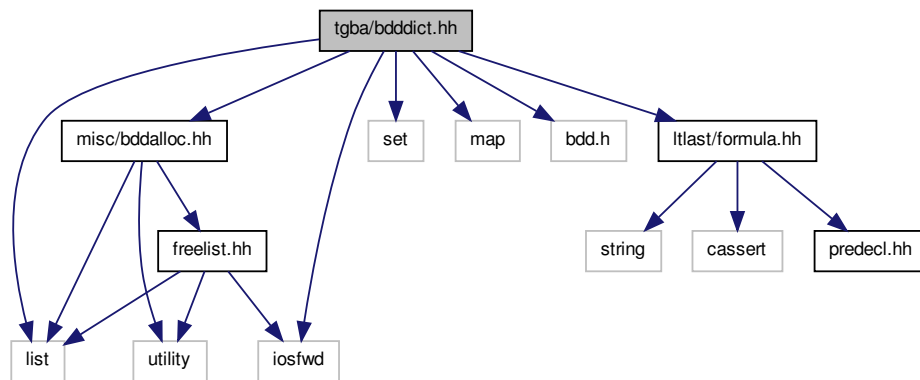
8.96 tgba/bdddict.hh File Reference

```

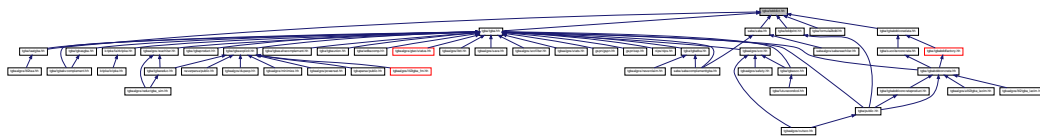
#include <list>
#include <set>
#include <map>
#include <iosfwd>
#include <bdd.h>
#include "ltlast/formula.hh"
#include "misc/bddalloc.hh"

```

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](#)

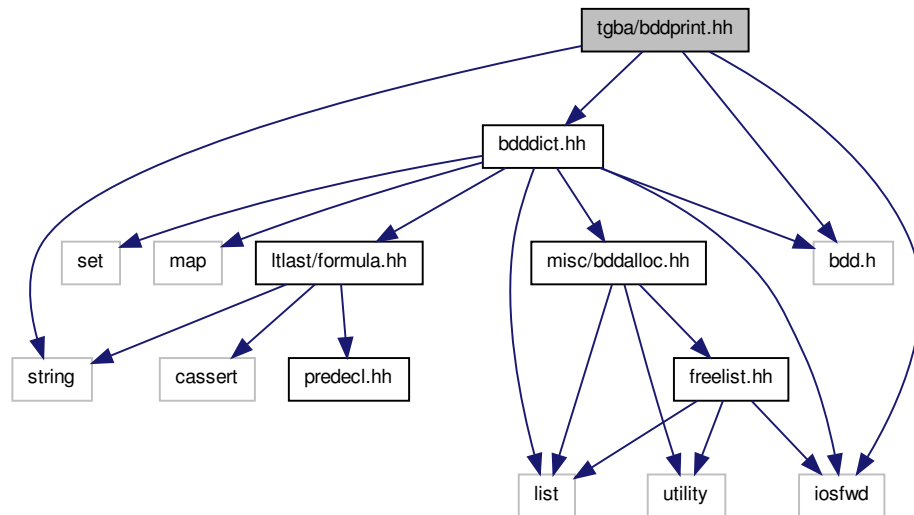
8.97 tgba/bddprint.hh File Reference

```

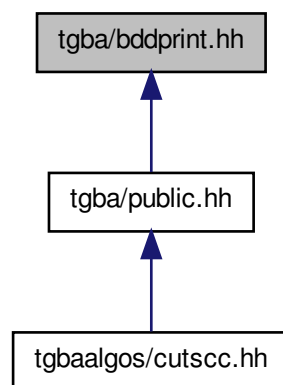
#include <string>
#include <iosfwd>
#include "bdddict.hh"
#include <bdd.h>

```

Include dependency graph for bddprint.hh:



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



Namespaces

- namespace `spot`

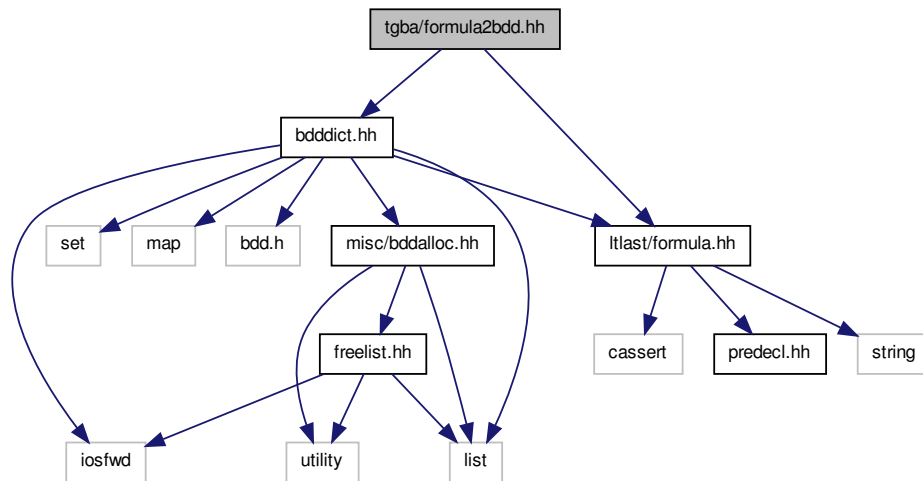
Functions

- `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 dotty format.
- `std::ostream & spot::bdd_print_table` (`std::ostream &os`, `const bdd_dict *dict`, `bdd b`)
Print a BDD as a table.

8.98 tgba/formula2bdd.hh File Reference

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

Include dependency graph for formula2bdd.hh:



Namespaces

- namespace `spot`

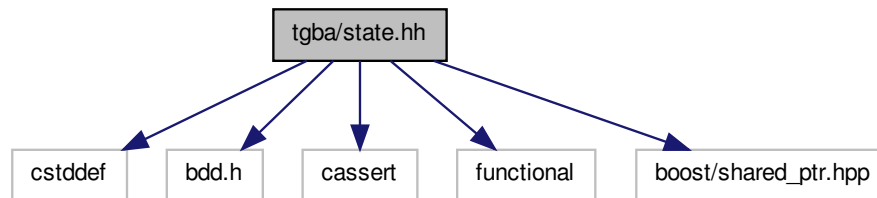
Functions

- `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)

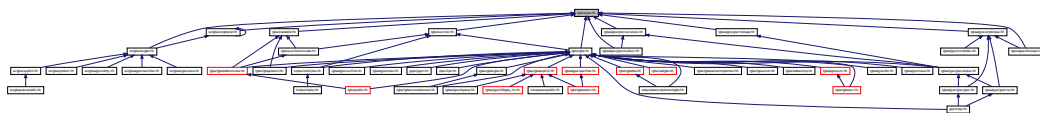
8.99 tgba/futurecondcol.hh File Reference

```
#include "tgbascc.hh"
```


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](#)

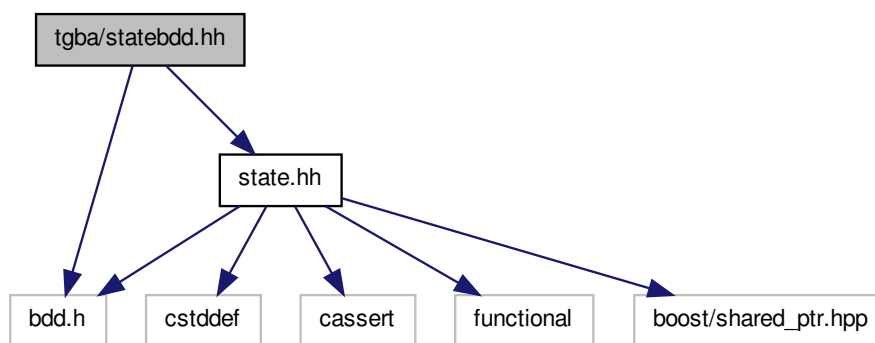
Functions

- void [spot::shared_state_deleter](#) (state *s)

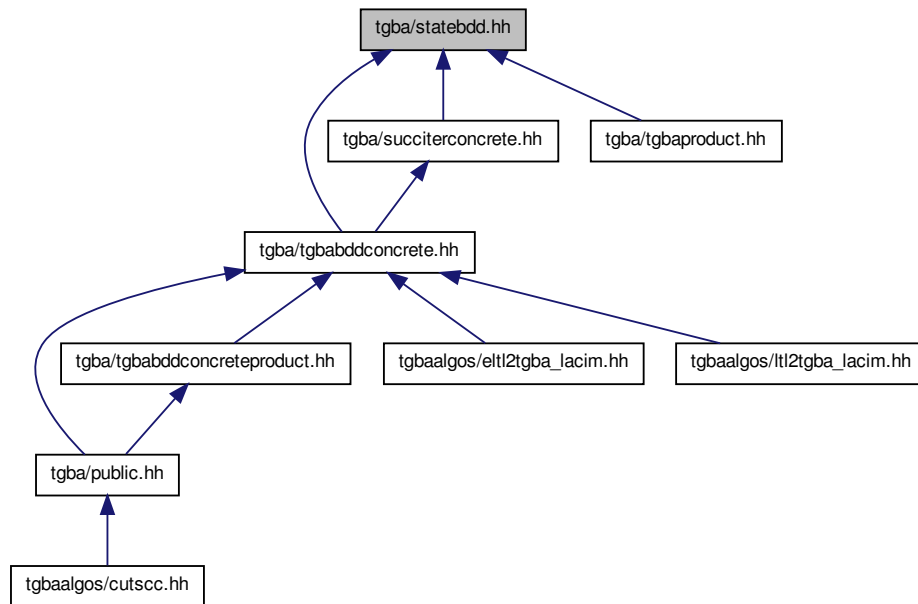
8.101 tgba/statebdd.hh File Reference

```
#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](#)

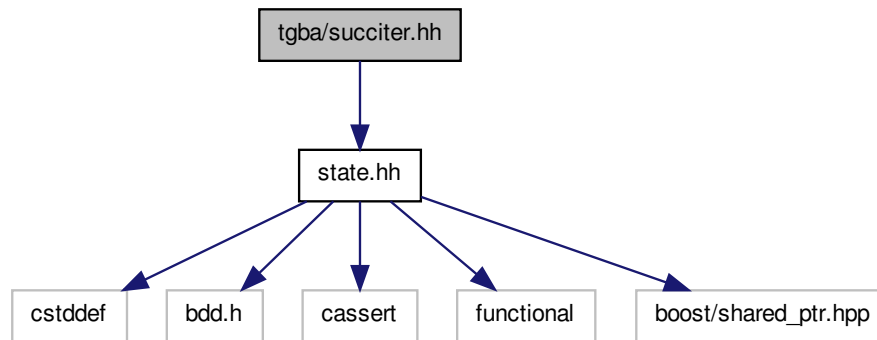
Namespaces

- namespace [spot](#)

8.102 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.

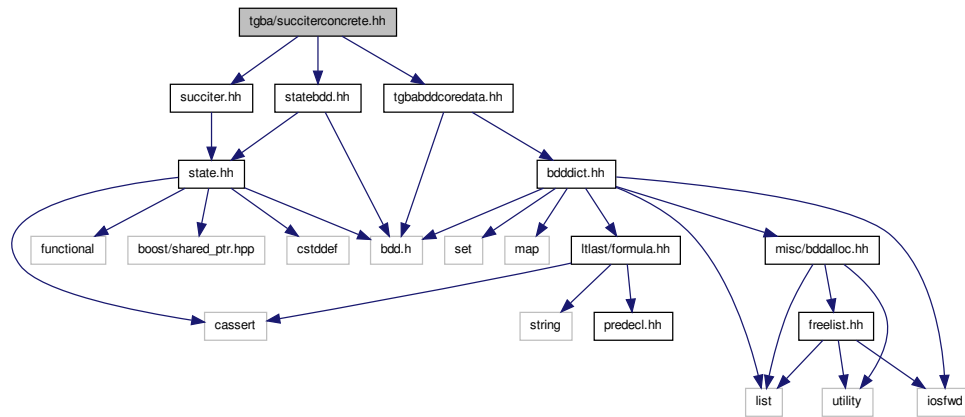
Namespaces

- namespace `spot`

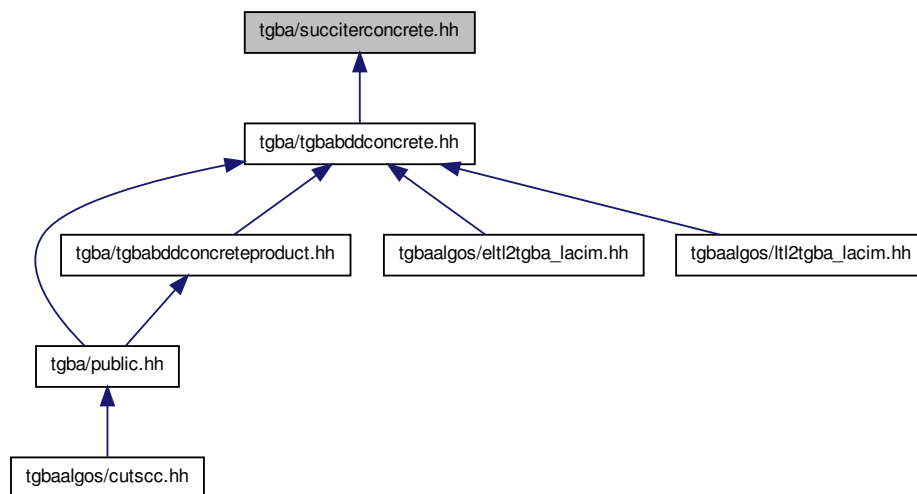
8.103 tgba/succiterconcrete.hh File Reference

```
#include "statebdd.hh"
#include "succiter.hh"
#include "tgbabddcoredata.hh"
```

Include dependency graph for succiterconcrete.hh:



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



Classes

- class [spot::tgba_succ_iterator_concrete](#)

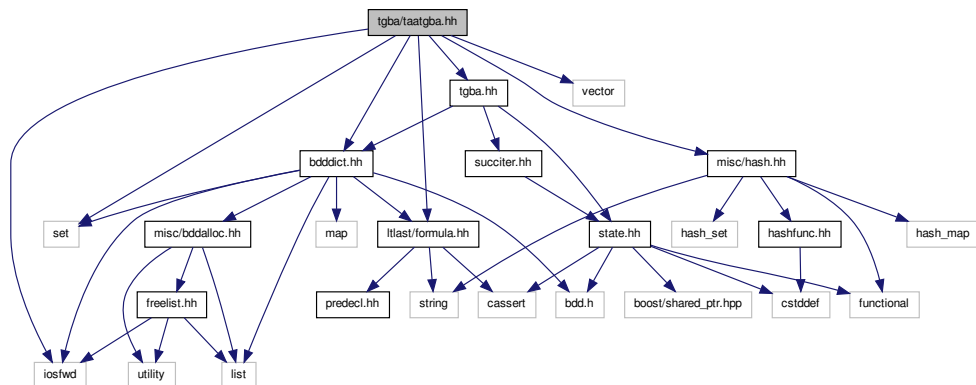
Namespaces

- namespace [spot](#)

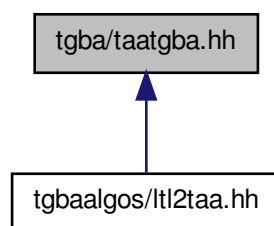
8.104 tgba/taatgba.hh File Reference

```
#include <set>
#include <iosfwd>
#include <vector>
#include "misc/hash.hh"
#include "ltlast/formula.hh"
#include "bdddict.hh"
#include "tgba.hh"
```

Include dependency graph for taatgba.hh:



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



Classes

- class [spot::taa_tgba](#)

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](#)

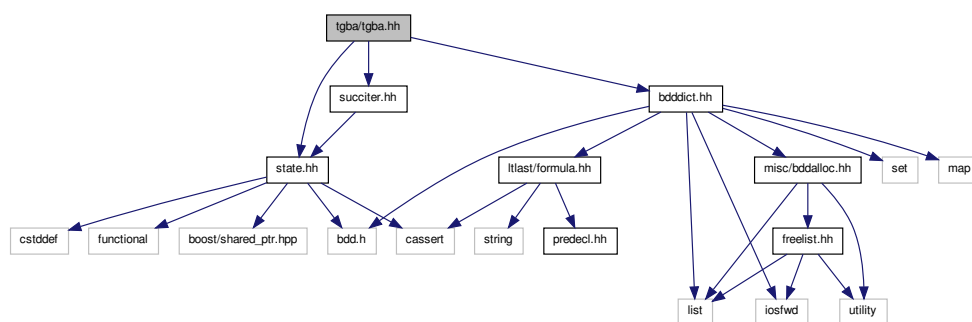
Namespaces

- namespace [spot](#)

8.105 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 Gianakopoulou and Flavio Lerda in "From States to Transitions: Improving Translation of LTL Formulae to Büchi Automata". (FORTE'02).

Namespaces

- namespace [spot](#)

8.106 tgba/tgbabddconcrete.hh File Reference

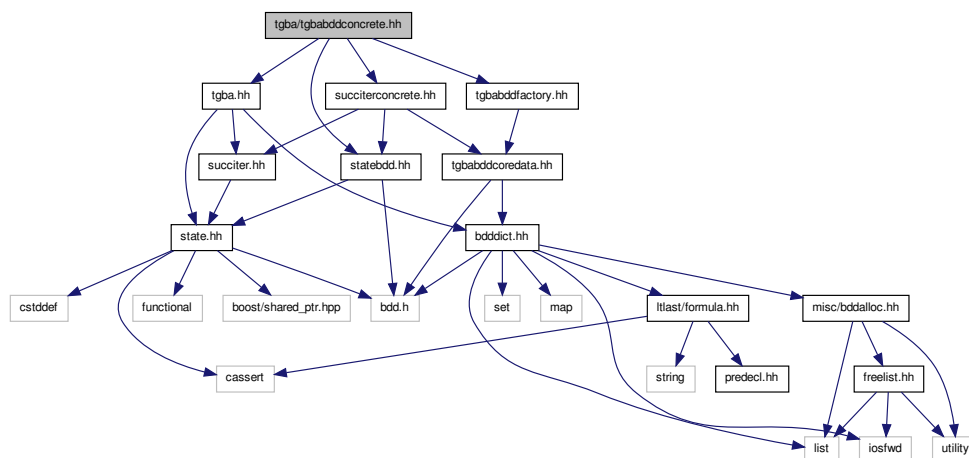
```
#include "tgba.hh"
```

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

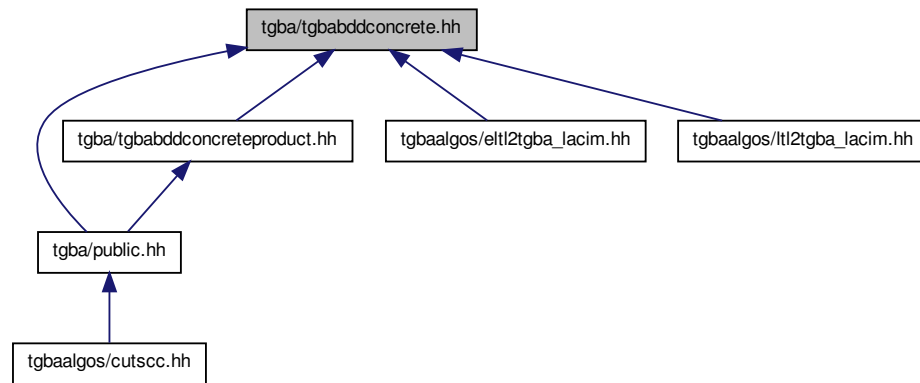
```
#include "tgbabddfactory.hh"
```

```
#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.

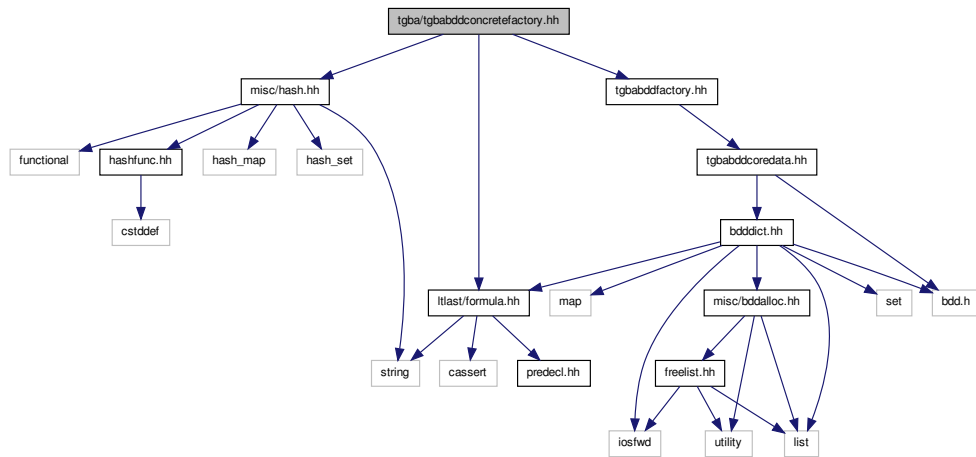
Namespaces

- namespace `spot`

8.107 tgba/tgbabddconcretefactory.hh File Reference

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

Include dependency graph for tgbabddconcretefactory.hh:



Classes

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

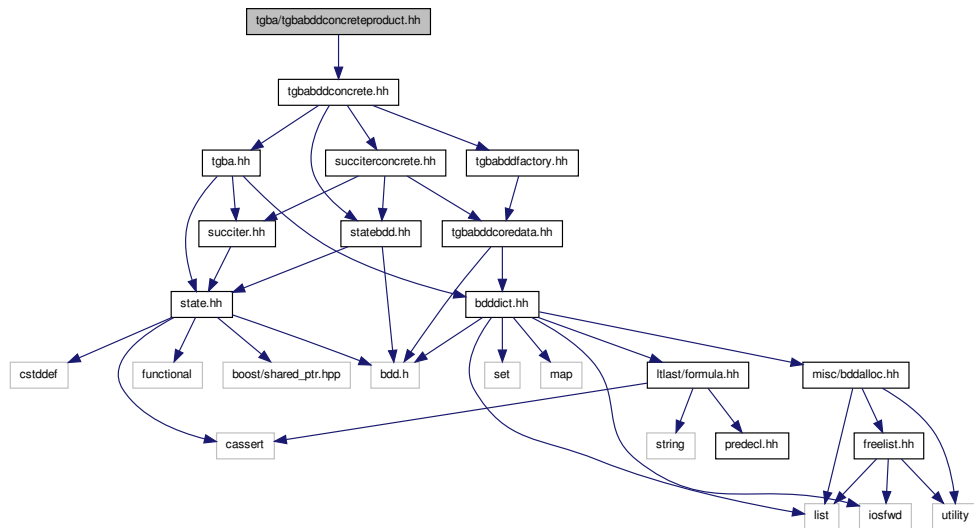
Namespaces

- namespace [spot](#)

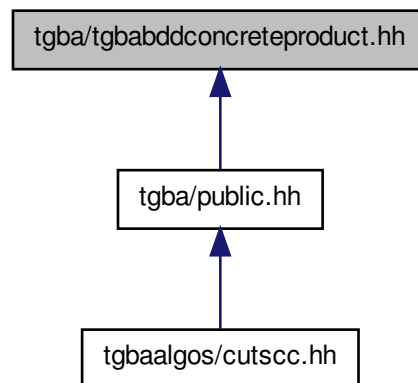
8.108 tgba/tgbabddconcreteproduct.hh File Reference

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


Include dependency graph for tgba/tgbabddconcreteproduct.hh:



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



Namespaces

- namespace [spot](#)

Functions

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

Multiplies two `spot::tgba_bdd_concrete` automata.

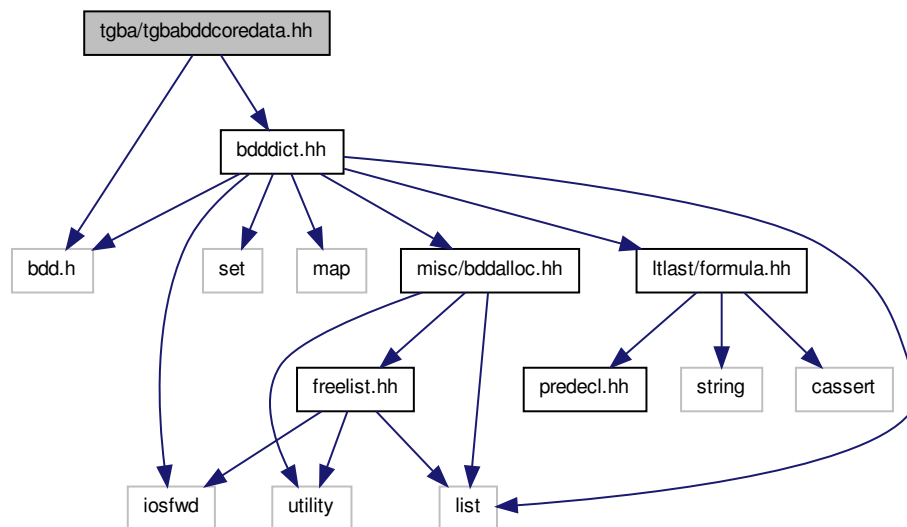
This function builds the resulting product as another `spot::tgba_bdd_concrete` automaton.

8.109 tgba/tgbabddcoredata.hh File Reference

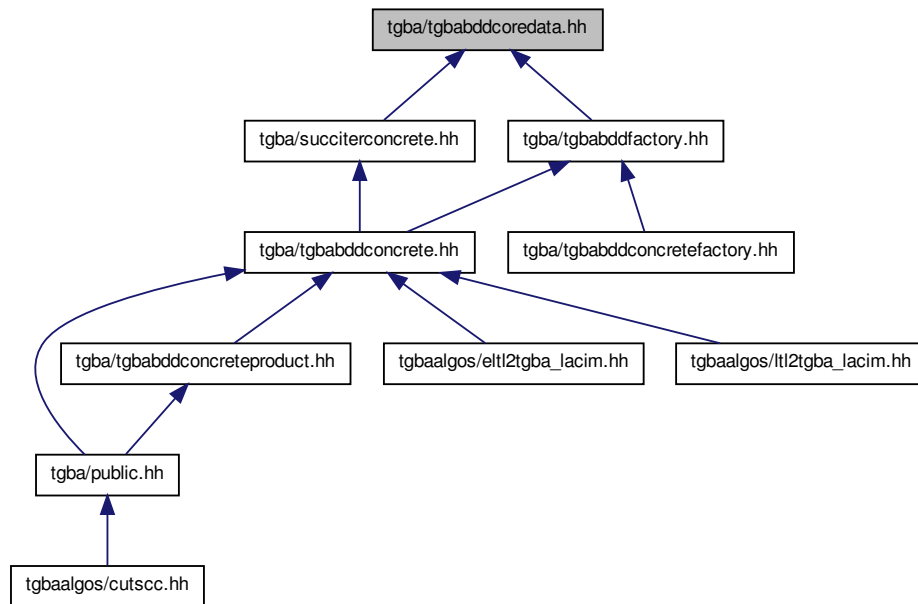
```
#include <bdd.h>
```

```
#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.

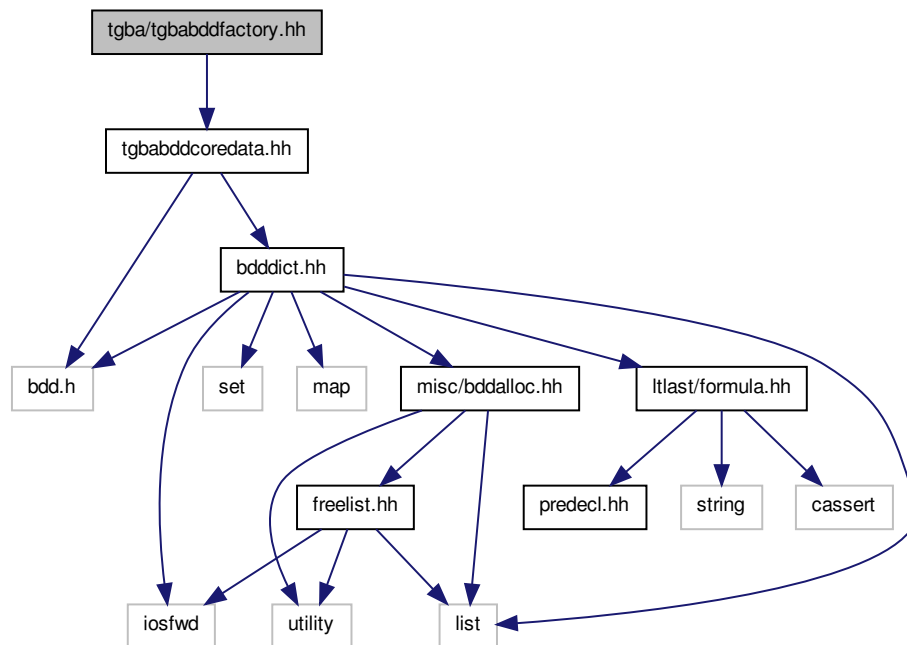
Namespaces

- namespace [spot](#)

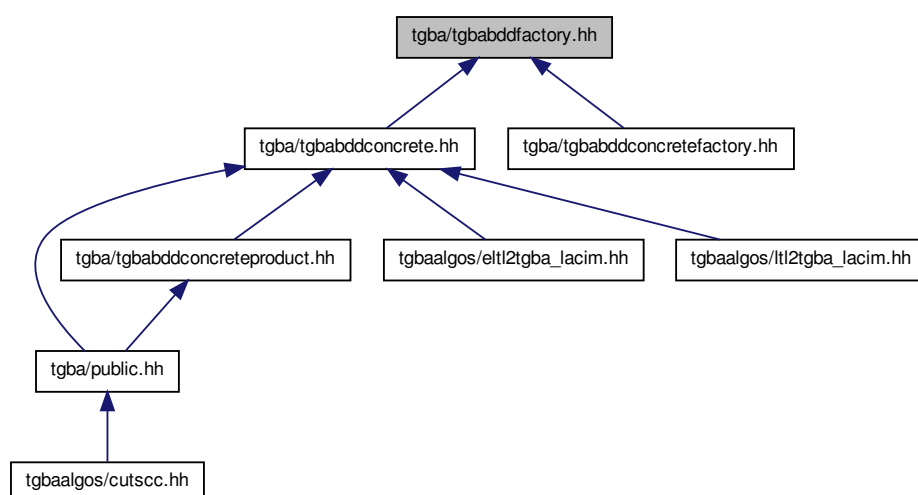
8.110 tgba/tgbabddfactory.hh File Reference

```
#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.

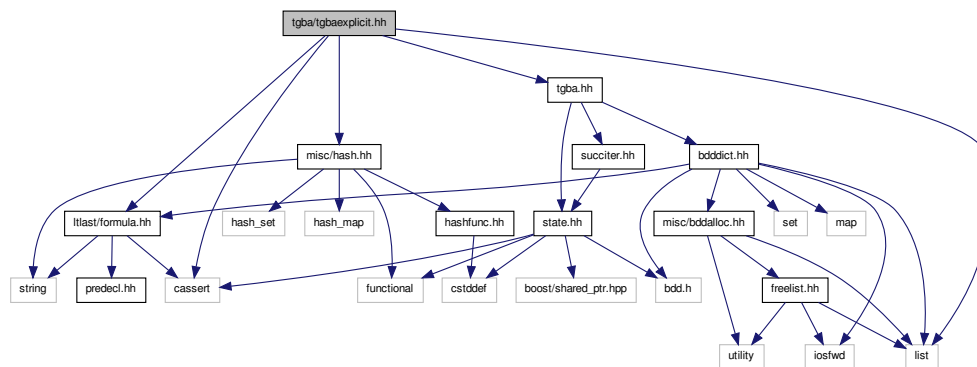
Namespaces

- namespace [spot](#)

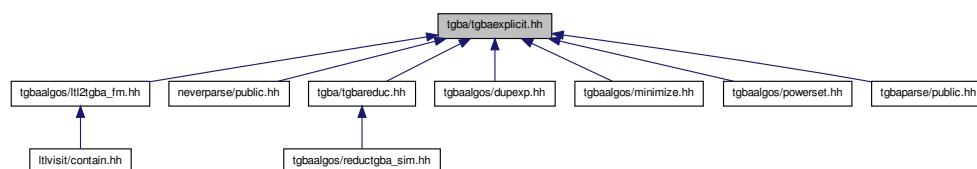
8.111 tgba/tgbaexplicit.hh File Reference

```
#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](#)
- class [spot::tgba_explicit_number](#)

Namespaces

- namespace [spot](#)

8.112 tgba/tgbakvcomplement.hh File Reference

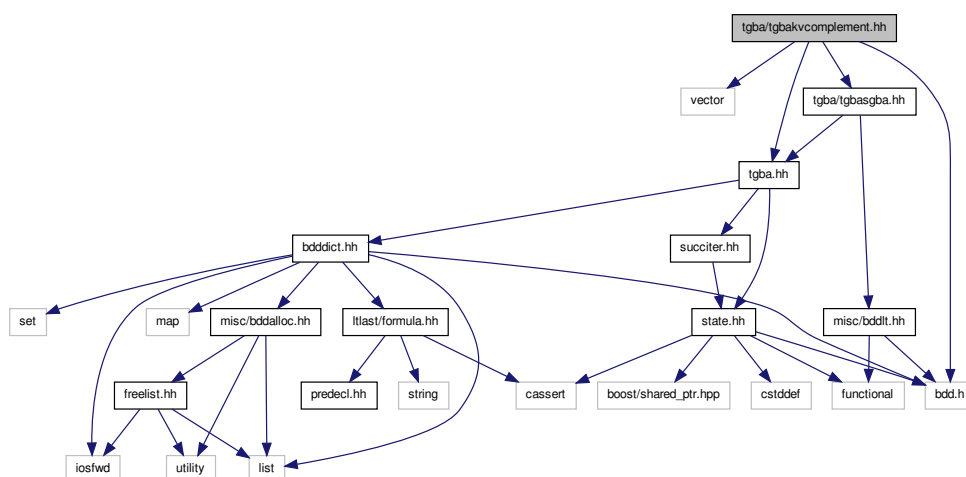
```
#include <vector>
```

```
#include "bdd.h"
```

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

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

Include dependency graph for `tgbakvcomplement.hh`:



Classes

- class [spot::bdd_ordered](#)
- class [spot::tgba_kv_complement](#)
*Build a complemented automaton.
The construction comes from:*

Namespaces

- namespace [spot](#)

Typedefs

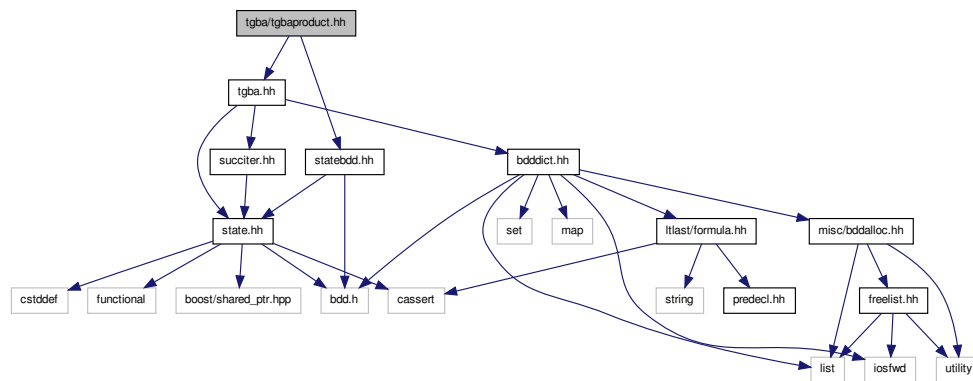
- typedef `std::vector< bdd_ordered >` [spot::acc_list_t](#)

8.113 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.).

- class `spot::tgba_product_init`
A lazy product with different initial states.

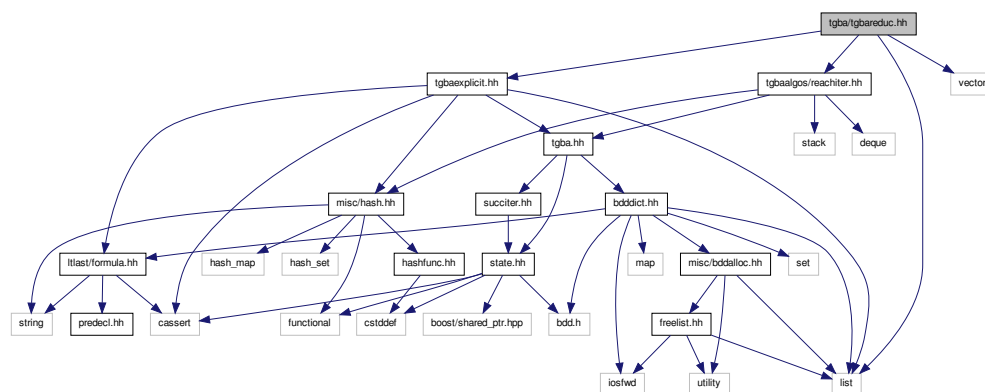
Namespaces

- namespace `spot`

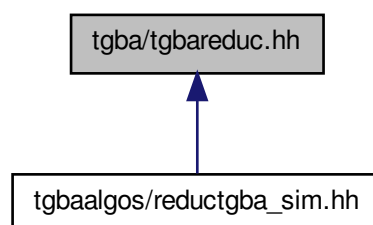
8.114 tgba/tgbareduc.hh File Reference

```
#include "tgbaexplicit.hh"
#include "tgbaalgos/reachiter.hh"
#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

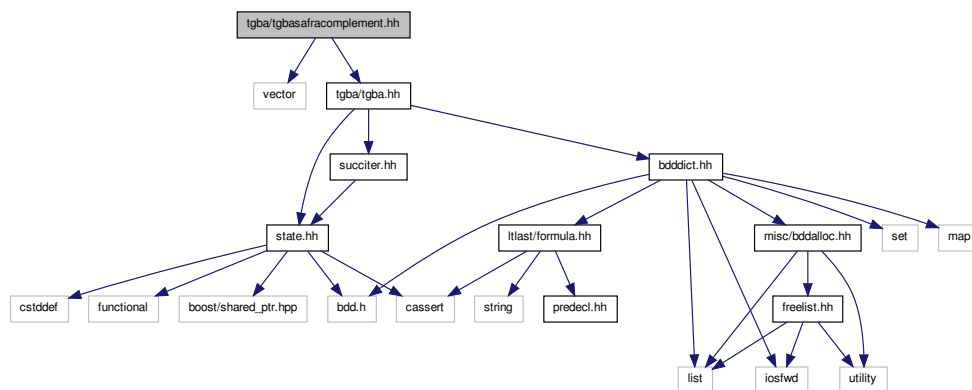
- typedef `std::pair< const spot::state *, const spot::state * >` [spot::state_couple](#)
- typedef `std::vector< state_couple * >` [spot::simulation_relation](#)

8.115 tgba/tgbasafracomplement.hh File Reference

```
#include <vector>
```

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

Include dependency graph for tgba/tgbasafracomplement.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_TBA](#) 0
- #define [TRANSFORM_TO_TGBA](#) (!TRANSFORM_TO_TBA)

Functions

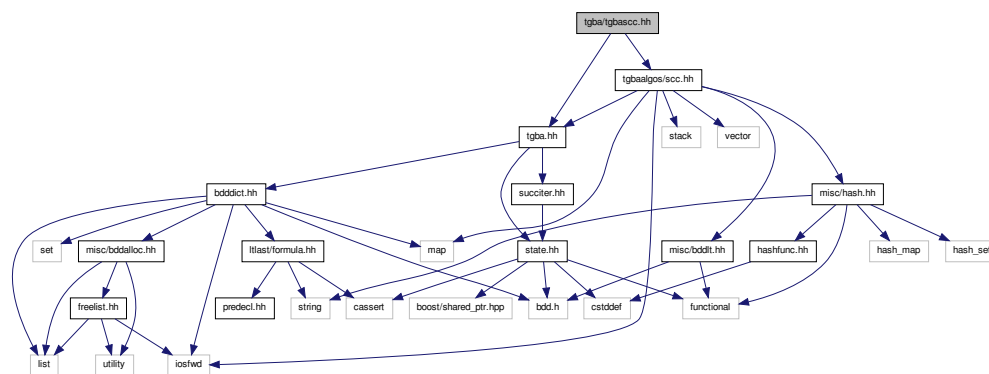
- void [spot::display_safra](#) (const tgba_safra_complement *a)
Produce a dot output of the Safra automaton associated to a.

8.115.1 Define Documentation**8.115.1.1 #define TRANSFORM_TO_TBA 0****8.115.1.2 #define TRANSFORM_TO_TGBA (!TRANSFORM_TO_TBA)****8.116 tgba/tgbascc.hh File Reference**

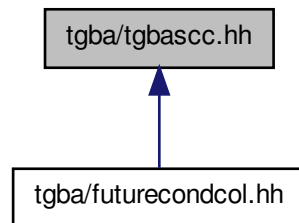
```
#include "tgba.hh"
```

```
#include "tgbaalgos/scc.hh"
```

Include dependency graph for tgbascc.hh:



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



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

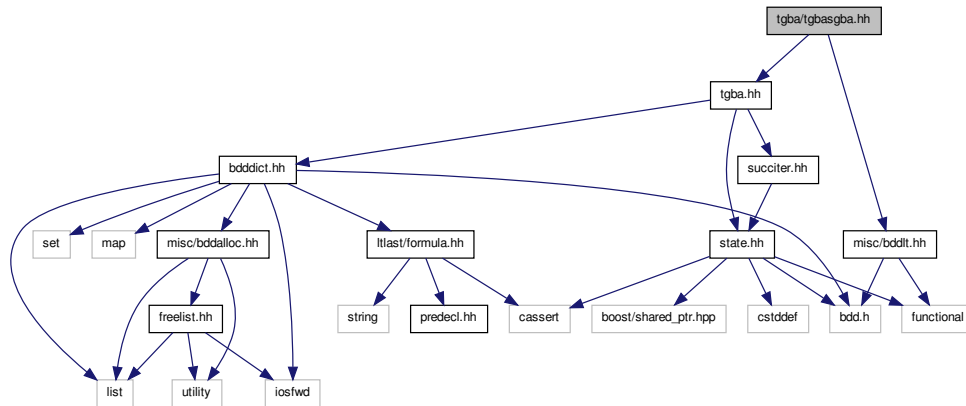
- namespace [spot](#)

8.117 tgba/tgbasgba.hh File Reference

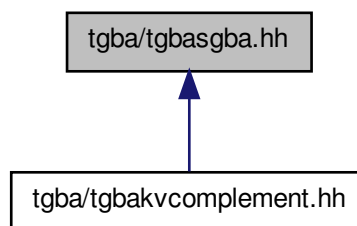
```
#include "tgba.hh"
```

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

Include dependency graph for tgba/tgbasgba.hh:



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



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.

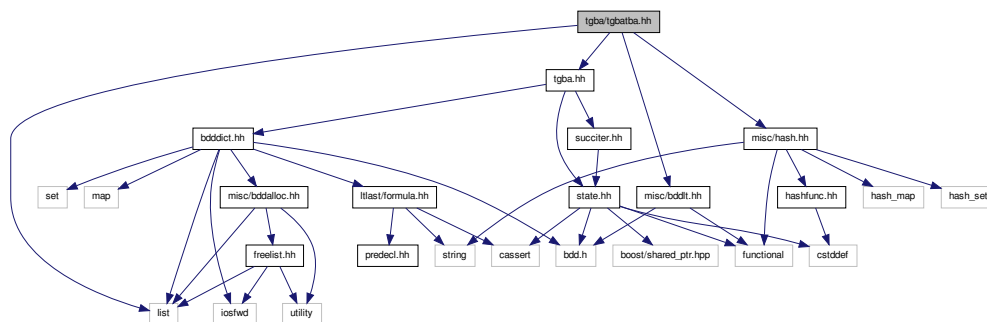
Namespaces

- namespace [spot](#)

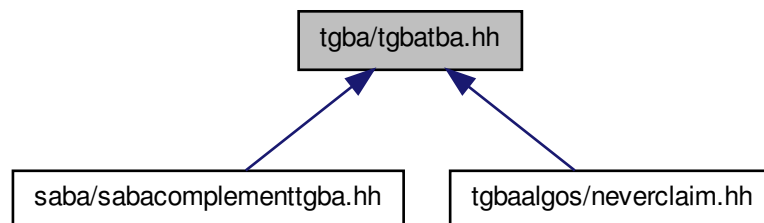
8.118 tgba/tgbatba.hh File Reference

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

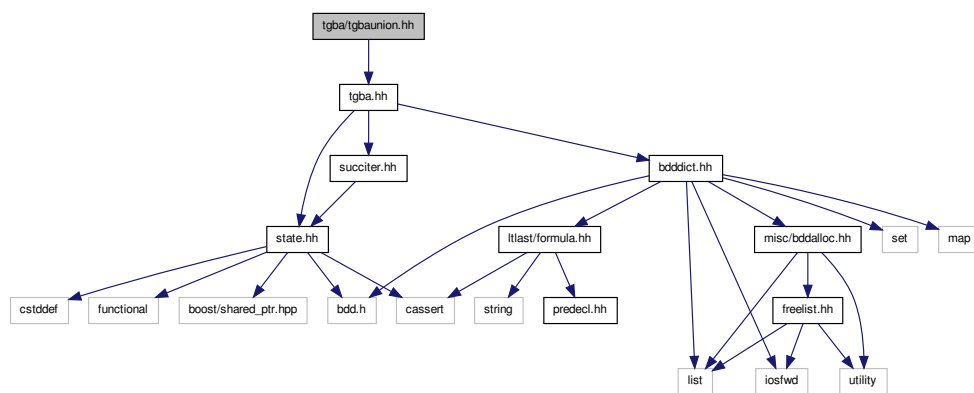
Namespaces

- namespace [spot](#)

8.119 tgba/tgbaunion.hh File Reference

```
#include "tgba.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.).

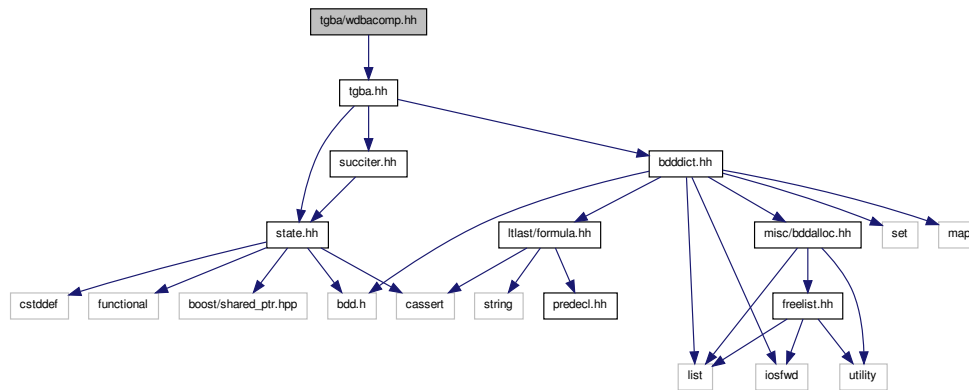
Namespaces

- namespace [spot](#)

8.120 tgba/wdbacomp.hh File Reference

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

Include dependency graph for wdbacomp.hh:



Namespaces

- namespace [spot](#)

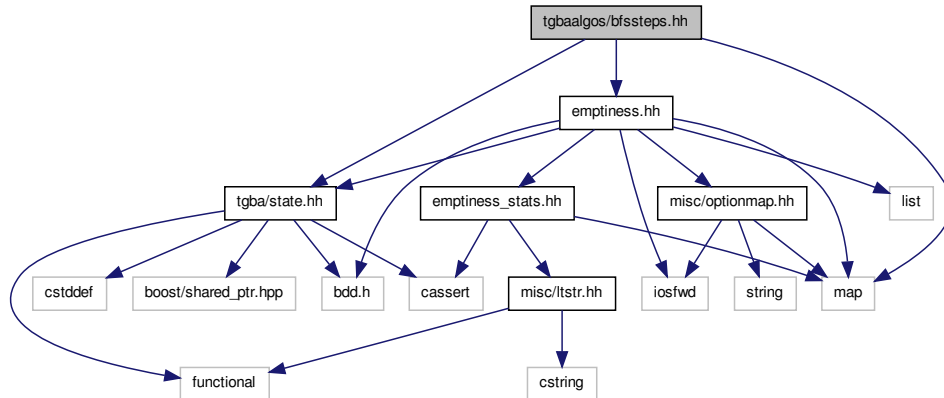
Functions

- `tgba * spot::wdba_complement (const tgba *aut)`
Complement a weak deterministic Büchi automaton.

8.121 tgbaalgos/bfssteps.hh File Reference

```
#include <map>
#include "tgba/state.hh"
#include "emptiness.hh"
```

Include dependency graph for bfssteps.hh:



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](#)

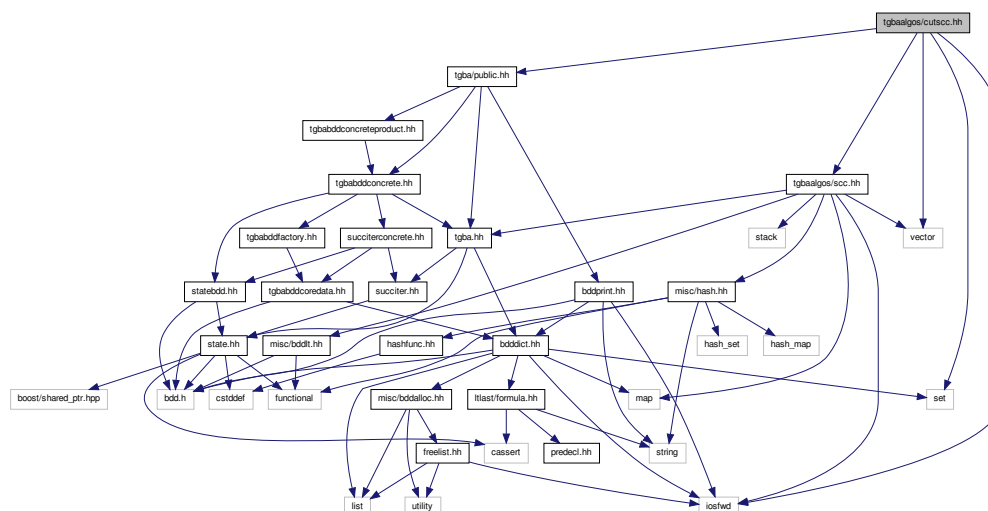
8.122 tgbaalgorithms/cutscc.hh File Reference

```

#include <iosfwd>
#include <set>
#include <vector>
#include "tgba/public.hh"
#include "tgbaalgorithms/scc.hh"

```


Include dependency graph for cutscc.hh:



Classes

- struct [spot::sccs_set](#)

Namespaces

- namespace [spot](#)

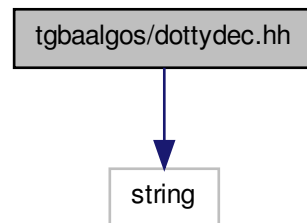
Functions

- `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)`

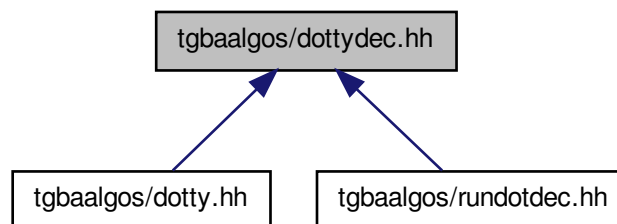
8.123 tgbaalgorithms/dottydec.hh File Reference

```
#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`.

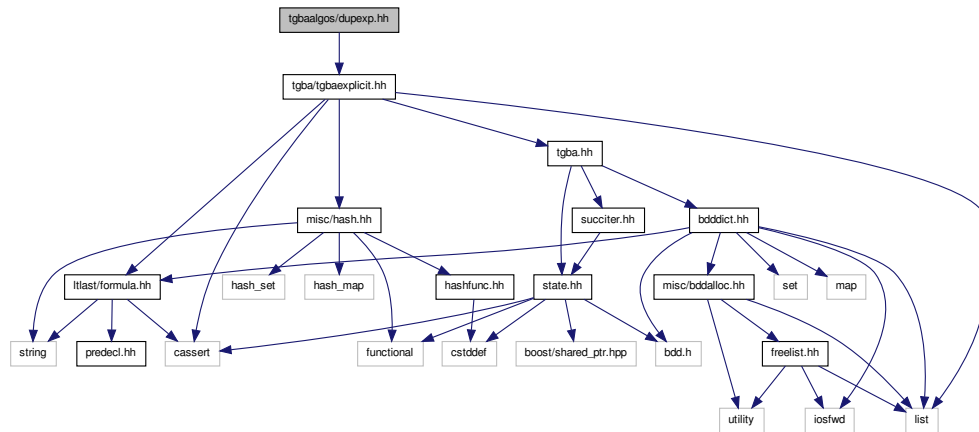
Namespaces

- namespace `spot`

8.124 tgbaalgos/dupeexp.hh File Reference

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

Include dependency graph for dupexp.hh:



Namespaces

- namespace [spot](#)

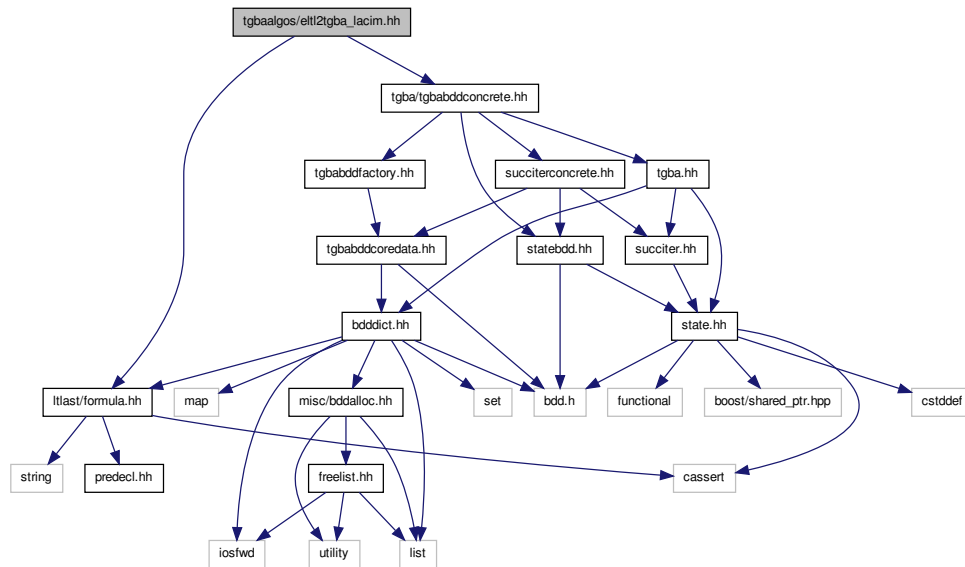
Functions

- 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.

8.125 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.*

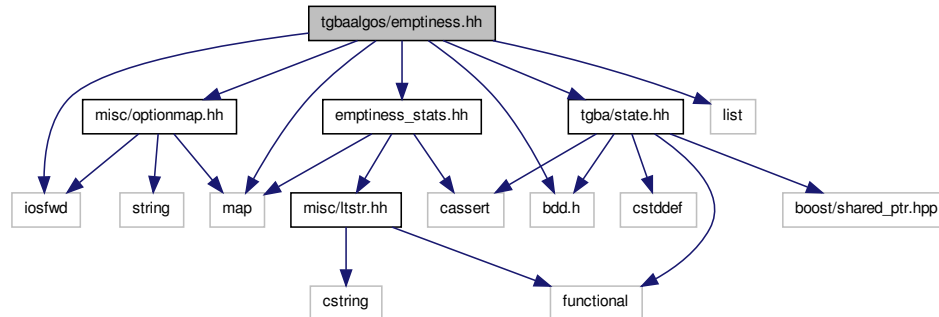
8.126 tgbaalgos/emptiness.hh File Reference

```

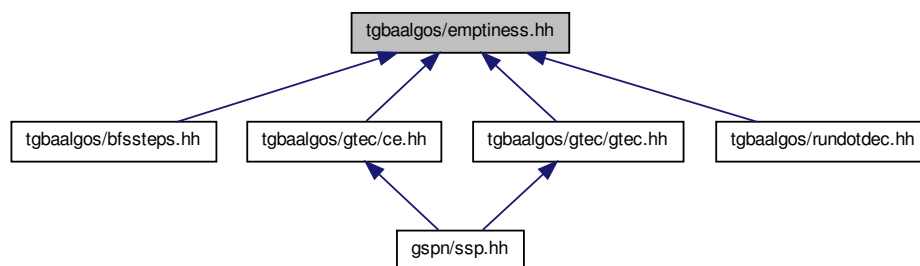
#include <map>
#include <list>
#include <iosfwd>
#include <bdd.h>
#include "misc/optionmap.hh"
#include "tgba/state.hh"
#include "emptiness_stats.hh"

```

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](#)

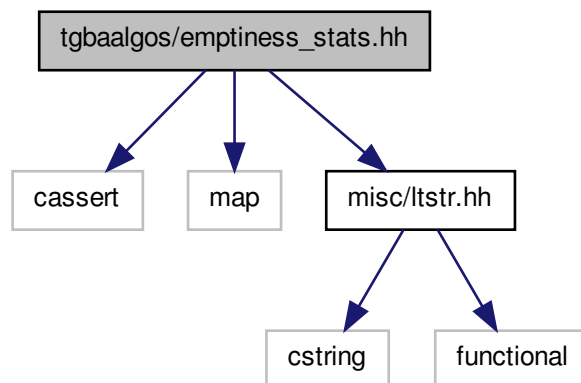
Functions

- `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).

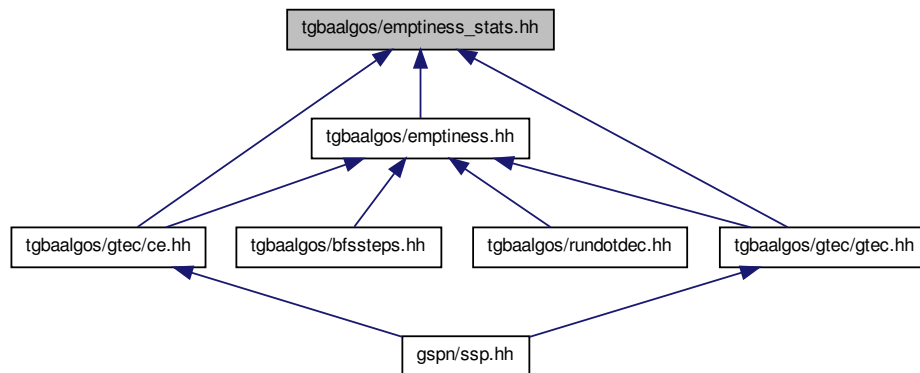
8.127 tgbaalgos/emptiness_stats.hh File Reference

```
#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.

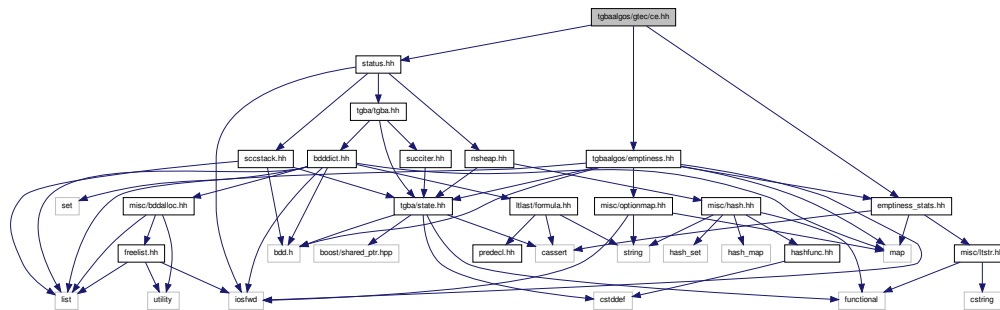
Namespaces

- namespace [spot](#)

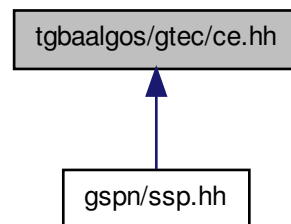
8.128 tgbaalgos/gtec/ce.hh File Reference

```
#include "status.hh"
#include "tgbaalgos/emptiness.hh"
#include "tgbaalgos/emptiness_stats.hh"
```

Include dependency graph for ce.hh:



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



Classes

- class `spot::couvreur99_check_result`
Compute a counter example from a `spot::couvreur99_check_status`.

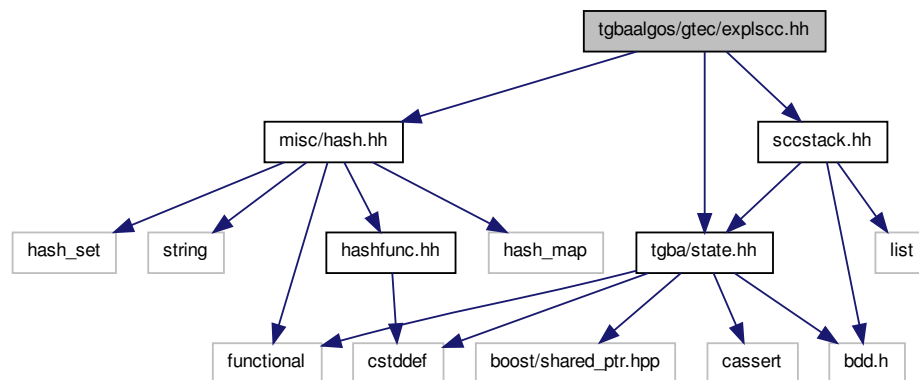
Namespaces

- namespace **spot**

8.129 tgbaalgos/gtec/explscc.hh 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`

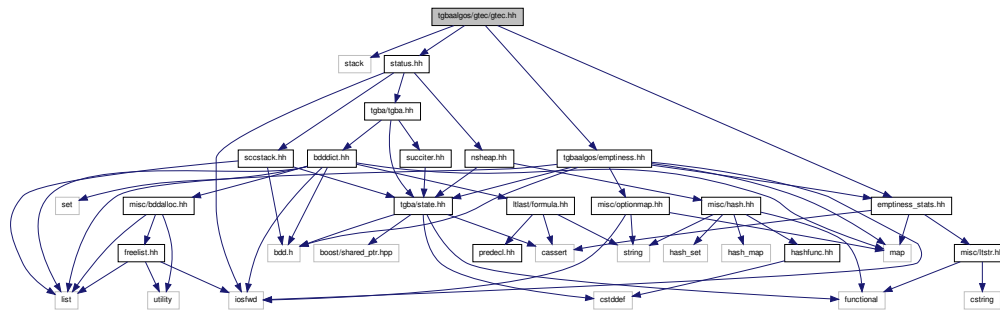
8.130 tgbaalgos/gtec/gtec.hh File Reference

```

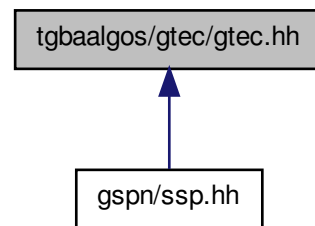
#include <stack>
#include "status.hh"
#include "tgbaalgos/emptiness.hh"
#include "tgbaalgos/emptiness_stats.hh"

```

Include dependency graph for gtec.hh:



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



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())`

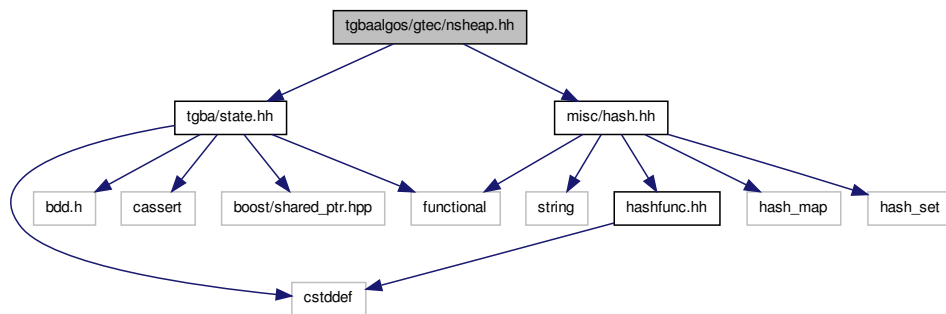
Check whether the language of an automate is empty.

8.131 tgbaalgos/gtec/nsheap.hh File Reference

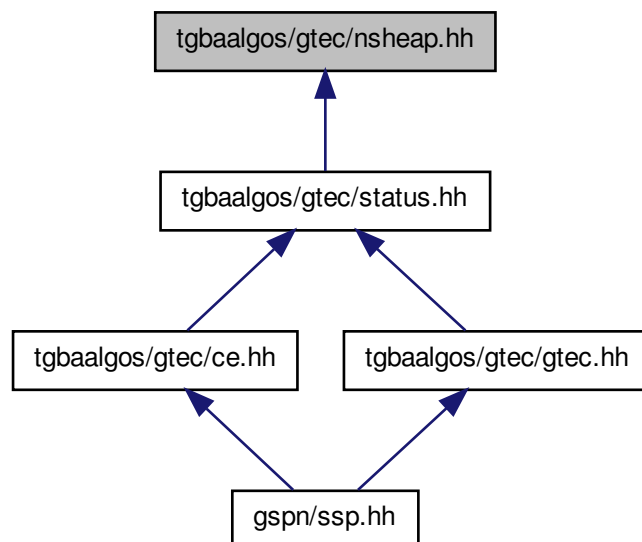
```
#include "tgba/state.hh"
```

```
#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](#)

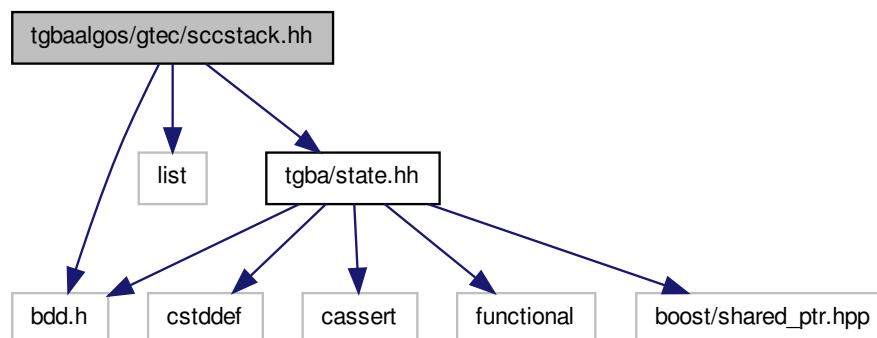
8.132 tgbaalgos/gtec/sccstack.hh File Reference

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

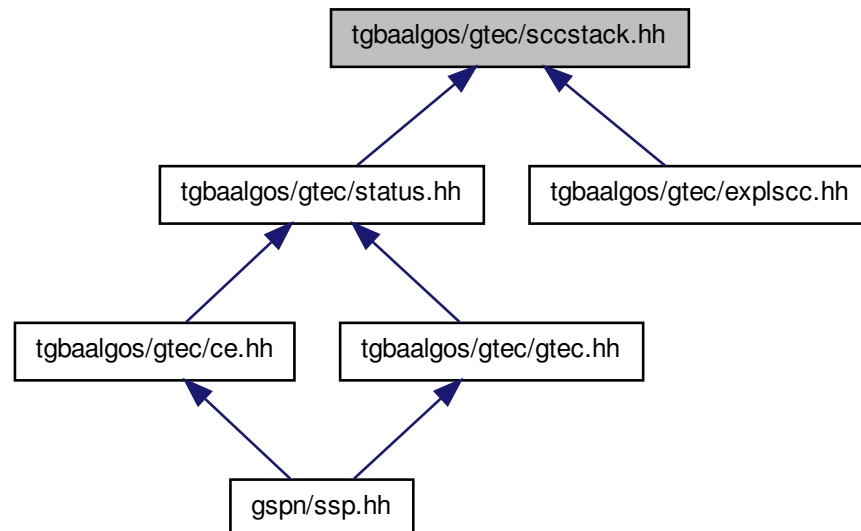
```
#include <list>
```

```
#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](#)

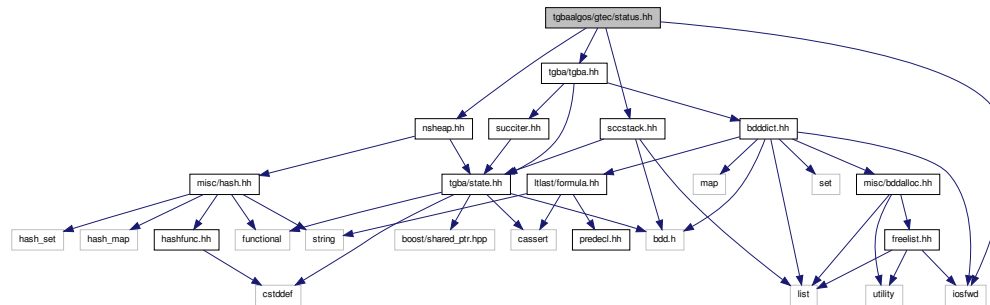
Namespaces

- namespace [spot](#)

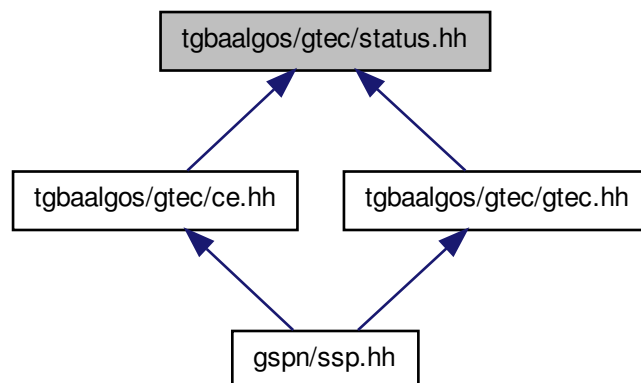
8.133 tgbaalgos/gtec/status.hh File Reference

```
#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.

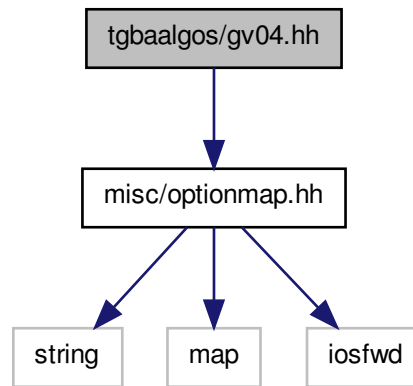
Namespaces

- namespace [spot](#)

8.134 tgbaalgos/gv04.hh File Reference

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

Include dependency graph for gv04.hh:



Namespaces

- namespace [spot](#)

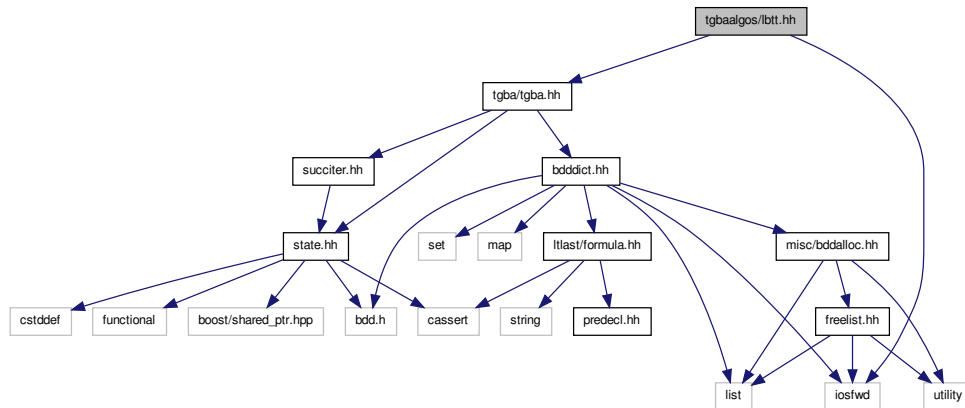
Functions

- 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.

8.135 tgbaalgos/lbtt.hh File Reference

```
#include "tgba/tgba.hh"
#include <iosfwd>
```

Include dependency graph for lbt.h:



Namespaces

- namespace [spot](#)

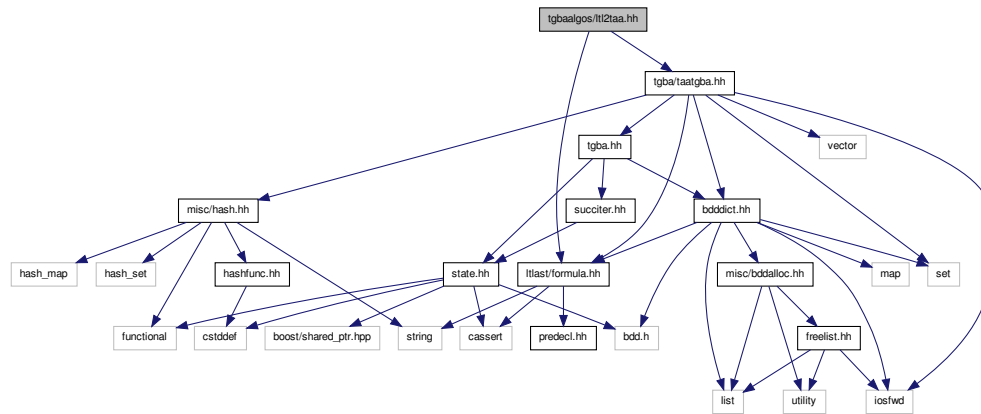
Functions

- `std::ostream & spot::lbt_reachable (std::ostream &os, const tgba *g)`
Print reachable states in LBT format.

8.136 tgbaalgos/ltl2taa.hh File Reference

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


Include dependency graph for ltl2taa.hh:



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.*

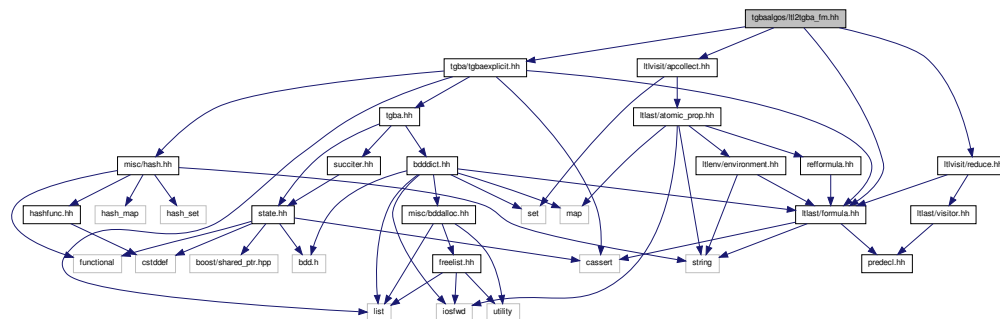
8.137 tgbaalgos/ltl2tgba_fm.hh File Reference

```

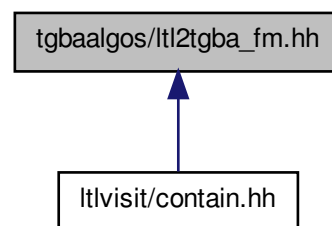
#include "ltlast/formula.hh"
#include "tgba/tgbaexplicit.hh"
#include "ltlvisit/apcollect.hh"
#include "ltlvisit/reduce.hh"

```

Include dependency graph for ltl2tgba_fm.hh:



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



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.*

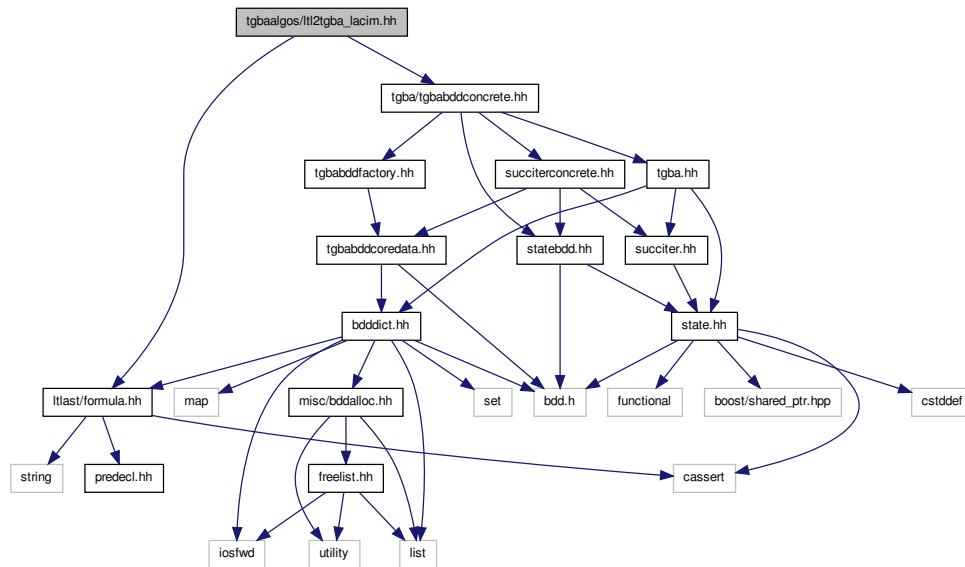
This is based on the following paper.

8.138 tgbaalgos/ltl2tgba_lacim.hh File Reference

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

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

Include dependency graph for `ltl2tgba_lacim.hh`:



Namespaces

- namespace [spot](#)

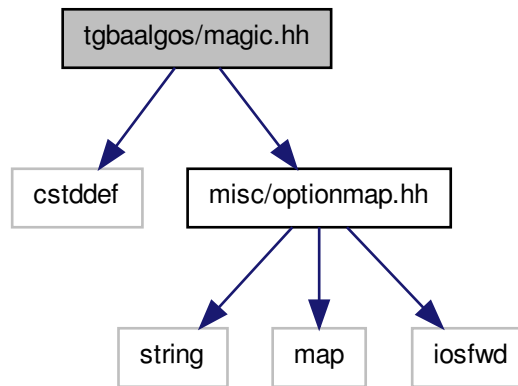
Functions

- `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:*

8.139 tgbaalgos/magic.hh File Reference

```
#include <cstdddef>
#include "misc/optionmap.hh"
```

Include dependency graph for magic.hh:



Namespaces

- namespace [spot](#)

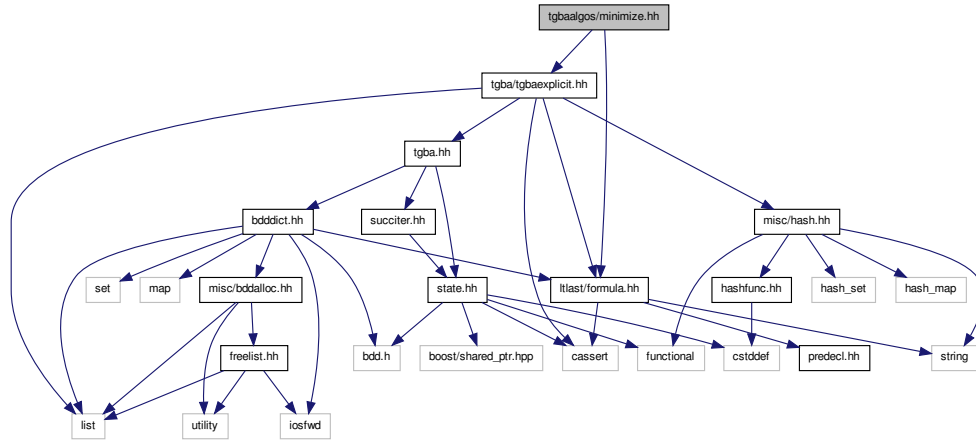
Functions

- 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.

8.140 tgbaalgos/minimize.hh File Reference

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

Include dependency graph for minimize.hh:



Namespaces

- namespace [spot](#)

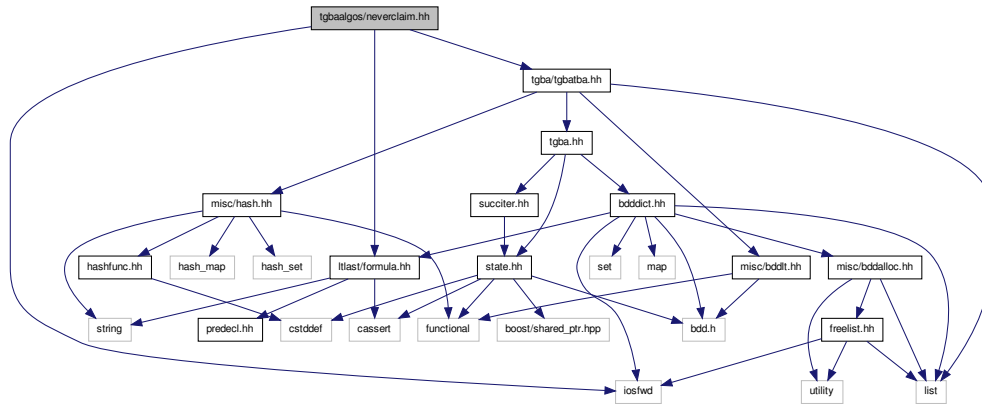
Functions

- tgba_explicit_number * [spot::minimize_monitor](#) (const tgba *a)
Construct a minimal deterministic monitor.
- tgba_explicit_number * [spot::minimize_wdba](#) (const tgba *a)
Minimize a Büchi automaton in the WDBA class.
- const tgba * [spot::minimize_obligation](#) (const tgba *aut_f, const ltlast::formula *f=0, const tgba *aut_neg_f=0)
Minimize an automaton if it represents an obligation property.

8.141 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

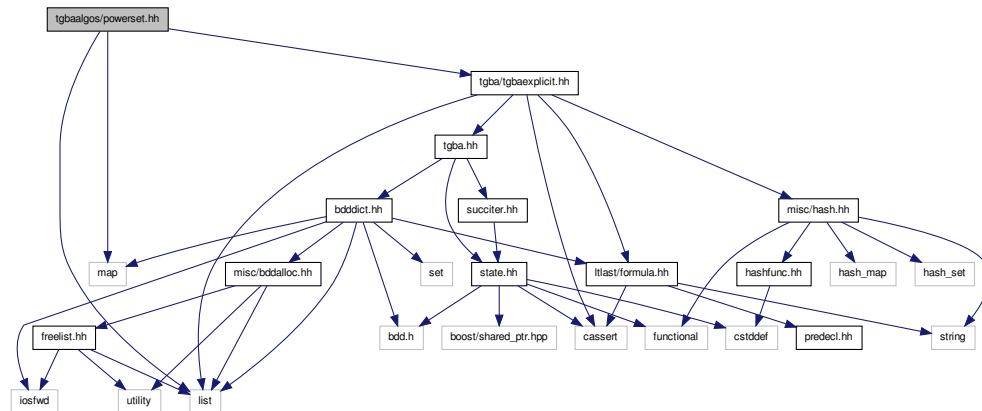
- `std::ostream & spot::never_claim_reachable (std::ostream &os, const tgba_sba_proxy *g, const ltl::formula *f=0, bool comments=false)`

Print reachable states in Spin never claim format.

8.142 tgbaalgos/powerset.hh File Reference

```
#include <list>
#include <map>
#include "tgba/tgbaexplicit.hh"
```

Include dependency graph for powerset.hh:



Classes

- struct [spot::power_map](#)

Namespaces

- namespace [spot](#)

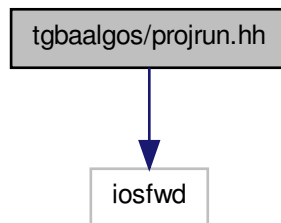
Functions

- tgba_explicit_number * [spot::tgba_powerset](#) (const tgba *aut, power_map &pm)
*Build a deterministic automaton, ignoring acceptance conditions.
This create a deterministic automaton that recognizes the same language as aut would if its acceptance conditions were ignored. This is the classical powerset algorithm.*
- tgba_explicit_number * [spot::tgba_powerset](#) (const tgba *aut)

8.143 tgbaalgos/projrun.hh File Reference

```
#include <iosfwd>
```

Include dependency graph for projrun.hh:



Namespaces

- namespace [spot](#)

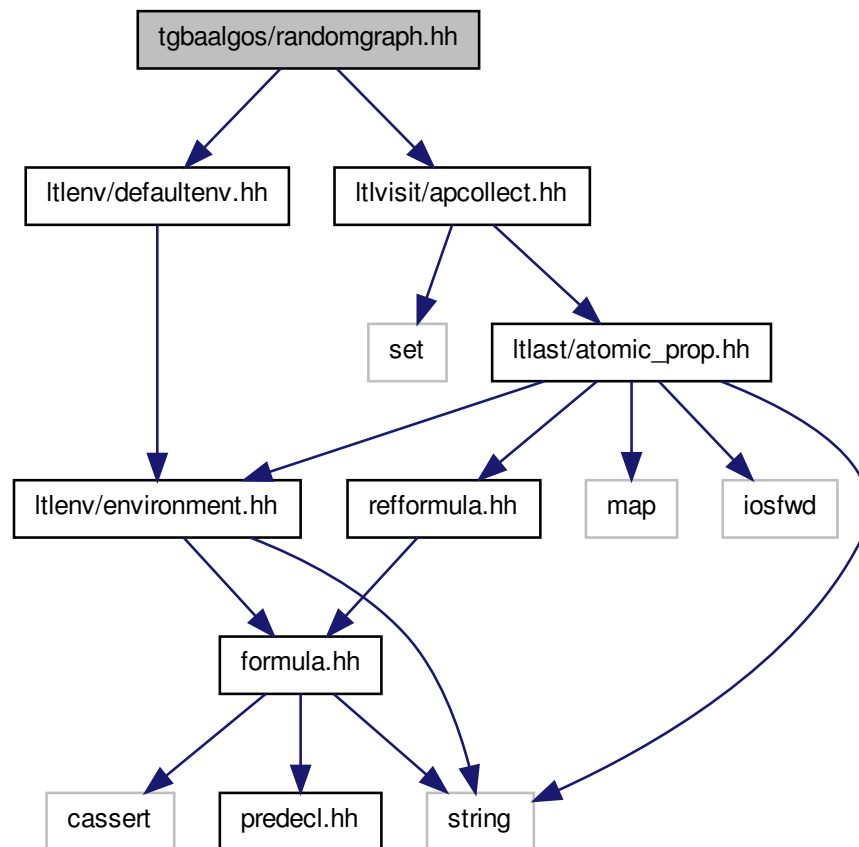
Functions

- `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,.

8.144 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.

8.145 tgbaalgos/reducerun.hh File Reference

Namespaces

- namespace [spot](#)

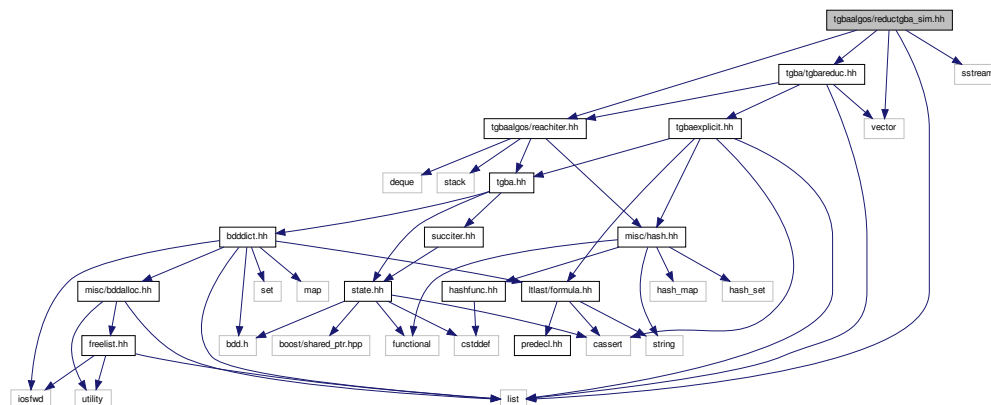
Functions

- `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.

8.146 tgbaalgos/reductgba_sim.hh File Reference

```
#include "tgba/tgbareduc.hh"
#include "tgbaalgos/reachiter.hh"
#include <vector>
#include <list>
#include <sstream>
```

Include dependency graph for reductgba_sim.hh:



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.

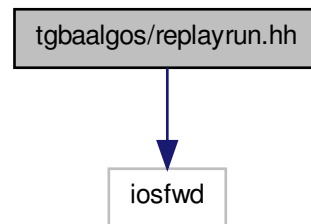
Functions

- 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.

8.147 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)

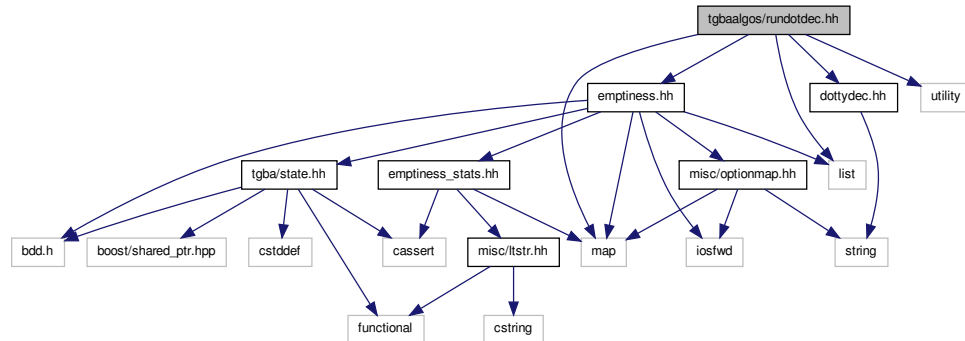
Replay a `tgba_run` on a `tgba`.

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.

8.148 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](#).

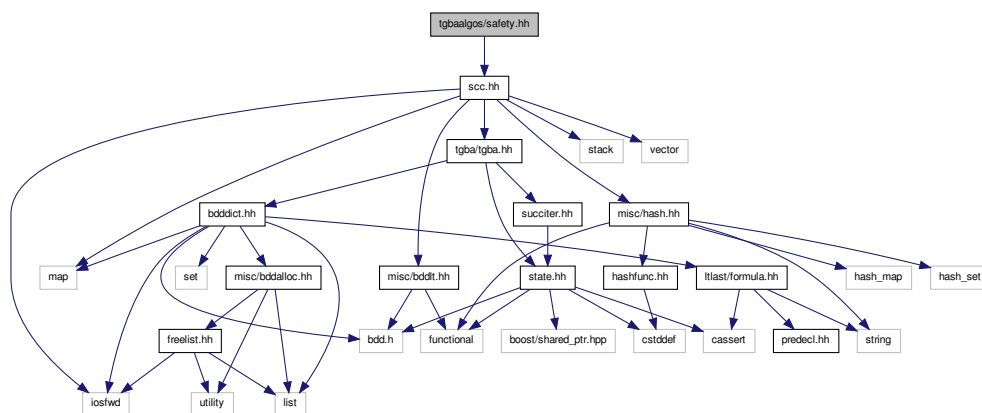
Namespaces

- namespace [spot](#)

8.149 tgbaalgos/safety.hh File Reference

```
#include "scc.hh"
```

Include dependency graph for safety.hh:



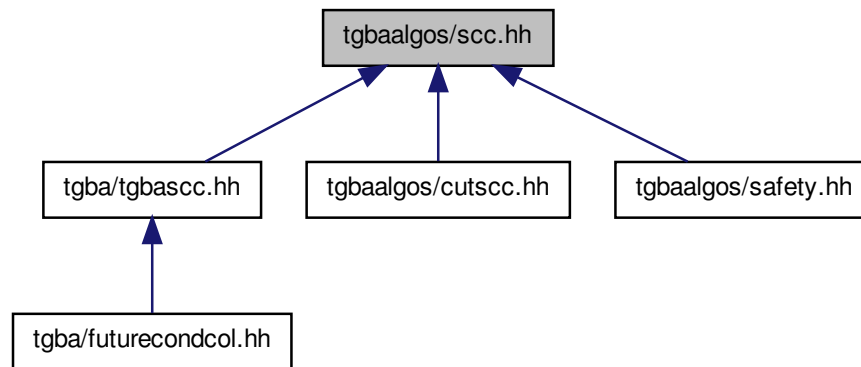
- namespace **spot**

- bool `spot::is_guarantee_automaton` (const tgba *aut, const scc_map *sm=0)

Whether a minimized WDBA represents a safety property.

```
#include <map>
#include <stack>
#include <vector>
#include "tgba/tgba.hh"
#include <iosfwd>
#include "misc/hash.hh"
#include "misc/bddlt.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](#)

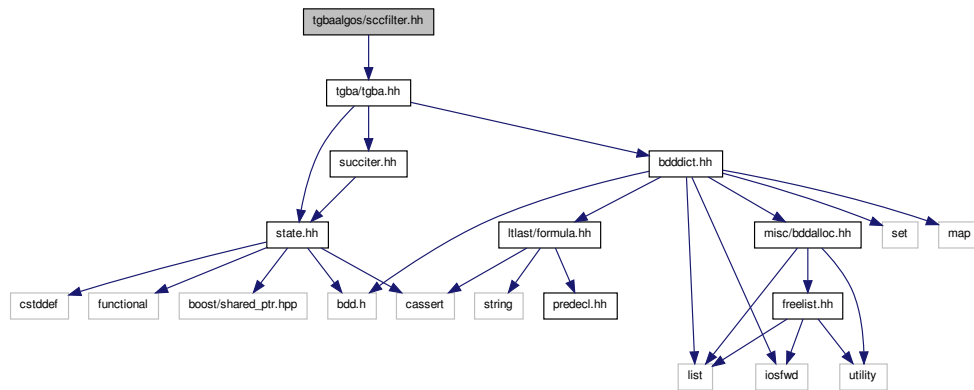
Functions

- 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)

8.151 tgbaalgos/sccfilter.hh File Reference

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

Include dependency graph for sccfilter.hh:



Namespaces

- namespace [spot](#)

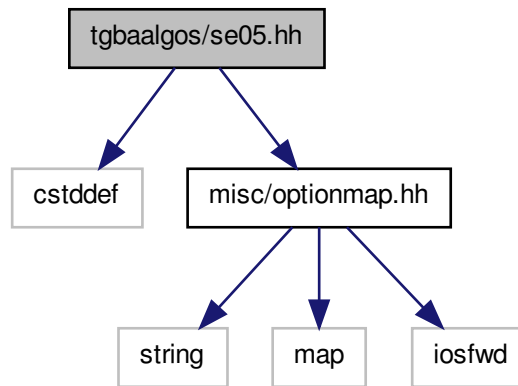
Functions

- `tgba * spot::scc_filter (const tgba *aut, bool remove_all_useless=false)`
Prune unaccepting SCCs and remove superfluous acceptance conditions.

8.152 tgbaalgos/se05.hh File Reference

```
#include <cstdint>
#include "misc/optionmap.hh"
```


Include dependency graph for se05.hh:



Namespaces

- namespace `spot`

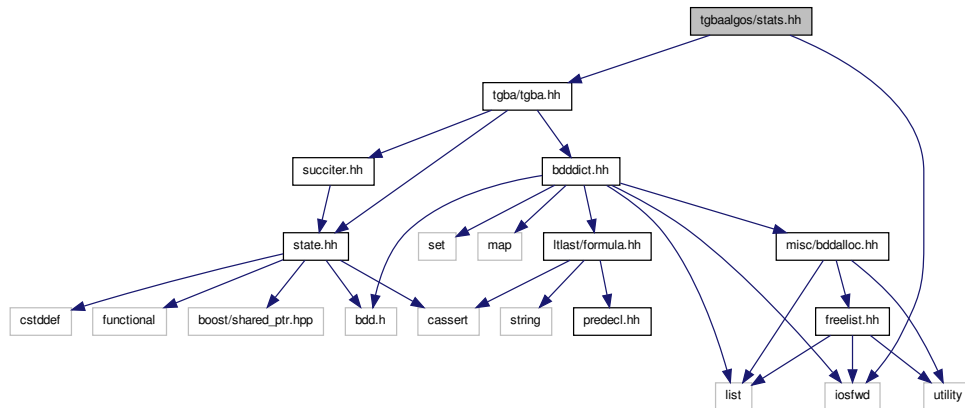
Functions

- 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.

8.153 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](#)

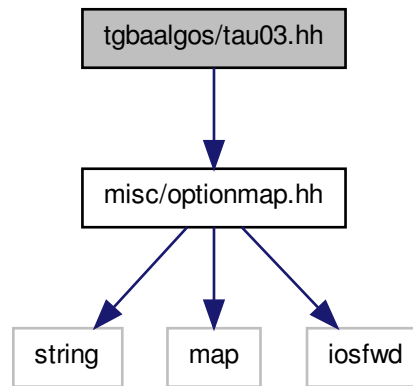
Functions

- tgba_statistics [spot::stats_reachable](#) (const tgba *g)
Compute statistics for an automaton.

8.154 tgbaalgos/tau03.hh File Reference

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

Include dependency graph for tau03.hh:



Namespaces

- namespace [spot](#)

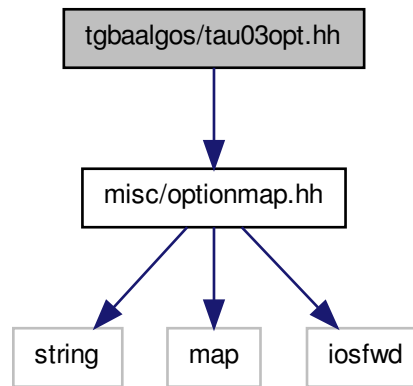
Functions

- emptiness_check * [spot::explicit_tau03_search](#) (const tgba *a, option_map o=option_map())
Returns an emptiness checker on the [spot::tgba](#) automaton a.

8.155 tgbaalgos/tau03opt.hh File Reference

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

Include dependency graph for tau03opt.hh:



Namespaces

- namespace [spot](#)

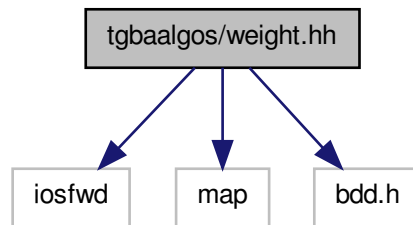
Functions

- 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.

8.156 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, [118](#)
- ~atomic_prop
 - spot::ltl::atomic_prop, [132](#)
- ~automatop
 - spot::ltl::automatop, [140](#)
- ~bdd_dict
 - spot::bdd_dict, [157](#)
- ~bfs_steps
 - spot::bfs_steps, [166](#)
- ~binop
 - spot::ltl::binop, [173](#)
- ~clone_visitor
 - spot::ltl::clone_visitor, [180](#)
- ~connected_component_hash_set
 - spot::connected_component_hash_set, [186](#)
- ~connected_component_hash_set_factory
 - spot::connected_component_hash_set_factory, [190](#)
- ~const_visitor
 - spot::ltl::const_visitor, [191](#)
- ~constant
 - spot::ltl::constant, [195](#)
- ~couvreur99_check
 - spot::couvreur99_check, [201](#)
- ~couvreur99_check_shy
 - spot::couvreur99_check_shy, [218](#)
- ~couvreur99_check_status
 - spot::couvreur99_check_status, [226](#)
- ~declarative_environment
 - spot::ltl::declarative_environment, [230](#)
- ~default_environment
 - spot::ltl::default_environment, [234](#)
- ~dotty_decorator
 - spot::dotty_decorator, [237](#)
- ~duplicator_node
 - spot::duplicator_node, [241](#)
- ~duplicator_node_delayed
 - spot::duplicator_node_delayed, [248](#)
- ~emptiness_check
 - spot::emptiness_check, [260](#)
- ~emptiness_check_result
 - spot::emptiness_check_result, [268](#)
- ~environment
 - spot::ltl::environment, [272](#)
- ~evtgba
 - spot::evtgba, [274](#)
- ~evtgba_explicit
 - spot::evtgba_explicit, [280](#)
- ~evtgba_iterator
 - spot::evtgba_iterator, [283](#)
- ~evtgba_product
 - spot::evtgba_product, [286](#)
- ~evtgba_reachable_iterator
 - spot::evtgba_reachable_iterator, [292](#)
- ~explicit_connected_component
 - spot::explicit_connected_component, [306](#)
- ~explicit_connected_component_factory
 - spot::explicit_connected_component_factory, [307](#)
- ~explicit_state_conjunction
 - spot::explicit_state_conjunction, [312](#)
- ~fair_kripke_succ_iterator
 - spot::fair_kripke_succ_iterator, [324](#)
- ~formula
 - spot::ltl::formula, [328](#)
- ~free_list
 - spot::free_list, [335](#)
- ~future_conditions_collector
 - spot::future_conditions_collector, [340](#)
- ~gspn_interface
 - spot::gspn_interface, [348](#)
- ~gspn_ssp_interface
 - spot::gspn_ssp_interface, [351](#)
- ~kripke
 - spot::kripke, [356](#)
- ~kripke_succ_iterator
 - spot::kripke_succ_iterator, [363](#)
- ~language_containment_checker
 - spot::ltl::language_containment_checker, [368](#)
- ~loopless_modular_mixed_radix_gray_code
 - spot::loopless_modular_mixed_radix_gray_code, [385](#)
- ~multop
 - spot::ltl::multop, [395](#)
- ~nfa
 - spot::ltl::nfa, [401](#)
- ~nips_interface
 - spot::nips_interface, [407](#)
- ~node
 - spot::ltl::formula_tree::node, [408](#)
- ~numbered_state_heap
 - spot::numbered_state_heap, [419](#)
- ~numbered_state_heap_const_iterator
 - spot::numbered_state_heap_const_iterator, [421](#)
- ~numbered_state_heap_factory
 - spot::numbered_state_heap_factory, [423](#)
- ~numbered_state_heap_hash_map
 - spot::numbered_state_heap_hash_map, [428](#)

- ~numbered_state_heap_hash_map_factory
 - spot::numbered_state_heap_hash_map_factory, [432](#)
- ~parity_game_graph
 - spot::parity_game_graph, [441](#)
- ~parity_game_graph_delayed
 - spot::parity_game_graph_delayed, [449](#)
- ~parity_game_graph_direct
 - spot::parity_game_graph_direct, [456](#)
- ~postfix_visitor
 - spot::ltl::postfix_visitor, [470](#)
- ~power_map
 - spot::power_map, [472](#)
- ~random_ltl
 - spot::ltl::random_ltl, [477](#)
- ~ref_formula
 - spot::ltl::ref_formula, [485](#)
- ~rsymbol
 - spot::rsymbol, [489](#)
- ~saba
 - spot::saba, [492](#)
- ~saba_complement_tgba
 - spot::saba_complement_tgba, [498](#)
- ~saba_reachable_iterator
 - spot::saba_reachable_iterator, [503](#)
- ~saba_state
 - spot::saba_state, [517](#)
- ~saba_state_conjunction
 - spot::saba_state_conjunction, [521](#)
- ~saba_succ_iterator
 - spot::saba_succ_iterator, [527](#)
- ~scc_map
 - spot::scc_map, [534](#)
- ~simplify_f_g_visitor
 - spot::ltl::simplify_f_g_visitor, [546](#)
- ~spoiler_node
 - spot::spoiler_node, [554](#)
- ~spoiler_node_delayed
 - spot::spoiler_node_delayed, [559](#)
- ~state
 - spot::state, [573](#)
- ~state_evtgba_explicit
 - spot::state_evtgba_explicit, [581](#)
- ~state_explicit
 - spot::state_explicit, [585](#)
- ~state_product
 - spot::state_product, [589](#)
- ~state_set
 - spot::state_set, [596](#)
- ~state_union
 - spot::state_union, [602](#)
- ~symbol
 - spot::symbol, [610](#)
- ~taa_succ_iterator
 - spot::taa_succ_iterator, [615](#)
- ~taa_tgba
 - spot::taa_tgba, [622](#)
- ~taa_tgba_formula
 - spot::taa_tgba_formula, [631](#)
- ~taa_tgba_string
 - spot::taa_tgba_string, [654](#)
- ~tgba
 - spot::tgba, [664](#)
- ~tgba_bdd_concrete
 - spot::tgba_bdd_concrete, [672](#)
- ~tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, [680](#)
- ~tgba_bdd_factory
 - spot::tgba_bdd_factory, [692](#)
- ~tgba_explicit
 - spot::tgba_explicit, [697](#)
- ~tgba_explicit_formula
 - spot::tgba_explicit_formula, [707](#)
- ~tgba_explicit_labelled
 - spot::tgba_explicit_labelled, [719](#)
- ~tgba_explicit_number
 - spot::tgba_explicit_number, [731](#)
- ~tgba_explicit_string
 - spot::tgba_explicit_string, [743](#)
- ~tgba_kv_complement
 - spot::tgba_kv_complement, [759](#)
- ~tgba_product
 - spot::tgba_product, [767](#)
- ~tgba_reachable_iterator
 - spot::tgba_reachable_iterator, [782](#)
- ~tgba_reduc
 - spot::tgba_reduc, [803](#)
- ~tgba_run
 - spot::tgba_run, [815](#)
- ~tgba_run_dotty_decorator
 - spot::tgba_run_dotty_decorator, [818](#)
- ~tgba_safra_complement
 - spot::tgba_safra_complement, [824](#)
- ~tgba_scc
 - spot::tgba_scc, [840](#)
- ~tgba_sgba_proxy
 - spot::tgba_sgba_proxy, [848](#)
- ~tgba_succ_iterator
 - spot::tgba_succ_iterator, [854](#)
- ~tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, [859](#)
- ~tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, [865](#)
- ~tgba_succ_iterator_union
 - spot::tgba_succ_iterator_union, [870](#)
- ~tgba_tba_proxy
 - spot::tgba_tba_proxy, [878](#)
- ~tgba_union

- spot::tgba_union, 886
- ~unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, 907
- ~unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, 912
- ~unop
 - spot::ltl::unop, 919
- ~unsigned_statistics
 - spot::unsigned_statistics, 924
- ~visitor
 - spot::ltl::visitor, 928
- a_
 - spot::bfs_steps, 167
 - spot::couvreur99_check, 205
 - spot::couvreur99_check_result, 212
 - spot::couvreur99_check_shy, 222
 - spot::emptiness_check, 262
 - spot::emptiness_check_result, 270
 - spot::loopless_modular_mixed_radix_gray_-code, 386
 - spot::tgba_sba_proxy, 836
 - spot::tgba_sgba_proxy, 852
 - spot::tgba_tba_proxy, 881
- a_first
 - spot::loopless_modular_mixed_radix_gray_-code, 385
- a_last
 - spot::loopless_modular_mixed_radix_gray_-code, 385
- a_next
 - spot::loopless_modular_mixed_radix_gray_-code, 385
- acc
 - spot::couvreur99_check_shy::successor, 609
 - spot::scc_map::scc, 529
 - spot::tgba_run::step, 605
- acc_
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 250
 - spot::tgba_bdd_concrete_factory, 682
- acc_cond_
 - spot::fair_kripke_succ_iterator, 326
- acc_cycle_
 - spot::tgba_sba_proxy, 836
 - spot::tgba_tba_proxy, 881
- acc_formula_map
 - spot::bdd_dict, 161
- acc_list_
 - spot::tgba_kv_complement, 763
- acc_list_t
 - spot, 100
- acc_map
 - spot::bdd_dict, 161
- acc_map_
 - spot::tgba_bdd_concrete_factory, 680
- acc_paths
 - spot::scc_stats, 541
- acc_scc
 - spot::scc_stats, 541
- acc_set
 - spot::tgba_bdd_core_data, 687
- acc_set_
 - spot::evtgba_explicit, 282
- acc_set_of
 - spot::scc_map, 534
- accept
 - spot::ltl::atomic_prop, 133
 - spot::ltl::automatop, 140
 - spot::ltl::binop, 174
 - spot::ltl::constant, 195
 - spot::ltl::formula, 329
 - spot::ltl::multop, 395
 - spot::ltl::ref_formula, 486
 - spot::ltl::unop, 919
- acceptance_cond_vec_
 - spot::tgba_safra_complement, 828
- acceptance_condition_
 - spot::tgba_sgba_proxy, 852
- acceptance_condition_visited_
 - spot::spoiler_node_delayed, 561
- acceptance_conditions
 - spot::evtgba_explicit::transition, 903
 - spot::saba_state, 518
 - spot::taa_tgba::transition, 900
 - spot::tgba_bdd_core_data, 687
 - spot::tgba_explicit::transition, 901
- acceptance_conditions_support
 - spot::tgba_bdd_core_data, 688
- accepting
 - spot::scc_map, 534
- accepting_cycle
 - spot::couvreur99_check_result, 210
- accepting_run
 - spot::couvreur99_check_result, 210
 - spot::emptiness_check_result, 269
- accmap_
 - spot::tgba_tba_proxy, 881
- accmap_t
 - spot::tgba_tba_proxy, 877
- acss_states
 - spot::acss_statistics, 118
 - spot::couvreur99_check_result, 210
- acss_statistics
 - spot::acss_statistics, 118
- add
 - spot::explicit_state_conjunction, 312
- add_acceptance_condition

- spot::taa_tgba_formula, 631
- spot::taa_tgba_labelled, 642
- spot::taa_tgba_string, 654
- spot::tgba_explicit, 697
- spot::tgba_explicit_formula, 707
- spot::tgba_explicit_labelled, 719
- spot::tgba_explicit_number, 731
- spot::tgba_explicit_string, 743
- spot::tgba_reduc, 803
- add_acceptance_conditions
 - spot::tgba_explicit, 697
 - spot::tgba_explicit_formula, 707
 - spot::tgba_explicit_labelled, 719
 - spot::tgba_explicit_number, 731
 - spot::tgba_explicit_string, 743
 - spot::tgba_reduc, 803
- add_condition
 - spot::taa_tgba, 622
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 654
 - spot::tgba_explicit, 697
 - spot::tgba_explicit_formula, 707
 - spot::tgba_explicit_labelled, 720
 - spot::tgba_explicit_number, 731
 - spot::tgba_explicit_string, 743
 - spot::tgba_reduc, 803
- add_conditions
 - spot::tgba_explicit, 697
 - spot::tgba_explicit_formula, 708
 - spot::tgba_explicit_labelled, 720
 - spot::tgba_explicit_number, 732
 - spot::tgba_explicit_string, 744
 - spot::tgba_reduc, 803
- add_default_init
 - spot::tgba_explicit, 697
 - spot::tgba_explicit_formula, 708
 - spot::tgba_explicit_labelled, 720
 - spot::tgba_explicit_number, 732
 - spot::tgba_explicit_string, 744
 - spot::tgba_reduc, 803
- add_duplicator_node_delayed
 - spot::parity_game_graph_delayed, 449
- add_pred
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- add_spoiler_node_delayed
 - spot::parity_game_graph_delayed, 449
- add_state
 - spot::evtgba_reachable_iterator, 292
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::ltl::nfa, 401
 - spot::parity_game_graph, 441
 - spot::parity_game_graph_delayed, 449
 - spot::parity_game_graph_direct, 456
 - spot::saba_reachable_iterator, 503
 - spot::saba_reachable_iterator_breadth_first, 509
 - spot::saba_reachable_iterator_depth_first, 515
 - spot::taa_tgba_labelled, 643
 - spot::tgba_explicit_formula, 708
 - spot::tgba_explicit_labelled, 720
 - spot::tgba_explicit_number, 732
 - spot::tgba_explicit_string, 744
 - spot::tgba_reachable_iterator, 782
 - spot::tgba_reachable_iterator_breadth_first, 788
 - spot::tgba_reachable_iterator_depth_first, 794
 - spot::tgba_reduc, 803, 804
- add_state_alias
 - spot::tgba_explicit_string, 744
 - spot::tgba_reduc, 804
- add_state_set
 - spot::taa_tgba_labelled, 643
- add_succ
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- add_transition
 - spot::evtgba_explicit, 280
 - spot::ltl::nfa, 401
- Algorithm patterns, 35
- Algorithms for LTL formulae, 12
- all_acc_
 - spot::evtgba_product, 288
- all_acceptance_cond_
 - spot::tgba_safra_complement, 828
- all_acceptance_conditions
 - spot::evtgba, 274
 - spot::evtgba_explicit, 280
 - spot::evtgba_product, 287
 - spot::fair_kripke, 317
 - spot::future_conditions_collector, 341
 - spot::kripke, 356
 - spot::saba, 492
 - spot::saba_complement_tgba, 498
 - spot::taa_tgba, 622
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 643
 - spot::taa_tgba_string, 654
 - spot::tgba, 664
 - spot::tgba_bdd_concrete, 673

- spot::tgba_bdd_core_data, 688
- spot::tgba_explicit, 697
- spot::tgba_explicit_formula, 708
- spot::tgba_explicit_labelled, 720
- spot::tgba_explicit_number, 732
- spot::tgba_explicit_string, 744
- spot::tgba_kv_complement, 759
- spot::tgba_product, 767
- spot::tgba_product_init, 775
- spot::tgba_reduc, 804
- spot::tgba_safra_complement, 824
- spot::tgba_sba_proxy, 832
- spot::tgba_scc, 840
- spot::tgba_sgba_proxy, 848
- spot::tgba_tba_proxy, 878
- spot::tgba_union, 887
- all_acceptance_conditions_
 - spot::taa_succ_iterator, 616
 - spot::taa_tgba, 625
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 659
 - spot::tgba_explicit, 701
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 725
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 749
 - spot::tgba_explicit_succ_iterator, 755
 - spot::tgba_product, 771
 - spot::tgba_reduc, 813
 - spot::tgba_union, 890
- all_acceptance_conditions_computed_
 - spot::taa_tgba, 625
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 659
 - spot::tgba_explicit, 701
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 725
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 749
 - spot::tgba_reduc, 813
- allocate_variables
 - spot::bdd_allocator, 149
 - spot::bdd_dict, 157
- alphabet
 - spot::evtgba, 274
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- alphabet_
 - spot::evtgba_explicit, 282
 - spot::evtgba_product, 288
- And
 - spot::ltl::multop, 395
- anon_free_list
 - spot::bdd_dict::anon_free_list, 123
- ap
 - spot::ltl::random_ltl, 477
- ap_
 - spot::ltl::random_ltl, 478
- ap_set_of
 - spot::scc_map, 535
- aprec_set_of
 - spot::scc_map, 535
- arc
 - spot::couvereur99_check_shy, 222
- arc_acc_
 - spot::scc_map, 538
- arc_cond_
 - spot::scc_map, 538
- arity
 - spot::ltl::formula_tree, 115
 - spot::ltl::nfa, 401
- arity_
 - spot::ltl::nfa, 403
- ars_cycle_states
 - spot::acss_statistics, 119
 - spot::ars_statistics, 127
 - spot::couvereur99_check_result, 211
- ars_prefix_states
 - spot::acss_statistics, 119
 - spot::ars_statistics, 127
 - spot::couvereur99_check_result, 211
- ars_statistics
 - spot::ars_statistics, 127
- as_bdd
 - spot::state_bdd, 577
- assert_emptiness
 - spot::bdd_dict, 157
- atomic_prop
 - spot::ltl::atomic_prop, 132
- atomic_prop_collect
 - ltl_misc, 16
- atomic_prop_set
 - ltl_misc, 16
- aut
 - spot::couvereur99_check_status, 226
- aut_
 - spot::future_conditions_collector, 344
 - spot::scc_map, 538
 - spot::tgba_scc, 843
- automata_
 - spot::evtgba_reachable_iterator, 293
 - spot::evtgba_reachable_iterator_breadth_first, 298
 - spot::evtgba_reachable_iterator_depth_first, 303
 - spot::parity_game_graph, 443

- spot::parity_game_graph_delayed, 452
- spot::parity_game_graph_direct, 458
- spot::saba_reachable_iterator, 505
- spot::saba_reachable_iterator_breadth_first, 510
- spot::saba_reachable_iterator_depth_first, 516
- spot::tgba_reachable_iterator, 783
- spot::tgba_reachable_iterator_breadth_first, 789
- spot::tgba_reachable_iterator_depth_first, 795
- spot::tgba_reduc, 813
- automaton
 - spot::couvreur99_check, 202
 - spot::couvreur99_check_result, 211
 - spot::couvreur99_check_shy, 219
 - spot::emptiness_check, 261
 - spot::emptiness_check_result, 269
 - spot::gspn_interface, 348
 - spot::gspn_ssp_interface, 351
 - spot::nips_interface, 407
- automaton_
 - spot::saba_complement_tgba, 499
 - spot::tgba_kv_complement, 763
 - spot::tgba_safra_complement, 828
- automatop
 - spot::ltl::automatop, 140
- barand
 - spot::barand, 144
- basic_reduce
 - ltl_rewriting, 14
- bdd_
 - spot::bdd_ordered, 164
- bdd_allocator
 - spot::bdd_allocator, 149
- bdd_dict
 - spot::bdd_dict, 157
- bdd_format_accset
 - spot, 101
- bdd_format_formula
 - spot, 101
- bdd_format_sat
 - spot, 101
- bdd_format_set
 - spot, 102
- bdd_ordered
 - spot::bdd_ordered, 163
- bdd_print_acc
 - spot, 102
- bdd_print_accset
 - spot, 102
- bdd_print_dot
 - spot, 103
- bdd_print_formula
 - spot, 103
- bdd_print_sat
 - spot, 103
- bdd_print_set
 - spot, 104
- bdd_print_table
 - spot, 104
- bdd_to_formula
 - spot, 104
- bdd_v
 - spot::parity_game_graph_delayed, 448
- begin
 - eltly::location, 377
 - eltly::stack, 568
 - ltly::location, 380
 - ltly::stack, 570
 - neverclaimy::location, 374
 - neverclaimy::stack, 563
 - sauty::location, 383
 - sauty::stack, 566
 - spot::ltl::nfa, 401
- bfs_steps
 - spot::bfs_steps, 166
- binop
 - spot::ltl::binop, 173
- bit_state_hashing_magic_search
 - emptiness_check_algorithms, 44
- bit_state_hashing_se05_search
 - emptiness_check_algorithms, 45
- bmrand
 - random, 22
- bounds_t
 - spot::taa_succ_iterator, 614
- branching_postponement_
 - spot::ltl::language_containment_checker, 369
- build
 - spot::connected_component_hash_set_factory, 190
 - spot::explicit_connected_component_factory, 308
 - spot::ltl::random_ltl::op_proba, 434
 - spot::numbered_state_heap_factory, 424
 - spot::numbered_state_heap_hash_map_factory, 432
- build_graph
 - spot::parity_game_graph, 441
 - spot::parity_game_graph_delayed, 449
 - spot::parity_game_graph_direct, 456
- build_link
 - spot::parity_game_graph_direct, 456
- build_map
 - spot::scc_map, 535
- build_recurse_successor_duplicator
 - spot::parity_game_graph_delayed, 449

- build_recurse_successor_spoiler
 - spot::parity_game_graph_delayed, 449
- build_scc_stats
 - spot, 104
- builder
 - spot::ltl::random_ltl::op_proba, 434
- bytecode_
 - spot::nips_interface, 407
- cancel
 - spot::timer_map, 896
- canonicalize
 - spot::power_map, 473
- cc_map
 - spot::bdd_dict, 156
- check
 - spot::couvreur99_check, 202
 - spot::couvreur99_check_shy, 219
 - spot::emptiness_check, 261
- child
 - spot::ltl::formula_tree::node_unop, 417
 - spot::ltl::unop, 919
- child_
 - spot::ltl::unop, 921
- children
 - spot::ltl::formula_tree::node_nfa, 415
- children_
 - spot::ltl::automatop, 143
 - spot::ltl::multop, 398
- clear_rem
 - spot::scc_stack, 539
- clear_todo
 - spot::couvreur99_check_shy, 219
- clone
 - ltl_essential, 11
 - spot::explicit_state_conjunction, 312
 - spot::ltl::atomic_prop, 133
 - spot::ltl::automatop, 140
 - spot::ltl::binop, 174
 - spot::ltl::constant, 195
 - spot::ltl::formula, 329
 - spot::ltl::multop, 396
 - spot::ltl::ref_formula, 486
 - spot::ltl::unop, 919
 - spot::saba_state, 518
 - spot::saba_state_conjunction, 521
 - spot::state, 573
 - spot::state_bdd, 577
 - spot::state_evtgba_explicit, 581
 - spot::state_explicit, 585
 - spot::state_product, 590
 - spot::state_set, 596
 - spot::state_union, 603
- clone_counts
 - spot::bdd_dict, 161
- clone_if
 - spot::taa_tgba_formula, 632
 - spot::taa_tgba_labelled, 643
 - spot::taa_tgba_string, 655
- clone_visitor
 - spot::ltl::clone_visitor, 180
- column
 - eltlyy::position, 465
 - ltlyy::position, 463
 - neverclaimyy::position, 461
 - sautyy::position, 467
- columns
 - eltlyy::location, 376
 - eltlyy::position, 464
 - ltlyy::location, 379
 - ltlyy::position, 462
 - neverclaimyy::location, 373
 - neverclaimyy::position, 460
 - sautyy::location, 382
 - sautyy::position, 466
- common_acceptance_conditions_of_original_state
 - spot::tgba_sba_proxy, 833
 - spot::tgba_tba_proxy, 878
- common_symbol_table
 - spot::evtgba_product, 286
- common_symbols_
 - spot::evtgba_product, 288
- compare
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::saba_state, 518
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
 - spot::state, 573
 - spot::state_bdd, 577
 - spot::state_evtgba_explicit, 581
 - spot::state_explicit, 585
 - spot::state_product, 590
 - spot::state_set, 596
 - spot::state_union, 603
- complement_all_acceptance_conditions
 - spot::tgba_explicit_formula, 708
 - spot::tgba_explicit_labelled, 720
 - spot::tgba_explicit_number, 732
 - spot::tgba_explicit_string, 744
 - spot::tgba_reduc, 804
- compute_all_acceptance_conditions
 - spot, 104
- compute_neg_acceptance_conditions
 - spot, 104
- compute_support_conditions
 - spot::fair_kripke, 317
 - spot::future_conditions_collector, 341

- spot::kripke, 356
- spot::taa_tgba, 622
- spot::taa_tgba_formula, 632
- spot::taa_tgba_labelled, 643
- spot::taa_tgba_string, 655
- spot::tgba, 664
- spot::tgba_bdd_concrete, 673
- spot::tgba_explicit, 698
- spot::tgba_explicit_formula, 708
- spot::tgba_explicit_labelled, 721
- spot::tgba_explicit_number, 732
- spot::tgba_explicit_string, 745
- spot::tgba_kv_complement, 759
- spot::tgba_product, 768
- spot::tgba_product_init, 775
- spot::tgba_reduc, 804
- spot::tgba_safra_complement, 824
- spot::tgba_sba_proxy, 833
- spot::tgba_scc, 840
- spot::tgba_sgba_proxy, 848
- spot::tgba_tba_proxy, 878
- spot::tgba_union, 887
- compute_support_variables
 - spot::fair_kripke, 317
 - spot::future_conditions_collector, 341
 - spot::kripke, 356
 - spot::taa_tgba, 622
 - spot::taa_tgba_formula, 632
 - spot::taa_tgba_labelled, 644
 - spot::taa_tgba_string, 655
 - spot::tgba, 665
 - spot::tgba_bdd_concrete, 673
 - spot::tgba_explicit, 698
 - spot::tgba_explicit_formula, 708
 - spot::tgba_explicit_labelled, 721
 - spot::tgba_explicit_number, 732
 - spot::tgba_explicit_string, 745
 - spot::tgba_kv_complement, 760
 - spot::tgba_product, 768
 - spot::tgba_product_init, 775
 - spot::tgba_reduc, 804
 - spot::tgba_safra_complement, 825
 - spot::tgba_sba_proxy, 833
 - spot::tgba_scc, 840
 - spot::tgba_sgba_proxy, 849
 - spot::tgba_tba_proxy, 878
 - spot::tgba_union, 887
- cond_
 - spot::fair_kripke_succ_iterator, 326
 - spot::kripke_succ_iterator, 365
- cond_set
 - spot::future_conditions_collector, 340
 - spot::scc_map, 533
- cond_set_of
 - spot::scc_map, 535
- condition
 - spot::connected_component_hash_set, 187
 - spot::explicit_connected_component, 306
 - spot::scc_stack::connected_component, 183
 - spot::taa_tgba::transition, 900
 - spot::tgba_explicit::transition, 901
- conds
 - spot::scc_map::scc, 529
- connected_component
 - spot::scc_stack::connected_component, 183
- connected_component_hash_set_factory
 - spot::connected_component_hash_set_factory, 190
- const_iterator
 - eltlyy::stack, 568
 - ltlyy::stack, 570
 - neverclaimyy::stack, 563
 - sautyy::stack, 565
- constant
 - spot::ltl::constant, 195
- constrain_relation
 - spot::tgba_bdd_concrete_factory, 681
- construct
 - spot::emptiness_check_instantiator, 264
- contained
 - spot::ltl::language_containment_checker, 368
- contained_neg
 - spot::ltl::language_containment_checker, 368
- copy_acceptance_conditions_of
 - spot::tgba_explicit, 698
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 721
 - spot::tgba_explicit_number, 733
 - spot::tgba_explicit_string, 745
 - spot::tgba_reduc, 805
- count_
 - spot::ltl::atomic_prop, 135
 - spot::ltl::automatop, 143
 - spot::ltl::binop, 176
 - spot::ltl::constant, 197
 - spot::ltl::formula, 330
 - spot::ltl::multop, 398
 - spot::ltl::ref_formula, 487
 - spot::ltl::unop, 921
- couvreur99
 - emptiness_check_algorithms, 45
- couvreur99_check
 - spot::couvreur99_check, 201
- couvreur99_check_result
 - spot::couvreur99_check_result, 210
- couvreur99_check_shy
 - spot::couvreur99_check_shy, 218
- 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, 226
- create_anonymous_state
 - spot::tgba_bdd_concrete_factory, 681
- create_atomic_prop
 - spot::tgba_bdd_concrete_factory, 681
- create_state
 - spot::tgba_bdd_concrete_factory, 681
- create_transition
 - spot::taa_tgba_formula, 632
 - spot::taa_tgba_labelled, 644
 - spot::taa_tgba_string, 655
 - spot::tgba_explicit, 698
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 721
 - spot::tgba_explicit_number, 733
 - spot::tgba_explicit_string, 745
 - spot::tgba_reduc, 805
- cube_
 - spot::minato_isop, 390
- current_
 - spot::tgba_succ_iterator_concrete, 860
- current_acc_
 - spot::tgba_succ_iterator_concrete, 860
- current_acceptance_conditions
 - spot::evtgba_iterator, 283
 - spot::fair_kripke_succ_iterator, 325
 - spot::kripke_succ_iterator, 364
 - spot::taa_succ_iterator, 615
 - spot::tgba_explicit_succ_iterator, 753
 - spot::tgba_succ_iterator, 854
 - spot::tgba_succ_iterator_concrete, 859
 - spot::tgba_succ_iterator_product, 865
 - spot::tgba_succ_iterator_union, 871
- current_cond_
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_succ_iterator_union, 872
- current_condition
 - spot::fair_kripke_succ_iterator, 325
 - spot::kripke_succ_iterator, 364
 - spot::saba_succ_iterator, 527
 - spot::taa_succ_iterator, 615
 - spot::tgba_explicit_succ_iterator, 753
 - spot::tgba_succ_iterator, 854
 - spot::tgba_succ_iterator_concrete, 859
 - spot::tgba_succ_iterator_product, 865
 - spot::tgba_succ_iterator_union, 871
- current_conjunction
 - spot::saba_succ_iterator, 528
- current_label
 - spot::evtgba_iterator, 283
- current_state
 - spot::evtgba_iterator, 283
 - spot::explicit_state_conjunction, 312
 - spot::fair_kripke_succ_iterator, 325
 - spot::kripke_succ_iterator, 364
 - spot::saba_state_conjunction, 521
 - spot::taa_succ_iterator, 615
 - spot::tgba_explicit_succ_iterator, 754
 - spot::tgba_succ_iterator, 854
 - spot::tgba_succ_iterator_concrete, 859
 - spot::tgba_succ_iterator_product, 865
 - spot::tgba_succ_iterator_union, 871
- current_state_
 - spot::tgba_succ_iterator_concrete, 860
- cycle
 - spot::tgba_run, 815
- cycle_list
 - spot::tgba_sba_proxy, 832
 - spot::tgba_tba_proxy, 877
- cycle_seed
 - spot::couvreur99_check_status, 226
- cycle_start_
 - spot::tgba_sba_proxy, 836
- cycle_states_
 - spot::ars_statistics, 128
- data_
 - spot::tgba_bdd_concrete, 677
 - spot::tgba_bdd_concrete_factory, 682
 - spot::tgba_succ_iterator_concrete, 861
- dead_
 - spot::gspn_interface, 348
- dead_paths
 - spot::scc_stats, 541
- dead_scc
 - spot::scc_stats, 542
- dec_depth
 - spot::couvreur99_check, 202
 - spot::couvreur99_check_shy, 219
 - spot::ec_statistics, 255
- dec_weight_handler
 - spot::weight, 930
- declarative_environment
 - spot::ltl::declarative_environment, 230
- declare
 - spot::ltl::declarative_environment, 230
- declare_acceptance_condition
 - spot::evtgba_explicit, 281
 - spot::tgba_bdd_concrete_factory, 681
 - spot::tgba_bdd_core_data, 687
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 721

- spot::tgba_explicit_number, 733
- spot::tgba_explicit_string, 745
- spot::tgba_reduc, 805
- declare_atomic_prop
 - spot::tgba_bdd_core_data, 687
- declare_now_next
 - spot::tgba_bdd_core_data, 687
- declare_state
 - spot::evtgba_explicit, 281
- Decorating the dot output, 42
- default_environment
 - spot::ltl::default_environment, 234
- del_pred
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- del_succ
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- delete_me_
 - spot::state_set, 597
- delete_scc
 - spot::tgba_reduc, 805
- delete_transitions
 - spot::tgba_reduc, 805
- delete_unaccepting_scc
 - spot::tgba_bdd_concrete, 673
 - spot::tgba_bdd_core_data, 687
- depth
 - spot::couvreur99_check, 202
 - spot::couvreur99_check_shy, 219
 - spot::ec_statistics, 255
- depth_
 - spot::ec_statistics, 256
- Derivable visitors, 12
- dest
 - spot::tgba_explicit::transition, 901
- destroy
 - ltl_essential, 11
 - spot::ltl::atomic_prop, 133
 - spot::ltl::automatop, 141
 - spot::ltl::binop, 174
 - spot::ltl::constant, 196
 - spot::ltl::formula, 329
 - spot::ltl::multop, 396
 - spot::ltl::ref_formula, 486
 - spot::ltl::unop, 920
 - spot::state, 574
 - spot::state_bdd, 577
 - spot::state_evtgba_explicit, 581
 - spot::state_explicit, 585
 - spot::state_product, 590
 - spot::state_set, 596
 - spot::state_union, 603
- dict
 - spot::tgba_bdd_core_data, 688
- dict_
 - spot::bdd_dict::anon_free_list, 124
 - spot::gspn_interface, 348
 - spot::gspn_ssp_interface, 351
 - spot::ltl::language_containment_checker, 369
 - spot::nips_interface, 407
 - spot::taa_tgba, 625
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 659
 - spot::tgba_explicit, 701
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 725
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 749
 - spot::tgba_product, 771
 - spot::tgba_reduc, 813
 - spot::tgba_union, 890
- display_rel_sim
 - spot::tgba_reduc, 806
- display_safra
 - spot, 105
- display_scc
 - spot::tgba_reduc, 806
- dn_v
 - tgba_reduction, 37
- doit
 - spot::ltl::postfix_visitor, 470, 471
- doit_default
 - spot::ltl::postfix_visitor, 471
- done
 - spot::evtgba_iterator, 283
 - spot::explicit_state_conjunction, 312
 - spot::fair_kripke_succ_iterator, 325
 - spot::kripke_succ_iterator, 364
 - spot::loopless_modular_mixed_radix_gray_code, 385
 - spot::numbered_state_heap_const_iterator, 421
 - spot::saba_state_conjunction, 522
 - spot::saba_succ_iterator, 528
 - spot::taa_succ_iterator, 616
 - spot::tgba_explicit_succ_iterator, 754
 - spot::tgba_succ_iterator, 854
 - spot::tgba_succ_iterator_concrete, 859
 - spot::tgba_succ_iterator_product, 865
 - spot::tgba_succ_iterator_union, 871
- done_

- spot::loopless_modular_mixed_radix_gray_code, 386
- dotty
 - ltl_io, 6
- dotty_decorator
 - spot::dotty_decorator, 237
- dotty_reachable
 - spot, 105
 - tgba_io, 25
- drand
 - random, 22
- dst
 - spot::ltl::nfa::transition, 900
 - spot::taa_tgba::transition, 900
- dump
 - ltl_io, 6
 - spot::bdd_dict, 157
 - spot::ltl::atomic_prop, 133
 - spot::ltl::automatop, 141
 - spot::ltl::binop, 174
 - spot::ltl::constant, 196
 - spot::ltl::formula, 329
 - spot::ltl::multop, 396
 - spot::ltl::ref_formula, 486
 - spot::ltl::unop, 920
 - spot::scc_stats, 541
 - spot::tgba_statistics, 852
- dump_free_list
 - spot::bdd_allocator, 149
 - spot::bdd_dict::anon_free_list, 123
 - spot::free_list, 335
- dump_instances
 - spot::ltl::atomic_prop, 133
 - spot::ltl::automatop, 141
 - spot::ltl::binop, 174
 - spot::ltl::multop, 396
 - spot::ltl::unop, 920
 - spot::symbol, 610
- dump_priorities
 - spot::ltl::random_ltl, 477
- dump_queue
 - spot::couvreur99_check_shy, 219
- dump_scc_dot
 - spot, 105
- duplicator_node
 - spot::duplicator_node, 241
- duplicator_node_delayed
 - spot::duplicator_node_delayed, 248
- duplicator_vertice_
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 452
 - spot::parity_game_graph_direct, 458
- ec_statistics
 - spot::ec_statistics, 254
- ecs_
 - spot::couvreur99_check, 205
 - spot::couvreur99_check_result, 212
 - spot::couvreur99_check_shy, 223
- eltl_to_tgba_lacim
 - tgba_ltl, 32
- eltlparse/ Directory Reference, 55
- eltlparse/location.hh, 936
- eltlparse/position.hh, 943
- eltlparse/public.hh, 949
- eltlparse/stack.hh, 956
- eltlyy, 73
 - operator<<, 75, 76
 - operator+, 74, 75
 - operator+=, 75
 - operator-, 75
 - operator=, 75
 - operator==, 76
- eltlyy::location, 374
 - begin, 377
 - columns, 376
 - end, 377
 - initialize, 376
 - lines, 376
 - location, 376
 - step, 377
- eltlyy::position, 463
 - column, 465
 - columns, 464
 - filename, 465
 - initialize, 464
 - line, 465
 - lines, 464
 - position, 464
- eltlyy::slice, 548
 - range_, 549
 - slice, 548
 - stack_, 549
- eltlyy::stack, 567
 - begin, 568
 - const_iterator, 568
 - end, 568
 - height, 568
 - iterator, 568
 - pop, 569
 - push, 569
 - seq_, 569
 - stack, 568
- Emptiness-check algorithms, 43
- Emptiness-check algorithms for SSP, 4
- Emptiness-check statistics, 54
- Emptiness-checks, 43
- emptiness_check

- spot::emptiness_check, 260
- emptiness_check_algorithms
 - bit_state_hashing_magic_search, 44
 - bit_state_hashing_se05_search, 45
 - couvreur99, 45
 - explicit_gv04_check, 47
 - explicit_magic_search, 47
 - explicit_se05_search, 49
 - explicit_tau03_opt_search, 50
 - explicit_tau03_search, 51
 - magic_search, 52
 - se05, 52
- emptiness_check_instantiator
 - spot::emptiness_check_instantiator, 264
- emptiness_check_result
 - spot::emptiness_check_result, 268
- emptiness_check_ssp
 - couvreur99_check_ssp_semi, 4
 - couvreur99_check_ssp_shy, 4
 - couvreur99_check_ssp_shy_semi, 4
- empty
 - spot::scc_stack, 540
 - spot::timer_map, 896
- emulate_acc_cond_
 - spot::tgba_sgba_proxy, 852
- end
 - eltlyy::location, 377
 - eltlyy::stack, 568
 - ltlyy::location, 380
 - ltlyy::stack, 570
 - neverclaimyy::location, 374
 - neverclaimyy::stack, 563
 - sautyy::location, 383
 - sautyy::stack, 566
 - spot::evtgba_reachable_iterator, 292
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::ltl::nfa, 401
 - spot::parity_game_graph, 441
 - spot::parity_game_graph_delayed, 450
 - spot::parity_game_graph_direct, 456
 - spot::saba_reachable_iterator, 503
 - spot::saba_reachable_iterator_breadth_first, 509
 - spot::saba_reachable_iterator_depth_first, 515
 - spot::tgba_reachable_iterator, 782
 - spot::tgba_reachable_iterator_breadth_first, 788
 - spot::tgba_reachable_iterator_depth_first, 794
 - spot::tgba_reduc, 806
- env
 - spot::ltl::atomic_prop, 133
- env_
 - spot::gspn_interface, 348
 - spot::gspn_ssp_interface, 351
 - spot::ltl::atomic_prop, 135
- equal
 - spot::ltl::language_containment_checker, 368
- Equiv
 - spot::ltl::binop, 173
- err_
 - spot::gspn_exception, 345
 - spot::nips_exception, 404
- err_defined_
 - spot::nips_exception, 404
- escape_str
 - misc_tools, 19
- Essential LTL types, 10
- Essential SABA types, 27
- Essential TGBA types, 28
- evtgba
 - spot::evtgba, 274
- evtgba/ Directory Reference, 56
- evtgba/evtgba.hh, 959
- evtgba/evtgbaiter.hh, 960
- evtgba/explicit.hh, 961
- evtgba/product.hh, 962
- evtgba/symbol.hh, 963
- evtgba_explicit
 - spot::evtgba_explicit, 280
- evtgba_parse
 - spot, 105
- evtgba_parse_error
 - spot, 100
- evtgba_parse_error_list
 - spot, 100
- evtgba_product
 - spot::evtgba_product, 286
- evtgba_product_operands
 - spot::evtgba_product, 286
- evtgba_reachable_iterator
 - spot::evtgba_reachable_iterator, 292
- evtgba_reachable_iterator_breadth_first
 - spot::evtgba_reachable_iterator_breadth_first, 297
- evtgba_reachable_iterator_depth_first
 - spot::evtgba_reachable_iterator_depth_first, 302
- evtgba_save_reachable
 - spot, 106
- evtgbaalgs/ Directory Reference, 57
- evtgbaalgs/dotty.hh, 965
- evtgbaalgs/reachiter.hh, 967
- evtgbaalgs/save.hh, 970
- evtgbaalgs/tgba2evtgba.hh, 971
- evtgbaparse/ Directory Reference, 58

- evtgba_parse/public.hh, 951
- explicit_gv04_check
 - emptiness_check_algorithms, 47
- explicit_magic_search
 - emptiness_check_algorithms, 47
- explicit_se05_search
 - emptiness_check_algorithms, 49
- explicit_state_conjunction
 - spot::explicit_state_conjunction, 312
- explicit_tau03_opt_search
 - emptiness_check_algorithms, 50
- explicit_tau03_search
 - emptiness_check_algorithms, 51
- exprop_
 - spot::ltl::language_containment_checker, 369
- extend
 - spot::bdd_allocator, 149
 - spot::bdd_dict::anon_free_list, 123
 - spot::free_list, 335
- extvarnum
 - spot::bdd_allocator, 150
- F
 - spot::ltl::unop, 919
- f0_max
 - spot::minato_isop::local_vars, 370
- f0_min
 - spot::minato_isop::local_vars, 370
- f1_max
 - spot::minato_isop::local_vars, 370
- f1_min
 - spot::minato_isop::local_vars, 371
- f_
 - spot::loopless_modular_mixed_radix_gray_code, 386
- f_max
 - spot::minato_isop::local_vars, 371
- f_min
 - spot::minato_isop::local_vars, 371
- fair_kripke_succ_iterator
 - spot::fair_kripke_succ_iterator, 324
- fair_loop_approx_
 - spot::ltl::language_containment_checker, 369
- False
 - spot::ltl::constant, 195
 - spot::ltl::formula_tree, 115
- false_instance
 - spot::ltl::constant, 196
- fc_map
 - spot::future_conditions_collector, 340
- filename
 - eltly::position, 465
 - ltly::position, 463
 - neverclaimy::position, 461
 - sautyy::position, 467
- filter
 - spot::bfs_steps, 166
- finalize
 - spot::bfs_steps, 167
- finals_
 - spot::ltl::nfa, 403
- find
 - spot::numbered_state_heap, 419, 420
 - spot::numbered_state_heap_hash_map, 428
- find_paths
 - spot, 106
- find_state
 - spot::couvereur99_check_shy, 220
- Finish
 - spot::ltl::unop, 919
- finish
 - spot::tgba_bdd_concrete_factory, 682
- first
 - spot::evtgba_iterator, 283
 - spot::explicit_state_conjunction, 312
 - spot::fair_kripke_succ_iterator, 325
 - spot::kripke_succ_iterator, 364
 - spot::loopless_modular_mixed_radix_gray_code, 386
 - spot::ltl::binop, 174
 - spot::numbered_state_heap_const_iterator, 421
 - spot::saba_state_conjunction, 522
 - spot::saba_succ_iterator, 528
 - spot::taa_succ_iterator, 616
 - spot::tgba_explicit_succ_iterator, 754
 - spot::tgba_succ_iterator, 855
 - spot::tgba_succ_iterator_concrete, 860
 - spot::tgba_succ_iterator_product, 866
 - spot::tgba_succ_iterator_union, 871
- first_
 - spot::ltl::binop, 176
- FirstStep
 - spot::minato_isop::local_vars, 370
- fl
 - spot::bdd_allocator, 151
 - spot::bdd_dict::anon_free_list, 124
 - spot::free_list, 336
- format_acceptance_condition
 - spot::evtgba, 274
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- format_acceptance_conditions
 - spot::evtgba, 275
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- format_evtgba_parse_errors
 - spot, 106

- format_label
 - spot::evtgba, 275
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- format_neverclaim_parse_errors
 - tgba_io, 25
- format_parse_errors
 - ltl_io, 7
- format_state
 - spot::evtgba, 275
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
 - spot::fair_kripke, 318
 - spot::future_conditions_collector, 341
 - spot::kripke, 357
 - spot::ltl::nfa, 402
 - spot::saba, 492
 - spot::saba_complement_tgba, 498
 - spot::taa_tgba, 622
 - spot::taa_tgba_formula, 632
 - spot::taa_tgba_labelled, 644
 - spot::taa_tgba_string, 656
 - spot::tgba, 665
 - spot::tgba_bdd_concrete, 673
 - spot::tgba_explicit, 698
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 721
 - spot::tgba_explicit_number, 733
 - spot::tgba_explicit_string, 746
 - spot::tgba_kv_complement, 760
 - spot::tgba_product, 768
 - spot::tgba_product_init, 775
 - spot::tgba_reduc, 806
 - spot::tgba_safracomplement, 825
 - spot::tgba_sba_proxy, 833
 - spot::tgba_scc, 841
 - spot::tgba_sgba_proxy, 849
 - spot::tgba_tba_proxy, 879
 - spot::tgba_union, 887
- format_state_set
 - spot::taa_tgba_labelled, 644
- format_tgba_parse_errors
 - tgba_io, 25
- formula
 - spot::ltl::formula, 328
- formula_to_bdd
 - spot, 106
- FourthStep
 - spot::minato_isop::local_vars, 370
- free_anonymous_list_of
 - spot::bdd_dict, 161
- free_anonymous_list_of_type
 - spot::bdd_dict, 156
- free_count
 - spot::bdd_allocator, 150
 - spot::bdd_dict::anon_free_list, 123
 - spot::free_list, 335
- free_list_type
 - spot::bdd_allocator, 149
 - spot::bdd_dict::anon_free_list, 123
 - spot::free_list, 335
- free_relation_simulation
 - tgba_reduction, 37
- future_conditions
 - spot::future_conditions_collector, 341
- future_conditions_collector
 - spot::future_conditions_collector, 340
- future_conds_
 - spot::future_conditions_collector, 344
- fv_map
 - spot::bdd_dict, 156
- G
 - spot::ltl::unop, 919
- g0
 - spot::minato_isop::local_vars, 371
- g1
 - spot::minato_isop::local_vars, 371
- generate
 - spot::ltl::random_ltl, 477
- get
 - spot::acss_statistics, 119
 - spot::ars_statistics, 127
 - spot::couvreur99_check, 202
 - spot::couvreur99_check_result, 211
 - spot::couvreur99_check_shy, 220
 - spot::ec_statistics, 255
 - spot::option_map, 435
 - spot::unsigned_statistics, 924
- get_acc
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
- get_acc_list
 - spot::tgba_kv_complement, 760
- get_acceptance_condition
 - spot::tgba_explicit, 698
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 722
 - spot::tgba_explicit_number, 733
 - spot::tgba_explicit_string, 746
 - spot::tgba_reduc, 806
- get_acceptance_condition_visited
 - spot::spoiler_node_delayed, 560
- get_aut
 - spot::scc_map, 535
- get_bdd
 - spot::bdd_ordered, 163
- get_core_data

- spot::tgba_bdd_concrete, 674
- spot::tgba_bdd_concrete_factory, 682
- spot::tgba_bdd_factory, 692
- get_delayed_relation_simulation
 - tgba_reduction, 37
- get_dict
 - spot::fair_kripke, 318
 - spot::future_conditions_collector, 341
 - spot::kripke, 357
 - spot::saba, 493
 - spot::saba_complement_tgba, 498
 - spot::taa_tgba, 623
 - spot::taa_tgba_formula, 633
 - spot::taa_tgba_labelled, 644
 - spot::taa_tgba_string, 656
 - spot::tgba, 665
 - spot::tgba_bdd_concrete, 674
 - spot::tgba_bdd_concrete_factory, 682
 - spot::tgba_explicit, 699
 - spot::tgba_explicit_formula, 709
 - spot::tgba_explicit_labelled, 722
 - spot::tgba_explicit_number, 733
 - spot::tgba_explicit_string, 746
 - spot::tgba_kv_complement, 760
 - spot::tgba_product, 768
 - spot::tgba_product_init, 775
 - spot::tgba_reduc, 806
 - spot::tgba_safra_complement, 825
 - spot::tgba_sba_proxy, 833
 - spot::tgba_scc, 841
 - spot::tgba_sgba_proxy, 849
 - spot::tgba_tba_proxy, 879
 - spot::tgba_union, 887
- get_direct_relation_simulation
 - tgba_reduction, 38
- get_duplicator_node
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- get_err
 - spot::gspn_exception, 345
 - spot::nips_exception, 404
- get_err_defined
 - spot::nips_exception, 404
- get_index
 - spot::numbered_state_heap_const_iterator, 422
- get_init_bdd
 - spot::tgba_bdd_concrete, 674
- get_init_state
 - spot::fair_kripke, 318
 - spot::future_conditions_collector, 342
 - spot::kripke, 357
 - spot::ltl::nfa, 402
 - spot::saba, 493
 - spot::saba_complement_tgba, 498
 - spot::taa_tgba, 623
 - spot::taa_tgba_formula, 633
 - spot::taa_tgba_labelled, 645
 - spot::taa_tgba_string, 656
 - spot::tgba, 665
 - spot::tgba_bdd_concrete, 674
 - spot::tgba_explicit, 699
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 722
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 746
 - spot::tgba_kv_complement, 760
 - spot::tgba_product, 768
 - spot::tgba_product_init, 776
 - spot::tgba_reduc, 806
 - spot::tgba_safra_complement, 825
 - spot::tgba_sba_proxy, 833
 - spot::tgba_scc, 841
 - spot::tgba_sgba_proxy, 849
 - spot::tgba_tba_proxy, 879
 - spot::tgba_union, 887
- get_label
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 248
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 722
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 746
 - spot::tgba_reduc, 807
- get_lead_2_acc_all
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node_delayed, 560
- get_name
 - spot::ltl::nfa, 402
- get_nb_succ
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- get_nfa
 - spot::ltl::automatop, 141
- get_pair
 - spot::duplicator_node, 242
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 560
- get_progress_measure
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node_delayed, 561
- get_prop_map
 - spot::ltl::declarative_environment, 230

- get_relation
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 450
 - spot::parity_game_graph_direct, 457
- get_removed_components
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
- get_safra
 - spot::tgba_safra_complement, 825
- get_spoiler_node
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 561
- get_state
 - spot::numbered_state_heap_const_iterator, 422
 - spot::state_evtgba_explicit, 582
 - spot::state_explicit, 586
 - spot::state_set, 597
- get_vmsize
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
- get_where
 - spot::gspn_exception, 345
 - spot::nips_exception, 404
- group2_
 - spot::couvreur99_check_shy, 223
- group_
 - spot::couvreur99_check_shy, 223
- gspn/ Directory Reference, 59
- gspn/common.hh, 931
- gspn/gspn.hh, 934
- gspn/ssp.hh, 934
- gspn_exception
 - spot::gspn_exception, 345
- gspn_interface
 - spot::gspn_interface, 348
- gspn_ssp_interface
 - spot::gspn_ssp_interface, 351
- h
 - spot::couvreur99_check_status, 227
 - spot::numbered_state_heap_hash_map, 429
- h_
 - spot::scc_map, 538
- has_acceptance_condition
 - spot::tgba_explicit, 699
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 722
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 747
 - spot::tgba_reduc, 807
- has_monitor
 - spot::nips_interface, 407
- has_state
 - spot::connected_component_hash_set, 186
 - spot::explicit_connected_component, 306
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 723
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 747
 - spot::tgba_reduc, 807
- hash
 - spot::ltl::atomic_prop, 134
 - spot::ltl::automatop, 141
 - spot::ltl::binop, 175
 - spot::ltl::constant, 196
 - spot::ltl::formula, 329
 - spot::ltl::multop, 396
 - spot::ltl::ref_formula, 486
 - spot::ltl::unop, 920
 - spot::saba_state, 518
 - spot::state, 574
 - spot::state_bdd, 578
 - spot::state_evtgba_explicit, 582
 - spot::state_explicit, 586
 - spot::state_product, 590
 - spot::state_set, 597
 - spot::state_union, 603
- hash_funcs
 - knuth32_hash, 21
 - wang32_hash, 21
- hash_type
 - spot::numbered_state_heap_hash_map, 427
 - spot::scc_map, 533
- Hashing functions, 20
- height
 - eltlyy::stack, 568
 - ltlyy::stack, 571
 - neverclaimyy::stack, 564
 - sautyy::stack, 566
- i
 - spot::ltl::formula_tree::node_atomic, 410
- i_
 - spot::ltl::succ_iterator, 607
 - spot::taa_succ_iterator, 616
 - spot::tgba_explicit_succ_iterator, 755
- Implies
 - spot::ltl::binop, 173
- implies
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 249
- implies_acc
 - spot::duplicator_node_delayed, 249
- implies_label
 - spot::duplicator_node_delayed, 249

- in
 - spot::evtgba_explicit::state, 572
 - spot::evtgba_explicit::transition, 903
 - spot::ltl::ltl_file, 388
- inc_ars_cycle_states
 - spot::acss_statistics, 119
 - spot::ars_statistics, 128
 - spot::couvreur99_check_result, 211
- inc_ars_prefix_states
 - spot::acss_statistics, 119
 - spot::ars_statistics, 128
 - spot::couvreur99_check_result, 211
- inc_depth
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
 - spot::ec_statistics, 255
- inc_states
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
 - spot::ec_statistics, 255
- inc_transitions
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
 - spot::ec_statistics, 255
- inc_weight_handler
 - spot::weight, 930
- incomp_map
 - spot::ltl::language_containment_checker::record_, 482
- incompatible
 - spot::ltl::language_containment_checker::record_, 482
- incompatible_
 - spot::ltl::language_containment_checker, 368
- index
 - spot::connected_component_hash_set, 187
 - spot::explicit_connected_component, 306
 - spot::numbered_state_heap, 420
 - spot::numbered_state_heap_hash_map, 428
 - spot::scc_map::scc, 529
 - spot::scc_stack::connected_component, 183
- infinitely_often
 - spot::tgba_bdd_core_data, 687
- info_
 - spot::emptiness_check_instantiator, 265
- init_
 - spot::ltl::nfa, 403
 - spot::taa_tgba, 625
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 659
 - spot::tgba_bdd_concrete, 677
 - spot::tgba_explicit, 702
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 725
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 750
 - spot::tgba_reduc, 813
- init_iter
 - spot::evtgba, 275
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- init_states_
 - spot::evtgba_explicit, 282
- initial
 - spot::scc_map, 536
- initialize
 - eltlyy::location, 376
 - eltlyy::position, 464
 - ltlyy::location, 379
 - ltlyy::position, 462
 - neverclaimyy::location, 373
 - neverclaimyy::position, 460
 - sautyy::location, 382
 - sautyy::position, 466
 - spot::bdd_allocator, 150
 - spot::bdd_dict, 158
- initialized
 - spot::bdd_allocator, 151
 - spot::bdd_dict, 161
- Input/Output of LTL formulae, 4
- Input/Output of TGBA, 23
- insert
 - spot::bdd_allocator, 150
 - spot::bdd_dict::anon_free_list, 123
 - spot::connected_component_hash_set, 186
 - spot::explicit_connected_component, 306
 - spot::free_list, 335
 - spot::numbered_state_heap, 420
 - spot::numbered_state_heap_hash_map, 428
- instance
 - spot::connected_component_hash_set_factory, 190
 - spot::dotty_decorator, 237
 - spot::ltl::atomic_prop, 134
 - spot::ltl::automatop, 141
 - spot::ltl::binop, 175
 - spot::ltl::default_environment, 234
 - spot::ltl::multop, 396, 397
 - spot::ltl::unop, 920
 - spot::numbered_state_heap_hash_map_factory, 432
 - spot::symbol, 610
 - spot::tgba_run_dotty_decorator, 819
- instance_count
 - spot::ltl::atomic_prop, 134
 - spot::ltl::automatop, 142
 - spot::ltl::binop, 175

- spot::ltl::multop, 397
- spot::ltl::unop, 920
- spot::symbol, 610
- instances
 - spot::ltl::atomic_prop, 135
 - spot::ltl::automatop, 143
 - spot::ltl::binop, 176
 - spot::ltl::multop, 398
 - spot::ltl::unop, 921
- instances_
 - spot::symbol, 611
- instanciate
 - spot::ltl::formula_tree, 115
- instantiate
 - spot::emptiness_check_instantiator, 264
- is_
 - spot::ltl::nfa, 403
- is_bare_word
 - misc_tools, 19
- is_eventual
 - ltl_misc, 16
- is_FG
 - ltl_misc, 16
- is_final
 - spot::ltl::nfa, 402
- is_GF
 - ltl_misc, 16
- is_guarantee_automaton
 - spot, 106
- is_loop
 - spot::ltl::nfa, 402
- is_map
 - spot::ltl::nfa, 400
- is_negated
 - spot::ltl::automatop, 142
- is_not_accepting
 - spot::tgba_reduc, 807
- is_registered_acceptance_variable
 - spot::bdd_dict, 158
- is_registered_proposition
 - spot::bdd_dict, 158
- is_registered_state
 - spot::bdd_dict, 158
- is_running
 - spot::timer, 894
- is_safety_mwdba
 - spot, 107
- is_terminal
 - spot::tgba_reduc, 807
- is_universal
 - ltl_misc, 17
- it_
 - spot::explicit_state_conjunction, 313
- item_type
 - spot::timer_map, 896
- iterator
 - eltlyy::stack, 568
 - ltlyy::stack, 570
 - neverclaimyy::stack, 563
 - sautyy::stack, 565
 - spot::ltl::nfa, 400
 - spot::numbered_state_heap, 420
 - spot::numbered_state_heap_hash_map, 429
 - spot::taa_succ_iterator, 614
- iterator_pair
 - spot::taa_succ_iterator, 615
- knuth32_hash
 - hash_funcs, 21
- Kripke Structures, 10
- kripke/ Directory Reference, 61
- kripke/fairkripke.hh, 971
- kripke/kripke.hh, 973
- kripke_succ_iterator
 - spot::kripke_succ_iterator, 363
- label
 - spot::evtgba_explicit::transition, 903
 - spot::ltl::nfa, 400
 - spot::tgba_run::step, 605
- label_
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 250
- label_t
 - spot::taa_tgba_formula, 630
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 653
 - spot::tgba_explicit_formula, 707
 - spot::tgba_explicit_labelled, 719
 - spot::tgba_explicit_number, 731
 - spot::tgba_explicit_string, 743
 - spot::tgba_reduc, 802
- label_to_string
 - spot::taa_tgba_formula, 633
 - spot::taa_tgba_labelled, 645
 - spot::taa_tgba_string, 656
- language_containment_checker
 - spot::ltl::language_containment_checker, 368
- last
 - spot::loopless_modular_mixed_radix_gray_code, 386
- last_support_conditions_input_
 - spot::tgba, 668
- last_support_conditions_output_
 - spot::tgba, 668
- last_support_variables_input_
 - spot::tgba, 668
- last_support_variables_output_
 - spot::tgba, 668

- spot::tgba, 668
- lbl
 - spot::ltl::nfa::transition, 900
- lbtt_reachable
 - tgba_io, 26
- lead_2_acc_all_
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node_delayed, 561
- left
 - spot::state_product, 591
 - spot::state_union, 604
- left_
 - spot::state_product, 591
 - spot::state_union, 604
 - spot::tgba_product, 771
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_succ_iterator_union, 872
 - spot::tgba_union, 890
- left_acc_complement_
 - spot::tgba_product, 771
 - spot::tgba_union, 890
- left_acc_missing_
 - spot::tgba_union, 890
- left_init_
 - spot::tgba_product_init, 778
- left_missing_
 - spot::tgba_succ_iterator_union, 872
- left_neg_
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_succ_iterator_union, 872
- left_var_missing_
 - spot::tgba_union, 890
- length
 - ltl_misc, 17
- lhs
 - spot::ltl::formula_tree::node_binop, 411
 - spot::ltl::formula_tree::node_multop, 413
- lift
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 450
 - spot::parity_game_graph_direct, 457
- line
 - eltlyy::position, 465
 - ltlyy::position, 463
 - neverclaimyy::position, 461
 - sautyy::position, 467
- lines
 - eltlyy::location, 376
 - eltlyy::position, 464
 - ltlyy::location, 379
 - ltlyy::position, 462
 - neverclaimyy::location, 373
 - neverclaimyy::position, 460
 - sautyy::location, 382
- sautyy::position, 466
- link_decl
 - spot::dotty_decorator, 237
 - spot::tgba_run_dotty_decorator, 819
- lnode_pred
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 561
- lnode_succ
 - spot::duplicator_node, 244
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 562
- local_vars
 - spot::minato_isop::local_vars, 370
- location
 - eltlyy::location, 376
 - ltlyy::location, 379
 - neverclaimyy::location, 373
 - sautyy::location, 382
- loopless_modular_mixed_radix_gray_code
 - spot::loopless_modular_mixed_radix_gray_code, 385
- LTl Abstract Syntax Tree, 11
- LTl environments, 12
- LTl formulae, 2
- 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, 11
 - destroy, 11
- ltl_file
 - spot::ltl::ltl_file, 387
- ltl_io
 - dotty, 6
 - dump, 6
 - format_parse_errors, 7
 - parse, 7
 - parse_error, 6
 - parse_error_list, 6
 - parse_file, 8
 - parse_string, 8
 - spair, 6
 - to_spin_string, 8, 9
 - to_string, 9
- ltl_misc
 - atomic_prop_collect, 16

- atomic_prop_set, 16
- is_eventual, 16
- is_FG, 16
- is_GF, 16
- is_universal, 17
- length, 17
- syntactic_implication, 17
- syntactic_implication_neg, 17
- ltl_rewriting
 - basic_reduce, 14
 - negative_normal_form, 14
 - reduce, 14
 - reduce_options, 13
 - simplify_f_g, 14
 - unabbreviate_logic, 14
- ltl_to_taa
 - tgba_ltl, 32
- ltl_to_tgba_fm
 - tgba_ltl, 33
- ltl_to_tgba_lacim
 - tgba_ltl, 34
- ltlast/ Directory Reference, 61
- ltlast/allnodes.hh, 973
- ltlast/atomic_prop.hh, 974
- ltlast/automatop.hh, 976
- ltlast/binop.hh, 977
- ltlast/constant.hh, 979
- ltlast/formula.hh, 980
- ltlast/formula_tree.hh, 982
- ltlast/multop.hh, 983
- ltlast/nfa.hh, 985
- ltlast/predecl.hh, 986
- ltlast/refformula.hh, 987
- ltlast/unop.hh, 988
- ltlast/visitor.hh, 990
- ltlenv/ Directory Reference, 63
- ltlenv/declenv.hh, 991
- ltlenv/defaultenv.hh, 993
- ltlenv/environment.hh, 994
- ltlparse/ Directory Reference, 64
- ltlparse/location.hh, 938
- ltlparse/ltlfile.hh, 995
- ltlparse/position.hh, 945
- ltlparse/public.hh, 952
- ltlparse/stack.hh, 957
- ltlvisit/ Directory Reference, 65
- ltlvisit/apcollect.hh, 996
- ltlvisit/basicreduce.hh, 998
- ltlvisit/clone.hh, 999
- ltlvisit/contain.hh, 1000
- ltlvisit/destroy.hh, 1001
- ltlvisit/dotty.hh, 965
- ltlvisit/dump.hh, 1002
- ltlvisit/length.hh, 1003
- ltlvisit/lunabbrev.hh, 1004
- ltlvisit/nenofrm.hh, 1006
- ltlvisit/postfix.hh, 1007
- ltlvisit/randomltl.hh, 1008
- ltlvisit/reduce.hh, 1009
- ltlvisit/simpfg.hh, 1011
- ltlvisit/syntimpl.hh, 1011
- ltlvisit/tostring.hh, 1012
- ltlvisit/tunabbrev.hh, 1013
- ltlty, 76
 - operator<<, 79
 - operator+, 78
 - operator+&, 78
 - operator-, 78
 - operator=, 79
 - operator==, 79
- ltlty::location, 377
 - begin, 380
 - columns, 379
 - end, 380
 - initialize, 379
 - lines, 379
 - location, 379
 - step, 380
- ltlty::position, 461
 - column, 463
 - columns, 462
 - filename, 463
 - initialize, 462
 - line, 463
 - lines, 462
 - position, 462
- ltlty::slice, 551
 - range_, 552
 - slice, 552
 - stack_, 552
- ltlty::stack, 569
 - begin, 570
 - const_iterator, 570
 - end, 570
 - height, 571
 - iterator, 570
 - pop, 571
 - push, 571
 - seq_, 571
 - stack, 570
- lvarnum
 - spot::bdd_allocator, 151
 - spot::bdd_dict, 161
- M
 - spot::ltl::binop, 173
- m
 - spot::weight, 931

- m_
 - spot::barand, 144
 - spot::loopless_modular_mixed_radix_gray_-code, 386
- magic_search
 - emptiness_check_algorithms, 52
- mainpage.dox, 931
- map
 - spot::ltl::atomic_prop, 132
 - spot::ltl::automatop, 140
 - spot::ltl::binop, 173
 - spot::ltl::multop, 395
 - spot::ltl::unop, 918
 - spot::symbol, 610
- map_
 - spot::power_map, 473
 - spot::tgba_run_dotty_decorator, 820
- map_builder_
 - spot::future_conditions_collector, 342
- match
 - spot::bfs_steps, 167
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 249
- max_acceptance_conditions
 - spot::emptiness_check_instantiator, 264
- max_count
 - spot::ltl::formula, 330
- max_depth
 - spot::couvreur99_check, 203
 - spot::couvreur99_check_shy, 220
 - spot::ec_statistics, 255
- max_depth_
 - spot::ec_statistics, 256
- max_spanning_paths
 - spot, 107
- memusage
 - spot, 107
- merge_state
 - spot::tgba_reduc, 808
- merge_state_delayed
 - spot::tgba_reduc, 808
- merge_transitions
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 723
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 747
 - spot::tgba_reduc, 808
- min_acceptance_conditions
 - spot::emptiness_check_instantiator, 265
- min_n
 - spot::ltl::random_ltl::op_proba, 434
- minato_isop
 - spot::minato_isop, 389
- minimize_monitor
 - tgba_reduction, 38
- minimize_obligation
 - tgba_reduction, 38
- minimize_wdba
 - tgba_reduction, 39
- misc/ Directory Reference, 66
- misc/bareword.hh, 1015
- misc/bddalloc.hh, 1015
- misc/bddlt.hh, 1016
- misc/bddop.hh, 1017
- misc/escape.hh, 1018
- misc/freelist.hh, 1019
- misc/hash.hh, 1020
- misc/hashfunc.hh, 1021
- misc/ltstr.hh, 1022
- misc/memusage.hh, 1023
- misc/minato.hh, 1023
- misc/modgray.hh, 1024
- misc/optionmap.hh, 1024
- misc/random.hh, 1025
- misc/timer.hh, 1026
- misc/version.hh, 1027
- 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, 40
- Miscellaneous helper algorithms, 18
- mrnd
 - random, 22
- multop
 - spot::ltl::multop, 395
- n
 - spot::couvreur99_check_shy::todo_item, 899
- n_
 - spot::barand, 144
 - spot::loopless_modular_mixed_radix_gray_-code, 387
- name
 - spot::ltl::atomic_prop, 134
 - spot::ltl::declarative_environment, 230
 - spot::ltl::default_environment, 234
 - spot::ltl::environment, 272
 - spot::ltl::random_ltl::op_proba, 434
 - spot::symbol, 610
- name_
 - spot::ltl::atomic_prop, 135
 - spot::ltl::nfa, 403
 - spot::symbol, 611
- name_state_map_
 - spot::evtgba_explicit, 282

- spot::taa_tgba_formula, 636
- spot::taa_tgba_labelled, 648
- spot::taa_tgba_string, 659
- spot::tgba_explicit_formula, 713
- spot::tgba_explicit_labelled, 725
- spot::tgba_explicit_number, 737
- spot::tgba_explicit_string, 750
- spot::tgba_reduc, 813
- nb_node_parity_game
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 452
 - spot::parity_game_graph_direct, 459
- nb_set_acc_cond
 - spot::parity_game_graph_delayed, 450
 - spot::tgba_reduc, 808
- nb_states_
 - spot::saba_complement_tgba, 499
 - spot::tgba_kv_complement, 763
- neg_acceptance_cond_
 - spot::tgba_safra_complement, 828
- neg_acceptance_conditions
 - spot::fair_kripke, 318
 - spot::future_conditions_collector, 342
 - spot::kripke, 357
 - spot::taa_tgba, 623
 - spot::taa_tgba_formula, 633
 - spot::taa_tgba_labelled, 645
 - spot::taa_tgba_string, 657
 - spot::tgba, 666
 - spot::tgba_bdd_concrete, 674
 - spot::tgba_explicit, 699
 - spot::tgba_explicit_formula, 710
 - spot::tgba_explicit_labelled, 723
 - spot::tgba_explicit_number, 734
 - spot::tgba_explicit_string, 747
 - spot::tgba_kv_complement, 760
 - spot::tgba_product, 769
 - spot::tgba_product_init, 776
 - spot::tgba_reduc, 808
 - spot::tgba_safra_complement, 825
 - spot::tgba_sba_proxy, 834
 - spot::tgba_scc, 841
 - spot::tgba_sgba_proxy, 849
 - spot::tgba_tba_proxy, 879
 - spot::tgba_union, 888
- neg_acceptance_conditions_
 - spot::taa_tgba, 626
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 660
 - spot::tgba_explicit, 702
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 726
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 750
 - spot::tgba_product, 771
 - spot::tgba_reduc, 813
 - spot::tgba_union, 890
- neg_all_acc
 - spot::weight, 931
- neg_contained
 - spot::ltl::language_containment_checker, 368
- negacc_set
 - spot::tgba_bdd_core_data, 688
- negated_
 - spot::ltl::automatop, 143
- negative_normal_form
 - ltl_rewriting, 14
- never_claim_reachable
 - tgba_io, 26
- neverclaim_parse
 - tgba_io, 26
- neverclaim_parse_error
 - tgba_io, 24
- neverclaim_parse_error_list
 - tgba_io, 24
- neverclaimyy, 80
 - operator<=, 82
 - operator+, 81
 - operator+=, 81, 82
 - operator-, 82
 - operator=, 82
 - operator==, 82, 83
- neverclaimyy::location, 371
 - begin, 374
 - columns, 373
 - end, 374
 - initialize, 373
 - lines, 373
 - location, 373
 - step, 374
- neverclaimyy::position, 459
 - column, 461
 - columns, 460
 - filename, 461
 - initialize, 460
 - line, 461
 - lines, 460
 - position, 460
- neverclaimyy::slice, 550
 - range_, 551
 - slice, 551
 - stack_, 551
- neverclaimyy::stack, 562
 - begin, 563
 - const_iterator, 563
 - end, 563
 - height, 564

- iterator, 563
- pop, 564
- push, 564
- seq_, 564
- stack, 563
- neverparse/ Directory Reference, 67
- neverparse/location.hh, 940
- neverparse/position.hh, 946
- neverparse/public.hh, 953
- neverparse/stack.hh, 958
- next
 - spot::evtgba_iterator, 283
 - spot::explicit_state_conjunction, 313
 - spot::fair_kripke_succ_iterator, 326
 - spot::kripke_succ_iterator, 365
 - spot::loopless_modular_mixed_radix_gray_-code, 386
 - spot::ltl::ltl_file, 388
 - spot::minato_isop, 389
 - spot::numbered_state_heap_const_iterator, 422
 - spot::saba_state_conjunction, 522
 - spot::saba_succ_iterator, 528
 - spot::taa_succ_iterator, 616
 - spot::tgba_explicit_succ_iterator, 754
 - spot::tgba_succ_iterator, 855
 - spot::tgba_succ_iterator_concrete, 860
 - spot::tgba_succ_iterator_product, 866
 - spot::tgba_succ_iterator_union, 872
- next_non_false_
 - spot::tgba_succ_iterator_product, 866
- next_set
 - spot::tgba_bdd_core_data, 688
- next_state
 - spot::evtgba_reachable_iterator, 292
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 450
 - spot::parity_game_graph_direct, 457
 - spot::saba_reachable_iterator, 503
 - spot::saba_reachable_iterator_breadth_first, 509
 - spot::saba_reachable_iterator_depth_first, 515
 - spot::tgba_reachable_iterator, 782
 - spot::tgba_reachable_iterator_breadth_first, 788
 - spot::tgba_reachable_iterator_depth_first, 794
 - spot::tgba_reduc, 808
- next_to_now
 - spot::bdd_dict, 161
- nfa
 - spot::ltl::formula_tree::node_nfa, 415
 - spot::ltl::nfa, 401
- nfa_
 - spot::ltl::automatop, 143
- nips.hh
 - nipsvm_bytecode_t, 936
 - nipsvm_t, 936
- nips/ Directory Reference, 67
- nips/common.hh, 933
- nips/nips.hh, 935
- nips_exception
 - spot::nips_exception, 404
- nips_interface
 - spot::nips_interface, 407
- nipsvm_
 - spot::nips_interface, 408
- nipsvm_bytecode_t
 - nips.hh, 936
- nipsvm_t
 - nips.hh, 936
- node_ptr
 - spot::ltl::formula_tree, 114
- non_one_radixes_
 - spot::loopless_modular_mixed_radix_gray_-code, 387
- Not
 - spot::ltl::unop, 919
- not_win
 - spot::duplicator_node, 244
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 562
- notacc_set
 - spot::tgba_bdd_core_data, 689
- notnext_set
 - spot::tgba_bdd_core_data, 689
- notnow_set
 - spot::tgba_bdd_core_data, 689
- notvar_set
 - spot::tgba_bdd_core_data, 689
- now_formula_map
 - spot::bdd_dict, 161
- now_map
 - spot::bdd_dict, 161
- now_set
 - spot::tgba_bdd_core_data, 689
- now_to_next
 - spot::bdd_dict, 162
- nownext_set
 - spot::tgba_bdd_core_data, 689
- nrand
 - random, 22
- ns_map
 - spot::evtgba_explicit, 280

- spot::taa_tgba_formula, 630
- spot::taa_tgba_labelled, 642
- spot::taa_tgba_string, 653
- spot::tgba_explicit_formula, 707
- spot::tgba_explicit_labelled, 719
- spot::tgba_explicit_number, 731
- spot::tgba_explicit_string, 743
- spot::tgba_reduc, 802
- nth
 - spot::ltl::automatop, 142
 - spot::ltl::multop, 397
- num
 - spot::couvereur99_check_shy, 223
- num_
 - spot::duplicator_node, 244
 - spot::duplicator_node_delayed, 250
 - spot::scc_map, 538
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 562
- num_acc_
 - spot::saba, 494
 - spot::tgba, 668
- number_of_acceptance_conditions
 - spot::fair_kripke, 319
 - spot::future_conditions_collector, 342
 - spot::kripke, 358
 - spot::saba, 493
 - spot::saba_complement_tgba, 499
 - spot::taa_tgba, 623
 - spot::taa_tgba_formula, 634
 - spot::taa_tgba_labelled, 645
 - spot::taa_tgba_string, 657
 - spot::tgba, 666
 - spot::tgba_bdd_concrete, 675
 - spot::tgba_explicit, 699
 - spot::tgba_explicit_formula, 711
 - spot::tgba_explicit_labelled, 723
 - spot::tgba_explicit_number, 735
 - spot::tgba_explicit_string, 747
 - spot::tgba_kv_complement, 761
 - spot::tgba_product, 769
 - spot::tgba_product_init, 776
 - spot::tgba_reduc, 809
 - spot::tgba_safracomplement, 826
 - spot::tgba_sba_proxy, 834
 - spot::tgba_scc, 841
 - spot::tgba_sgba_proxy, 850
 - spot::tgba_tba_proxy, 879
 - spot::tgba_union, 888
- numbered_state_heap_hash_map_factory
 - spot::numbered_state_heap_hash_map_factory, 432
- o_
 - spot::couvereur99_check, 205
 - spot::couvereur99_check_result, 212
 - spot::couvereur99_check_shy, 223
 - spot::emptiness_check, 262
 - spot::emptiness_check_instantiator, 265
 - spot::emptiness_check_result, 270
- one_state_of
 - spot::scc_map, 536
- onepass_
 - spot::couvereur99_check_shy, 223
- op
 - spot::ltl::binop, 175
 - spot::ltl::formula_tree::node_binop, 411
 - spot::ltl::formula_tree::node_multop, 413
 - spot::ltl::formula_tree::node_unop, 417
 - spot::ltl::multop, 397
 - spot::ltl::unop, 920
- op_
 - spot::evtgba_product, 288
 - spot::ltl::binop, 176
 - spot::ltl::multop, 398
 - spot::ltl::unop, 921
- op_name
 - spot::ltl::binop, 175
 - spot::ltl::multop, 397
 - spot::ltl::unop, 921
- operator const symbol *
 - spot::rsymbol, 489
- operator<
 - spot::rsymbol, 490
- operator<<
 - eltlyy, 75, 76
 - ltlyy, 79
 - neverclaimyy, 82
 - sautyy, 85
 - spot, 107
 - spot::option_map, 437
 - spot::weight, 931
- operator*
 - spot::ltl::succ_iterator, 607
- operator()
 - spot::bdd_less_than, 163
 - spot::char_ptr_less_than, 177
 - spot::identity_hash, 352
 - spot::ltl::automatop::tripletcmp, 903
 - spot::ltl::formula_ptr_hash, 331
 - spot::ltl::formula_ptr_less_than, 331
 - spot::ltl::multop::paircmp, 437
 - spot::ptr_hash, 474
 - spot::saba_state_ptr_equal, 523
 - spot::saba_state_ptr_hash, 524
 - spot::saba_state_ptr_less_than, 524
 - spot::saba_state_shared_ptr_equal, 525
 - spot::saba_state_shared_ptr_hash, 526

- spot::saba_state_shared_ptr_less_than, 526
- spot::state_ptr_equal, 592
- spot::state_ptr_hash, 592
- spot::state_ptr_less_than, 593
- spot::state_shared_ptr_equal, 598
- spot::state_shared_ptr_hash, 599
- spot::state_shared_ptr_less_than, 599
- spot::string_hash, 606
- spot::taa_succ_iterator::distance_sort, 235
- operator+
 - eltlyy, 74, 75
 - ltlyy, 78
 - neverclaimyy, 81
 - sautyy, 84
- operator++
 - spot::ltl::succ_iterator, 607
- operator+=
 - eltlyy, 75
 - ltlyy, 78
 - neverclaimyy, 81, 82
 - sautyy, 84
 - spot::weight, 930
- operator-
 - eltlyy, 75
 - ltlyy, 78
 - neverclaimyy, 82
 - sautyy, 84
 - spot::weight, 930
- operator-=
 - eltlyy, 75
 - ltlyy, 79
 - neverclaimyy, 82
 - sautyy, 85
 - spot::weight, 930
- operator=
 - spot::bdd_dict, 158
 - spot::explicit_state_conjunction, 313
 - spot::ltl::nfa, 402
 - spot::rsymbol, 490
 - spot::taa_tgba, 623
 - spot::tgba_bdd_concrete, 675
 - spot::tgba_bdd_core_data, 687
 - spot::tgba_explicit, 699
 - spot::tgba_product, 769
 - spot::tgba_run, 815
 - spot::tgba_sgba_proxy, 850
 - spot::tgba_tba_proxy, 880
 - spot::tgba_union, 888
- operator==
 - eltlyy, 76
 - ltlyy, 79
 - neverclaimyy, 82, 83
 - spot::rsymbol, 490
 - spot::unsigned_statistics_copy, 926
- options
 - spot::couvereur99_check, 203
 - spot::couvereur99_check_result, 211
 - spot::couvereur99_check_shy, 221
 - spot::emptiness_check, 261
 - spot::emptiness_check_instantiator, 265
 - spot::emptiness_check_result, 269
- options_
 - spot::option_map, 437
- options_updated
 - spot::couvereur99_check, 203
 - spot::couvereur99_check_result, 211
 - spot::couvereur99_check_shy, 221
 - spot::emptiness_check, 261
 - spot::emptiness_check_result, 269
- Or
 - spot::ltl::multop, 395
- order
 - spot::bdd_ordered, 163
- order_
 - spot::bdd_ordered, 164
- out
 - spot::evtgba_explicit::state, 572
 - spot::evtgba_explicit::transition, 903
- output
 - spot::taa_tgba_formula, 634
 - spot::taa_tgba_labelled, 645
 - spot::taa_tgba_string, 657
- pair
 - spot::ltl::atomic_prop, 132
 - spot::ltl::binop, 173
 - spot::ltl::multop, 395
 - spot::ltl::unop, 918
- pair_state_iter
 - spot::scc_map, 534
- pairf
 - spot::ltl::binop, 173
- parity_game_graph
 - spot::parity_game_graph, 441
- parity_game_graph_delayed
 - spot::parity_game_graph_delayed, 449
- parity_game_graph_direct
 - spot::parity_game_graph_direct, 456
- parse
 - ltl_io, 7
- parse_error
 - ltl_io, 6
- parse_error_list
 - ltl_io, 6
- parse_file
 - ltl_io, 8
- parse_options
 - spot::couvereur99_check, 204

- spot::couvreur99_check_result, 212
- spot::couvreur99_check_shy, 221
- spot::emptiness_check, 261
- spot::emptiness_check_result, 269
- spot::ltl::random_ltl, 477
- spot::option_map, 436
- parse_string
 - ltl_io, 8
- pm
 - spot::weight, 931
- pop
 - eltlyy::stack, 569
 - ltlyy::stack, 571
 - neverclaimyy::stack, 564
 - sautyy::stack, 566
 - spot::scc_stack, 540
- poprem_
 - spot::couvreur99_check, 205
 - spot::couvreur99_check_shy, 223
- pos
 - spot::couvreur99_check_shy, 223
- pos_lenght_pair
 - spot::bdd_allocator, 149
 - spot::bdd_dict::anon_free_list, 123
 - spot::free_list, 335
- position
 - eltlyy::position, 464
 - ltlyy::position, 462
 - neverclaimyy::position, 460
 - sautyy::position, 466
- postfix_visitor
 - spot::ltl::postfix_visitor, 470
- power_map_data
 - spot::power_map, 472
- power_state
 - spot::power_map, 472
- prand
 - random, 22
- pred_iter
 - spot::evtgba, 275
 - spot::evtgba_explicit, 281
 - spot::evtgba_product, 287
- prefix
 - spot::tgba_run, 815
- prefix_states_
 - spot::ars_statistics, 128
- print
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 450
 - spot::parity_game_graph_direct, 457
 - spot::timer_map, 897
- print_stats
 - spot::couvreur99_check, 204
 - spot::couvreur99_check_result, 212
 - spot::couvreur99_check_shy, 221
 - spot::couvreur99_check_status, 226
 - spot::emptiness_check, 261
- print_tgba_run
 - tgba_run, 53
- proba
 - spot::ltl::random_ltl::op_proba, 434
- proba_
 - spot::ltl::random_ltl, 478
- proba_2_
 - spot::ltl::random_ltl, 478
- process_link
 - spot::evtgba_reachable_iterator, 292
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 450, 451
 - spot::parity_game_graph_direct, 457
 - spot::saba_reachable_iterator, 503
 - spot::saba_reachable_iterator_breadth_first, 509
 - spot::saba_reachable_iterator_depth_first, 515
 - spot::tgba_reachable_iterator, 782
 - spot::tgba_reachable_iterator_breadth_first, 788
 - spot::tgba_reachable_iterator_depth_first, 794
 - spot::tgba_reduc, 809
- process_state
 - spot::evtgba_reachable_iterator, 292
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::parity_game_graph, 442
 - spot::parity_game_graph_delayed, 451
 - spot::parity_game_graph_direct, 458
 - spot::saba_reachable_iterator, 504
 - spot::saba_reachable_iterator_breadth_first, 509
 - spot::saba_reachable_iterator_depth_first, 515
 - spot::tgba_reachable_iterator, 782
 - spot::tgba_reachable_iterator_breadth_first, 788
 - spot::tgba_reachable_iterator_depth_first, 794
 - spot::tgba_reduc, 809
- process_state_conjunction
 - spot::saba_reachable_iterator, 504
 - spot::saba_reachable_iterator_breadth_first, 510
 - spot::saba_reachable_iterator_depth_first, 516
- product
 - tgba_algorithms, 30

- progress_measure_
 - spot::duplicator_node_delayed, [251](#)
 - spot::spoiler_node_delayed, [562](#)
- project_state
 - spot::fair_kripke, [319](#)
 - spot::future_conditions_collector, [342](#)
 - spot::kripke, [358](#)
 - spot::taa_tgba, [624](#)
 - spot::taa_tgba_formula, [634](#)
 - spot::taa_tgba_labelled, [646](#)
 - spot::taa_tgba_string, [657](#)
 - spot::tgba, [666](#)
 - spot::tgba_bdd_concrete, [675](#)
 - spot::tgba_explicit, [700](#)
 - spot::tgba_explicit_formula, [711](#)
 - spot::tgba_explicit_labelled, [723](#)
 - spot::tgba_explicit_number, [735](#)
 - spot::tgba_explicit_string, [747](#)
 - spot::tgba_kv_complement, [761](#)
 - spot::tgba_product, [769](#)
 - spot::tgba_product_init, [776](#)
 - spot::tgba_reduc, [809](#)
 - spot::tgba_safra_complement, [826](#)
 - spot::tgba_sba_proxy, [834](#)
 - spot::tgba_scc, [842](#)
 - spot::tgba_sgba_proxy, [850](#)
 - spot::tgba_tba_proxy, [880](#)
 - spot::tgba_union, [888](#)
- project_tgba_run
 - tgba_run, [53](#)
- prop_map
 - spot::ltl::declarative_environment, [230](#)
- props_
 - spot::ltl::declarative_environment, [231](#)
- prune
 - spot::duplicator_node, [243](#)
 - spot::duplicator_node_delayed, [249](#)
 - spot::spoiler_node, [555](#)
 - spot::spoiler_node_delayed, [561](#)
- ptr
 - spot::ltl::nfa, [400](#)
- push
 - eltlyy::stack, [569](#)
 - ltlyy::stack, [571](#)
 - neverclaimyy::stack, [564](#)
 - sautyy::stack, [566](#)
 - spot::scc_stack, [540](#)
- q
 - spot::couvereur99_check_shy::todo_item, [899](#)
- quote_unless_bare_word
 - misc_tools, [19](#)
- quotient_state
 - spot::tgba_reduc, [810](#)
- R
 - spot::ltl::binop, [173](#)
- rand
 - spot::barand, [144](#)
- random
 - bmrand, [22](#)
 - drand, [22](#)
 - mrnd, [22](#)
 - nrnd, [22](#)
 - prand, [22](#)
 - rrand, [23](#)
 - srnd, [23](#)
- Random functions, [21](#)
- random_graph
 - tgba_misc, [41](#)
- random_ltl
 - spot::ltl::random_ltl, [477](#)
- range_
 - eltlyy::slice, [549](#)
 - ltlyy::slice, [552](#)
 - neverclaimyy::slice, [551](#)
 - sautyy::slice, [550](#)
- recurse
 - spot::ltl::clone_visitor, [180](#)
 - spot::ltl::simplify_f_g_visitor, [547](#)
 - spot::ltl::unabbreviate_logic_visitor, [908](#)
 - spot::ltl::unabbreviate_ltl_visitor, [913](#)
- redirect_transition
 - spot::tgba_reduc, [810](#)
- reduc_tgba_sim
 - tgba_reduction, [40](#)
- reduce
 - ltl_rewriting, [14](#)
- Reduce_All
 - ltl_rewriting, [13](#)
 - tgba_reduction, [37](#)
- 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, [37](#)
- Reduce_quotient_Del_Sim
 - tgba_reduction, [37](#)
- Reduce_quotient_Dir_Sim
 - tgba_reduction, [37](#)
- Reduce_Scc
 - tgba_reduction, [37](#)
- Reduce_Syntactic_Implications

- ltl_rewriting, [13](#)
- Reduce_transition_De1_Sim
 - tgba_reduction, [37](#)
- Reduce_transition_Dir_Sim
 - tgba_reduction, [37](#)
- reduce_options
 - ltl_rewriting, [13](#)
- reduce_run
 - tgba_run, [54](#)
- reduce_tau03
 - spot::ltl, [113](#)
- reduce_tgba_options
 - tgba_reduction, [37](#)
- ref
 - spot::symbol, [611](#)
- ref_
 - spot::ltl::atomic_prop, [134](#)
 - spot::ltl::automatop, [142](#)
 - spot::ltl::binop, [175](#)
 - spot::ltl::constant, [196](#)
 - spot::ltl::formula, [330](#)
 - spot::ltl::multop, [397](#)
 - spot::ltl::ref_formula, [487](#)
 - spot::ltl::unop, [921](#)
- ref_count_
 - spot::ltl::atomic_prop, [134](#)
 - spot::ltl::automatop, [142](#)
 - spot::ltl::binop, [175](#)
 - spot::ltl::multop, [398](#)
 - spot::ltl::ref_formula, [487](#)
 - spot::ltl::unop, [921](#)
 - spot::symbol, [611](#)
- ref_counter_
 - spot::ltl::ref_formula, [487](#)
- ref_formula
 - spot::ltl::ref_formula, [485](#)
- ref_set
 - spot::bdd_dict, [157](#)
- refs_
 - spot::symbol, [611](#)
- register_acceptance_variable
 - spot::bdd_dict, [158](#)
- register_acceptance_variables
 - spot::bdd_dict, [158](#)
- register_all_variables_of
 - spot::bdd_dict, [159](#)
- register_anonymous_variables
 - spot::bdd_dict, [159](#)
- register_clone_acc
 - spot::bdd_dict, [159](#)
- register_formula_
 - spot::ltl::language_containment_checker, [368](#)
- register_n
 - spot::bdd_allocator, [150](#)
 - spot::bdd_dict::anon_free_list, [123](#)
 - spot::free_list, [335](#)
- register_proposition
 - spot::bdd_dict, [159](#)
- register_propositions
 - spot::bdd_dict, [159](#)
- register_state
 - spot::bdd_dict, [160](#)
- relabel_component
 - spot::scc_map, [536](#)
- relation
 - spot::tgba_bdd_core_data, [689](#)
- release_n
 - spot::bdd_allocator, [150](#)
 - spot::bdd_dict::anon_free_list, [124](#)
 - spot::free_list, [336](#)
- release_variables
 - spot::bdd_allocator, [150](#)
 - spot::bdd_dict, [160](#)
- rem
 - spot::connected_component_hash_set, [187](#)
 - spot::explicit_connected_component, [306](#)
 - spot::scc_stack, [540](#)
 - spot::scc_stack::connected_component, [183](#)
- remove
 - spot::bdd_allocator, [151](#)
 - spot::bdd_dict::anon_free_list, [124](#)
 - spot::free_list, [336](#)
- remove_acc
 - spot::tgba_reduc, [810](#)
- remove_component
 - spot::couvreur99_check, [204](#)
 - spot::couvreur99_check_shy, [221](#)
 - spot::tgba_reduc, [810](#)
- remove_predecessor_state
 - spot::tgba_reduc, [810](#)
- remove_scc
 - spot::tgba_reduc, [810](#)
- remove_state
 - spot::tgba_reduc, [810](#)
- removed_components
 - spot::couvreur99_check, [205](#)
 - spot::couvreur99_check_shy, [223](#)
- replay_tgba_run
 - tgba_run, [54](#)
- require
 - spot::ltl::declarative_environment, [230](#)
 - spot::ltl::default_environment, [234](#)
 - spot::ltl::environment, [272](#)
- result
 - spot::couvreur99_check, [204](#)
 - spot::couvreur99_check_shy, [221](#)
 - spot::ltl::clone_visitor, [180](#)
 - spot::ltl::simplify_f_g_visitor, [547](#)

- spot::ltl::unabbreviate_logic_visitor, 908
- spot::ltl::unabbreviate_ltl_visitor, 913
- result_
 - spot::ltl::clone_visitor, 181
 - spot::ltl::simplify_f_g_visitor, 548
 - spot::ltl::unabbreviate_logic_visitor, 909
 - spot::ltl::unabbreviate_ltl_visitor, 914
- ret_
 - spot::minato_isop, 390
- Rewriting LTL formulae, 13
- rhs
 - spot::ltl::formula_tree::node_binop, 411
 - spot::ltl::formula_tree::node_multop, 413
- right
 - spot::state_product, 591
 - spot::state_union, 604
- right_
 - spot::state_product, 591
 - spot::state_union, 604
 - spot::tgba_product, 771
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_succ_iterator_union, 872
 - spot::tgba_union, 890
- right_acc_complement_
 - spot::tgba_product, 771
 - spot::tgba_union, 890
- right_acc_missing_
 - spot::tgba_union, 891
- right_common_acc_
 - spot::tgba_product, 771
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_union, 891
- right_init_
 - spot::tgba_product_init, 778
- right_missing_
 - spot::tgba_succ_iterator_union, 873
- right_neg_
 - spot::tgba_succ_iterator_product, 867
 - spot::tgba_succ_iterator_union, 873
- right_var_missing_
 - spot::tgba_union, 891
- root
 - spot::couvereur99_check_status, 227
- root_
 - spot::scc_map, 538
- rrand
 - random, 23
- rsymbol
 - spot::rsymbol, 489
- rsymbol_set
 - spot, 100
- run
 - spot::evtgba_reachable_iterator, 293
 - spot::evtgba_reachable_iterator_breadth_first, 297
 - spot::evtgba_reachable_iterator_depth_first, 302
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 451
 - spot::parity_game_graph_direct, 458
 - spot::saba_reachable_iterator, 504
 - spot::saba_reachable_iterator_breadth_first, 510
 - spot::saba_reachable_iterator_depth_first, 516
 - spot::tgba_reachable_iterator, 783
 - spot::tgba_reachable_iterator_breadth_first, 789
 - spot::tgba_reachable_iterator_depth_first, 795
 - spot::tgba_reduc, 811
- run_
 - spot::couvereur99_check_result, 212
 - spot::tgba_run_dotty_decorator, 820
- running
 - spot::timer, 895
- s
 - spot::couvereur99_check_shy::successor, 609
 - spot::couvereur99_check_shy::todo_item, 899
 - spot::scc_stack, 540
 - spot::tgba_run::step, 605
- s_
 - spot::barand, 144
 - spot::loopless_modular_mixed_radix_gray_code, 387
 - spot::rsymbol, 490
 - spot::state_set, 597
 - spot::tgba_explicit_succ_iterator, 755
- s_v
 - tgba_reduction, 37
- saba
 - spot::saba, 492
- SABA (State-based Alternating Büchi Automata), 3
- saba/ Directory Reference, 68
- saba/explicitstateconjunction.hh, 1027
- saba/saba.hh, 1028
- saba/sabacomplementtgba.hh, 1029
- saba/sabastate.hh, 1030
- saba/sabasucciter.hh, 1032
- saba_complement_tgba
 - spot::saba_complement_tgba, 498
- saba_dotty_reachable
 - spot, 107
- saba_reachable_iterator
 - spot::saba_reachable_iterator, 503
- saba_reachable_iterator_breadth_first
 - spot::saba_reachable_iterator_breadth_first, 509

- saba_reachable_iterator_depth_first
 - spot::saba_reachable_iterator_depth_first, 515
- saba_state_set_t
 - spot::explicit_state_conjunction, 311
- sabaalgorithms/ Directory Reference, 69
- sabaalgorithms/sabadotty.hh, 1034
- sabaalgorithms/sabareachiter.hh, 1034
- safe
 - spot::couvreur99_check, 204
 - spot::couvreur99_check_shy, 222
 - spot::emptiness_check, 262
- safra_
 - spot::tgba_safra_complement, 828
- sautparse/ Directory Reference, 69
- sautparse/location.hh, 941
- sautparse/position.hh, 948
- sautparse/stack.hh, 958
- sautty, 83
 - operator<<, 85
 - operator+, 84
 - operator+=, 84
 - operator-, 84
 - operator=, 85
- sautty::location, 380
 - begin, 383
 - columns, 382
 - end, 383
 - initialize, 382
 - lines, 382
 - location, 382
 - step, 383
- sautty::position, 465
 - column, 467
 - columns, 466
 - filename, 467
 - initialize, 466
 - line, 467
 - lines, 466
 - position, 466
- sautty::slice, 549
 - range_, 550
 - slice, 549
 - stack_, 550
- sautty::stack, 565
 - begin, 566
 - const_iterator, 565
 - end, 566
 - height, 566
 - iterator, 565
 - pop, 566
 - push, 566
 - seq_, 567
 - stack, 565
- sc_
 - spot::duplicator_node, 244
 - spot::duplicator_node_delayed, 251
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 562
- scc
 - spot::scc_map::scc, 529
- scc_count
 - spot::scc_map, 536
- scc_filter
 - spot, 107
- scc_map
 - spot::scc_map, 534
- scc_map_
 - spot::future_conditions_collector, 344
 - spot::scc_map, 538
 - spot::tgba_scc, 843
- scc_map_type
 - spot::scc_map, 534
- scc_of_state
 - spot::future_conditions_collector, 343
 - spot::scc_map, 536
 - spot::tgba_scc, 842
- scc_total
 - spot::scc_stats, 542
- sccs
 - spot::sccs_set, 543
- se05
 - emptiness_check_algorithms, 52
- search
 - spot::bfs_steps, 167
- second
 - spot::ltl::binop, 175, 176
- second_
 - spot::ltl::binop, 176
- SecondStep
 - spot::minato_isop::local_vars, 370
- seen
 - spot::evtgba_reachable_iterator, 293
 - spot::evtgba_reachable_iterator_breadth_first, 298
 - spot::evtgba_reachable_iterator_depth_first, 303
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 452
 - spot::parity_game_graph_direct, 459
 - spot::saba_reachable_iterator, 505
 - spot::saba_reachable_iterator_breadth_first, 510
 - spot::saba_reachable_iterator_depth_first, 516
 - spot::tgba_reachable_iterator, 783
 - spot::tgba_reachable_iterator_breadth_first, 789
 - spot::tgba_reachable_iterator_depth_first, 795
 - spot::tgba_reduc, 813

- seen_
 - spot::duplicator_node_delayed, 251
 - spot::spoiler_node_delayed, 562
 - spot::taa_succ_iterator, 617
- seen_map
 - spot::evtgba_reachable_iterator, 291
 - spot::evtgba_reachable_iterator_breadth_first, 296
 - spot::evtgba_reachable_iterator_depth_first, 301
 - spot::parity_game_graph, 441
 - spot::parity_game_graph_delayed, 448
 - spot::parity_game_graph_direct, 456
 - spot::saba_reachable_iterator, 502
 - spot::saba_reachable_iterator_breadth_first, 508
 - spot::saba_reachable_iterator_depth_first, 514
 - spot::taa_succ_iterator, 615
 - spot::tgba_reachable_iterator, 781
 - spot::tgba_reachable_iterator_breadth_first, 787
 - spot::tgba_reachable_iterator_depth_first, 793
 - spot::tgba_reduc, 802
- self_loops
 - spot::scc_map, 536
 - spot::scc_stats, 542
- self_loops_
 - spot::scc_map, 538
- seq_
 - eltlyy::stack, 569
 - ltlyy::stack, 571
 - neverclaimyy::stack, 564
 - sautyy::stack, 567
- set
 - spot::option_map, 436
 - spot::unsigned_statistics_copy, 926
- set_
 - spot::explicit_state_conjunction, 313
- set_acceptance_conditions
 - spot::tgba_explicit, 700
 - spot::tgba_explicit_formula, 711
 - spot::tgba_explicit_labelled, 723
 - spot::tgba_explicit_number, 735
 - spot::tgba_explicit_string, 748
 - spot::tgba_reduc, 811
- set_final
 - spot::ltl::nfa, 402
- set_init_state
 - spot::evtgba_explicit, 282
 - spot::ltl::nfa, 402
 - spot::taa_tgba_formula, 634
 - spot::taa_tgba_labelled, 646
 - spot::taa_tgba_string, 657, 658
 - spot::tgba_bdd_concrete, 675
 - spot::tgba_explicit_formula, 711
 - spot::tgba_explicit_labelled, 724
 - spot::tgba_explicit_number, 735
 - spot::tgba_explicit_string, 748
 - spot::tgba_reduc, 811
- set_lead_2_acc_all
 - spot::duplicator_node_delayed, 249
 - spot::spoiler_node_delayed, 561
- set_name
 - spot::ltl::nfa, 402
- set_states
 - spot::couvereur99_check, 204
 - spot::couvereur99_check_shy, 222
 - spot::ec_statistics, 255
- set_type
 - spot::connected_component_hash_set, 186
- set_win
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node, 555
 - spot::spoiler_node_delayed, 561
- seteq
 - spot::unsigned_statistics_copy, 926
- setup
 - spot::ltl::random_ltl::op_proba, 434
- shared_saba_state
 - spot, 100
- shared_state
 - spot, 100
- shared_state_deleter
 - spot, 108
- show_
 - spot::future_conditions_collector, 344
 - spot::tgba_scc, 843
- si_
 - spot::ltl::nfa, 403
- si_map
 - spot::ltl::nfa, 400
- simplify_f_g
 - ltl_rewriting, 14
- simplify_f_g_visitor
 - spot::ltl::simplify_f_g_visitor, 546
- simulation_relation
 - spot, 101
- size
 - spot::ltl::automatop, 142
 - spot::ltl::multop, 398
 - spot::numbered_state_heap, 420
 - spot::numbered_state_heap_hash_map, 429
 - spot::scc_stack, 540
 - spot::sccs_set, 543
- slice
 - eltlyy::slice, 548
 - ltlyy::slice, 552

- neverclaimyy::slice, 551
- sautyy::slice, 549
- sn_map
 - spot::evtgba_explicit, 280
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 653
 - spot::tgba_explicit_formula, 707
 - spot::tgba_explicit_labelled, 719
 - spot::tgba_explicit_number, 731
 - spot::tgba_explicit_string, 743
 - spot::tgba_reduc, 802
- sn_v
 - tgba_reduction, 37
- sp_map
 - spot::tgba_reduc, 802
- spair
 - ltl_io, 6
- split_tgba
 - spot, 108
- spoiler_node
 - spot::spoiler_node, 554
- spoiler_node_delayed
 - spot::spoiler_node_delayed, 559
- spoiler_vertice_
 - spot::parity_game_graph, 444
 - spot::parity_game_graph_delayed, 452
 - spot::parity_game_graph_direct, 459
- spot, 85
 - acc_list_t, 100
 - bdd_format_accset, 101
 - bdd_format_formula, 101
 - bdd_format_sat, 101
 - bdd_format_set, 102
 - bdd_print_acc, 102
 - bdd_print_accset, 102
 - bdd_print_dot, 103
 - bdd_print_formula, 103
 - bdd_print_sat, 103
 - bdd_print_set, 104
 - bdd_print_table, 104
 - bdd_to_formula, 104
 - build_scc_stats, 104
 - compute_all_acceptance_conditions, 104
 - compute_neg_acceptance_conditions, 104
 - display_safra, 105
 - dotty_reachable, 105
 - dump_scc_dot, 105
 - evtgba_parse, 105
 - evtgba_parse_error, 100
 - evtgba_parse_error_list, 100
 - evtgba_save_reachable, 106
 - find_paths, 106
 - format_evtgba_parse_errors, 106
 - formula_to_bdd, 106
 - is_guarantee_automaton, 106
 - is_safety_mwdba, 107
 - max_spanning_paths, 107
 - memusage, 107
 - operator<<, 107
 - rsymbol_set, 100
 - saba_dotty_reachable, 107
 - scc_filter, 107
 - shared_saba_state, 100
 - shared_state, 100
 - shared_state_deleter, 108
 - simulation_relation, 101
 - split_tgba, 108
 - state_couple, 101
 - symbol_set, 101
 - tgba_to_evtgba, 108
- spot::acss_statistics, 115
 - ~acss_statistics, 118
- acss_states, 118
- acss_statistics, 118
- ars_cycle_states, 119
- ars_prefix_states, 119
- get, 119
- inc_ars_cycle_states, 119
- inc_ars_prefix_states, 119
- stats, 119
- stats_map, 118
- unsigned_fun, 118
- spot::ars_statistics, 124
 - ars_cycle_states, 127
 - ars_prefix_states, 127
 - ars_statistics, 127
 - cycle_states_, 128
 - get, 127
 - inc_ars_cycle_states, 128
 - inc_ars_prefix_states, 128
 - prefix_states_, 128
 - stats, 128
 - stats_map, 127
 - unsigned_fun, 127
- spot::barand, 143
 - barand, 144
 - m_, 144
 - n_, 144
 - rand, 144
 - s_, 144
- spot::bdd_allocator, 145
 - allocate_variables, 149
 - bdd_allocator, 149
 - dump_free_list, 149
 - extend, 149
 - extvarnum, 150
 - fl, 151

- free_count, 150
- free_list_type, 149
- initialize, 150
- initialized, 151
- insert, 150
- lvarnum, 151
- pos_lenght_pair, 149
- register_n, 150
- release_n, 150
- release_variables, 150
- remove, 151
- spot::bdd_dict, 151
 - ~bdd_dict, 157
 - acc_formula_map, 161
 - acc_map, 161
 - allocate_variables, 157
 - assert_emptiness, 157
 - bdd_dict, 157
 - cc_map, 156
 - clone_counts, 161
 - dump, 157
 - free_anonymous_list_of, 161
 - free_anonymous_list_of_type, 156
 - fv_map, 156
 - initialize, 158
 - initialized, 161
 - is_registered_acceptance_variable, 158
 - is_registered_proposition, 158
 - is_registered_state, 158
 - lvarnum, 161
 - next_to_now, 161
 - now_formula_map, 161
 - now_map, 161
 - now_to_next, 162
 - operator=, 158
 - ref_set, 157
 - register_acceptance_variable, 158
 - register_acceptance_variables, 158
 - register_all_variables_of, 159
 - register_anonymous_variables, 159
 - register_clone_acc, 159
 - register_proposition, 159
 - register_propositions, 159
 - register_state, 160
 - release_variables, 160
 - unregister_all_my_variables, 160
 - unregister_variable, 160
 - var_formula_map, 162
 - var_map, 162
 - var_refs, 162
 - vf_map, 157
 - vr_map, 157
- spot::bdd_dict::anon_free_list, 119
 - anon_free_list, 123
 - dict_, 124
 - dump_free_list, 123
 - extend, 123
 - fl, 124
 - free_count, 123
 - free_list_type, 123
 - insert, 123
 - pos_lenght_pair, 123
 - register_n, 123
 - release_n, 124
 - remove, 124
- spot::bdd_less_than, 162
 - operator(), 163
- spot::bdd_ordered, 163
 - bdd_, 164
 - bdd_ordered, 163
 - get_bdd, 163
 - order, 163
 - order_, 164
- spot::bfs_steps, 164
 - ~bfs_steps, 166
 - a_, 167
 - bfs_steps, 166
 - filter, 166
 - finalize, 167
 - match, 167
 - search, 167
- spot::char_ptr_less_than, 177
 - operator(), 177
- spot::connected_component_hash_set, 183
 - ~connected_component_hash_set, 186
 - condition, 187
 - has_state, 186
 - index, 187
 - insert, 186
 - rem, 187
 - set_type, 186
 - states, 187
- spot::connected_component_hash_set_factory, 187
 - ~connected_component_hash_set_factory, 190
 - build, 190
 - connected_component_hash_set_factory, 190
 - instance, 190
- spot::couvreur99_check, 197
 - ~couvreur99_check, 201
 - a_, 205
 - automaton, 202
 - check, 202
 - couvreur99_check, 201
 - dec_depth, 202
 - depth, 202
 - ecs_, 205
 - get, 202

- get_removed_components, 203
- get_vmsize, 203
- inc_depth, 203
- inc_states, 203
- inc_transitions, 203
- max_depth, 203
- o_, 205
- options, 203
- options_updated, 203
- parse_options, 204
- poprem_, 205
- print_stats, 204
- remove_component, 204
- removed_components, 205
- result, 204
- safe, 204
- set_states, 204
- states, 204
- statistics, 205
- stats, 205
- stats_map, 201
- transitions, 205
- unsigned_fun, 201
- spot::couvreur99_check_result, 206
 - a_, 212
 - accepting_cycle, 210
 - accepting_run, 210
 - acss_states, 210
 - ars_cycle_states, 211
 - ars_prefix_states, 211
 - automaton, 211
 - couvreur99_check_result, 210
 - ecs_, 212
 - get, 211
 - inc_ars_cycle_states, 211
 - inc_ars_prefix_states, 211
 - o_, 212
 - options, 211
 - options_updated, 211
 - parse_options, 212
 - print_stats, 212
 - run_, 212
 - statistics, 212
 - stats, 212
 - stats_map, 210
 - unsigned_fun, 210
- spot::couvreur99_check_shy, 213
 - ~couvreur99_check_shy, 218
 - a_, 222
 - arc, 222
 - automaton, 219
 - check, 219
 - clear_todo, 219
 - couvreur99_check_shy, 218
 - dec_depth, 219
 - depth, 219
 - dump_queue, 219
 - ecs_, 223
 - find_state, 220
 - get, 220
 - get_removed_components, 220
 - get_vmsize, 220
 - group2_, 223
 - group_, 223
 - inc_depth, 220
 - inc_states, 220
 - inc_transitions, 220
 - max_depth, 220
 - num, 223
 - o_, 223
 - onepass_, 223
 - options, 221
 - options_updated, 221
 - parse_options, 221
 - poprem_, 223
 - pos, 223
 - print_stats, 221
 - remove_component, 221
 - removed_components, 223
 - result, 221
 - safe, 222
 - set_states, 222
 - states, 222
 - statistics, 222
 - stats, 224
 - stats_map, 218
 - succ_queue, 218
 - todo, 224
 - todo_list, 218
 - transitions, 222
 - unsigned_fun, 218
- spot::couvreur99_check_shy::successor, 607
 - acc, 609
 - s, 609
 - successor, 608
- spot::couvreur99_check_shy::todo_item, 898
 - n, 899
 - q, 899
 - s, 899
 - todo_item, 899
- spot::couvreur99_check_status, 224
 - ~couvreur99_check_status, 226
 - aut, 226
 - couvreur99_check_status, 226
 - cycle_seed, 226
 - h, 227
 - print_stats, 226
 - root, 227

- states, 226
- spot::delayed_simulation_relation, 235
- spot::direct_simulation_relation, 235
- spot::dotty_decorator, 235
 - ~dotty_decorator, 237
 - dotty_decorator, 237
 - instance, 237
 - link_decl, 237
 - state_decl, 237
- spot::duplicator_node, 238
 - ~duplicator_node, 241
 - acc_, 243
 - add_pred, 242
 - add_succ, 242
 - compare, 242
 - del_pred, 242
 - del_succ, 242
 - duplicator_node, 241
 - get_acc, 242
 - get_duplicator_node, 242
 - get_label, 242
 - get_nb_succ, 242
 - get_pair, 242
 - get_spoiler_node, 243
 - implies, 243
 - label_, 243
 - lnode_pred, 243
 - lnode_succ, 244
 - match, 243
 - not_win, 244
 - num_, 244
 - prune, 243
 - sc_, 244
 - set_win, 243
 - succ_to_string, 243
 - to_string, 243
- spot::duplicator_node_delayed, 244
 - ~duplicator_node_delayed, 248
 - acc_, 250
 - add_pred, 248
 - add_succ, 248
 - compare, 248
 - del_pred, 248
 - del_succ, 248
 - duplicator_node_delayed, 248
 - get_acc, 248
 - get_duplicator_node, 248
 - get_label, 248
 - get_lead_2_acc_all, 249
 - get_nb_succ, 249
 - get_pair, 249
 - get_progress_measure, 249
 - get_spoiler_node, 249
 - implies, 249
 - implies_acc, 249
 - implies_label, 249
 - label_, 250
 - lead_2_acc_all_, 250
 - lnode_pred, 250
 - lnode_succ, 250
 - match, 249
 - not_win, 250
 - num_, 250
 - progress_measure_, 251
 - prune, 249
 - sc_, 251
 - seen_, 251
 - set_lead_2_acc_all, 249
 - set_win, 250
 - succ_to_string, 250
 - to_string, 250
- spot::ec_statistics, 251
 - dec_depth, 255
 - depth, 255
 - depth_, 256
 - ec_statistics, 254
 - get, 255
 - inc_depth, 255
 - inc_states, 255
 - inc_transitions, 255
 - max_depth, 255
 - max_depth_, 256
 - set_states, 255
 - states, 255
 - states_, 256
 - stats, 256
 - stats_map, 254
 - transitions, 256
 - transitions_, 256
 - unsigned_fun, 254
- spot::eltl, 108
- spot::emptiness_check, 257
 - ~emptiness_check, 260
 - a_, 262
 - automaton, 261
 - check, 261
 - emptiness_check, 260
 - o_, 262
 - options, 261
 - options_updated, 261
 - parse_options, 261
 - print_stats, 261
 - safe, 262
 - statistics, 262
- spot::emptiness_check_instantiator, 262
 - construct, 264
 - emptiness_check_instantiator, 264
 - info_, 265

- instantiate, 264
- max_acceptance_conditions, 264
- min_acceptance_conditions, 265
- o_, 265
- options, 265
- spot::emptiness_check_result, 265
 - ~emptiness_check_result, 268
 - a_, 270
 - accepting_run, 269
 - automaton, 269
 - emptiness_check_result, 268
 - o_, 270
 - options, 269
 - options_updated, 269
 - parse_options, 269
 - statistics, 269
- spot::evtgba, 272
 - ~evtgba, 274
 - all_acceptance_conditions, 274
 - alphabet, 274
 - evtgba, 274
 - format_acceptance_condition, 274
 - format_acceptance_conditions, 275
 - format_label, 275
 - format_state, 275
 - init_iter, 275
 - pred_iter, 275
 - succ_iter, 275
- spot::evtgba_explicit, 276
 - ~evtgba_explicit, 280
 - acc_set_, 282
 - add_transition, 280
 - all_acceptance_conditions, 280
 - alphabet, 281
 - alphabet_, 282
 - declare_acceptance_condition, 281
 - declare_state, 281
 - evtgba_explicit, 280
 - format_acceptance_condition, 281
 - format_acceptance_conditions, 281
 - format_label, 281
 - format_state, 281
 - init_iter, 281
 - init_states_, 282
 - name_state_map_, 282
 - ns_map, 280
 - pred_iter, 281
 - set_init_state, 282
 - sn_map, 280
 - state_name_map_, 282
 - succ_iter, 282
 - transition_list, 280
- spot::evtgba_explicit::state, 572
 - in, 572
 - out, 572
- spot::evtgba_explicit::transition, 901
 - acceptance_conditions, 903
 - in, 903
 - label, 903
 - out, 903
- spot::evtgba_iterator, 282
 - ~evtgba_iterator, 283
 - current_acceptance_conditions, 283
 - current_label, 283
 - current_state, 283
 - done, 283
 - first, 283
 - next, 283
- spot::evtgba_product, 284
 - ~evtgba_product, 286
 - all_acc_, 288
 - all_acceptance_conditions, 287
 - alphabet, 287
 - alphabet_, 288
 - common_symbol_table, 286
 - common_symbols_, 288
 - evtgba_product, 286
 - evtgba_product_operands, 286
 - format_acceptance_condition, 287
 - format_acceptance_conditions, 287
 - format_label, 287
 - format_state, 287
 - init_iter, 287
 - op_, 288
 - pred_iter, 287
 - succ_iter, 288
- spot::evtgba_reachable_iterator, 288
 - ~evtgba_reachable_iterator, 292
 - add_state, 292
 - automata_, 293
 - end, 292
 - evtgba_reachable_iterator, 292
 - next_state, 292
 - process_link, 292
 - process_state, 292
 - run, 293
 - seen, 293
 - seen_map, 291
 - start, 293
- spot::evtgba_reachable_iterator_breadth_first, 293
 - add_state, 297
 - automata_, 298
 - end, 297
 - evtgba_reachable_iterator_breadth_first, 297
 - next_state, 297
 - process_link, 297
 - process_state, 297
 - run, 297

- seen, 298
- seen_map, 296
- start, 298
- todo, 298
- spot::evtgba_reachable_iterator_depth_first, 298
 - add_state, 302
 - automata_, 303
 - end, 302
 - evtgba_reachable_iterator_depth_first, 302
 - next_state, 302
 - process_link, 302
 - process_state, 302
 - run, 302
 - seen, 303
 - seen_map, 301
 - start, 303
 - todo, 303
- spot::explicit_connected_component, 303
 - ~explicit_connected_component, 306
 - condition, 306
 - has_state, 306
 - index, 306
 - insert, 306
 - rem, 306
- spot::explicit_connected_component_factory, 307
 - ~explicit_connected_component_factory, 307
 - build, 308
- spot::explicit_state_conjunction, 308
 - ~explicit_state_conjunction, 312
 - add, 312
 - clone, 312
 - current_state, 312
 - done, 312
 - explicit_state_conjunction, 312
 - first, 312
 - it_, 313
 - next, 313
 - operator=, 313
 - saba_state_set_t, 311
 - set_, 313
- spot::fair_kripke, 313
 - all_acceptance_conditions, 317
 - compute_support_conditions, 317
 - compute_support_variables, 317
 - format_state, 318
 - get_dict, 318
 - get_init_state, 318
 - neg_acceptance_conditions, 318
 - number_of_acceptance_conditions, 319
 - project_state, 319
 - state_acceptance_conditions, 319
 - state_condition, 319
 - succ_iter, 319
 - support_conditions, 320
 - support_variables, 320
 - transition_annotation, 320
- spot::fair_kripke_succ_iterator, 321
 - ~fair_kripke_succ_iterator, 324
 - acc_cond_, 326
 - cond_, 326
 - current_acceptance_conditions, 325
 - current_condition, 325
 - current_state, 325
 - done, 325
 - fair_kripke_succ_iterator, 324
 - first, 325
 - next, 326
- spot::free_list, 332
 - ~free_list, 335
 - dump_free_list, 335
 - extend, 335
 - fl, 336
 - free_count, 335
 - free_list_type, 335
 - insert, 335
 - pos_lenght_pair, 335
 - register_n, 335
 - release_n, 336
 - remove, 336
- spot::future_conditions_collector, 336
 - ~future_conditions_collector, 340
 - all_acceptance_conditions, 341
 - aut_, 344
 - compute_support_conditions, 341
 - compute_support_variables, 341
 - cond_set, 340
 - fc_map, 340
 - format_state, 341
 - future_conditions, 341
 - future_conditions_collector, 340
 - future_conds_, 344
 - get_dict, 341
 - get_init_state, 342
 - map_builder_, 342
 - neg_acceptance_conditions, 342
 - number_of_acceptance_conditions, 342
 - project_state, 342
 - scc_map_, 344
 - scc_of_state, 343
 - show_, 344
 - succ_iter, 343
 - support_conditions, 343
 - support_variables, 344
 - transition_annotation, 344
- spot::gspn_exception, 345
 - err_, 345
 - get_err, 345
 - get_where, 345

- gspn_exception, 345
 - where_, 345
- spot::gspn_interface, 346
 - ~gspn_interface, 348
 - automaton, 348
 - dead_, 348
 - dict_, 348
 - env_, 348
 - gspn_interface, 348
- spot::gspn_ssp_interface, 349
 - ~gspn_ssp_interface, 351
 - automaton, 351
 - dict_, 351
 - env_, 351
 - gspn_ssp_interface, 351
- spot::identity_hash, 351
 - operator(), 352
- spot::kripke, 352
 - ~kripke, 356
 - all_acceptance_conditions, 356
 - compute_support_conditions, 356
 - compute_support_variables, 356
 - format_state, 357
 - get_dict, 357
 - get_init_state, 357
 - neg_acceptance_conditions, 357
 - number_of_acceptance_conditions, 358
 - project_state, 358
 - state_acceptance_conditions, 358
 - state_condition, 358
 - succ_iter, 358
 - support_conditions, 359
 - support_variables, 359
 - transition_annotation, 359
- spot::kripke_succ_iterator, 360
 - ~kripke_succ_iterator, 363
 - cond_, 365
 - current_acceptance_conditions, 364
 - current_condition, 364
 - current_state, 364
 - done, 364
 - first, 364
 - kripke_succ_iterator, 363
 - next, 365
- spot::loopless_modular_mixed_radix_gray_code, 383
 - ~loopless_modular_mixed_radix_gray_code, 385
 - a_, 386
 - a_first, 385
 - a_last, 385
 - a_next, 385
 - done, 385
 - done_, 386
 - f_, 386
 - first, 386
 - last, 386
 - loopless_modular_mixed_radix_gray_code, 385
 - m_, 386
 - n_, 387
 - next, 386
 - non_one_radixes_, 387
 - s_, 387
- spot::ltl, 109
 - reduce_tau03, 113
 - unabbreviate_ltl, 113
- spot::ltl::atomic_prop, 128
 - ~atomic_prop, 132
 - accept, 133
 - atomic_prop, 132
 - clone, 133
 - count_, 135
 - destroy, 133
 - dump, 133
 - dump_instances, 133
 - env, 133
 - env_, 135
 - hash, 134
 - instance, 134
 - instance_count, 134
 - instances, 135
 - map, 132
 - name, 134
 - name_, 135
 - pair, 132
 - ref_, 134
 - ref_count_, 134
 - unref_, 134
- spot::ltl::automatop, 135
 - ~automatop, 140
 - accept, 140
 - automatop, 140
 - children_, 143
 - clone, 140
 - count_, 143
 - destroy, 141
 - dump, 141
 - dump_instances, 141
 - get_nfa, 141
 - hash, 141
 - instance, 141
 - instance_count, 142
 - instances, 143
 - is_negated, 142
 - map, 140
 - negated_, 143
 - nfa_, 143

- nth, 142
- ref_, 142
- ref_count_, 142
- size, 142
- triplet, 140
- unref_, 142
- vec, 140
- spot::ltl::automatop::tripletcmp, 903
 - operator(), 903
- spot::ltl::binop, 168
 - ~binop, 173
 - accept, 174
 - binop, 173
 - clone, 174
 - count_, 176
 - destroy, 174
 - dump, 174
 - dump_instances, 174
 - Equiv, 173
 - first, 174
 - first_, 176
 - hash, 175
 - Implies, 173
 - instance, 175
 - instance_count, 175
 - instances, 176
 - M, 173
 - map, 173
 - op, 175
 - op_, 176
 - op_name, 175
 - pair, 173
 - pairf, 173
 - R, 173
 - ref_, 175
 - ref_count_, 175
 - second, 175, 176
 - second_, 176
 - type, 173
 - U, 173
 - unref_, 176
 - W, 173
 - Xor, 173
- spot::ltl::clone_visitor, 177
 - ~clone_visitor, 180
 - clone_visitor, 180
 - recurse, 180
 - result, 180
 - result_, 181
 - visit, 180, 181
- spot::ltl::const_visitor, 190
 - ~const_visitor, 191
 - visit, 191
- spot::ltl::constant, 192
 - ~constant, 195
 - accept, 195
 - clone, 195
 - constant, 195
 - count_, 197
 - destroy, 196
 - dump, 196
 - False, 195
 - false_instance, 196
 - hash, 196
 - ref_, 196
 - True, 195
 - true_instance, 196
 - type, 195
 - unref_, 196
 - val, 197
 - val_, 197
 - val_name, 197
- spot::ltl::declarative_environment, 227
 - ~declarative_environment, 230
 - declarative_environment, 230
 - declare, 230
 - get_prop_map, 230
 - name, 230
 - prop_map, 230
 - props_, 231
 - require, 230
- spot::ltl::default_environment, 231
 - ~default_environment, 234
 - default_environment, 234
 - instance, 234
 - name, 234
 - require, 234
- spot::ltl::environment, 270
 - ~environment, 272
 - name, 272
 - require, 272
- spot::ltl::formula, 326
 - ~formula, 328
 - accept, 329
 - clone, 329
 - count_, 330
 - destroy, 329
 - dump, 329
 - formula, 328
 - hash, 329
 - max_count, 330
 - ref_, 330
 - unref_, 330
- spot::ltl::formula_tree
 - False, 115
 - True, 115
- spot::ltl::formula_ptr_hash, 330
 - operator(), 331

- spot::ltl::formula_ptr_less_than, 331
 - operator(), 331
- spot::ltl::formula_tree, 114
 - arity, 115
 - instantiate, 115
 - node_ptr, 114
- spot::ltl::formula_tree::node, 408
 - ~node, 408
- spot::ltl::formula_tree::node_atomic, 408
 - i, 410
- spot::ltl::formula_tree::node_binop, 410
 - lhs, 411
 - op, 411
 - rhs, 411
- spot::ltl::formula_tree::node_multop, 412
 - lhs, 413
 - op, 413
 - rhs, 413
- spot::ltl::formula_tree::node_nfa, 414
 - children, 415
 - nfa, 415
- spot::ltl::formula_tree::node_unop, 415
 - child, 417
 - op, 417
- spot::ltl::language_containment_checker, 365
 - ~language_containment_checker, 368
 - branching_postponement_, 369
 - contained, 368
 - contained_neg, 368
 - dict_, 369
 - equal, 368
 - exprop_, 369
 - fair_loop_approx_, 369
 - incompatible_, 368
 - language_containment_checker, 368
 - neg_contained, 368
 - register_formula_, 368
 - symb_merge_, 369
 - trans_map, 367
 - translated_, 369
- spot::ltl::language_containment_checker::record_, 481
 - incomp_map, 482
 - incompatible, 482
 - translation, 482
- spot::ltl::ltl_file, 387
 - in, 388
 - ltl_file, 387
 - next, 388
- spot::ltl::multop, 390
 - ~multop, 395
 - accept, 395
 - And, 395
 - children_, 398
 - clone, 396
 - count_, 398
 - destroy, 396
 - dump, 396
 - dump_instances, 396
 - hash, 396
 - instance, 396, 397
 - instance_count, 397
 - instances, 398
 - map, 395
 - multop, 395
 - nth, 397
 - op, 397
 - op_, 398
 - op_name, 397
 - Or, 395
 - pair, 395
 - ref_, 397
 - ref_count_, 398
 - size, 398
 - type, 395
 - unref_, 398
 - vec, 395
- spot::ltl::multop::paircmp, 437
 - operator(), 437
- spot::ltl::nfa, 399
 - ~nfa, 401
 - add_state, 401
 - add_transition, 401
 - arity, 401
 - arity_, 403
 - begin, 401
 - end, 401
 - finals_, 403
 - format_state, 402
 - get_init_state, 402
 - get_name, 402
 - init_, 403
 - is_, 403
 - is_final, 402
 - is_loop, 402
 - is_map, 400
 - iterator, 400
 - label, 400
 - name_, 403
 - nfa, 401
 - operator=, 402
 - ptr, 400
 - set_final, 402
 - set_init_state, 402
 - set_name, 402
 - si_, 403
 - si_map, 400
 - state, 401

- spot::ltl::nfa::transition, 900
 - dst, 900
 - lbl, 900
- spot::ltl::postfix_visitor, 467
 - ~postfix_visitor, 470
 - doit, 470, 471
 - doit_default, 471
 - postfix_visitor, 470
 - visit, 471
- spot::ltl::random_ltl, 474
 - ~random_ltl, 477
 - ap, 477
 - ap_, 478
 - dump_priorities, 477
 - generate, 477
 - parse_options, 477
 - proba_, 478
 - proba_2_, 478
 - random_ltl, 477
 - total_1_, 478
 - total_2_, 478
 - total_2_and_more_, 478
 - update_sums, 478
- spot::ltl::random_ltl::op_proba, 433
 - build, 434
 - builder, 434
 - min_n, 434
 - name, 434
 - proba, 434
 - setup, 434
- spot::ltl::ref_formula, 482
 - ~ref_formula, 485
 - accept, 486
 - clone, 486
 - count_, 487
 - destroy, 486
 - dump, 486
 - hash, 486
 - ref_, 487
 - ref_count_, 487
 - ref_counter_, 487
 - ref_formula, 485
 - unref_, 487
- spot::ltl::simplify_f_g_visitor, 543
 - ~simplify_f_g_visitor, 546
 - recurse, 547
 - result, 547
 - result_, 548
 - simplify_f_g_visitor, 546
 - super, 546
 - visit, 547
- spot::ltl::succ_iterator, 606
 - i_, 607
 - operator*, 607
 - operator++, 607
 - succ_iterator, 607
- spot::ltl::unabbreviate_logic_visitor, 904
 - ~unabbreviate_logic_visitor, 907
 - recurse, 908
 - result, 908
 - result_, 909
 - super, 907
 - unabbreviate_logic_visitor, 907
 - visit, 908
- spot::ltl::unabbreviate_ltl_visitor, 909
 - ~unabbreviate_ltl_visitor, 912
 - recurse, 913
 - result, 913
 - result_, 914
 - super, 912
 - unabbreviate_ltl_visitor, 912
 - visit, 913
- spot::ltl::unop, 914
 - ~unop, 919
 - accept, 919
 - child, 919
 - child_, 921
 - clone, 919
 - count_, 921
 - destroy, 920
 - dump, 920
 - dump_instances, 920
 - F, 919
 - Finish, 919
 - G, 919
 - hash, 920
 - instance, 920
 - instance_count, 920
 - instances, 921
 - map, 918
 - Not, 919
 - op, 920
 - op_, 921
 - op_name, 921
 - pair, 918
 - ref_, 921
 - ref_count_, 921
 - type, 919
 - unop, 919
 - unref_, 921
 - X, 919
- spot::ltl::visitor, 926
 - ~visitor, 928
 - visit, 928, 929
- spot::minato_isop::local_vars
 - FirstStep, 370
 - FourthStep, 370
 - SecondStep, 370

- ThirdStep, 370
- spot::minato_isop, 388
 - cube_, 390
 - minato_isop, 389
 - next, 389
 - ret_, 390
 - todo_, 390
- spot::minato_isop::local_vars, 369
 - f0_max, 370
 - f0_min, 370
 - f1_max, 370
 - f1_min, 371
 - f_max, 371
 - f_min, 371
 - g0, 371
 - g1, 371
 - local_vars, 370
 - step, 371
 - v1, 371
 - vars, 371
- spot::nips_exception, 403
 - err_, 404
 - err_defined_, 404
 - get_err, 404
 - get_err_defined, 404
 - get_where, 404
 - nips_exception, 404
 - where_, 404
- spot::nips_interface, 405
 - ~nips_interface, 407
 - automaton, 407
 - bytecode_, 407
 - dict_, 407
 - has_monitor, 407
 - nips_interface, 407
 - nipsvm_, 408
- spot::numbered_state_heap, 417
 - ~numbered_state_heap, 419
 - find, 419, 420
 - index, 420
 - insert, 420
 - iterator, 420
 - size, 420
 - state_index, 419
 - state_index_p, 419
- spot::numbered_state_heap_const_iterator, 421
 - ~numbered_state_heap_const_iterator, 421
 - done, 421
 - first, 421
 - get_index, 422
 - get_state, 422
 - next, 422
- spot::numbered_state_heap_factory, 422
 - ~numbered_state_heap_factory, 423
 - build, 424
- spot::numbered_state_heap_hash_map, 424
 - ~numbered_state_heap_hash_map, 428
 - find, 428
 - h, 429
 - hash_type, 427
 - index, 428
 - insert, 428
 - iterator, 429
 - size, 429
 - state_index, 427
 - state_index_p, 427
- spot::numbered_state_heap_hash_map_factory, 429
 - ~numbered_state_heap_hash_map_factory, 432
 - build, 432
 - instance, 432
 - numbered_state_heap_hash_map_factory, 432
- spot::option_map, 434
 - get, 435
 - operator<<, 437
 - options_, 437
 - parse_options, 436
 - set, 436
- spot::parity_game_graph, 437
 - ~parity_game_graph, 441
 - add_state, 441
 - automata_, 443
 - build_graph, 441
 - duplicator_vertice_, 443
 - end, 441
 - get_relation, 442
 - lift, 442
 - nb_node_parity_game, 443
 - next_state, 442
 - parity_game_graph, 441
 - print, 442
 - process_link, 442
 - process_state, 442
 - run, 443
 - seen, 443
 - seen_map, 441
 - spoiler_vertice_, 444
 - start, 443
 - tgba_state_, 444
 - todo, 444
 - want_state, 443
- spot::parity_game_graph_delayed, 444
 - ~parity_game_graph_delayed, 449
 - add_duplicator_node_delayed, 449
 - add_spoiler_node_delayed, 449
 - add_state, 449
 - automata_, 452

- bdd_v, 448
- build_graph, 449
- build_recurse_successor_duplicator, 449
- build_recurse_successor_spoiler, 449
- duplicator_vertice_, 452
- end, 450
- get_relation, 450
- lift, 450
- nb_node_parity_game, 452
- nb_set_acc_cond, 450
- next_state, 450
- parity_game_graph_delayed, 449
- print, 450
- process_link, 450, 451
- process_state, 451
- run, 451
- seen, 452
- seen_map, 448
- spoiler_vertice_, 452
- start, 451
- sub_set_acc_cond_, 452
- tgba_state_, 452
- todo, 452
- want_state, 451
- spot::parity_game_graph_direct, 452
 - ~parity_game_graph_direct, 456
 - add_state, 456
 - automata_, 458
 - build_graph, 456
 - build_link, 456
 - duplicator_vertice_, 458
 - end, 456
 - get_relation, 457
 - lift, 457
 - nb_node_parity_game, 459
 - next_state, 457
 - parity_game_graph_direct, 456
 - print, 457
 - process_link, 457
 - process_state, 458
 - run, 458
 - seen, 459
 - seen_map, 456
 - spoiler_vertice_, 459
 - start, 458
 - tgba_state_, 459
 - todo, 459
 - want_state, 458
- spot::power_map, 472
 - ~power_map, 472
 - canonicalize, 473
 - map_, 473
 - power_map_data, 472
 - power_state, 472
 - state_set, 472
 - states, 473
 - states_of, 473
- spot::ptr_hash, 473
 - operator(), 474
- spot::rsymbol, 487
 - ~rsymbol, 489
 - operator const symbol *, 489
 - operator<, 490
 - operator=, 490
 - operator==, 490
 - rsymbol, 489
 - s_, 490
- spot::saba, 490
 - ~saba, 492
 - all_acceptance_conditions, 492
 - format_state, 492
 - get_dict, 493
 - get_init_state, 493
 - num_acc_, 494
 - number_of_acceptance_conditions, 493
 - saba, 492
 - succ_iter, 493
- spot::saba_complement_tgba, 494
 - ~saba_complement_tgba, 498
 - all_acceptance_conditions, 498
 - automaton_, 499
 - format_state, 498
 - get_dict, 498
 - get_init_state, 498
 - nb_states_, 499
 - number_of_acceptance_conditions, 499
 - saba_complement_tgba, 498
 - succ_iter, 499
 - the_acceptance_cond_, 499
- spot::saba_reachable_iterator, 499
 - ~saba_reachable_iterator, 503
 - add_state, 503
 - automata_, 505
 - end, 503
 - next_state, 503
 - process_link, 503
 - process_state, 504
 - process_state_conjunction, 504
 - run, 504
 - saba_reachable_iterator, 503
 - seen, 505
 - seen_map, 502
 - start, 504
 - want_state, 504
- spot::saba_reachable_iterator_breadth_first, 505
 - add_state, 509
 - automata_, 510
 - end, 509

- next_state, 509
- process_link, 509
- process_state, 509
- process_state_conjunction, 510
- run, 510
- saba_reachable_iterator_breadth_first, 509
- seen, 510
- seen_map, 508
- start, 510
- todo, 511
- want_state, 510
- spot::saba_reachable_iterator_depth_first, 511
 - add_state, 515
 - automata_, 516
 - end, 515
 - next_state, 515
 - process_link, 515
 - process_state, 515
 - process_state_conjunction, 516
 - run, 516
 - saba_reachable_iterator_depth_first, 515
 - seen, 516
 - seen_map, 514
 - start, 516
 - todo, 517
 - want_state, 516
- spot::saba_state, 517
 - ~saba_state, 517
 - acceptance_conditions, 518
 - clone, 518
 - compare, 518
 - hash, 518
- spot::saba_state_conjunction, 519
 - ~saba_state_conjunction, 521
 - clone, 521
 - current_state, 521
 - done, 522
 - first, 522
 - next, 522
- spot::saba_state_ptr_equal, 523
 - operator(), 523
- spot::saba_state_ptr_hash, 523
 - operator(), 524
- spot::saba_state_ptr_less_than, 524
 - operator(), 524
- spot::saba_state_shared_ptr_equal, 525
 - operator(), 525
- spot::saba_state_shared_ptr_hash, 525
 - operator(), 526
- spot::saba_state_shared_ptr_less_than, 526
 - operator(), 526
- spot::saba_succ_iterator, 527
 - ~saba_succ_iterator, 527
 - current_condition, 527
 - current_conjunction, 528
 - done, 528
 - first, 528
 - next, 528
- spot::scc_map, 530
 - ~scc_map, 534
 - acc_set_of, 534
 - accepting, 534
 - ap_set_of, 535
 - aprec_set_of, 535
 - arc_acc_, 538
 - arc_cond_, 538
 - aut_, 538
 - build_map, 535
 - cond_set, 533
 - cond_set_of, 535
 - get_aut, 535
 - h_, 538
 - hash_type, 533
 - initial, 536
 - num_, 538
 - one_state_of, 536
 - pair_state_iter, 534
 - relabel_component, 536
 - root_, 538
 - scc_count, 536
 - scc_map, 534
 - scc_map_, 538
 - scc_map_type, 534
 - scc_of_state, 536
 - self_loops, 536
 - self_loops_, 538
 - stack_type, 534
 - states_of, 537
 - succ, 537
 - succ_type, 534
 - todo_, 538
 - trivial, 537
 - update_supp_rec, 537
 - useful_acc_of, 537
- spot::scc_map::scc, 528
 - acc, 529
 - conds, 529
 - index, 529
 - scc, 529
 - states, 530
 - succ, 530
 - supp, 530
 - supp_rec, 530
 - trivial, 530
 - useful_acc, 530
- spot::scc_stack, 538
 - clear_rem, 539
 - empty, 540

- pop, 540
- push, 540
- rem, 540
- s, 540
- size, 540
- stack_type, 539
- top, 540
- spot::scc_stack::connected_component, 181
 - condition, 183
 - connected_component, 183
 - index, 183
 - rem, 183
- spot::scc_stats, 541
 - acc_paths, 541
 - acc_scc, 541
 - dead_paths, 541
 - dead_scc, 542
 - dump, 541
 - scc_total, 542
 - self_loops, 542
 - useful_acc, 542
 - useless_scc_map, 542
- spot::sccs_set, 542
 - sccs, 543
 - size, 543
- spot::spoiler_node, 552
 - ~spoiler_node, 554
 - add_pred, 555
 - add_succ, 555
 - compare, 555
 - del_pred, 555
 - del_succ, 555
 - get_duplicator_node, 555
 - get_nb_succ, 555
 - get_pair, 555
 - get_spoiler_node, 555
 - lnode_pred, 556
 - lnode_succ, 556
 - not_win, 556
 - num_, 556
 - prune, 555
 - sc_, 556
 - set_win, 555
 - spoiler_node, 554
 - succ_to_string, 556
 - to_string, 556
- spot::spoiler_node_delayed, 556
 - ~spoiler_node_delayed, 559
 - acceptance_condition_visited_, 561
 - add_pred, 560
 - add_succ, 560
 - compare, 560
 - del_pred, 560
 - del_succ, 560
 - get_acceptance_condition_visited, 560
 - get_duplicator_node, 560
 - get_lead_2_acc_all, 560
 - get_nb_succ, 560
 - get_pair, 560
 - get_progress_measure, 561
 - get_spoiler_node, 561
 - lead_2_acc_all_, 561
 - lnode_pred, 561
 - lnode_succ, 562
 - not_win, 562
 - num_, 562
 - progress_measure_, 562
 - prune, 561
 - sc_, 562
 - seen_, 562
 - set_lead_2_acc_all, 561
 - set_win, 561
 - spoiler_node_delayed, 559
 - succ_to_string, 561
 - to_string, 561
- spot::state, 572
 - ~state, 573
 - clone, 573
 - compare, 573
 - destroy, 574
 - hash, 574
- spot::state_bdd, 574
 - as_bdd, 577
 - clone, 577
 - compare, 577
 - destroy, 577
 - hash, 578
 - state_, 578
 - state_bdd, 577
- spot::state_evtgba_explicit, 578
 - ~state_evtgba_explicit, 581
 - clone, 581
 - compare, 581
 - destroy, 581
 - get_state, 582
 - hash, 582
 - state_, 582
 - state_evtgba_explicit, 581
- spot::state_explicit, 582
 - ~state_explicit, 585
 - clone, 585
 - compare, 585
 - destroy, 585
 - get_state, 586
 - hash, 586
 - state_, 586
 - state_explicit, 585
- spot::state_product, 586

- ~state_product, 589
- clone, 590
- compare, 590
- destroy, 590
- hash, 590
- left, 591
- left_, 591
- right, 591
- right_, 591
- state_product, 589
- spot::state_ptr_equal, 591
 - operator(), 592
- spot::state_ptr_hash, 592
 - operator(), 592
- spot::state_ptr_less_than, 593
 - operator(), 593
- spot::state_set, 593
 - ~state_set, 596
 - clone, 596
 - compare, 596
 - delete_me_, 597
 - destroy, 596
 - get_state, 597
 - hash, 597
 - s_, 597
 - state_set, 596
- spot::state_shared_ptr_equal, 598
 - operator(), 598
- spot::state_shared_ptr_hash, 598
 - operator(), 599
- spot::state_shared_ptr_less_than, 599
 - operator(), 599
- spot::state_union, 599
 - ~state_union, 602
 - clone, 603
 - compare, 603
 - destroy, 603
 - hash, 603
 - left, 604
 - left_, 604
 - right, 604
 - right_, 604
 - state_union, 602
- spot::string_hash, 606
 - operator(), 606
- spot::symbol, 609
 - ~symbol, 610
 - dump_instances, 610
 - instance, 610
 - instance_count, 610
 - instances_, 611
 - map, 610
 - name, 610
 - name_, 611
 - ref, 611
 - ref_count_, 611
 - refs_, 611
 - symbol, 610
 - unref, 611
- spot::taa_succ_iterator, 611
 - ~taa_succ_iterator, 615
 - all_acceptance_conditions_, 616
 - bounds_t, 614
 - current_acceptance_conditions, 615
 - current_condition, 615
 - current_state, 615
 - done, 616
 - first, 616
 - i_, 616
 - iterator, 614
 - iterator_pair, 615
 - next, 616
 - seen_, 617
 - seen_map, 615
 - succ_, 617
 - taa_succ_iterator, 615
- spot::taa_succ_iterator::distance_sort, 235
 - operator(), 235
- spot::taa_tgba, 617
 - ~taa_tgba, 622
 - add_condition, 622
 - all_acceptance_conditions, 622
 - all_acceptance_conditions_, 625
 - all_acceptance_conditions_computed_, 625
 - compute_support_conditions, 622
 - compute_support_variables, 622
 - dict_, 625
 - format_state, 622
 - get_dict, 623
 - get_init_state, 623
 - init_, 625
 - neg_acceptance_conditions, 623
 - neg_acceptance_conditions_, 626
 - number_of_acceptance_conditions, 623
 - operator=, 623
 - project_state, 624
 - ss_vec, 621
 - state, 621
 - state_set, 621
 - state_set_vec_, 626
 - succ_iter, 624
 - support_conditions, 624
 - support_variables, 625
 - taa_tgba, 622
 - transition_annotation, 625
- spot::taa_tgba::transition, 899
 - acceptance_conditions, 900
 - condition, 900

- dst, 900
- spot::taa_tgba_formula, 626
 - ~taa_tgba_formula, 631
 - add_acceptance_condition, 631
 - add_condition, 631
 - all_acceptance_conditions, 631
 - all_acceptance_conditions_, 636
 - all_acceptance_conditions_computed_, 636
 - clone_if, 632
 - compute_support_conditions, 632
 - compute_support_variables, 632
 - create_transition, 632
 - dict_, 636
 - format_state, 632
 - get_dict, 633
 - get_init_state, 633
 - init_, 636
 - label_t, 630
 - label_to_string, 633
 - name_state_map_, 636
 - neg_acceptance_conditions, 633
 - neg_acceptance_conditions_, 636
 - ns_map, 630
 - number_of_acceptance_conditions, 634
 - output, 634
 - project_state, 634
 - set_init_state, 634
 - sn_map, 631
 - ss_vec, 631
 - state, 631
 - state_name_map_, 636
 - state_set, 631
 - state_set_vec_, 636
 - succ_iter, 634
 - support_conditions, 635
 - support_variables, 635
 - taa_tgba_formula, 631
 - transition_annotation, 635
- spot::taa_tgba_labelled, 637
 - add_acceptance_condition, 642
 - add_condition, 642
 - add_state, 643
 - add_state_set, 643
 - all_acceptance_conditions, 643
 - all_acceptance_conditions_, 648
 - all_acceptance_conditions_computed_, 648
 - clone_if, 643
 - compute_support_conditions, 643
 - compute_support_variables, 644
 - create_transition, 644
 - dict_, 648
 - format_state, 644
 - format_state_set, 644
 - get_dict, 644
 - get_init_state, 645
 - init_, 648
 - label_t, 642
 - label_to_string, 645
 - name_state_map_, 648
 - neg_acceptance_conditions, 645
 - neg_acceptance_conditions_, 648
 - ns_map, 642
 - number_of_acceptance_conditions, 645
 - output, 645
 - project_state, 646
 - set_init_state, 646
 - sn_map, 642
 - ss_vec, 642
 - state, 642
 - state_name_map_, 648
 - state_set, 642
 - state_set_vec_, 648
 - succ_iter, 646
 - support_conditions, 647
 - support_variables, 647
 - taa_tgba_labelled, 642
 - transition_annotation, 647
- spot::taa_tgba_string, 649
 - ~taa_tgba_string, 654
 - add_acceptance_condition, 654
 - add_condition, 654
 - all_acceptance_conditions, 654
 - all_acceptance_conditions_, 659
 - all_acceptance_conditions_computed_, 659
 - clone_if, 655
 - compute_support_conditions, 655
 - compute_support_variables, 655
 - create_transition, 655
 - dict_, 659
 - format_state, 656
 - get_dict, 656
 - get_init_state, 656
 - init_, 659
 - label_t, 653
 - label_to_string, 656
 - name_state_map_, 659
 - neg_acceptance_conditions, 657
 - neg_acceptance_conditions_, 660
 - ns_map, 653
 - number_of_acceptance_conditions, 657
 - output, 657
 - project_state, 657
 - set_init_state, 657, 658
 - sn_map, 653
 - ss_vec, 654
 - state, 654
 - state_name_map_, 660
 - state_set, 654

- state_set_vec_, 660
- succ_iter, 658
- support_conditions, 658
- support_variables, 658
- taa_tgba_string, 654
- transition_annotation, 659
- spot::tgba, 660
 - ~tgba, 664
 - all_acceptance_conditions, 664
 - compute_support_conditions, 664
 - compute_support_variables, 665
 - format_state, 665
 - get_dict, 665
 - get_init_state, 665
 - last_support_conditions_input_, 668
 - last_support_conditions_output_, 668
 - last_support_variables_input_, 668
 - last_support_variables_output_, 668
 - neg_acceptance_conditions, 666
 - num_acc_, 668
 - number_of_acceptance_conditions, 666
 - project_state, 666
 - succ_iter, 666
 - support_conditions, 667
 - support_variables, 667
 - tgba, 664
 - transition_annotation, 667
- spot::tgba_bdd_concrete, 668
 - ~tgba_bdd_concrete, 672
 - all_acceptance_conditions, 673
 - compute_support_conditions, 673
 - compute_support_variables, 673
 - data_, 677
 - delete_unaccepting_scc, 673
 - format_state, 673
 - get_core_data, 674
 - get_dict, 674
 - get_init_bdd, 674
 - get_init_state, 674
 - init_, 677
 - neg_acceptance_conditions, 674
 - number_of_acceptance_conditions, 675
 - operator=, 675
 - project_state, 675
 - set_init_state, 675
 - succ_iter, 675
 - support_conditions, 676
 - support_variables, 676
 - tgba_bdd_concrete, 672, 673
 - transition_annotation, 676
- spot::tgba_bdd_concrete_factory, 677
 - ~tgba_bdd_concrete_factory, 680
 - acc_, 682
 - acc_map_, 680
 - constrain_relation, 681
 - create_anonymous_state, 681
 - create_atomic_prop, 681
 - create_state, 681
 - data_, 682
 - declare_acceptance_condition, 681
 - finish, 682
 - get_core_data, 682
 - get_dict, 682
 - tgba_bdd_concrete_factory, 680
- spot::tgba_bdd_core_data, 683
 - acc_set, 687
 - acceptance_conditions, 687
 - acceptance_conditions_support, 688
 - all_acceptance_conditions, 688
 - declare_acceptance_condition, 687
 - declare_atomic_prop, 687
 - declare_now_next, 687
 - delete_unaccepting_scc, 687
 - dict, 688
 - infinitely_often, 687
 - negacc_set, 688
 - next_set, 688
 - notacc_set, 689
 - notnext_set, 689
 - notnow_set, 689
 - notvar_set, 689
 - now_set, 689
 - nownext_set, 689
 - operator=, 687
 - relation, 689
 - tgba_bdd_core_data, 686
 - var_set, 690
 - varandnext_set, 690
- spot::tgba_bdd_factory, 690
 - ~tgba_bdd_factory, 692
 - get_core_data, 692
- spot::tgba_explicit, 692
 - ~tgba_explicit, 697
 - add_acceptance_condition, 697
 - add_acceptance_conditions, 697
 - add_condition, 697
 - add_conditions, 697
 - add_default_init, 697
 - all_acceptance_conditions, 697
 - all_acceptance_conditions_, 701
 - all_acceptance_conditions_computed_, 701
 - compute_support_conditions, 698
 - compute_support_variables, 698
 - copy_acceptance_conditions_of, 698
 - create_transition, 698
 - dict_, 701
 - format_state, 698
 - get_acceptance_condition, 698

- get_dict, 699
- get_init_state, 699
- has_acceptance_condition, 699
- init_, 702
- neg_acceptance_conditions, 699
- neg_acceptance_conditions_, 702
- number_of_acceptance_conditions, 699
- operator=, 699
- project_state, 700
- set_acceptance_conditions, 700
- state, 696
- succ_iter, 700
- support_conditions, 700
- support_variables, 701
- tgba_explicit, 697
- transition_annotation, 701
- spot::tgba_explicit::transition, 901
 - acceptance_conditions, 901
 - condition, 901
 - dest, 901
- spot::tgba_explicit_formula, 702
 - ~tgba_explicit_formula, 707
 - add_acceptance_condition, 707
 - add_acceptance_conditions, 707
 - add_condition, 707
 - add_conditions, 708
 - add_default_init, 708
 - add_state, 708
 - all_acceptance_conditions, 708
 - all_acceptance_conditions_, 713
 - all_acceptance_conditions_computed_, 713
 - complement_all_acceptance_conditions, 708
 - compute_support_conditions, 708
 - compute_support_variables, 708
 - copy_acceptance_conditions_of, 709
 - create_transition, 709
 - declare_acceptance_condition, 709
 - dict_, 713
 - format_state, 709
 - get_acceptance_condition, 709
 - get_dict, 709
 - get_init_state, 710
 - get_label, 710
 - has_acceptance_condition, 710
 - has_state, 710
 - init_, 713
 - label_t, 707
 - merge_transitions, 710
 - name_state_map_, 713
 - neg_acceptance_conditions, 710
 - neg_acceptance_conditions_, 713
 - ns_map, 707
 - number_of_acceptance_conditions, 711
 - project_state, 711
 - set_acceptance_conditions, 711
 - set_init_state, 711
 - sn_map, 707
 - state, 707
 - state_name_map_, 713
 - succ_iter, 711
 - support_conditions, 712
 - support_variables, 712
 - tgba_explicit_formula, 707
 - transition_annotation, 712
- spot::tgba_explicit_labelled, 714
 - ~tgba_explicit_labelled, 719
 - add_acceptance_condition, 719
 - add_acceptance_conditions, 719
 - add_condition, 720
 - add_conditions, 720
 - add_default_init, 720
 - add_state, 720
 - all_acceptance_conditions, 720
 - all_acceptance_conditions_, 725
 - all_acceptance_conditions_computed_, 725
 - complement_all_acceptance_conditions, 720
 - compute_support_conditions, 721
 - compute_support_variables, 721
 - copy_acceptance_conditions_of, 721
 - create_transition, 721
 - declare_acceptance_condition, 721
 - dict_, 725
 - format_state, 721
 - get_acceptance_condition, 722
 - get_dict, 722
 - get_init_state, 722
 - get_label, 722
 - has_acceptance_condition, 722
 - has_state, 723
 - init_, 725
 - label_t, 719
 - merge_transitions, 723
 - name_state_map_, 725
 - neg_acceptance_conditions, 723
 - neg_acceptance_conditions_, 726
 - ns_map, 719
 - number_of_acceptance_conditions, 723
 - project_state, 723
 - set_acceptance_conditions, 723
 - set_init_state, 724
 - sn_map, 719
 - state, 719
 - state_name_map_, 726
 - succ_iter, 724
 - support_conditions, 724
 - support_variables, 724
 - tgba_explicit_labelled, 719
 - transition_annotation, 725

- spot::tgba_explicit_number, 726
 - ~tgba_explicit_number, 731
 - add_acceptance_condition, 731
 - add_acceptance_conditions, 731
 - add_condition, 731
 - add_conditions, 732
 - add_default_init, 732
 - add_state, 732
 - all_acceptance_conditions, 732
 - all_acceptance_conditions_, 737
 - all_acceptance_conditions_computed_, 737
 - complement_all_acceptance_conditions, 732
 - compute_support_conditions, 732
 - compute_support_variables, 732
 - copy_acceptance_conditions_of, 733
 - create_transition, 733
 - declare_acceptance_condition, 733
 - dict_, 737
 - format_state, 733
 - get_acceptance_condition, 733
 - get_dict, 733
 - get_init_state, 734
 - get_label, 734
 - has_acceptance_condition, 734
 - has_state, 734
 - init_, 737
 - label_t, 731
 - merge_transitions, 734
 - name_state_map_, 737
 - neg_acceptance_conditions, 734
 - neg_acceptance_conditions_, 737
 - ns_map, 731
 - number_of_acceptance_conditions, 735
 - project_state, 735
 - set_acceptance_conditions, 735
 - set_init_state, 735
 - sn_map, 731
 - state, 731
 - state_name_map_, 737
 - succ_iter, 735
 - support_conditions, 736
 - support_variables, 736
 - tgba_explicit_number, 731
 - transition_annotation, 736
- spot::tgba_explicit_string, 738
 - ~tgba_explicit_string, 743
 - add_acceptance_condition, 743
 - add_acceptance_conditions, 743
 - add_condition, 743
 - add_conditions, 744
 - add_default_init, 744
 - add_state, 744
 - add_state_alias, 744
 - all_acceptance_conditions, 744
 - all_acceptance_conditions_, 749
 - all_acceptance_conditions_computed_, 749
 - complement_all_acceptance_conditions, 744
 - compute_support_conditions, 745
 - compute_support_variables, 745
 - copy_acceptance_conditions_of, 745
 - create_transition, 745
 - declare_acceptance_condition, 745
 - dict_, 749
 - format_state, 746
 - get_acceptance_condition, 746
 - get_dict, 746
 - get_init_state, 746
 - get_label, 746
 - has_acceptance_condition, 747
 - has_state, 747
 - init_, 750
 - label_t, 743
 - merge_transitions, 747
 - name_state_map_, 750
 - neg_acceptance_conditions, 747
 - neg_acceptance_conditions_, 750
 - ns_map, 743
 - number_of_acceptance_conditions, 747
 - project_state, 747
 - set_acceptance_conditions, 748
 - set_init_state, 748
 - sn_map, 743
 - state, 743
 - state_name_map_, 750
 - succ_iter, 748
 - support_conditions, 748
 - support_variables, 749
 - tgba_explicit_string, 743
 - transition_annotation, 749
- spot::tgba_explicit_succ_iterator, 750
 - all_acceptance_conditions_, 755
 - current_acceptance_conditions, 753
 - current_condition, 753
 - current_state, 754
 - done, 754
 - first, 754
 - i_, 755
 - next, 754
 - s_, 755
 - tgba_explicit_succ_iterator, 753
- spot::tgba_kv_complement, 755
 - ~tgba_kv_complement, 759
 - acc_list_, 763
 - all_acceptance_conditions, 759
 - automaton_, 763
 - compute_support_conditions, 759
 - compute_support_variables, 760
 - format_state, 760

- get_acc_list, 760
- get_dict, 760
- get_init_state, 760
- nb_states_, 763
- neg_acceptance_conditions, 760
- number_of_acceptance_conditions, 761
- project_state, 761
- succ_iter, 761
- support_conditions, 762
- support_variables, 762
- tgba_kv_complement, 759
- the_acceptance_cond_, 763
- transition_annotation, 762
- spot::tgba_product, 763
 - ~tgba_product, 767
 - all_acceptance_conditions, 767
 - all_acceptance_conditions_, 771
 - compute_support_conditions, 768
 - compute_support_variables, 768
 - dict_, 771
 - format_state, 768
 - get_dict, 768
 - get_init_state, 768
 - left_, 771
 - left_acc_complement_, 771
 - neg_acceptance_conditions, 769
 - neg_acceptance_conditions_, 771
 - number_of_acceptance_conditions, 769
 - operator=, 769
 - project_state, 769
 - right_, 771
 - right_acc_complement_, 771
 - right_common_acc_, 771
 - succ_iter, 769
 - support_conditions, 770
 - support_variables, 770
 - tgba_product, 767
 - transition_annotation, 770
- spot::tgba_product_init, 771
 - all_acceptance_conditions, 775
 - compute_support_conditions, 775
 - compute_support_variables, 775
 - format_state, 775
 - get_dict, 775
 - get_init_state, 776
 - left_init_, 778
 - neg_acceptance_conditions, 776
 - number_of_acceptance_conditions, 776
 - project_state, 776
 - right_init_, 778
 - succ_iter, 777
 - support_conditions, 777
 - support_variables, 777
 - tgba_product_init, 775
 - transition_annotation, 778
- spot::tgba_reachable_iterator, 778
 - ~tgba_reachable_iterator, 782
 - add_state, 782
 - automata_, 783
 - end, 782
 - next_state, 782
 - process_link, 782
 - process_state, 782
 - run, 783
 - seen, 783
 - seen_map, 781
 - start, 783
 - tgba_reachable_iterator, 782
 - want_state, 783
- spot::tgba_reachable_iterator_breadth_first, 784
 - add_state, 788
 - automata_, 789
 - end, 788
 - next_state, 788
 - process_link, 788
 - process_state, 788
 - run, 789
 - seen, 789
 - seen_map, 787
 - start, 789
 - tgba_reachable_iterator_breadth_first, 788
 - todo, 789
 - want_state, 789
- spot::tgba_reachable_iterator_depth_first, 790
 - add_state, 794
 - automata_, 795
 - end, 794
 - next_state, 794
 - process_link, 794
 - process_state, 794
 - run, 795
 - seen, 795
 - seen_map, 793
 - start, 795
 - tgba_reachable_iterator_depth_first, 794
 - todo, 795
 - want_state, 795
- spot::tgba_reduc, 796
 - ~tgba_reduc, 803
 - add_acceptance_condition, 803
 - add_acceptance_conditions, 803
 - add_condition, 803
 - add_conditions, 803
 - add_default_init, 803
 - add_state, 803, 804
 - add_state_alias, 804
 - all_acceptance_conditions, 804
 - all_acceptance_conditions_, 813

all_acceptance_conditions_computed_, 813
automata_, 813
complement_all_acceptance_conditions, 804
compute_support_conditions, 804
compute_support_variables, 804
copy_acceptance_conditions_of, 805
create_transition, 805
declare_acceptance_condition, 805
delete_scc, 805
delete_transitions, 805
dict_, 813
display_rel_sim, 806
display_scc, 806
end, 806
format_state, 806
get_acceptance_condition, 806
get_dict, 806
get_init_state, 806
get_label, 807
has_acceptance_condition, 807
has_state, 807
init_, 813
is_not_accepting, 807
is_terminal, 807
label_t, 802
merge_state, 808
merge_state_delayed, 808
merge_transitions, 808
name_state_map_, 813
nb_set_acc_cond, 808
neg_acceptance_conditions, 808
neg_acceptance_conditions_, 813
next_state, 808
ns_map, 802
number_of_acceptance_conditions, 809
process_link, 809
process_state, 809
project_state, 809
quotient_state, 810
redirect_transition, 810
remove_acc, 810
remove_component, 810
remove_predecessor_state, 810
remove_scc, 810
remove_state, 810
run, 811
seen, 813
seen_map, 802
set_acceptance_conditions, 811
set_init_state, 811
sn_map, 802
sp_map, 802
start, 811
state, 802
state_name_map_, 813
state_predecessor_map_, 814
succ_iter, 811
support_conditions, 812
support_variables, 812
tgba_reduc, 803
todo, 814
transition_annotation, 812
want_state, 812
spot::tgba_run, 814
 ~tgba_run, 815
 cycle, 815
 operator=, 815
 prefix, 815
 steps, 815
 tgba_run, 815
spot::tgba_run::step, 604
 acc, 605
 label, 605
 s, 605
spot::tgba_run_dotty_decorator, 815
 ~tgba_run_dotty_decorator, 818
 instance, 819
 link_decl, 819
 map_, 820
 run_, 820
 state_decl, 819
 step_map, 818
 step_num, 818
 step_set, 818
 tgba_run_dotty_decorator, 818
spot::tgba_safra_complement, 820
 ~tgba_safra_complement, 824
 acceptance_cond_vec_, 828
 all_acceptance_cond_, 828
 all_acceptance_conditions, 824
 autaton_, 828
 compute_support_conditions, 824
 compute_support_variables, 825
 format_state, 825
 get_dict, 825
 get_init_state, 825
 get_safra, 825
 neg_acceptance_cond_, 828
 neg_acceptance_conditions, 825
 number_of_acceptance_conditions, 826
 project_state, 826
 safra_, 828
 succ_iter, 826
 support_conditions, 827
 support_variables, 827
 tgba_safra_complement, 824
 transition_annotation, 827
spot::tgba_sba_proxy, 828

- a_, 836
- acc_cycle_, 836
- all_acceptance_conditions, 832
- common_acceptance_conditions_of_-
 - original_state, 833
- compute_support_conditions, 833
- compute_support_variables, 833
- cycle_list, 832
- cycle_start_, 836
- format_state, 833
- get_dict, 833
- get_init_state, 833
- neg_acceptance_conditions, 834
- number_of_acceptance_conditions, 834
- project_state, 834
- state_is_accepting, 834
- succ_iter, 835
- support_conditions, 835
- support_variables, 835
- tgba_sba_proxy, 832
- transition_annotation, 836
- spot::tgba_scc, 836
 - ~tgba_scc, 840
 - all_acceptance_conditions, 840
 - aut_, 843
 - compute_support_conditions, 840
 - compute_support_variables, 840
 - format_state, 841
 - get_dict, 841
 - get_init_state, 841
 - neg_acceptance_conditions, 841
 - number_of_acceptance_conditions, 841
 - project_state, 842
 - scc_map_, 843
 - scc_of_state, 842
 - show_, 843
 - succ_iter, 842
 - support_conditions, 842
 - support_variables, 843
 - tgba_scc, 840
 - transition_annotation, 843
- spot::tgba_sgba_proxy, 844
 - ~tgba_sgba_proxy, 848
- a_, 852
- acceptance_condition_, 852
- all_acceptance_conditions, 848
- compute_support_conditions, 848
- compute_support_variables, 849
- emulate_acc_cond_, 852
- format_state, 849
- get_dict, 849
- get_init_state, 849
- neg_acceptance_conditions, 849
- number_of_acceptance_conditions, 850
- operator=, 850
- project_state, 850
- state_acceptance_conditions, 850
- succ_iter, 850
- support_conditions, 851
- support_variables, 851
- tgba_sgba_proxy, 848
- transition_annotation, 851
- spot::tgba_statistics, 852
 - dump, 852
 - states, 852
 - transitions, 852
- spot::tgba_succ_iterator, 853
 - ~tgba_succ_iterator, 854
 - current_acceptance_conditions, 854
 - current_condition, 854
 - current_state, 854
 - done, 854
 - first, 855
 - next, 855
- spot::tgba_succ_iterator_concrete, 855
 - ~tgba_succ_iterator_concrete, 859
 - current_, 860
 - current_acc_, 860
 - current_acceptance_conditions, 859
 - current_condition, 859
 - current_state, 859
 - current_state_, 860
 - data_, 861
 - done, 859
 - first, 860
 - next, 860
 - succ_set_, 861
 - succ_set_left_, 861
 - tgba_succ_iterator_concrete, 859
- spot::tgba_succ_iterator_product, 861
 - ~tgba_succ_iterator_product, 865
 - current_acceptance_conditions, 865
 - current_cond_, 867
 - current_condition, 865
 - current_state, 865
 - done, 865
 - first, 866
 - left_, 867
 - left_neg_, 867
 - next, 866
 - next_non_false_, 866
 - right_, 867
 - right_common_acc_, 867
 - right_neg_, 867
 - step_, 866
 - tgba_product, 866
 - tgba_succ_iterator_product, 865
- spot::tgba_succ_iterator_union, 867

- ~tgba_succ_iterator_union, 870
- current_acceptance_conditions, 871
- current_cond_, 872
- current_condition, 871
- current_state, 871
- done, 871
- first, 871
- left_, 872
- left_missing_, 872
- left_neg_, 872
- next, 872
- right_, 872
- right_missing_, 873
- right_neg_, 873
- tgba_succ_iterator_union, 870
- tgba_union, 872
- spot::tgba_tba_proxy, 873
 - ~tgba_tba_proxy, 878
 - a_, 881
 - acc_cycle_, 881
 - accmap_, 881
 - accmap_t, 877
 - all_acceptance_conditions, 878
 - common_acceptance_conditions_of_-
original_state, 878
 - compute_support_conditions, 878
 - compute_support_variables, 878
 - cycle_list, 877
 - format_state, 879
 - get_dict, 879
 - get_init_state, 879
 - neg_acceptance_conditions, 879
 - number_of_acceptance_conditions, 879
 - operator=, 880
 - project_state, 880
 - succ_iter, 880
 - support_conditions, 880
 - support_variables, 881
 - tgba_tba_proxy, 878
 - the_acceptance_cond_, 882
 - transition_annotation, 881
- spot::tgba_union, 882
 - ~tgba_union, 886
 - all_acceptance_conditions, 887
 - all_acceptance_conditions_, 890
 - compute_support_conditions, 887
 - compute_support_variables, 887
 - dict_, 890
 - format_state, 887
 - get_dict, 887
 - get_init_state, 887
 - left_, 890
 - left_acc_complement_, 890
 - left_acc_missing_, 890
 - left_var_missing_, 890
 - neg_acceptance_conditions, 888
 - neg_acceptance_conditions_, 890
 - number_of_acceptance_conditions, 888
 - operator=, 888
 - project_state, 888
 - right_, 890
 - right_acc_complement_, 890
 - right_acc_missing_, 891
 - right_common_acc_, 891
 - right_var_missing_, 891
 - succ_iter, 888
 - support_conditions, 889
 - support_variables, 889
 - tgba_union, 886
 - transition_annotation, 889
- spot::time_info, 891
 - stime, 892
 - time_info, 891
 - utime, 892
- spot::timer, 892
 - is_running, 894
 - running, 895
 - start, 894
 - start_, 895
 - stime, 894
 - stop, 894
 - timer, 894
 - total_, 895
 - utime, 895
- spot::timer_map, 895
 - cancel, 896
 - empty, 896
 - item_type, 896
 - print, 897
 - start, 897
 - stop, 897
 - timer, 897
 - tm, 898
 - tm_type, 896
- spot::unsigned_statistics, 922
 - ~unsigned_statistics, 924
 - get, 924
 - stats, 924
 - stats_map, 924
 - unsigned_fun, 924
- spot::unsigned_statistics_copy, 925
 - operator==, 926
 - set, 926
 - seteq, 926
 - stats, 926
 - stats_map, 925
 - unsigned_statistics_copy, 925
- spot::weight, 929

- dec_weight_handler, 930
- inc_weight_handler, 930
- m, 931
- neg_all_acc, 931
- operator<<, 931
- operator+&, 930
- operator-, 930
- operator=, 930
- pm, 931
- weight, 930
- weight_vector, 930
- srand
 - random, 23
- ss_vec
 - spot::taa_tgba, 621
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 654
- stack
 - eltlyy::stack, 568
 - ltlyy::stack, 570
 - neverclaimyy::stack, 563
 - sautyy::stack, 565
- stack_
 - eltlyy::slice, 549
 - ltlyy::slice, 552
 - neverclaimyy::slice, 551
 - sautyy::slice, 550
- stack_type
 - spot::scc_map, 534
 - spot::scc_stack, 539
- start
 - spot::evtgba_reachable_iterator, 293
 - spot::evtgba_reachable_iterator_breadth_first, 298
 - spot::evtgba_reachable_iterator_depth_first, 303
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 451
 - spot::parity_game_graph_direct, 458
 - spot::saba_reachable_iterator, 504
 - spot::saba_reachable_iterator_breadth_first, 510
 - spot::saba_reachable_iterator_depth_first, 516
 - spot::tgba_reachable_iterator, 783
 - spot::tgba_reachable_iterator_breadth_first, 789
 - spot::tgba_reachable_iterator_depth_first, 795
 - spot::tgba_reduc, 811
 - spot::timer, 894
 - spot::timer_map, 897
- start_
 - spot::timer, 895
- state
 - spot::ltl::nfa, 401
 - spot::taa_tgba, 621
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 654
 - spot::tgba_explicit, 696
 - spot::tgba_explicit_formula, 707
 - spot::tgba_explicit_labelled, 719
 - spot::tgba_explicit_number, 731
 - spot::tgba_explicit_string, 743
 - spot::tgba_reduc, 802
- state_
 - spot::state_bdd, 578
 - spot::state_evtgba_explicit, 582
 - spot::state_explicit, 586
- state_acceptance_conditions
 - spot::fair_kripke, 319
 - spot::kripke, 358
 - spot::tgba_sgba_proxy, 850
- state_bdd
 - spot::state_bdd, 577
- state_condition
 - spot::fair_kripke, 319
 - spot::kripke, 358
- state_couple
 - spot, 101
- state_decl
 - spot::dotty_decorator, 237
 - spot::tgba_run_dotty_decorator, 819
- state_evtgba_explicit
 - spot::state_evtgba_explicit, 581
- state_explicit
 - spot::state_explicit, 585
- state_index
 - spot::numbered_state_heap, 419
 - spot::numbered_state_heap_hash_map, 427
- state_index_p
 - spot::numbered_state_heap, 419
 - spot::numbered_state_heap_hash_map, 427
- state_is_accepting
 - spot::tgba_sba_proxy, 834
- state_name_map_
 - spot::evtgba_explicit, 282
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 660
 - spot::tgba_explicit_formula, 713
 - spot::tgba_explicit_labelled, 726
 - spot::tgba_explicit_number, 737
 - spot::tgba_explicit_string, 750
 - spot::tgba_reduc, 813
- state_predecessor_map_
 - spot::tgba_reduc, 814
- state_product

- spot::state_product, 589
- state_set
 - spot::power_map, 472
 - spot::state_set, 596
 - spot::taa_tgba, 621
 - spot::taa_tgba_formula, 631
 - spot::taa_tgba_labelled, 642
 - spot::taa_tgba_string, 654
- state_set_vec_
 - spot::taa_tgba, 626
 - spot::taa_tgba_formula, 636
 - spot::taa_tgba_labelled, 648
 - spot::taa_tgba_string, 660
- state_union
 - spot::state_union, 602
- states
 - spot::connected_component_hash_set, 187
 - spot::couvreur99_check, 204
 - spot::couvreur99_check_shy, 222
 - spot::couvreur99_check_status, 226
 - spot::ec_statistics, 255
 - spot::power_map, 473
 - spot::scc_map::scc, 530
 - spot::tgba_statistics, 852
- states_
 - spot::ec_statistics, 256
- states_of
 - spot::power_map, 473
 - spot::scc_map, 537
- statistics
 - spot::couvreur99_check, 205
 - spot::couvreur99_check_result, 212
 - spot::couvreur99_check_shy, 222
 - spot::emptiness_check, 262
 - spot::emptiness_check_result, 269
- stats
 - spot::acss_statistics, 119
 - spot::ars_statistics, 128
 - spot::couvreur99_check, 205
 - spot::couvreur99_check_result, 212
 - spot::couvreur99_check_shy, 224
 - spot::ec_statistics, 256
 - spot::unsigned_statistics, 924
 - spot::unsigned_statistics_copy, 926
- stats_map
 - spot::acss_statistics, 118
 - spot::ars_statistics, 127
 - spot::couvreur99_check, 201
 - spot::couvreur99_check_result, 210
 - spot::couvreur99_check_shy, 218
 - spot::ec_statistics, 254
 - spot::unsigned_statistics, 924
 - spot::unsigned_statistics_copy, 925
- stats_reachable
 - tgba_misc, 42
- step
 - eltlyy::location, 377
 - ltlyy::location, 380
 - neverclaimyy::location, 374
 - sautyy::location, 383
 - spot::minato_isop::local_vars, 371
- step_
 - spot::tgba_succ_iterator_product, 866
- step_map
 - spot::tgba_run_dotty_decorator, 818
- step_num
 - spot::tgba_run_dotty_decorator, 818
- step_set
 - spot::tgba_run_dotty_decorator, 818
- steps
 - spot::tgba_run, 815
- stime
 - spot::time_info, 892
 - spot::timer, 894
- stop
 - spot::timer, 894
 - spot::timer_map, 897
- sub_set_acc_cond_
 - spot::parity_game_graph_delayed, 452
- succ
 - spot::scc_map, 537
 - spot::scc_map::scc, 530
- succ_
 - spot::taa_succ_iterator, 617
- succ_iter
 - spot::evtgba, 275
 - spot::evtgba_explicit, 282
 - spot::evtgba_product, 288
 - spot::fair_kripke, 319
 - spot::future_conditions_collector, 343
 - spot::kripke, 358
 - spot::saba, 493
 - spot::saba_complement_tgba, 499
 - spot::taa_tgba, 624
 - spot::taa_tgba_formula, 634
 - spot::taa_tgba_labelled, 646
 - spot::taa_tgba_string, 658
 - spot::tgba, 666
 - spot::tgba_bdd_concrete, 675
 - spot::tgba_explicit, 700
 - spot::tgba_explicit_formula, 711
 - spot::tgba_explicit_labelled, 724
 - spot::tgba_explicit_number, 735
 - spot::tgba_explicit_string, 748
 - spot::tgba_kv_complement, 761
 - spot::tgba_product, 769
 - spot::tgba_product_init, 777
 - spot::tgba_reduc, 811

- spot::tgba_safra_complement, 826
- spot::tgba_sba_proxy, 835
- spot::tgba_scc, 842
- spot::tgba_sgba_proxy, 850
- spot::tgba_tba_proxy, 880
- spot::tgba_union, 888
- succ_iterator
 - spot::ltl::succ_iterator, 607
- succ_queue
 - spot::couvreur99_check_shy, 218
- succ_set_
 - spot::tgba_succ_iterator_concrete, 861
- succ_set_left_
 - spot::tgba_succ_iterator_concrete, 861
- succ_to_string
 - spot::duplicator_node, 243
 - spot::duplicator_node_delayed, 250
 - spot::spoiler_node, 556
 - spot::spoiler_node_delayed, 561
- succ_type
 - spot::scc_map, 534
- successor
 - spot::couvreur99_check_shy::successor, 608
- super
 - spot::ltl::simplify_f_g_visitor, 546
 - spot::ltl::unabbreviate_logic_visitor, 907
 - spot::ltl::unabbreviate_ltl_visitor, 912
- supp
 - spot::scc_map::scc, 530
- supp_rec
 - spot::scc_map::scc, 530
- support_conditions
 - spot::fair_kripke, 320
 - spot::future_conditions_collector, 343
 - spot::kripke, 359
 - spot::taa_tgba, 624
 - spot::taa_tgba_formula, 635
 - spot::taa_tgba_labelled, 647
 - spot::taa_tgba_string, 658
 - spot::tgba, 667
 - spot::tgba_bdd_concrete, 676
 - spot::tgba_explicit, 700
 - spot::tgba_explicit_formula, 712
 - spot::tgba_explicit_labelled, 724
 - spot::tgba_explicit_number, 736
 - spot::tgba_explicit_string, 748
 - spot::tgba_kv_complement, 762
 - spot::tgba_product, 770
 - spot::tgba_product_init, 777
 - spot::tgba_reduc, 812
 - spot::tgba_safra_complement, 827
 - spot::tgba_sba_proxy, 835
 - spot::tgba_scc, 843
 - spot::tgba_sgba_proxy, 851
 - spot::tgba_tba_proxy, 881
 - spot::tgba_union, 889
- symb_merge_
 - spot::ltl::language_containment_checker, 369
- symbol
 - spot::symbol, 610
- symbol_set
 - spot, 101
- syntactic_implication
 - ltl_misc, 17
- syntactic_implication_neg
 - ltl_misc, 17
- taa_succ_iterator
 - spot::taa_succ_iterator, 615
- taa_tgba
 - spot::taa_tgba, 622
- taa_tgba_formula
 - spot::taa_tgba_formula, 631
- taa_tgba_labelled
 - spot::taa_tgba_labelled, 642
- taa_tgba_string
 - spot::taa_tgba_string, 654
- tgba
 - spot::tgba, 664
- TGBA (Transition-based Generalized Büchi Automata), 3
- TGBA algorithms, 30
- TGBA on-the-fly algorithms, 30

- TGBA representations, 29
- TGBA runs and supporting functions, 52
- TGBA simplifications, 35
- tgba/ Directory Reference, 70
- tgba/bdddict.hh, 1035
- tgba/bddprint.hh, 1036
- tgba/formula2bdd.hh, 1038
- tgba/futurecondcol.hh, 1039
- tgba/public.hh, 954
- tgba/state.hh, 1040
- tgba/statebdd.hh, 1042
- tgba/succiter.hh, 1043
- tgba/succiterconcrete.hh, 1044
- tgba/taatgba.hh, 1046
- tgba/tgba.hh, 1047
- tgba/tgbabddconcrete.hh, 1048
- tgba/tgbabddconcretefactory.hh, 1049
- tgba/tgbabddconcreteproduct.hh, 1050
- tgba/tgbabddcoredata.hh, 1052
- tgba/tgbabddfactory.hh, 1053
- tgba/tgbaexplicit.hh, 1055
- tgba/tgbakvcomplement.hh, 1056
- tgba/tgbaproduct.hh, 1057
- tgba/tgbareduc.hh, 1058
- tgba/tgbasafracomplement.hh, 1059
- tgba/tgbascc.hh, 1060
- tgba/tgbasgba.hh, 1061
- tgba/tgbatba.hh, 1063
- tgba/tgbaunion.hh, 1064
- tgba/wdbacomp.hh, 1065
- tgba_reduction
 - Reduce_All, 37
 - Reduce_None, 37
 - Reduce_quotient_Del_Sim, 37
 - Reduce_quotient_Dir_Sim, 37
 - Reduce_Scc, 37
 - Reduce_transition_Del_Sim, 37
 - Reduce_transition_Dir_Sim, 37
- tgba_algorithms
 - product, 30
- tgba_bdd_concrete
 - spot::tgba_bdd_concrete, 672, 673
- tgba_bdd_concrete_factory
 - spot::tgba_bdd_concrete_factory, 680
- tgba_bdd_core_data
 - spot::tgba_bdd_core_data, 686
- tgba_dupexp_bfs
 - tgba_misc, 42
- tgba_dupexp_dfs
 - tgba_misc, 42
- tgba_explicit
 - spot::tgba_explicit, 697
- tgba_explicit_formula
 - spot::tgba_explicit_formula, 707
- tgba_explicit_labelled
 - spot::tgba_explicit_labelled, 719
- tgba_explicit_number
 - spot::tgba_explicit_number, 731
- tgba_explicit_string
 - spot::tgba_explicit_string, 743
- tgba_explicit_succ_iterator
 - spot::tgba_explicit_succ_iterator, 753
- tgba_io
 - dotty_reachable, 25
 - format_neverclaim_parse_errors, 25
 - format_tgba_parse_errors, 25
 - lbt_reachable, 26
 - never_claim_reachable, 26
 - neverclaim_parse, 26
 - neverclaim_parse_error, 24
 - neverclaim_parse_error_list, 24
 - tgba_parse, 27
 - tgba_parse_error, 24
 - tgba_parse_error_list, 25
 - tgba_save_reachable, 27
- tgba_kv_complement
 - spot::tgba_kv_complement, 759
- tgba_ltl
 - eltl_to_tgba_lacim, 32
 - ltl_to_taa, 32
 - ltl_to_tgba_fm, 33
 - ltl_to_tgba_lacim, 34
- tgba_misc
 - random_graph, 41
 - stats_reachable, 42
 - tgba_dupexp_bfs, 42
 - tgba_dupexp_dfs, 42
 - tgba_powerset, 42
- tgba_on_the_fly_algorithms
 - wdba_complement, 31
- tgba_parse
 - tgba_io, 27
- tgba_parse_error
 - tgba_io, 24
- tgba_parse_error_list
 - tgba_io, 25
- tgba_powerset
 - tgba_misc, 42
- tgba_product
 - spot::tgba_product, 767
 - spot::tgba_succ_iterator_product, 866
- tgba_product_init
 - spot::tgba_product_init, 775
- tgba_reachable_iterator
 - spot::tgba_reachable_iterator, 782
- tgba_reachable_iterator_breadth_first
 - spot::tgba_reachable_iterator_breadth_first, 788

- tgba_reachable_iterator_depth_first
 - spot::tgba_reachable_iterator_depth_first, 794
- tgba_reduc
 - spot::tgba_reduc, 803
- tgba_reduction
 - dn_v, 37
 - free_relation_simulation, 37
 - get_delayed_relation_simulation, 37
 - get_direct_relation_simulation, 38
 - minimize_monitor, 38
 - minimize_obligation, 38
 - minimize_wdba, 39
 - reduc_tgba_sim, 40
 - reduce_tgba_options, 37
 - s_v, 37
 - sn_v, 37
- tgba_run
 - print_tgba_run, 53
 - project_tgba_run, 53
 - reduce_run, 54
 - replay_tgba_run, 54
 - spot::tgba_run, 815
 - tgba_run_to_tgba, 54
- tgba_run_dotty_decorator
 - spot::tgba_run_dotty_decorator, 818
- tgba_run_to_tgba
 - tgba_run, 54
- tgba_safracomplement
 - spot::tgba_safracomplement, 824
- tgba_save_reachable
 - tgba_io, 27
- tgba_sba_proxy
 - spot::tgba_sba_proxy, 832
- tgba_scc
 - spot::tgba_scc, 840
- tgba_sgba_proxy
 - spot::tgba_sgba_proxy, 848
- tgba_state_
 - spot::parity_game_graph, 444
 - spot::parity_game_graph_delayed, 452
 - spot::parity_game_graph_direct, 459
- tgba_succ_iterator_concrete
 - spot::tgba_succ_iterator_concrete, 859
- tgba_succ_iterator_product
 - spot::tgba_succ_iterator_product, 865
- tgba_succ_iterator_union
 - spot::tgba_succ_iterator_union, 870
- tgba_tba_proxy
 - spot::tgba_tba_proxy, 878
- tgba_to_evtgba
 - spot, 108
- tgba_union
 - spot::tgba_succ_iterator_union, 872
 - spot::tgba_union, 886
- tgbaalgorithms/ Directory Reference, 71
- tgbaalgorithms/bfssteps.hh, 1065
- tgbaalgorithms/cutsgcc.hh, 1066
- tgbaalgorithms/dotty.hh, 966
- tgbaalgorithms/dottydec.hh, 1067
- tgbaalgorithms/dupeexp.hh, 1068
- tgbaalgorithms/eltl2tgba_lacim.hh, 1069
- tgbaalgorithms/emptiness.hh, 1070
- tgbaalgorithms/emptiness_stats.hh, 1072
- tgbaalgorithms/gtec/ Directory Reference, 60
- tgbaalgorithms/gtec/ce.hh, 1073
- tgbaalgorithms/gtec/expsgcc.hh, 1074
- tgbaalgorithms/gtec/gtec.hh, 1075
- tgbaalgorithms/gtec/nsheap.hh, 1077
- tgbaalgorithms/gtec/sccstack.hh, 1078
- tgbaalgorithms/gtec/status.hh, 1079
- tgbaalgorithms/gv04.hh, 1080
- tgbaalgorithms/lbtt.hh, 1081
- tgbaalgorithms/ltl2taa.hh, 1082
- tgbaalgorithms/ltl2tgba_fm.hh, 1083
- tgbaalgorithms/ltl2tgba_lacim.hh, 1084
- tgbaalgorithms/magic.hh, 1085
- tgbaalgorithms/minimize.hh, 1086
- tgbaalgorithms/neverclaim.hh, 1087
- tgbaalgorithms/powerset.hh, 1088
- tgbaalgorithms/projrun.hh, 1089
- tgbaalgorithms/randomgraph.hh, 1090
- tgbaalgorithms/reachiter.hh, 968
- tgbaalgorithms/reducerun.hh, 1092
- tgbaalgorithms/reducttgba_sim.hh, 1092
- tgbaalgorithms/replayrun.hh, 1094
- tgbaalgorithms/rundotdec.hh, 1094
- tgbaalgorithms/safety.hh, 1095
- tgbaalgorithms/save.hh, 970
- tgbaalgorithms/scc.hh, 1096
- tgbaalgorithms/sccfilter.hh, 1097
- tgbaalgorithms/se05.hh, 1098
- tgbaalgorithms/stats.hh, 1099
- tgbaalgorithms/tau03.hh, 1100
- tgbaalgorithms/tau03opt.hh, 1101
- tgbaalgorithms/weight.hh, 1102
- tgbaalgorithms/ Directory Reference, 73
- tgbaalgorithms/public.hh, 955
- tgbaalgorithms/safracomplement.hh
 - TRANSFORM_TO_TBA, 1060
 - TRANSFORM_TO_TGBA, 1060
- the_acceptance_cond_
 - spot::saba_complement_tgba, 499
 - spot::tgba_kv_complement, 763
 - spot::tgba_tba_proxy, 882
- ThirdStep
 - spot::minato_isop::local_vars, 370
- time_info
 - spot::time_info, 891

- timer
 - spot::timer, [894](#)
 - spot::timer_map, [897](#)
- tm
 - spot::timer_map, [898](#)
- tm_type
 - spot::timer_map, [896](#)
- to_spin_string
 - ltl_io, [8](#), [9](#)
- to_string
 - ltl_io, [9](#)
 - spot::duplicator_node, [243](#)
 - spot::duplicator_node_delayed, [250](#)
 - spot::spoiler_node, [556](#)
 - spot::spoiler_node_delayed, [561](#)
- todo
 - spot::couvreur99_check_shy, [224](#)
 - spot::evtgba_reachable_iterator_breadth_first, [298](#)
 - spot::evtgba_reachable_iterator_depth_first, [303](#)
 - spot::parity_game_graph, [444](#)
 - spot::parity_game_graph_delayed, [452](#)
 - spot::parity_game_graph_direct, [459](#)
 - spot::saba_reachable_iterator_breadth_first, [511](#)
 - spot::saba_reachable_iterator_depth_first, [517](#)
 - spot::tgba_reachable_iterator_breadth_first, [789](#)
 - spot::tgba_reachable_iterator_depth_first, [795](#)
 - spot::tgba_reduc, [814](#)
- todo_
 - spot::minato_isop, [390](#)
 - spot::scc_map, [538](#)
- todo_item
 - spot::couvreur99_check_shy::todo_item, [899](#)
- todo_list
 - spot::couvreur99_check_shy, [218](#)
- top
 - spot::scc_stack, [540](#)
- total_
 - spot::timer, [895](#)
- total_1_
 - spot::ltl::random_ltl, [478](#)
- total_2_
 - spot::ltl::random_ltl, [478](#)
- total_2_and_more_
 - spot::ltl::random_ltl, [478](#)
- trans_map
 - spot::ltl::language_containment_checker, [367](#)
- TRANSFORM_TO_TBA
 - tgbafracomplement.hh, [1060](#)
- TRANSFORM_TO_TGBA
 - tgbafracomplement.hh, [1060](#)
- transition_annotation
 - spot::fair_kripke, [320](#)
 - spot::future_conditions_collector, [344](#)
 - spot::kripke, [359](#)
 - spot::taa_tgba, [625](#)
 - spot::taa_tgba_formula, [635](#)
 - spot::taa_tgba_labelled, [647](#)
 - spot::taa_tgba_string, [659](#)
 - spot::tgba, [667](#)
 - spot::tgba_bdd_concrete, [676](#)
 - spot::tgba_explicit, [701](#)
 - spot::tgba_explicit_formula, [712](#)
 - spot::tgba_explicit_labelled, [725](#)
 - spot::tgba_explicit_number, [736](#)
 - spot::tgba_explicit_string, [749](#)
 - spot::tgba_kv_complement, [762](#)
 - spot::tgba_product, [770](#)
 - spot::tgba_product_init, [778](#)
 - spot::tgba_reduc, [812](#)
 - spot::tgba_safra_complement, [827](#)
 - spot::tgba_sba_proxy, [836](#)
 - spot::tgba_scc, [843](#)
 - spot::tgba_sgba_proxy, [851](#)
 - spot::tgba_tba_proxy, [881](#)
 - spot::tgba_union, [889](#)
- transition_list
 - spot::evtgba_explicit, [280](#)
- transitions
 - spot::couvreur99_check, [205](#)
 - spot::couvreur99_check_shy, [222](#)
 - spot::ec_statistics, [256](#)
 - spot::tgba_statistics, [852](#)
- transitions_
 - spot::ec_statistics, [256](#)
- translated_
 - spot::ltl::language_containment_checker, [369](#)
- Translating LTL formulae into TGBA, [31](#)
- translation
 - spot::ltl::language_containment_checker::record_, [482](#)
- triplet
 - spot::ltl::automatop, [140](#)
- trivial
 - spot::scc_map, [537](#)
 - spot::scc_map::scc, [530](#)
- True
 - spot::ltl::constant, [195](#)
 - spot::ltl::formula_tree, [115](#)
- true_instance
 - spot::ltl::constant, [196](#)
- type
 - spot::ltl::binop, [173](#)
 - spot::ltl::constant, [195](#)
 - spot::ltl::multop, [395](#)

- spot::ltl::unop, 919
- U
 - spot::ltl::binop, 173
- unabbreviate_logic
 - ltl_rewriting, 14
- unabbreviate_logic_visitor
 - spot::ltl::unabbreviate_logic_visitor, 907
- unabbreviate_ltl
 - spot::ltl, 113
- unabbreviate_ltl_visitor
 - spot::ltl::unabbreviate_ltl_visitor, 912
- unop
 - spot::ltl::unop, 919
- unref
 - spot::symbol, 611
- unref_
 - spot::ltl::atomic_prop, 134
 - spot::ltl::automatop, 142
 - spot::ltl::binop, 176
 - spot::ltl::constant, 196
 - spot::ltl::formula, 330
 - spot::ltl::multop, 398
 - spot::ltl::ref_formula, 487
 - spot::ltl::unop, 921
- unregister_all_my_variables
 - spot::bdd_dict, 160
- unregister_variable
 - spot::bdd_dict, 160
- unsigned_fun
 - spot::acss_statistics, 118
 - spot::ars_statistics, 127
 - spot::couvreur99_check, 201
 - spot::couvreur99_check_result, 210
 - spot::couvreur99_check_shy, 218
 - spot::ec_statistics, 254
 - spot::unsigned_statistics, 924
- unsigned_statistics_copy
 - spot::unsigned_statistics_copy, 925
- update_sums
 - spot::ltl::random_ltl, 478
- update_supp_rec
 - spot::scc_map, 537
- useful_acc
 - spot::scc_map::scc, 530
 - spot::scc_stats, 542
- useful_acc_of
 - spot::scc_map, 537
- useless_scc_map
 - spot::scc_stats, 542
- utime
 - spot::time_info, 892
 - spot::timer, 895
- v1
 - spot::minato_isop::local_vars, 371
- val
 - spot::ltl::constant, 197
- val_
 - spot::ltl::constant, 197
- val_name
 - spot::ltl::constant, 197
- var_formula_map
 - spot::bdd_dict, 162
- var_map
 - spot::bdd_dict, 162
- var_refs
 - spot::bdd_dict, 162
- var_set
 - spot::tgba_bdd_core_data, 690
- varandnext_set
 - spot::tgba_bdd_core_data, 690
- vars
 - spot::minato_isop::local_vars, 371
- vec
 - spot::ltl::automatop, 140
 - spot::ltl::multop, 395
- version
 - misc_tools, 20
- vf_map
 - spot::bdd_dict, 157
- visit
 - spot::ltl::clone_visitor, 180, 181
 - spot::ltl::const_visitor, 191
 - spot::ltl::postfix_visitor, 471
 - spot::ltl::simplify_f_g_visitor, 547
 - spot::ltl::unabbreviate_logic_visitor, 908
 - spot::ltl::unabbreviate_ltl_visitor, 913
 - spot::ltl::visitor, 928, 929
- vr_map
 - spot::bdd_dict, 157
- W
 - spot::ltl::binop, 173
- wang32_hash
 - hash_funcs, 21
- want_state
 - spot::parity_game_graph, 443
 - spot::parity_game_graph_delayed, 451
 - spot::parity_game_graph_direct, 458
 - spot::saba_reachable_iterator, 504
 - spot::saba_reachable_iterator_breadth_first, 510
 - spot::saba_reachable_iterator_depth_first, 516
 - spot::tgba_reachable_iterator, 783
 - spot::tgba_reachable_iterator_breadth_first, 789
 - spot::tgba_reachable_iterator_depth_first, 795

- spot::tgba_reduc, [812](#)
- wdba_complement
 - tgba_on_the_fly_algorithms, [31](#)
- weight
 - spot::weight, [930](#)
- weight_vector
 - spot::weight, [930](#)
- where_
 - spot::gspn_exception, [345](#)
 - spot::nips_exception, [404](#)
- X
 - spot::lfl::unop, [919](#)
- Xor
 - spot::lfl::binop, [173](#)