

NVIDIOSO

1.0

Generated by Doxygen 1.8.7

Mon Aug 11 2014 17:35:03



# Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>NVIDIOSO</b>	<b>3</b>
<b>3</b>	<b>Todo List</b>	<b>5</b>
<b>4</b>	<b>Hierarchical Index</b>	<b>7</b>
4.1	Class Hierarchy . . . . .	7
<b>5</b>	<b>Class Index</b>	<b>9</b>
5.1	Class List . . . . .	9
<b>6</b>	<b>Class Documentation</b>	<b>11</b>
6.1	BacktrackableObject Class Reference . . . . .	11
6.1.1	Member Function Documentation . . . . .	11
6.1.1.1	create_memento . . . . .	11
6.1.1.2	get_backtrackable_id . . . . .	12
6.1.1.3	restore_state . . . . .	12
6.1.1.4	set_backtrackable_id . . . . .	12
6.1.1.5	set_memento . . . . .	12
6.1.1.6	set_state . . . . .	12
6.1.1.7	set_state . . . . .	12
6.2	BacktrackManager Class Reference . . . . .	12
6.2.1	Member Function Documentation . . . . .	13
6.2.1.1	add_changed . . . . .	13
6.2.1.2	attach_backtracable . . . . .	13
6.2.1.3	detach_backtracable . . . . .	13
6.2.1.4	force_storage . . . . .	14
6.2.1.5	get_level . . . . .	14
6.2.1.6	number_backtracable . . . . .	14
6.2.1.7	number_changed_backtracable . . . . .	14
6.2.1.8	remove_level . . . . .	14
6.2.1.9	remove_until_level . . . . .	14

6.2.1.10	set_level	15
6.3	BoolDomain Class Reference	15
6.3.1	Member Function Documentation	16
6.3.1.1	get_event	16
6.3.1.2	reset_event	16
6.4	ConcreteDomain< T > Class Template Reference	16
6.4.1	Member Function Documentation	17
6.4.1.1	add	17
6.4.1.2	add	17
6.4.1.3	contains	17
6.4.1.4	get_representation	17
6.4.1.5	get_singleton	17
6.4.1.6	in_max	18
6.4.1.7	in_min	19
6.4.1.8	is_empty	19
6.4.1.9	is_singleton	19
6.4.1.10	print	19
6.4.1.11	set_domain	19
6.4.1.12	shrink	20
6.4.1.13	size	20
6.4.1.14	subtract	20
6.5	Constraint Class Reference	20
6.5.1	Constructor & Destructor Documentation	22
6.5.1.1	Constraint	22
6.5.2	Member Function Documentation	22
6.5.2.1	arguments	22
6.5.2.2	attach_me_to_vars	22
6.5.2.3	changed_vars	22
6.5.2.4	changed_vars_from_event	22
6.5.2.5	consistency	22
6.5.2.6	decompose	23
6.5.2.7	decrease_weight	23
6.5.2.8	events	23
6.5.2.9	fix_point	23
6.5.2.10	get_number_id	23
6.5.2.11	get_scope_size	23
6.5.2.12	get_this_shared_ptr	23
6.5.2.13	increase_weight	23
6.5.2.14	remove_constraint	24
6.5.2.15	satisfied	24

6.5.2.16	scope	24
6.5.2.17	set_consistency_level	24
6.5.2.18	set_event	24
6.5.2.19	unsat_level	24
6.5.2.20	update	25
6.5.3	Member Data Documentation	25
6.5.3.1	_arguments	25
6.5.3.2	_consistency	25
6.5.3.3	_number_id	25
6.5.3.4	_str_id	25
6.5.3.5	_trigger_events	25
6.6	ConstraintStore Class Reference	25
6.6.1	Member Function Documentation	26
6.6.1.1	add_changed	26
6.6.1.2	clear_queue	26
6.6.1.3	consistency	26
6.6.1.4	fail	26
6.6.1.5	getConstraint	27
6.6.1.6	impose	27
6.6.1.7	num_constraints	27
6.6.1.8	num_constraints_to_reevaluate	27
6.6.1.9	sat_check	27
6.7	CPModel Class Reference	27
6.7.1	Member Function Documentation	28
6.7.1.1	add_constraint	28
6.7.1.2	add_constraint_store	28
6.7.1.3	add_search_engine	29
6.7.1.4	add_variable	30
6.7.1.5	attach_constraint_store	30
6.7.1.6	create_constraint_graph	30
6.7.1.7	get_id	30
6.7.1.8	get_search_engine	30
6.7.1.9	init_constraint_store	30
6.7.1.10	set_solutions_limit	30
6.7.1.11	set_timeout_limit	31
6.8	CPSolver Class Reference	31
6.8.1	Constructor & Destructor Documentation	32
6.8.1.1	CPSolver	32
6.8.2	Member Function Documentation	32
6.8.2.1	add_model	32

6.8.2.2	customize	32
6.8.2.3	get_model	32
6.8.2.4	num_models	33
6.8.2.5	num_solved_models	33
6.8.2.6	remove_model	33
6.8.2.7	run	33
6.8.2.8	run	33
6.8.2.9	run_model	33
6.8.2.10	sat_models	34
6.8.2.11	unsat_models	34
6.8.3	Member Data Documentation	34
6.8.3.1	_models	34
6.9	CPStore Class Reference	34
6.9.1	Member Function Documentation	35
6.9.1.1	init_model	35
6.10	CudaConcreteBitmapList Class Reference	35
6.10.1	Constructor & Destructor Documentation	36
6.10.1.1	CudaConcreteBitmapList	36
6.10.2	Member Function Documentation	36
6.10.2.1	add	36
6.10.2.2	add	36
6.10.2.3	contains	37
6.10.2.4	find_next_pair	37
6.10.2.5	find_pair	37
6.10.2.6	find_prev_pair	38
6.10.2.7	get_id_representation	39
6.10.2.8	in_max	39
6.10.2.9	in_min	39
6.10.2.10	print	39
6.10.2.11	set_domain	39
6.10.2.12	shrink	40
6.10.2.13	subtract	40
6.10.3	Member Data Documentation	40
6.10.3.1	_domain_size	40
6.11	CudaConcreteDomain Class Reference	40
6.11.1	Constructor & Destructor Documentation	41
6.11.1.1	CudaConcreteDomain	41
6.11.2	Member Function Documentation	42
6.11.2.1	allocated_bytes	42
6.11.2.2	flush_domain	42

6.11.2.3	<a href="#">get_id_representation</a>	42
6.11.2.4	<a href="#">get_num_chunks</a>	42
6.11.2.5	<a href="#">get_representation</a>	42
6.11.2.6	<a href="#">is_empty</a>	42
6.11.2.7	<a href="#">set_domain</a>	43
6.11.2.8	<a href="#">set_empty</a>	44
6.11.3	<a href="#">Member Data Documentation</a>	44
6.11.3.1	<a href="#">_concrete_domain</a>	44
6.12	<a href="#">CudaConcreteDomainBitmap Class Reference</a>	44
6.12.1	<a href="#">Constructor &amp; Destructor Documentation</a>	45
6.12.1.1	<a href="#">CudaConcreteDomainBitmap</a>	45
6.12.1.2	<a href="#">CudaConcreteDomainBitmap</a>	45
6.12.2	<a href="#">Member Function Documentation</a>	46
6.12.2.1	<a href="#">add</a>	46
6.12.2.2	<a href="#">add</a>	46
6.12.2.3	<a href="#">contains</a>	46
6.12.2.4	<a href="#">get_id_representation</a>	47
6.12.2.5	<a href="#">get_singleton</a>	47
6.12.2.6	<a href="#">IDX_BIT</a>	47
6.12.2.7	<a href="#">IDX_CHUNK</a>	47
6.12.2.8	<a href="#">in_max</a>	47
6.12.2.9	<a href="#">in_min</a>	48
6.12.2.10	<a href="#">is_singleton</a>	48
6.12.2.11	<a href="#">NUM_CHUNKS</a>	48
6.12.2.12	<a href="#">print</a>	48
6.12.2.13	<a href="#">set_domain</a>	48
6.12.2.14	<a href="#">shrink</a>	49
6.12.2.15	<a href="#">subtract</a>	49
6.12.3	<a href="#">Member Data Documentation</a>	49
6.12.3.1	<a href="#">BITS_IN_BYTE</a>	49
6.12.3.2	<a href="#">BITS_IN_CHUNK</a>	49
6.13	<a href="#">CudaConcreteDomainList Class Reference</a>	50
6.13.1	<a href="#">Constructor &amp; Destructor Documentation</a>	50
6.13.1.1	<a href="#">CudaConcreteDomainList</a>	50
6.13.2	<a href="#">Member Function Documentation</a>	51
6.13.2.1	<a href="#">add</a>	51
6.13.2.2	<a href="#">add</a>	51
6.13.2.3	<a href="#">contains</a>	51
6.13.2.4	<a href="#">find_next_pair</a>	51
6.13.2.5	<a href="#">find_pair</a>	51

6.13.2.6	find_prev_pair	52
6.13.2.7	get_id_representation	52
6.13.2.8	get_singleton	52
6.13.2.9	in_max	52
6.13.2.10	in_min	52
6.13.2.11	is_singleton	53
6.13.2.12	print	53
6.13.2.13	set_domain	53
6.13.2.14	shrink	53
6.13.2.15	subtract	53
6.13.3	Member Data Documentation	54
6.13.3.1	_domain_size	54
6.14	CudaDomain Class Reference	54
6.14.1	Member Function Documentation	56
6.14.1.1	add_element	56
6.14.1.2	allocated_bytes	56
6.14.1.3	clone	56
6.14.1.4	contains	56
6.14.1.5	EVT_IDX	57
6.14.1.6	get_concrete_domain	57
6.14.1.7	get_size	57
6.14.1.8	IDX_BIT	57
6.14.1.9	IDX_CHUNK	57
6.14.1.10	init_domain	58
6.14.1.11	num_chunks	58
6.14.1.12	reset_event	58
6.14.1.13	set_bit_representation	59
6.14.1.14	set_bitlist_representation	59
6.14.1.15	set_bounds	60
6.14.1.16	set_concrete_domain	60
6.14.1.17	set_list_representation	60
6.14.1.18	set_singleton	60
6.14.1.19	shrink	60
6.14.1.20	switch_list_to_bitmaplist	60
6.14.2	Member Data Documentation	61
6.14.2.1	_concrete_domain	61
6.14.2.2	_domain	61
6.14.2.3	_num_allocated_bytes	61
6.14.2.4	_num_int_chunks	61
6.14.2.5	BITS_IN_BYTE	61



6.14.2.6	MAX_BYTES_SIZE	61
6.14.2.7	MAX_DOMAIN_VALUES	61
6.14.2.8	MAX_STATUS_SIZE	62
6.14.2.9	SHARED_MEM_KB	62
6.15	CudaGenerator Class Reference	62
6.16	CudaMementoState Class Reference	63
6.16.1	Constructor & Destructor Documentation	63
6.16.1.1	CudaMementoState	63
6.16.2	Member Function Documentation	63
6.16.2.1	set_memento	63
6.17	CudaVariable Class Reference	63
6.17.1	Constructor & Destructor Documentation	64
6.17.1.1	CudaVariable	64
6.17.1.2	CudaVariable	64
6.17.2	Member Function Documentation	64
6.17.2.1	restore_state	64
6.17.2.2	set_domain	64
6.17.2.3	set_domain	64
6.17.2.4	set_domain	65
6.17.2.5	set_state	65
6.18	DataStore Class Reference	65
6.18.1	Constructor & Destructor Documentation	66
6.18.1.1	DataStore	66
6.18.2	Member Function Documentation	66
6.18.2.1	load_model	66
6.19	DepthFirstSearch Class Reference	66
6.19.1	Member Function Documentation	68
6.19.1.1	get_backtracks	68
6.19.1.2	get_nodes	68
6.19.1.3	get_solution	68
6.19.1.4	get_solution	69
6.19.1.5	get_wrong_decisions	69
6.19.1.6	init_search	69
6.19.1.7	label	69
6.19.1.8	labeling	69
6.19.1.9	print_solution	70
6.19.1.10	search_out	70
6.19.1.11	set_backtrack_manager	70
6.19.1.12	set_backtrack_out	70
6.19.1.13	set_debug	70

6.19.1.14	set_heuristic	71
6.19.1.15	set_nodes_out	71
6.19.1.16	set_solution_limit	71
6.19.1.17	set_solution_manager	71
6.19.1.18	set_store	71
6.19.1.19	set_timeout_limit	72
6.19.1.20	set_trail_debug	72
6.19.1.21	set_wrong_decisions_out	72
6.19.2	Member Data Documentation	72
6.19.2.1	_num_backtracks	72
6.19.2.2	_num_nodes	72
6.19.2.3	_num_wrong_decisions	72
6.20	Domain Class Reference	73
6.20.1	Member Function Documentation	74
6.20.1.1	clone	74
6.20.1.2	get_event	74
6.20.1.3	get_size	74
6.20.1.4	get_string_representation	74
6.20.1.5	is_empty	74
6.20.1.6	is_numeric	75
6.20.1.7	is_singleton	75
6.20.1.8	reset_event	75
6.20.1.9	set_type	75
6.21	DomainIterator Class Reference	75
6.21.1	Member Function Documentation	76
6.21.1.1	domain_size	76
6.21.1.2	get_string_representation	76
6.21.1.3	is_numeric	76
6.21.1.4	max_val	76
6.21.1.5	min_val	76
6.21.1.6	random_val	76
6.22	FactoryModelGenerator Class Reference	77
6.23	FactoryParser Class Reference	77
6.24	FirstFail Class Reference	77
6.24.1	Member Function Documentation	77
6.24.1.1	compare	77
6.24.1.2	compare	78
6.25	FZNConstraint Class Reference	78
6.25.1	Constructor & Destructor Documentation	80
6.25.1.1	FZNConstraint	80

6.25.2	Member Function Documentation	81
6.25.2.1	attach_me_to_vars	81
6.25.2.2	consistency	81
6.25.2.3	int_to_type	81
6.25.2.4	name_to_id	81
6.25.2.5	remove_constraint	81
6.25.2.6	satisfied	82
6.25.2.7	setup	82
6.25.2.8	type_to_int	82
6.26	FZNConstraintFactory Class Reference	82
6.26.1	Member Function Documentation	83
6.26.1.1	get_fzn_constraint	83
6.26.1.2	get_fzn_constraint_shr_ptr	84
6.27	FZNParser Class Reference	84
6.27.1	Member Function Documentation	85
6.27.1.1	get_constraint	85
6.27.1.2	get_next_content	85
6.27.1.3	get_search_engine	85
6.27.1.4	get_variable	85
6.28	FZNSearchFactory Class Reference	85
6.28.1	Member Function Documentation	86
6.28.1.1	get_fzn_search_shr_ptr	86
6.29	FZNTokenization Class Reference	87
6.29.1	Member Function Documentation	87
6.29.1.1	get_token	87
6.30	Heuristic Class Reference	87
6.30.1	Member Function Documentation	88
6.30.1.1	get_choice_value	88
6.30.1.2	get_choice_variable	88
6.30.1.3	get_index	88
6.31	IdGenerator Class Reference	89
6.31.1	Constructor & Destructor Documentation	89
6.31.1.1	IdGenerator	89
6.32	InDomainMax Class Reference	90
6.32.1	Member Function Documentation	90
6.32.1.1	metric_value	90
6.33	InDomainMin Class Reference	90
6.33.1	Member Function Documentation	91
6.33.1.1	metric_value	91
6.34	InputData Class Reference	91

6.34.1	Constructor & Destructor Documentation	91
6.34.1.1	InputData	91
6.34.2	Member Function Documentation	91
6.34.2.1	get_in_file	91
6.34.2.2	get_out_file	92
6.34.2.3	max_n_sol	92
6.34.2.4	timeout	92
6.34.2.5	timer	92
6.34.2.6	verbose	92
6.35	InputOrder Class Reference	93
6.35.1	Member Function Documentation	93
6.35.1.1	compare	93
6.35.1.2	compare	93
6.36	IntDomain Class Reference	93
6.36.1	Member Function Documentation	94
6.36.1.1	add_element	94
6.36.1.2	contains	94
6.36.1.3	in_max	94
6.36.1.4	in_min	95
6.36.1.5	init_domain	95
6.36.1.6	set_singleton	95
6.36.1.7	shrink	95
6.36.1.8	subtract	95
6.37	IntLinNe Class Reference	96
6.37.1	Constructor & Destructor Documentation	96
6.37.1.1	IntLinNe	96
6.37.1.2	IntLinNe	97
6.37.2	Member Function Documentation	97
6.37.2.1	consistency	97
6.37.2.2	scope	97
6.38	IntNe Class Reference	97
6.38.1	Constructor & Destructor Documentation	98
6.38.1.1	IntNe	98
6.38.1.2	IntNe	98
6.38.1.3	IntNe	98
6.38.1.4	IntNe	98
6.38.1.5	IntNe	98
6.38.1.6	IntNe	99
6.38.2	Member Function Documentation	99
6.38.2.1	scope	99

6.39 IntVariable Class Reference . . . . .	99
6.39.1 Member Function Documentation . . . . .	100
6.39.1.1 get_size . . . . .	100
6.39.1.2 in_max . . . . .	100
6.39.1.3 in_min . . . . .	100
6.39.1.4 is_empty . . . . .	101
6.39.1.5 is_singleton . . . . .	101
6.39.1.6 max . . . . .	101
6.39.1.7 min . . . . .	101
6.39.1.8 notify_backtrack_manager . . . . .	101
6.39.1.9 notify_observers . . . . .	102
6.39.1.10 set_backtrack_manager . . . . .	102
6.39.1.11 set_backtrackable_id . . . . .	102
6.39.1.12 set_domain . . . . .	102
6.39.1.13 set_domain . . . . .	102
6.39.1.14 set_domain . . . . .	102
6.39.1.15 set_domain_type . . . . .	103
6.39.1.16 shrink . . . . .	103
6.39.1.17 subtract . . . . .	103
6.39.2 Member Data Documentation . . . . .	103
6.39.2.1 _backtrack_manager . . . . .	103
6.39.2.2 _domain_ptr . . . . .	103
6.40 Logger Class Reference . . . . .	104
6.41 Memento Class Reference . . . . .	104
6.41.1 Member Function Documentation . . . . .	105
6.41.1.1 get_state . . . . .	105
6.41.1.2 set_state . . . . .	105
6.42 MementoState Class Reference . . . . .	105
6.43 ModelGenerator Class Reference . . . . .	105
6.43.1 Member Function Documentation . . . . .	106
6.43.1.1 get_constraint . . . . .	106
6.43.1.2 get_search_engine . . . . .	106
6.43.1.3 get_store . . . . .	106
6.43.1.4 get_variable . . . . .	106
6.44 NvdException Class Reference . . . . .	107
6.44.1 Constructor & Destructor Documentation . . . . .	107
6.44.1.1 NvdException . . . . .	107
6.44.1.2 NvdException . . . . .	107
6.44.1.3 NvdException . . . . .	107
6.44.2 Member Function Documentation . . . . .	108

6.44.2.1	what	108
6.45	Parser Class Reference	108
6.45.1	Member Function Documentation	109
6.45.1.1	close	109
6.45.1.2	get_next_content	109
6.45.1.3	get_next_token	109
6.45.1.4	get_variable	110
6.45.1.5	more_tokens	110
6.45.1.6	more_variables	110
6.45.1.7	open	110
6.46	SearchEngine Class Reference	110
6.46.1	Member Function Documentation	111
6.46.1.1	get_backtracks	111
6.46.1.2	get_nodes	111
6.46.1.3	get_solution	111
6.46.1.4	get_solution	111
6.46.1.5	get_wrong_decisions	112
6.46.1.6	label	112
6.46.1.7	labeling	112
6.46.1.8	print_solution	112
6.46.1.9	set_backtrack_manager	113
6.46.1.10	set_backtrack_out	113
6.46.1.11	set_debug	113
6.46.1.12	set_heuristic	113
6.46.1.13	set_nodes_out	113
6.46.1.14	set_solution_limit	114
6.46.1.15	set_solution_manager	114
6.46.1.16	set_store	114
6.46.1.17	set_timeout_limit	114
6.46.1.18	set_trail_debug	114
6.46.1.19	set_wrong_decisions_out	115
6.47	SetDomain Class Reference	115
6.47.1	Member Function Documentation	116
6.47.1.1	get_event	116
6.47.1.2	get_values	116
6.47.1.3	reset_event	116
6.47.1.4	set_values	116
6.48	SimpleBacktrackManager Class Reference	116
6.48.1	Member Function Documentation	117
6.48.1.1	add_changed	117

6.48.1.2	<a href="#">attach_backtracable</a>	117
6.48.1.3	<a href="#">detach_backtracable</a>	118
6.48.1.4	<a href="#">force_storage</a>	118
6.48.1.5	<a href="#">get_level</a>	118
6.48.1.6	<a href="#">number_backtracable</a>	118
6.48.1.7	<a href="#">number_changed_backtracable</a>	118
6.48.1.8	<a href="#">remove_level</a>	118
6.48.1.9	<a href="#">remove_until_level</a>	119
6.48.1.10	<a href="#">set_level</a>	119
6.48.2	<a href="#">Member Data Documentation</a>	119
6.48.2.1	<a href="#">_backtrackable_objects</a>	119
6.48.2.2	<a href="#">_changed_backtrackables</a>	119
6.48.2.3	<a href="#">_trail_stack</a>	119
6.48.2.4	<a href="#">_trail_stack_info</a>	119
6.49	<a href="#">SimpleConstraintStore Class Reference</a>	120
6.49.1	<a href="#">Constructor &amp; Destructor Documentation</a>	120
6.49.1.1	<a href="#">SimpleConstraintStore</a>	120
6.49.2	<a href="#">Member Function Documentation</a>	121
6.49.2.1	<a href="#">add_changed</a>	121
6.49.2.2	<a href="#">clear_queue</a>	121
6.49.2.3	<a href="#">consistency</a>	121
6.49.2.4	<a href="#">fail</a>	121
6.49.2.5	<a href="#">getConstraint</a>	121
6.49.2.6	<a href="#">impose</a>	121
6.49.2.7	<a href="#">num_constraints</a>	122
6.49.2.8	<a href="#">num_constraints_to_reevaluate</a>	122
6.49.2.9	<a href="#">sat_check</a>	122
6.49.3	<a href="#">Member Data Documentation</a>	122
6.49.3.1	<a href="#">_constraint_queue</a>	122
6.49.3.2	<a href="#">_failure</a>	122
6.49.3.3	<a href="#">_lookup_table</a>	123
6.49.3.4	<a href="#">_satisfiability_check</a>	123
6.50	<a href="#">SimpleHeuristic Class Reference</a>	123
6.50.1	<a href="#">Constructor &amp; Destructor Documentation</a>	123
6.50.1.1	<a href="#">SimpleHeuristic</a>	123
6.50.2	<a href="#">Member Function Documentation</a>	124
6.50.2.1	<a href="#">get_choice_value</a>	124
6.50.2.2	<a href="#">get_choice_variable</a>	124
6.50.3	<a href="#">Member Data Documentation</a>	124
6.50.3.1	<a href="#">_fd_variables</a>	124

6.50.3.2	<code>_value_metric</code>	124
6.50.3.3	<code>_variable_metric</code>	124
6.51	SimpleSolutionManager Class Reference	125
6.51.1	Constructor & Destructor Documentation	125
6.51.1.1	SimpleSolutionManager	125
6.51.2	Member Function Documentation	126
6.51.2.1	<code>get_all_solutions</code>	126
6.51.2.2	<code>get_solution</code>	126
6.51.2.3	<code>get_solution</code>	126
6.51.2.4	<code>notify</code>	126
6.51.2.5	<code>number_of_solutions</code>	126
6.51.2.6	<code>print_solution</code>	127
6.51.2.7	<code>set_solution_limit</code>	127
6.51.2.8	<code>set_variables</code>	127
6.51.3	Member Data Documentation	127
6.51.3.1	<code>_max_number_of_solutions</code>	127
6.51.3.2	<code>_solution_strings</code>	127
6.51.3.3	<code>_variables</code>	127
6.52	SolutionManager Class Reference	128
6.52.1	Member Function Documentation	128
6.52.1.1	<code>get_all_solutions</code>	128
6.52.1.2	<code>get_solution</code>	128
6.52.1.3	<code>get_solution</code>	128
6.52.1.4	<code>notify</code>	129
6.52.1.5	<code>number_of_solutions</code>	129
6.52.1.6	<code>print_solution</code>	129
6.52.1.7	<code>set_solution_limit</code>	129
6.52.1.8	<code>set_variables</code>	130
6.53	Solver Class Reference	131
6.53.1	Member Function Documentation	131
6.53.1.1	<code>add_model</code>	131
6.53.1.2	<code>customize</code>	131
6.53.1.3	<code>get_model</code>	132
6.53.1.4	<code>num_models</code>	132
6.53.1.5	<code>num_solved_models</code>	132
6.53.1.6	<code>remove_model</code>	132
6.53.1.7	<code>run</code>	132
6.53.1.8	<code>run</code>	132
6.53.1.9	<code>sat_models</code>	133
6.53.1.10	<code>unsat_models</code>	133



6.54	Statistics Class Reference	133
6.54.1	Member Function Documentation	134
6.54.1.1	get_timer	134
6.54.1.2	set_timer	134
6.54.1.3	stopwatch	134
6.54.1.4	stopwatch_and_add	134
6.55	Token Class Reference	135
6.56	TokenArr Class Reference	135
6.56.1	Member Function Documentation	136
6.56.1.1	get_lower_var	136
6.56.1.2	get_upper_var	136
6.56.1.3	is_var_in	136
6.56.1.4	set_array_bounds	137
6.57	TokenCon Class Reference	137
6.57.1	Member Function Documentation	138
6.57.1.1	add_expr	138
6.57.1.2	get_expr	138
6.57.1.3	get_expr_array	138
6.57.1.4	get_expr_elements_array	138
6.57.1.5	get_expr_not_var_elements_array	139
6.57.1.6	get_expr_var_elements_array	139
6.58	Tokenization Class Reference	139
6.58.1	Member Function Documentation	140
6.58.1.1	analyze_token	140
6.58.1.2	clear_line	140
6.58.1.3	set_new_line	140
6.58.1.4	set_new_tokenizer	140
6.59	TokenSol Class Reference	141
6.59.1	Member Function Documentation	142
6.59.1.1	get_var_to_label	142
6.59.1.2	get_var_to_label	142
6.59.1.3	num_var_to_label	142
6.59.2	Member Data Documentation	142
6.59.2.1	_var_to_label	142
6.60	TokenVar Class Reference	143
6.60.1	Member Function Documentation	144
6.60.1.1	get_range	144
6.60.1.2	get_subset	144
6.60.1.3	get_subset_domain	144
6.60.1.4	set_range_domain	144

6.60.1.5	set_range_domain . . . . .	144
6.60.1.6	set_subset_domain . . . . .	145
6.60.1.7	set_subset_domain . . . . .	145
6.60.1.8	set_subset_domain . . . . .	145
6.60.1.9	set_subset_domain . . . . .	145
6.60.1.10	set_subset_domain . . . . .	145
6.60.1.11	set_var_dom_type . . . . .	145
6.60.1.12	set_var_id . . . . .	146
6.61	ValueChoiceMetric Class Reference . . . . .	146
6.61.1	Member Function Documentation . . . . .	146
6.61.1.1	metric_type . . . . .	146
6.61.1.2	metric_value . . . . .	147
6.62	Variable Class Reference . . . . .	148
6.62.1	Constructor & Destructor Documentation . . . . .	149
6.62.1.1	Variable . . . . .	149
6.62.1.2	Variable . . . . .	149
6.62.2	Member Function Documentation . . . . .	149
6.62.2.1	attach_constraint . . . . .	149
6.62.2.2	attach_store . . . . .	150
6.62.2.3	detach_constraint . . . . .	150
6.62.2.4	detach_constraint . . . . .	150
6.62.2.5	get_size . . . . .	150
6.62.2.6	get_str_id . . . . .	150
6.62.2.7	is_attached . . . . .	151
6.62.2.8	is_empty . . . . .	152
6.62.2.9	is_singleton . . . . .	152
6.62.2.10	notify_constraint . . . . .	152
6.62.2.11	notify_store . . . . .	152
6.62.2.12	set_domain_type . . . . .	152
6.62.2.13	set_str_id . . . . .	152
6.62.2.14	size_constraints . . . . .	153
6.62.2.15	size_constraints_original . . . . .	153
6.62.3	Member Data Documentation . . . . .	153
6.62.3.1	_attached_constraints . . . . .	153
6.62.3.2	_constraint_store . . . . .	153
6.62.3.3	_detach_constraints . . . . .	153
6.62.3.4	domain_iterator . . . . .	153
6.63	VariableChoiceMetric Class Reference . . . . .	154
6.63.1	Member Function Documentation . . . . .	154
6.63.1.1	compare . . . . .	154

---

6.63.1.2	<a href="#">compare</a>	154
6.63.1.3	<a href="#">metric_type</a>	155
6.63.1.4	<a href="#">metric_value</a>	155
<b>Index</b>		<b>156</b>



# Chapter 1

## Main Page

NVIDIOSO NVIDIA-based cOnstraint Solver v. 1.0

\_\_\_CSP/COP REPRESENTATION\_\_\_

VARIABLES:

[Variable](#) has variable types.

- bool: true, false
- int: -42, 0, 69
- set of int: {}, {2, 3, 4}, 1..10

We distinguish between four different types of variables, namely:

- FD Variables: standard Finite [Domain](#) variables
- SUP Variables: SUPport variable introduced to compute the objective function. These variables have unbounded int domains.
- OBJ Variables: OBJective variables. These variables store the objective value as calculated by the objective function through standard propagation. These variables have unbounded int domains.

DOMAINS:

[Domain](#) representation may vary depending on the type of model that is instantiated. In particular, for a CPU model the domains can be represented by lists of sets of domain value. For CUDA models domains are represented as follows. There are two internal representations for an finite domain D depending on whether  $|D| \leq \text{max\_vector}$  or not:

- Bitmap: if  $|D| \leq \text{max\_vector}$ ;
- List of bounds: otherwise.

By default, `max_vector` is equal to 256. This value can be redefined via an environment variable `VECTOR_MAX`.

Domains have the following structure:

| EVT | REP | LB | UB | DSZ || ... BIT ... |

where

- EVT: represents the EVenT happened on the domain;
- REP: is the REPresentation currently used; This value can be one of the following:

- -1, -2, -3, ...: BIT represents a set of 1, 2, 3, ... bitmaps respectively. Each bitmap represents a domain subset of values {LB, UB};
- 0 : BIT represents a Bitmap of contiguous values starting from LB: LB..VECTOR\_MAX.
- 1, 2, 3, ... : in BIT there are respectively 1, 2, 3, ... pairs of bound. If there are 0 pairs, then there is a unique pair of bounds {LB, UB} in the LB/UB field respectively.
- LB: Lower Bound of the current domain;
- UB: Upper Bound of the current domain;
- DSZ: **Domain** SiZe where  $DSZ \leq \max\_vector -> REP = 0$ . Moreover,
  - $\{LB, UB\}' = \{LB, k\} \{k', UB\} -> DSZ' = DSZ - (k' - k + 1)$ ;
  - $LB' = LB + k -> DSZ' = DSZ - (k - LB + 1)$ ;
  - $UB' = UB - k -> DSZ' = DSZ - (UB - k + 1)$ ;
- BIT: bit vector where
  - $REP < 0$ : there is a total of ( $\leq$ ) VECTOR\_MAX bits representing REP pairs of bounds. The first part of BIT is used to store REP pairs <LB, UB>. This bounds do not change anymore even if the correspondend bitmap changes. This is done in order to keep the original offset when clearing bits from the bitmap. The second part of BIT stores the actual bitmaps. Using  $UB - LB + 1$  it is possible to calculate the size of the bitmap and hence the position in BIT of the next pair <LB, UB>. When  $REP < 0$  the BIT field does not change anymore. The system will use the LB/UB fields to check for the right bitmap in the BIT field.
  - $REP = 0$ : there are  $UB - LB + 1 \leq VECTOR\_MAX$  bits of contiguous domain values starting from 0;
  - $REP > 0$ : each pair of bound is identified as LB, UB (LB = UB if singlet). If  $REP = 1$ , then there is only 1 pair of bounds represented by {LB, UB}. If  $REP > 1$ , then there are at least 2 pairs in BIT and the LB/UB fields represent respectively the min/max values among all the pairs.

OBSERVATIONS (CUDA implementation):

Shared Memory:  $49152 = 48$  kB per block -> keep 47 kB available.

- $REP < 0$  there are  $47 * 1024 = 48128 -> (48128 - 5 * 32) / 32 = 1499$  possible storable values. Worst case:  $REP = -256 -> 3 * 256$  triples =  $3 * 256 = 768 < 1499$  ( $-8 = 256 / 32$ ).
- $REP = 0$  and  $VECTOR\_MAX = 4096$  the worst case is when there are 4096 sing.:  $((4096 + 4096 * 2 * 32) / 8) / 1024 = 32.5$  kB  $< 45$  kB  $((tot\_bits + tot\_bits * 2 \text{ int} * bit\_per\_int) / B) / \text{kB}$ .
- $REP > 0$ :  $45 \text{ kB} = 11520 \text{ int} -> 11520 - 5 = 11515 -> 11515 / 2$  (used two int to represent a pair of bounds) = 5757 pairs separated by at least one "hole" from each other ->  $5757 * 2 = 11514$  such as {0, 1}, {3, 4}, ... .

#### Note

The above observation means that when the domains are greater than 11514 then a check must be performed in order to apply multiple copies from global to share memory if needed.

A domain such as {300, 450} has 150 values  $< VECTOR\_MAX$  but it still represented as  $REP < 0$ . This is done for efficiency reasons, avoiding to store a further base-offset for contiguous domains of size  $< VECTOR\_MAX$ .

When a domain (or subsets of it) is (are) represented using a bitmap, the values are stored from right to left using "chunks" of 32 bits (considering a 32bit representation for an unsigned int), where the most significant bit is in the leftmost position of the chunk, i.e., it is the 31th bit. For example, the domain {0, 63} is stored as [63...32|31...0]. The chunk containing a value val is easily computing by  $tot\_chunks - (val / 32)$ , where  $tot\_chunks$  is the total number of chunks used for representing a domain. The position of val within the chunk is given by  $val \% 32$ .

## Chapter 2

# NVIDIOSO

NVIDIOSO - NVidia-based cOnstraint SOLver v. 1.0





## Chapter 3

# Todo List

**Member `BoolDomain::get_event () const`**

implement this function

**Member `CudaConcreteBitmapList::add (int min, int max)`**

complete add function to add any bitmap.

**Member `CudaConcreteDomainBitmap::add (int min, int max)`**

implement using checks on chunks of bits (i.e. sublinear cost).

**Member `CudaVariable::set_domain (std::vector< std::vector< int > > elems)`**

implement set of sets of elements.

**Member `IntVariable::set_domain (std::vector< std::vector< int > > elems)=0`**

implement set of sets of elements.

**Member `SetDomain::get_event () const`**

implement this function

**Member `SimpleBacktrackManager::_trail_stack_info`**

implement this functionality.



## Chapter 4

# Hierarchical Index

### 4.1 Class Hierarchy

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

BacktrackableObject . . . . .	??
IntVariable . . . . .	??
CudaVariable . . . . .	??
BacktrackManager . . . . .	??
SimpleBacktrackManager . . . . .	??
ConcreteDomain< T > . . . . .	??
ConcreteDomain< int > . . . . .	??
CudaConcreteDomain . . . . .	??
CudaConcreteDomainBitmap . . . . .	??
CudaConcreteBitmapList . . . . .	??
CudaConcreteDomainList . . . . .	??
ConstraintStore . . . . .	??
SimpleConstraintStore . . . . .	??
CPModel . . . . .	??
DataStore . . . . .	??
CPStore . . . . .	??
Domain . . . . .	??
BoolDomain . . . . .	??
IntDomain . . . . .	??
CudaDomain . . . . .	??
SetDomain . . . . .	??
DomainIterator . . . . .	??
enable_shared_from_this	
Constraint . . . . .	??
FZNConstraint . . . . .	??
IntLinNe . . . . .	??
IntNe . . . . .	??
exception	
NvdException . . . . .	??
FactoryModelGenerator . . . . .	??
FactoryParser . . . . .	??
FZNConstraintFactory . . . . .	??
FZNSearchFactory . . . . .	??
Heuristic . . . . .	??
SimpleHeuristic . . . . .	??

IdGenerator . . . . .	??
InputData . . . . .	??
Logger . . . . .	??
Memento . . . . .	??
MementoState . . . . .	??
CudaMementoState . . . . .	??
ModelGenerator . . . . .	??
CudaGenerator . . . . .	??
Parser . . . . .	??
FZNParser . . . . .	??
SearchEngine . . . . .	??
DepthFirstSearch . . . . .	??
SolutionManager . . . . .	??
SimpleSolutionManager . . . . .	??
Solver . . . . .	??
CPSolver . . . . .	??
Statistics . . . . .	??
Token . . . . .	??
TokenCon . . . . .	??
TokenSol . . . . .	??
TokenVar . . . . .	??
TokenArr . . . . .	??
Tokenization . . . . .	??
FZNTokenization . . . . .	??
ValueChoiceMetric . . . . .	??
InDomainMax . . . . .	??
InDomainMin . . . . .	??
Variable . . . . .	??
IntVariable . . . . .	??
VariableChoiceMetric . . . . .	??
FirstFail . . . . .	??
InputOrder . . . . .	??

## Chapter 5

# Class Index

### 5.1 Class List

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

BacktrackableObject	??
BacktrackManager	??
BoolDomain	??
ConcreteDomain< T >	??
Constraint	??
ConstraintStore	??
CPModel	??
CPSolver	??
CPStore	??
CudaConcreteBitmapList	??
CudaConcreteDomain	??
CudaConcreteDomainBitmap	??
CudaConcreteDomainList	??
CudaDomain	??
CudaGenerator	??
CudaMementoState	??
CudaVariable	??
DataStore	??
DepthFirstSearch	??
Domain	??
DomainIterator	??
FactoryModelGenerator	??
FactoryParser	??
FirstFail	??
FZNConstraint	??
FZNConstraintFactory	??
FZNParser	??
FZNSearchFactory	??
FZNTokenization	??
Heuristic	??
IdGenerator	??
InDomainMax	??
InDomainMin	??
InputData	??
InputOrder	??
IntDomain	??
IntLinNe	??
IntNe	??

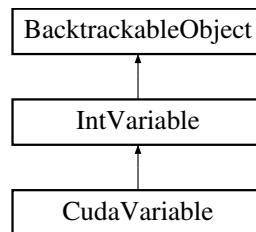
<a href="#">IntVariable</a>	??
<a href="#">Logger</a>	??
<a href="#">Memento</a>	??
<a href="#">MementoState</a>	??
<a href="#">ModelGenerator</a>	??
<a href="#">NvdException</a>	??
<a href="#">Parser</a>	??
<a href="#">SearchEngine</a>	??
<a href="#">SetDomain</a>	??
<a href="#">SimpleBacktrackManager</a>	??
<a href="#">SimpleConstraintStore</a>	??
<a href="#">SimpleHeuristic</a>	??
<a href="#">SimpleSolutionManager</a>	??
<a href="#">SolutionManager</a>	??
<a href="#">Solver</a>	??
<a href="#">Statistics</a>	??
<a href="#">Token</a>	??
<a href="#">TokenArr</a>	??
<a href="#">TokenCon</a>	??
<a href="#">Tokenization</a>	??
<a href="#">TokenSol</a>	??
<a href="#">TokenVar</a>	??
<a href="#">ValueChoiceMetric</a>	??
<a href="#">Variable</a>	??
<a href="#">VariableChoiceMetric</a>	??

## Chapter 6

# Class Documentation

### 6.1 BacktrackableObject Class Reference

Inheritance diagram for BacktrackableObject:



#### Public Member Functions

- virtual [Memento](#) \* [create\\_memento](#) ()
- virtual void [set\\_memento](#) ([Memento](#) &m)
- virtual void [set\\_state](#) ([MementoState](#) \*state)
- virtual int [get\\_backtrackable\\_id](#) () const
- virtual void [set\\_backtrackable\\_id](#) ()=0
- virtual void [restore\\_state](#) ()=0
- virtual void [set\\_state](#) ()=0

#### Protected Attributes

- int [\\_backtrackable\\_id](#)  
*Unique identifier for this backtrackable object.*
- [MementoState](#) \* [\\_current\\_state](#)  
*[Memento](#) hold by this this backtrackable object.*

#### 6.1.1 Member Function Documentation

6.1.1.1 virtual [Memento](#)\* [BacktrackableObject::create\\_memento](#) ( ) [inline],[virtual]

Create a new memento object (state).

**Returns**

a reference to a new memento.

**6.1.1.2** `virtual int BacktrackableObject::get_backtrackable_id ( ) const [inline],[virtual]`

Returns the unique id of this backtrackable object.

**Returns**

the unique id of this backtrackable object.

**6.1.1.3** `virtual void BacktrackableObject::restore_state ( ) [pure virtual]`

Restore a state from the current state hold by the [BacktrackableObject](#).

Implemented in [CudaVariable](#).

**6.1.1.4** `virtual void BacktrackableObject::set_backtrackable_id ( ) [pure virtual]`

Set unique id for this backtrackable object. Concrete backtracable objects are required to implement this method so any backtrackable object has its unique id.

Implemented in [IntVariable](#).

**6.1.1.5** `virtual void BacktrackableObject::set_memento ( Memento & m ) [inline],[virtual]`

Set a memento as current state.

**Parameters**

<i>m</i>	the memento to set as current state.
----------	--------------------------------------

**6.1.1.6** `virtual void BacktrackableObject::set_state ( MementoState * state ) [inline],[virtual]`

Set the current state of this backtrackable object.

**Parameters**

<i>state</i>	the current state to set.
--------------	---------------------------

**6.1.1.7** `virtual void BacktrackableObject::set_state ( ) [pure virtual]`

Set internal state with other information hold by concrete [BacktrackableObject](#) objects.

Implemented in [CudaVariable](#).

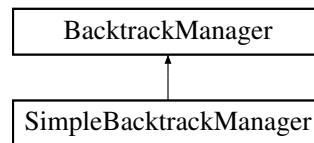
The documentation for this class was generated from the following file:

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/backtrackable\_object.h

## 6.2 BacktrackManager Class Reference

Inheritance diagram for BacktrackManager:





## Public Member Functions

- virtual void [attach\\_backtracable](#) ([BacktrackableObject](#) \*bkt\_obj)=0
- virtual void [detach\\_backtracable](#) (size\_t bkt\_id)=0
- virtual void [add\\_changed](#) (size\_t idx)=0
- virtual size\_t [get\\_level](#) () const =0
- virtual void [set\\_level](#) (size\_t lvl)=0
- virtual void [force\\_storage](#) ()=0
- virtual void [remove\\_level](#) (size\_t lvl)=0
- virtual void [remove\\_until\\_level](#) (size\_t lvl)=0
- virtual size\_t [number\\_backtracable](#) () const =0
- virtual size\_t [number\\_changed\\_backtracable](#) () const =0
- virtual void [print](#) () const =0

*Print information about this backtrack manager.*

### 6.2.1 Member Function Documentation

#### 6.2.1.1 virtual void BacktrackManager::add\_changed ( size\_t idx ) [pure virtual]

Informs the manager that a given backtrackable object has changed at a given level.

##### Parameters

<i>idx</i>	the (unique) id of the backtrackable object which is changed.
------------	---

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.2 virtual void BacktrackManager::attach\_backtracable ( BacktrackableObject \* bkt\_obj ) [pure virtual]

Register a backtrackable object to this manager using the unique id of the backtrackable object.

##### Parameters

<i>bkt_obj</i>	a reference to a backtrackable object.
----------------	--

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.3 virtual void BacktrackManager::detach\_backtracable ( size\_t bkt\_id ) [pure virtual]

Detaches a backtrackable object from this manager, so its state won't be restored anymore.

##### Parameters

<i>bkt_id</i>	the id of the backtrackable object to detach.
---------------	---

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.4 virtual void BacktrackManager::force\_storage ( ) [pure virtual]

Forces the storage of all the backtrackable objects attached to this manager (at next set\_level call), no matter if a backtrackable object has been modified or not.

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.5 virtual size\_t BacktrackManager::get\_level ( ) const [pure virtual]

Get the current active level.

##### Returns

current active level in the manager.

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.6 virtual size\_t BacktrackManager::number\_backtracable ( ) const [pure virtual]

Returns the number of backtrackable objects attached to this backtrack manager.

##### Returns

number of objects attached to this manager.

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.7 virtual size\_t BacktrackManager::number\_changed\_backtracable ( ) const [pure virtual]

Returns the number of changed backtrackable objects from last call to set\_level in this backtrack manager.

##### Returns

number of changed objects.

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.8 virtual void BacktrackManager::remove\_level ( size\_t lvl ) [pure virtual]

Removes a level. It performs a backtrack from that level.

##### Parameters

lvl	the level which is being removed.
-----	-----------------------------------

Implemented in [SimpleBacktrackManager](#).

#### 6.2.1.9 virtual void BacktrackManager::remove\_until\_level ( size\_t lvl ) [pure virtual]

Removes all levels until the one given as input. It performs backtrack until the level given as input.

##### Parameters

<code>lvl</code>	the level to backtrack to.
------------------	----------------------------

Implemented in [SimpleBacktrackManager](#).

6.2.1.10 `virtual void BacktrackManager::set_level ( size_t lvl ) [pure virtual]`

Specifies the level which should become the active one in the manager.

Parameters

<code>lvl</code>	the active level at which the changes will be recorded.
------------------	---

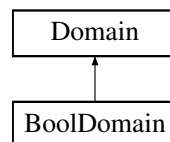
Implemented in [SimpleBacktrackManager](#).

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/backtrack_manager.h`

## 6.3 BoolDomain Class Reference

Inheritance diagram for BoolDomain:



### Public Member Functions

- `DomainPtr clone () const`  
*Clone the current domain and returns a pointer to it.*
- `EventType get_event () const`
- `void reset_event ()`
- `size_t get_size () const`  
*Returns the size of the domain.*
- `bool is_empty () const`  
*Returns true if the domain is empty.*
- `bool is_singleton () const`  
*Returns true if the domain has only one element.*
- `bool is_numeric () const`  
*Returns true if this is a numeric finite domain.*
- `std::string get_string_representation () const`  
*Get string rep. of this domain.*
- `void print () const`  
*Print info about the domain.*

### Protected Member Functions

- `DomainPtr clone_impl () const`  
*Clone the current domain.*

## Protected Attributes

- BoolValue [\\_bool\\_value](#)  
*Current domain value.*

## Additional Inherited Members

### 6.3.1 Member Function Documentation

#### 6.3.1.1 EventType BoolDomain::get\_event ( ) const [virtual]

Get event on this domain

**Todo** implement this function

Implements [Domain](#).

#### 6.3.1.2 void BoolDomain::reset\_event ( ) [virtual]

Sets the no event on this domain.

#### Note

No event won't trigger any propagation on this domain.

Implements [Domain](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/bool\_domain.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/bool\_domain.cpp

## 6.4 ConcreteDomain< T > Class Template Reference

### Public Member Functions

- virtual unsigned int [size](#) ( ) const =0
- virtual T [lower\\_bound](#) ( ) const =0  
*Returns lower bound.*
- virtual T [upper\\_bound](#) ( ) const =0  
*Returns upper bound.*
- virtual void [shrink](#) (T min, T max)=0
- virtual void [subtract](#) (T value)=0
- virtual void [in\\_min](#) (T min)=0
- virtual void [in\\_max](#) (T max)=0
- virtual void [add](#) (T value)=0
- virtual void [add](#) (T min, T max)=0
- virtual bool [contains](#) (T value) const =0
- virtual bool [is\\_empty](#) ( ) const =0
- virtual bool [is\\_singleton](#) ( ) const =0
- virtual T [get\\_singleton](#) ( ) const =0
- virtual void [set\\_domain](#) (void \*const domain, int rep, int min, int max, int dsz)=0
- virtual const void \* [get\\_representation](#) ( ) const =0
- virtual void [print](#) ( ) const =0

### 6.4.1 Member Function Documentation

6.4.1.1 `template<class T> virtual void ConcreteDomain< T >::add ( T value ) [pure virtual]`

It computes union of this domain and {value}.

Parameters

<i>value</i>	it specifies the value which is being added.
--------------	--

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

6.4.1.2 `template<class T> virtual void ConcreteDomain< T >::add ( T min, T max ) [pure virtual]`

It computes union of this domain and {min, max}.

Parameters

<i>min</i>	lower bound of the new domain which is being added.
<i>max</i>	upper bound of the new domain which is being added.

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

6.4.1.3 `template<class T> virtual bool ConcreteDomain< T >::contains ( T value ) const [pure virtual]`

It checks whether the value belongs to the domain or not.

Parameters

<i>value</i>	to check whether it is in the current domain.
--------------	---

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

6.4.1.4 `template<class T> virtual const void* ConcreteDomain< T >::get_representation ( ) const [pure virtual]`

It returns a void pointer to an object representing the current representation of the domain (e.g., bitmap).

Returns

void pointer to the concrete domain representation.

Implemented in [CudaConcreteDomain](#).

6.4.1.5 `template<class T> virtual T ConcreteDomain< T >::get_singleton ( ) const [pure virtual]`

It returns the value of type T of the domain if it is a singleton.

Returns

the value of the singleton element.

Note

Classes that specialize this method should handle the case of an invocation of the method and a non-singleton domain. For example, throw an exception or returning the lower bound.

Implemented in [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

6.4.1.6 `template<class T> virtual void ConcreteDomain< T >::in_max ( T max )` [pure virtual]

It updates the domain according to the maximum value.

## Parameters

<i>max</i>	domain value.
------------	---------------

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

**6.4.1.7** `template<class T> virtual void ConcreteDomain< T >::in_min ( T min ) [pure virtual]`

It updates the domain according to the minimum value.

## Parameters

<i>min</i>	domain value.
------------	---------------

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

**6.4.1.8** `template<class T> virtual bool ConcreteDomain< T >::is_empty ( ) const [pure virtual]`

It checks whether the current domain is empty.

## Returns

true if the current domain is empty, false otherwise.

Implemented in [CudaConcreteDomain](#).

**6.4.1.9** `template<class T> virtual bool ConcreteDomain< T >::is_singleton ( ) const [pure virtual]`

It checks whether the current domain contains only an element (i.e., it is a singleton).

## Returns

true if the current domain is singleton, false otherwise.

Implemented in [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

**6.4.1.10** `template<class T> virtual void ConcreteDomain< T >::print ( ) const [pure virtual]`

It prints the current domain representation (its state).

## Note

it prints the content of the object given by "get\_representation ()" .

Implemented in [CudaConcreteDomainBitmap](#), [CudaConcreteBitmapList](#), and [CudaConcreteDomainList](#).

**6.4.1.11** `template<class T> virtual void ConcreteDomain< T >::set_domain ( void *const domain, int rep, int min, int max, int dsz ) [pure virtual]`

Sets the internal representation of the domain from a given concrete domain and given lower/upper bounds.

## Parameters

<i>domain</i>	a reference to a given concrete domain.
<i>rep</i>	current internal's domain representation.
<i>min</i>	lower bound to set.
<i>max</i>	upper bound to set.
<i>dsz</i>	domain size to set.

**Note**

the client must pass a valid concrete domain's representation.

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), [CudaConcreteDomain](#), and [CudaConcreteDomainList](#).

**6.4.1.12** `template<class T> virtual void ConcreteDomain< T >::shrink ( T min, T max )` [pure virtual]

It updates the domain to have values only within min/max.

**Parameters**

<i>min</i>	new lower bound to set for the current domain.
<i>max</i>	new upper bound to set for the current domain.

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

**6.4.1.13** `template<class T> virtual unsigned int ConcreteDomain< T >::size ( ) const` [pure virtual]

It returns the number of elements in the domain. It returns the current size of the domain.

Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

**6.4.1.14** `template<class T> virtual void ConcreteDomain< T >::subtract ( T value )` [pure virtual]

It subtracts {value} from the current domain.

**Parameters**

<i>value</i>	the value to subtract from the current domain.
--------------	--

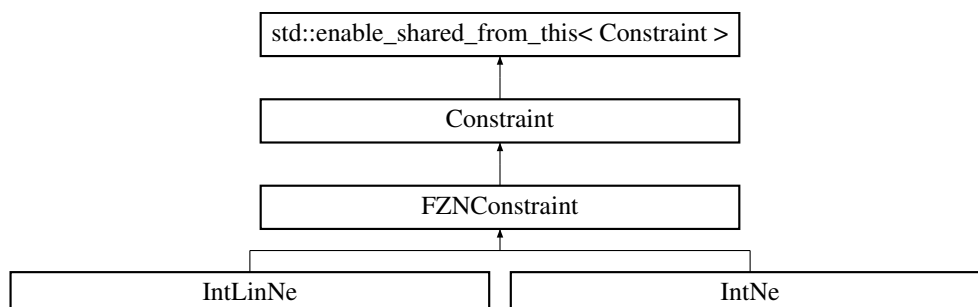
Implemented in [CudaConcreteBitmapList](#), [CudaConcreteDomainBitmap](#), and [CudaConcreteDomainList](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/concrete\_domain.h

## 6.5 Constraint Class Reference

Inheritance diagram for Constraint:





## Public Member Functions

- `size_t get_unique_id () const`  
*Get unique (global) id of this constraint.*
- `int get_number_id () const`
- `std::string get_name () const`  
*Get the name id of this constraint.*
- `int get_weight () const`  
*Get the weight of this constraint.*
- `void set_consistency_level (ConsistencyType con_type)`
- `void increase_weight (int weight=1)`
- `void decrease_weight (int weight=1)`
- `size_t get_scope_size () const`
- `size_t get_arguments_size () const`  
*Get the size of the auxiliary arguments of this constraint.*
- `virtual void set_event (EventType event=EventType::CHANGE_EVT)`
- `const std::vector< EventType > & events () const`
- `const std::vector< int > & arguments () const`
- `virtual void update (EventType e)`
- `virtual std::vector< ConstraintPtr > decompose () const`
- `virtual std::vector< VariablePtr > changed_vars_from_event (EventType event) const`
- `virtual std::vector< VariablePtr > changed_vars () const`
- `virtual bool fix_point () const`
- `virtual int unsat_level () const`
- `virtual const std::vector< VariablePtr > scope () const =0`
- `virtual void attach_me_to_vars ()=0`
- `virtual void consistency ()=0`
- `virtual bool satisfied ()=0`
- `virtual void remove_constraint ()=0`
- `virtual void print () const =0`  
*Prints info.*
- `virtual void print_semantic () const =0`  
*Prints the semantic of this constraint.*

## Protected Member Functions

- `Constraint ()`
- `virtual ConstraintPtr get_this_shared_ptr ()`

## Protected Attributes

- `std::string _dbg`  
*Debug string.*
- `int _number_id`
- `std::string _str_id`
- `ConsistencyType _consistency`
- `std::vector< EventType > _trigger_events`
- `std::vector< int > _arguments`

## 6.5.1 Constructor & Destructor Documentation

### 6.5.1.1 `Constraint::Constraint ( )` `[protected]`

Default constructor. It creates a new instance of a null constraint with a new unique id. It sets all the other members to null.

## 6.5.2 Member Function Documentation

### 6.5.2.1 `const std::vector< int > & Constraint::arguments ( ) const`

It returns the list of auxiliary arguments of a given constraint.

### 6.5.2.2 `virtual void Constraint::attach_me_to_vars ( )` `[pure virtual]`

It attaches this constraint (observer) to the list of the variables in its scope. When a variable changes state, this constraint could be automatically notified (depending on the variable).

Implemented in [FZNConstraint](#).

### 6.5.2.3 `std::vector< VariablePtr > Constraint::changed_vars ( ) const` `[virtual]`

It returns the vector of (pointers to) all variables for which the corresponding domains have been modified by the propagation/consistency of this constraint.

#### Returns

a vector of (pointers to) variables which domains have been modified after the propagation of this constraint. It returns null if no domain has been modified.

### 6.5.2.4 `std::vector< VariablePtr > Constraint::changed_vars_from_event ( EventType event ) const` `[virtual]`

It returns the vector of (pointers to) variables that correspond to the variables for which the domains have been modified by the propagation/consistency of this constraint w.r.t. a given event.

#### Parameters

<i>event</i>	the event to that may be happened on some domain of the variables of the scope of this constraint.
--------------	--

#### Returns

a vector of (pointers to) variables which domains have been modified after the propagation of this constraint. It returns null if no domain has been modified.

### 6.5.2.5 `virtual void Constraint::consistency ( )` `[pure virtual]`

It is a (most probably incomplete) consistency function which removes the values from variable domains. Only values which do not have any support in a solution space are removed.

Implemented in [FZNConstraint](#), [IntNe](#), and [IntLinNe](#).

**6.5.2.6** `std::vector< ConstraintPtr > Constraint::decompose ( ) const` `[virtual]`

It returns a vector of (pointers to) constraints which are used to decompose this constraint. It actually creates a decomposition (possibly also creating variables), but it does not impose the constraints.

#### Returns

a vector of (pointers to) constraints used to decompose this constraint.

**6.5.2.7** `void Constraint::decrease_weight ( int weight = 1 )`

Decrease current weight.

#### Parameters

<i>weight</i>	the weight to decrease from the current weight (default: 1).
---------------	--

**6.5.2.8** `const std::vector< EventType > & Constraint::events ( ) const`

It returns the list of events that trigger a given constraint.

**6.5.2.9** `bool Constraint::fix_point ( ) const` `[virtual]`

It checks if the constraint has reached the fixed point, i.e., it checks whether no events happened on the domains of the variables in the scope of the this constraint.

**6.5.2.10** `int Constraint::get_number_id ( ) const`

Get number id of this constraint.

#### Note

same type of constraints have same number\_id.

**6.5.2.11** `size_t Constraint::get_scope_size ( ) const`

Get the size of the scope of this constraint, i.e., the number of FD variables which is defined on.

#### Note

The size of the scope does not correspond to the formal definition of the constraint but with the actual number of variables within the scope of a given constraint. For example: `int_eq ( x, y )` has `_scope_size` equal to 2; `int_eq ( x, 1 )` has `_scope_size` equal to 1.

**6.5.2.12** `ConstraintPtr Constraint::get_this_shared_ptr ( )` `[protected], [virtual]`

Create a shared pointer from this instance.

#### Returns

a shared pointer to [Constraint](#) object.

**6.5.2.13** `void Constraint::increase_weight ( int weight = 1 )`

Increase current weight.

## Parameters

<i>weight</i>	the weight to add to the current weight (default: 1).
---------------	---

**6.5.2.14** `virtual void Constraint::remove_constraint ( ) [pure virtual]`

It removes the constraint by removing this constraint from all variables in its scope.

Implemented in [FZNConstraint](#).

**6.5.2.15** `virtual bool Constraint::satisfied ( ) [pure virtual]`

It checks if the constraint is satisfied.

## Returns

true if the constraint is for certain satisfied, false otherwise.

## Note

If this function is incorrectly implemented, a constraint may not be satisfied in a solution.

Implemented in [FZNConstraint](#), [IntNe](#), and [IntLinNe](#).

**6.5.2.16** `virtual const std::vector<VariablePtr> Constraint::scope ( ) const [pure virtual]`

It returns the vector of (shared) pointers of all the variables involved in a given constraint (i.e., its scope).

Implemented in [IntNe](#), and [IntLinNe](#).

**6.5.2.17** `void Constraint::set_consistency_level ( ConsistencyType con_type )`

Set the consistency level for this constraints. Different consistency levels are implemented with different algorithms and may require different computational times.

**6.5.2.18** `void Constraint::set_event ( EventType event = EventType::CHANGE_EVT ) [virtual]`

Set an event as triggering event for re-evaluation of this constraint.

## Parameters

<i>event</i>	the event that will trigger the re-evaluation of this constraint.
--------------	---

## Note

default: CHANGE\_EVT.

different constraints should specialize this method with the appropriate list of events.

**6.5.2.19** `int Constraint::unsat_level ( ) const [virtual]`

It returns an integer value that can be used to represent how much the current constraint is unsatisfied. This function can be used to implement some heuristics for optimization problems.

## Returns

an integer value representing how much this constraint is unsatisfied. It returns 0 if this constraint is satisfied.

**6.5.2.20 void Constraint::update ( EventType e ) [virtual]**

It receives an update about an action that has been performed on some variables and it acts accordingly. This method is used to trigger some actions when this observer observes a change in the state of some observed subject.

**Parameters**

<b>e</b>	an object of type Event that specifies the event that triggered the update.
----------	---

**6.5.3 Member Data Documentation****6.5.3.1 std::vector<int> Constraint::\_arguments [protected]**

It represents the array of auxiliary arguments needed by a given constraint in order to be propagated. For example: `int_eq ( x, 2 )` has 2 as auxiliary argument.

**6.5.3.2 ConsistencyType Constraint::\_consistency [protected]**

It specifies which kind of consistency the constraint must ensure. There are at least two types of consistency: 1 - bound consistency 2 - domain consistency Default is bound consistency.

**6.5.3.3 int Constraint::\_number\_id [protected]**

It specifies the number id for a given constraint. All constraints within the same type have unique number ids.

**6.5.3.4 std::string Constraint::\_str\_id [protected]**

It specifies the string id of the constraint. If it is null, then the string id is created from string associated for the constraint type and the `_number_id` of the constraint.

**6.5.3.5 std::vector<EventType> Constraint::\_trigger\_events [protected]**

It specifies the events which trigger the propagation of a given constraint.

**Note**

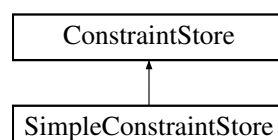
see [domain.h](#) for the list of events of type "EventType".

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/constraint.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/constraint.cpp

**6.6 ConstraintStore Class Reference**

Inheritance diagram for ConstraintStore:



## Public Member Functions

- virtual void [fail](#) ()=0
- virtual void [sat\\_check](#) (bool sat\_check=true)=0
- virtual void [add\\_changed](#) (std::vector< size\_t > &c\_id, EventType event)=0
- virtual void [impose](#) (ConstraintPtr c)=0
- virtual bool [consistency](#) ()=0
- virtual [Constraint](#) \*const [getConstraint](#) ()=0
- virtual void [clear\\_queue](#) ()=0
- virtual size\_t [num\\_constraints](#) () const =0
- virtual size\_t [num\\_constraints\\_to\\_reevaluate](#) () const =0
- virtual void [print](#) () const =0

*Print information about this constraint store.*

### 6.6.1 Member Function Documentation

**6.6.1.1** virtual void [ConstraintStore::add\\_changed](#) ( std::vector< size\_t > & *c\_id*, EventType *event* ) [pure virtual]

It adds the constraints given in input to the queue of constraint to re-evaluate.

#### Parameters

<i>c_id</i>	the vector of constraints ids to re-evaluate.
<i>event</i>	the event that has triggered the re-evaluation of the given list of constraints.

#### Note

only constraints that have been previously attached/imposed to this constraint store will be re-evaluated.

Implemented in [SimpleConstraintStore](#).

**6.6.1.2** virtual void [ConstraintStore::clear\\_queue](#) ( ) [pure virtual]

Clears the queue of constraints to re-evaluate. It can be used when implementing different scheme of constraint propagation.

Implemented in [SimpleConstraintStore](#).

**6.6.1.3** virtual bool [ConstraintStore::consistency](#) ( ) [pure virtual]

Computes the consistency function. This function propagates the constraints that are in the constraint queue until the queue is empty.

#### Returns

true if all propagate constraints are consistent, false otherwise.

Implemented in [SimpleConstraintStore](#).

**6.6.1.4** virtual void [ConstraintStore::fail](#) ( ) [pure virtual]

Informs the constraint store that something bad happened somewhere else. This forces the store to clean up everything and exit as soon as possible without re-evaluating any constraint.

Implemented in [SimpleConstraintStore](#).

#### 6.6.1.5 `virtual Constraint* const ConstraintStore::getConstraint ( ) [pure virtual]`

Returns a constraint that is scheduled for re-evaluation. The basic implementation is first-in-first-out. The constraint is hence remove from the constraint queue, since it is assumed that it will be re-evaluated right away.

##### Returns

a const pointer to a constraint to re-evaluate.

Implemented in [SimpleConstraintStore](#).

#### 6.6.1.6 `virtual void ConstraintStore::impose ( ConstraintPtr c ) [pure virtual]`

Imposes a constraint to the store. The constraint is added to the list of constraints in this constraint store as well as to the queue of constraint to re-evaluate next call to consistency. Most probably this function is called every time a new constraint is instantiated.

##### Parameters

<code>c</code>	the constraint to impose in this constraint store.
----------------	--

Implemented in [SimpleConstraintStore](#).

#### 6.6.1.7 `virtual size_t ConstraintStore::num_constraints ( ) const [pure virtual]`

Returns the total number of constraints in this constraint store.

Implemented in [SimpleConstraintStore](#).

#### 6.6.1.8 `virtual size_t ConstraintStore::num_constraints_to_reevaluate ( ) const [pure virtual]`

Returns the number of constraints to re-evaluate.

##### Returns

number of constraints to re-evaluate.

Implemented in [SimpleConstraintStore](#).

#### 6.6.1.9 `virtual void ConstraintStore::sat_check ( bool sat_check = true ) [pure virtual]`

Sets the satisfiability check during constraint propagation. Thic check increases the time spent for consistency but reduces the total exectuion time.

##### Parameters

<code>sat_check</code>	boolean value representing whether or not the satisfiability check should be performed (default: true).
------------------------	---

Implemented in [SimpleConstraintStore](#).

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/constraint_store.h`

## 6.7 CPMoDel Class Reference

## Public Member Functions

- virtual int [get\\_id](#) () const
- virtual void [add\\_variable](#) (VariablePtr ptr)
- virtual void [add\\_constraint](#) (ConstraintPtr ptr)
- virtual void [add\\_search\\_engine](#) (SearchEnginePtr ptr)
- virtual SearchEnginePtr [get\\_search\\_engine](#) ()
- virtual void [add\\_constraint\\_store](#) (ConstraintStorePtr store)
- virtual void [init\\_constraint\\_store](#) ()
- virtual void [create\\_constraint\\_graph](#) ()
- virtual void [attach\\_constraint\\_store](#) ()
- virtual void [set\\_solutions\\_limit](#) (size\_t sol\_limit)
- virtual void [set\\_timeout\\_limit](#) (double timeout)
- virtual void [print](#) () const

*Print information about this CP Model.*

## Protected Attributes

- int [\\_model\\_id](#)  
*Unique id for this model.*
- std::vector< VariablePtr > [\\_variables](#)  
*Variables.*
- std::vector< ConstraintPtr > [\\_constraints](#)  
*Constraint Store.*
- SearchEnginePtr [\\_search\\_engine](#)  
*Search engine.*
- ConstraintStorePtr [\\_store](#)  
*Constraint store.*

### 6.7.1 Member Function Documentation

#### 6.7.1.1 void CPMModel::add\_constraint ( ConstraintPtr ptr ) [virtual]

Add a constraint to the model. It links constraints to variables, actually defining the constraint graph.

##### Parameters

<i>ptr</i>	pointer to the constraint to add to the model
------------	---

#### 6.7.1.2 void CPMModel::add\_constraint\_store ( ConstraintStorePtr store ) [virtual]

Add a constraint store to the model.

##### Parameters

<i>store</i>	pointer to the constraint store to attach to the variables and propagate constraints.
--------------	---

##### Note

this represents at least the first instance of constraint store. Every time this method is called, the variable's store will be updated with the given instance.

If a search engine is already present in the model, it sets the given constraint store to the search engine.



6.7.1.3 void CPMoDel::add\_search\_engine ( SearchEnginePtr *ptr* ) [virtual]

Add a search engine to the model.

## Parameters

<i>ptr</i>	pointer to the search engine to use in order to explore the search space.
------------	---

## Note

if a constraint store is already present in the model, it sets the store into the given search engine.

#### 6.7.1.4 void CPMModel::add\_variable ( VariablePtr ptr ) [virtual]

Add a variable to the model. It links variables to constraints, actually defining the constraint graph.

## Parameters

<i>ptr</i>	pointer to the variable to add to the model
------------	---

#### 6.7.1.5 void CPMModel::attach\_constraint\_store ( ) [virtual]

Sets the constraint store as current constraint store for all the variables in the model. When a variable changes its state, the constraint store is automatically notified.

#### 6.7.1.6 void CPMModel::create\_constraint\_graph ( ) [virtual]

Defines the constraint graphs actually attaching the constraints to the variables.

#### 6.7.1.7 int CPMModel::get\_id ( ) const [virtual]

Get the (unique) id of this model.

## Returns

the model's id.

#### 6.7.1.8 SearchEnginePtr CPMModel::get\_search\_engine ( ) [virtual]

Gets the search engine in order to run it.

## Returns

a reference to the search engine in this model.

#### 6.7.1.9 void CPMModel::init\_constraint\_store ( ) [virtual]

Initializes the constraint store filling it with the all the constraints into the model.

#### 6.7.1.10 void CPMModel::set\_solutions\_limit ( size\_t sol\_limit ) [virtual]

Imposes a limit on the number of solutions.

## Parameters

<i>sol_limit</i>	the maximum number of solutions for this model.
------------------	---

## Note

-1 means find all solutions.

6.7.1.11 void CPMModel::set\_timeout\_limit ( double *timeout* ) [virtual]

Imposes a timeout limit.

## Parameters

<i>timeout</i>	timeout limit.
----------------	----------------

## Note

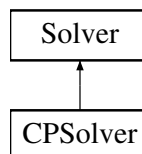
-1 means no timeout.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_model.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_model.cpp

## 6.8 CPSolver Class Reference

Inheritance diagram for CPSolver:



### Public Member Functions

- [CPSolver](#) ()  
*Constructor.*
- [CPSolver](#) (CPModel \*model)
- void [add\\_model](#) (CPModel \*model) override
- void [remove\\_model](#) (int model\_idx) override
- CPModel \* [get\\_model](#) (int model\_idx) const override
- virtual void [customize](#) (const [InputData](#) &i\_data, int model\_idx=0) override
- void [run](#) ()
- void [run](#) (int model\_idx) override
- int [num\\_models](#) () const override
- int [num\\_solved\\_models](#) () const override
- int [sat\\_models](#) () const override
- int [unsat\\_models](#) () const override
- void [print](#) () const override  
*Print information about this solver.*

## Protected Member Functions

- void [run\\_model](#) ([CPModel](#) \*model)

## Protected Attributes

- std::string [\\_dbg](#)  
*Debug info.*
- std::vector< [CPModel](#) \* > [\\_models](#)
- int [\\_solved\\_models](#)  
*Number of solved models.*
- int [\\_sat\\_models](#)  
*Number of models which have a solution.*
- int [\\_unsat\\_models](#)  
*Number of unsatisfiable models.*

## 6.8.1 Constructor & Destructor Documentation

### 6.8.1.1 [CPSolver::CPSolver](#) ( [CPModel](#) \* *model* )

Constructor.

Parameters

<i>model</i>	a model to add to this <a href="#">CPSolver</a> .
--------------	---

## 6.8.2 Member Function Documentation

### 6.8.2.1 void [CPSolver::add\\_model](#) ( [CPModel](#) \* *model* ) [override],[virtual]

Add a model to the solver.

Parameters

<i>model</i>	the reference to the (CP) model to add to the solver.
--------------	---

Note

a solver can hold several models and decide both the model to run and the order in which run each model.

Implements [Solver](#).

### 6.8.2.2 void [CPSolver::customize](#) ( const [InputData](#) & *i\_data*, int *model\_idx* = 0 ) [override],[virtual]

Further customizes a given model (identified by its index) with user options.

Parameters

<i>i_data</i>	a reference to a <a href="#">input_data</a> class where options are retrieved.
<i>model_idx</i>	the index of the model to customize (default: 0, i.e., first model).

Implements [Solver](#).

### 6.8.2.3 [CPModel](#) \* [CPSolver::get\\_model](#) ( int *model\_idx* ) const [override],[virtual]

Returns a reference to model.

## Parameters

<i>model_idx</i>	the index of the model to return.
------------------	-----------------------------------

Implements [Solver](#).

**6.8.2.4** `int CPSolver::num_models ( ) const` `[override],[virtual]`

Returns the number of models that are managed by this solver.

## Returns

the number of models managed by this solver.

Implements [Solver](#).

**6.8.2.5** `int CPSolver::num_solved_models ( ) const` `[override],[virtual]`

Returns the current number of runned models.

## Returns

the number of models for which the run function has been called.

Implements [Solver](#).

**6.8.2.6** `void CPSolver::remove_model ( int model_idx )` `[override],[virtual]`

Removes a model actually destroying it.

## Parameters

<i>model_idx</i>	the index of the model to destroy.
------------------	------------------------------------

Implements [Solver](#).

**6.8.2.7** `void CPSolver::run ( )` `[virtual]`

It runs the solver in order to find a solution, the best solutions or other solutions w.r.t. the model given to the solver.

Implements [Solver](#).

**6.8.2.8** `void CPSolver::run ( int model_idx )` `[override],[virtual]`

It runs the solver in order to find a solution, the best solutions or other solutions for the model specified by its index.

## Parameters

<i>model_idx</i>	the index of the model to solve.
------------------	----------------------------------

Implements [Solver](#).

**6.8.2.9** `void CPSolver::run_model ( CPMModel * model )` `[protected]`

It actually runs a CP Model.

## Parameters

<i>a</i>	reference to a CP Model.
----------	--------------------------

#### 6.8.2.10 `int CPSolver::sat_models ( ) const` [override],[virtual]

Returns the number of models for which a solution has been found (out of the number of solved models).

## Returns

the number of models for which a solution has been found.

Implements [Solver](#).

#### 6.8.2.11 `int CPSolver::unsat_models ( ) const` [override],[virtual]

Returns the number of unsatisfiable models, i.e., the number of models with no solutions among those that have been solved so far.

## Returns

the number of unsatisfiable models.

Implements [Solver](#).

### 6.8.3 Member Data Documentation

#### 6.8.3.1 `std::vector< CPMModel * > CPSolver::_models` [protected]

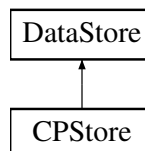
CP models to be considered by this [CPSolver](#). The solver may decide which model to solve and in which order solve it.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_solver.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_solver.cpp

## 6.9 CPStore Class Reference

Inheritance diagram for CPStore:



### Public Member Functions

- virtual bool [load\\_model](#) (std::string= "")  
Load model from input file (FlatZinc model)
- virtual void [init\\_model](#) ()
- virtual void [print\\_model\\_info](#) ()

*Print info about the model.*

- virtual void **print\_model\_variable\_info** ()
- virtual void **print\_model\_domain\_info** ()
- virtual void **print\_model\_constraint\_info** ()

### Static Public Member Functions

- static [CPStore](#) \* **get\_store** (std::string in\_file)  
*Constructor get (static) instance.*

### Protected Member Functions

- [CPStore](#) (std::string)  
*Protected constructor for singleton pattern.*

### Additional Inherited Members

#### 6.9.1 Member Function Documentation

##### 6.9.1.1 void CPStore::init\_model ( ) [virtual]

Init store with the loaded model. This method works on the internal state of the store. It uses a generator to generate the right instances of the objects (e.g. CUDA-FD variabes) and add them to the model. A generator takes tokens as input and returns the corresponding pointer to the instantiated objects.

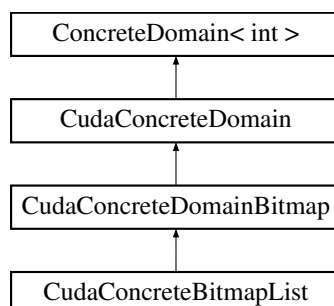
Implements [DataStore](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_store.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cp\_store.cpp

## 6.10 CudaConcreteBitmapList Class Reference

Inheritance diagram for CudaConcreteBitmapList:



### Public Member Functions

- [CudaConcreteBitmapList](#) (size\_t size, std::vector< std::pair< int, int > > pairs)
- void **set\_domain** (void \*const domain, int rep, int min, int max, int dsz) override
- unsigned int **size** () const

*It returns the current size of the domain.*

- void [shrink](#) (int min, int max)
- void [subtract](#) (int value)
- void [in\\_min](#) (int min)
- void [in\\_max](#) (int max)
- void [add](#) (int value)
- void [add](#) (int min, int max)
- bool [contains](#) (int val) const
- int [get\\_id\\_representation](#) () const override
- void [print](#) () const

## Protected Member Functions

- int [find\\_pair](#) (int val) const
- int [find\\_prev\\_pair](#) (int val) const
- int [find\\_next\\_pair](#) (int val) const

## Protected Attributes

- int [\\_num\\_bitmaps](#)  
*Number of pairs in the list (list size).*
- int [\\_bitmap\\_size](#)  
*Fixed size of each bitmap in the list.*
- unsigned int [\\_domain\\_size](#)

## Additional Inherited Members

### 6.10.1 Constructor & Destructor Documentation

#### 6.10.1.1 `CudaConcreteBitmapList::CudaConcreteBitmapList ( size_t size, std::vector< std::pair< int, int > > pairs )`

Constructor. It allocates size bytes for the internal domain's representation and it initializes it with the pairs of bounds contained in pairs.

Parameters

<i>size</i>	the number of bytes to allocate.
<i>pairs</i>	the SORTED list of pairs to allocate.

### 6.10.2 Member Function Documentation

#### 6.10.2.1 `void CudaConcreteBitmapList::add ( int value ) [virtual]`

It computes union of this domain and {value}.

Parameters

<i>value</i>	it specifies the value which is being added.
--------------	--

Reimplemented from [CudaConcreteDomainBitmap](#).

#### 6.10.2.2 `void CudaConcreteBitmapList::add ( int min, int max ) [virtual]`

It computes union of this domain and {min, max}.



## Parameters

<i>min</i>	lower bound of the new domain which is being added.
<i>max</i>	upper bound of the new domain which is being added.

## Note

it is possible to add only bitmaps with empty intersection with previous bitmaps and which min is greater than current lower bound.

**Todo** complete add function to add any bitmap.

Reimplemented from [CudaConcreteDomainBitmap](#).

### 6.10.2.3 bool CudaConcreteBitmapList::contains ( int *val* ) const [virtual]

It checks whether the value belongs to the domain or not.

## Parameters

<i>val</i>	to check whether it is in the current domain.
------------	---

## Note

*val* is given w.r.t. the lower bound of 0.

Reimplemented from [CudaConcreteDomainBitmap](#).

### 6.10.2.4 int CudaConcreteBitmapList::find\_next\_pair ( int *val* ) const [protected]

Find the index of the first pair with values greater than *val*.

## Parameters

<i>val</i>	to be compared in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair with *val* greater than *val*, -1 if no such pair exists.

## Note

it returns the index of the pair regardless of whether the element is present or not.

### 6.10.2.5 int CudaConcreteBitmapList::find\_pair ( int *val* ) const [protected]

Find the index of the pair containing *val*.

## Parameters

<i>val</i>	to be searched in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair containing *val*, -1 otherwise.

## Note

it returns the index of the pair regardless of whether the element is present or not.

6.10.2.6 `int CudaConcreteBitmapList::find_prev_pair ( int val ) const` `[protected]`

Find the index of the last pair with values smaller than `val`.

## Parameters

<i>val</i>	to be compared in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair with *val* lower than *val*, -1 if no such pair exists.

## Note

it returns the index of the pair regardless of whether the element is present or not.

#### 6.10.2.7 `int CudaConcreteBitmapList::get_id_representation ( ) const` [override], [virtual]

Returns the current CUDA concrete domain's representation.

## Returns

an integer id indicating the current representation of this domain.

Reimplemented from [CudaConcreteDomainBitmap](#).

#### 6.10.2.8 `void CudaConcreteBitmapList::in_max ( int max )` [virtual]

It updates the domain according to *max* value.

## Parameters

<i>max</i>	domain value.
------------	---------------

Reimplemented from [CudaConcreteDomainBitmap](#).

#### 6.10.2.9 `void CudaConcreteBitmapList::in_min ( int min )` [virtual]

It updates the domain according to *min* value.

## Parameters

<i>min</i>	domain value.
------------	---------------

Reimplemented from [CudaConcreteDomainBitmap](#).

#### 6.10.2.10 `void CudaConcreteBitmapList::print ( ) const` [virtual]

It prints the current domain representation (its state).

## Note

it prints the content of the object given by "get\_representation ()".

Reimplemented from [CudaConcreteDomainBitmap](#).

#### 6.10.2.11 `void CudaConcreteBitmapList::set_domain ( void *const domain, int rep, int min, int max, int dsz )` [override], [virtual]

Sets the internal representation of the domain from a given concrete domain and given lower/upper bounds.

**Parameters**

<i>domain</i>	a reference to a given concrete domain.
<i>rep</i>	current internal's domain representation.
<i>min</i>	lower bound to set.
<i>max</i>	upper bound to set.
<i>dsz</i>	domain size to set.

**Note**

the client must pass a valid concrete domain's representation.

Reimplemented from [CudaConcreteDomainBitmap](#).

**6.10.2.12** void CudaConcreteBitmapList::shrink ( int *min*, int *max* ) [virtual]

It updates the domain to have values only within min/max.

**Parameters**

<i>min</i>	new lower bound to set for the current domain.
<i>max</i>	new upper bound to set for the current domain.

Reimplemented from [CudaConcreteDomainBitmap](#).

**6.10.2.13** void CudaConcreteBitmapList::subtract ( int *value* ) [virtual]

It subtracts {value} from the current domain.

**Parameters**

<i>value</i>	the value to subtract from the current domain.
--------------	--

Reimplemented from [CudaConcreteDomainBitmap](#).

**6.10.3 Member Data Documentation**

**6.10.3.1** unsigned int CudaConcreteBitmapList::\_domain\_size [protected]

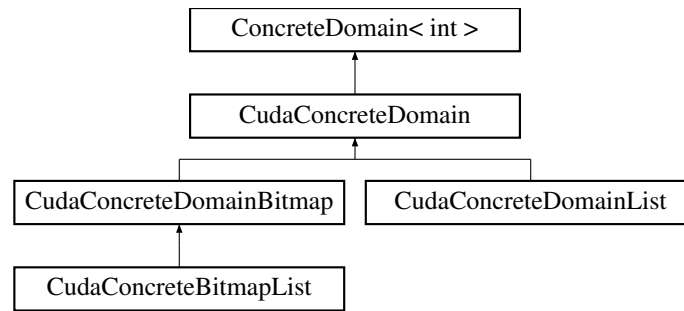
Current domain size, i.e., sum of the elements on each bitmap.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_bitmaplist.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_bitmaplist.cpp

**6.11 CudaConcreteDomain Class Reference**

Inheritance diagram for CudaConcreteDomain:



## Public Member Functions

- int [lower\\_bound](#) () const  
*Returns lower bound.*
- int [upper\\_bound](#) () const  
*Returns upper bound.*
- int [get\\_num\\_chunks](#) () const
- size\_t [allocated\\_bytes](#) () const
- bool [is\\_empty](#) () const
- void [set\\_domain](#) (void \*const domain, int rep, int min, int max, int dsz) override
- const void \* [get\\_representation](#) () const override
- virtual int [get\\_id\\_representation](#) () const =0

## Protected Member Functions

- void [flush\\_domain](#) ()
- void [set\\_empty](#) ()
- [CudaConcreteDomain](#) (size\_t size)

## Protected Attributes

- std::string [\\_dbg](#)
- int [\\_num\\_chunks](#)  
*Number of allocated (32 bit int) chunks.*
- int [\\_lower\\_bound](#)  
*Lower bound.*
- int [\\_upper\\_bound](#)  
*Upper bound.*
- int \* [\\_concrete\\_domain](#)

### 6.11.1 Constructor & Destructor Documentation

#### 6.11.1.1 CudaConcreteDomain::CudaConcreteDomain ( size\_t size ) [protected]

Constructor for [CudaConcreteDomain](#). It instantiates a new object and allocate size bytes for the array of integers

#### Parameters

---

<i>size</i>	the number of bytes to allocate.
-------------	----------------------------------

**Note**

the client should check whether integers are represented by 32 bit values.

**6.11.2 Member Function Documentation****6.11.2.1 `size_t CudaConcreteDomain::allocated_bytes ( ) const`**

Get the number of allocated bytes, i.e., the size of the internal domain's representation.

**6.11.2.2 `void CudaConcreteDomain::flush_domain ( ) [protected]`**

Flush domain: reduces its domain size to zero by flushing all values in the internal domain's representation. It sets the current domain's state as empty.

**Note**

it sets upper bound < lower bound.

**6.11.2.3 `virtual int CudaConcreteDomain::get_id_representation ( ) const [pure virtual]`**

Returns the current CUDA concrete domain's representation.

**Returns**

an integer id indicating the current representation of this domain.

Implemented in [CudaConcreteDomainBitmap](#), [CudaConcreteBitmapList](#), and [CudaConcreteDomainList](#).

**6.11.2.4 `int CudaConcreteDomain::get_num_chunks ( ) const`**

Get the number of allocated chunks (in terms of 32 bit integers).

**6.11.2.5 `const void * CudaConcreteDomain::get_representation ( ) const [override],[virtual]`**

It returns a void pointer to an object representing the current representation of the domain (e.g., bitmap).

**Returns**

void pointer to the concrete domain representation.

Implements [ConcreteDomain< int >](#).

**6.11.2.6 `bool CudaConcreteDomain::is_empty ( ) const [virtual]`**

It checks whether the current domain is empty.

**Returns**

true if the current domain is empty, false otherwise.

Implements [ConcreteDomain< int >](#).

6.11.2.7 void CudaConcreteDomain::set\_domain ( void \*const *domain*, int *rep*, int *min*, int *max*, int *dsz* ) [override],  
[virtual]

Sets the internal representation of the domain from a given concrete domain and given lower/upper bounds.

## Parameters

<i>domain</i>	a reference to a given concrete domain.
<i>rep</i>	current internal's domain representation.
<i>min</i>	lower bound to set.
<i>max</i>	upper bound to set.
<i>dsz</i>	domain size to set.

## Note

the client must pass a valid concrete domain's representation.

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteDomainList](#).

#### 6.11.2.8 void CudaConcreteDomain::set\_empty ( ) [protected]

Empty domain: reduces its domain size to zero by setting the current domain's state as empty.

## Note

it does not flush the current internal domain's representation.

### 6.11.3 Member Data Documentation

#### 6.11.3.1 int\* CudaConcreteDomain::\_concrete\_domain [protected]

Concrete domain is represented by an array of (32 bit) integers.

## Note

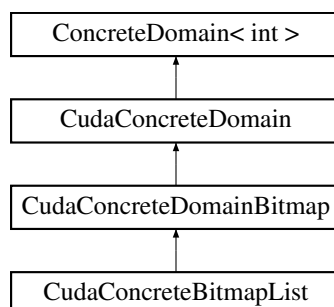
actual internal representation of domain.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_domain.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_domain.cpp

## 6.12 CudaConcreteDomainBitmap Class Reference

Inheritance diagram for CudaConcreteDomainBitmap:





## Public Member Functions

- [CudaConcreteDomainBitmap](#) (size\_t [size](#))
- [CudaConcreteDomainBitmap](#) (size\_t [size](#), int min, int max)
- void [set\\_domain](#) (void \*const domain, int rep, int min, int max, int dsz) override
- unsigned int [size](#) () const  
*It returns the current size of the domain.*
- void [shrink](#) (int min, int max)
- void [subtract](#) (int value)
- void [in\\_min](#) (int min)
- void [in\\_max](#) (int max)
- void [add](#) (int value)
- void [add](#) (int min, int max)
- bool [contains](#) (int value) const
- bool [is\\_singleton](#) () const
- int [get\\_singleton](#) () const
- int [get\\_id\\_representation](#) () const override
- void [print](#) () const

## Static Protected Member Functions

- static constexpr int [IDX\\_CHUNK](#) (int val)
- static constexpr int [IDX\\_BIT](#) (int val)
- static constexpr int [NUM\\_CHUNKS](#) (int [size](#))

## Protected Attributes

- unsigned int [\\_num\\_valid\\_bits](#)  
*Number of bits set to 1.*

## Static Protected Attributes

- static constexpr int [BITS\\_IN\\_BYTE](#) = INT8\_C( 8 )
- static constexpr int [BITS\\_IN\\_CHUNK](#) = sizeof( int ) \* [BITS\\_IN\\_BYTE](#)

## Additional Inherited Members

### 6.12.1 Constructor & Destructor Documentation

#### 6.12.1.1 CudaConcreteDomainBitmap::CudaConcreteDomainBitmap ( size\_t *size* )

Constructor for [CudaConcreteDomainBitmap](#).

Parameters

<i>size</i>	the size in bytes to allocate for the bitmap.
-------------	---

#### Note

the bitmap is represented considering lower bound = 0 and upper bound given by the parameter size.  
initially all bits are set to 1 (i.e. valid bits).

#### 6.12.1.2 CudaConcreteDomainBitmap::CudaConcreteDomainBitmap ( size\_t *size*, int *min*, int *max* )

Constructor for [CudaConcreteDomainBitmap](#).

## Parameters

<i>size</i>	the size in bytes to allocate for the bitmap.
<i>min</i>	lower bound for {min, max} set initialization. min must be greater than or equal to 0 and less than or equal to the max number of bits storable using size bytes.
<i>max</i>	upper bound for {min, max} set initialization. max must be less than or equal to max number of bits storable using size bytes and greater than or equal to 0.

## Note

the bitmap is represented considering lower bound = 0 and upper bound given by the parameter size.  
initially all bits in {min, max} are set to 1 (i.e. valid bits).

## 6.12.2 Member Function Documentation

6.12.2.1 void CudaConcreteDomainBitmap::add ( int *value* ) [virtual]

It computes union of this domain and {value}.

## Parameters

<i>value</i>	it specifies the value which is being added.
--------------	--

## Note

value is given w.r.t. a lower bound of 0.

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

6.12.2.2 void CudaConcreteDomainBitmap::add ( int *min*, int *max* ) [virtual]

It computes union of this domain and {min, max}.

## Parameters

<i>min</i>	lower bound of the new domain which is being added.
<i>max</i>	upper bound of the new domain which is being added.

**Todo** implement using checks on chunks of bits (i.e. sublinear cost).

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

6.12.2.3 bool CudaConcreteDomainBitmap::contains ( int *value* ) const [virtual]

It checks whether the value belongs to the domain or not.

## Parameters

<i>value</i>	to check whether it is in the current domain.
--------------	---

## Note

value is given w.r.t. the lower bound of 0.

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

6.12.2.4 `int CudaConcreteDomainBitmap::get_id_representation ( ) const` `[override], [virtual]`

Returns the current CUDA concrete domain's representation.

#### Returns

an integer id indicating the current representation of this domain.

Implements [CudaConcreteDomain](#).

Reimplemented in [CudaConcreteBitmapList](#).

6.12.2.5 `int CudaConcreteDomainBitmap::get_singleton ( ) const` `[virtual]`

It returns the value of the domain element if it is a singleton.

#### Returns

the value of the singleton element.

#### Note

it throws an exception if domain is not singleton.

Implements [ConcreteDomain< int >](#).

6.12.2.6 `static constexpr int CudaConcreteDomainBitmap::IDX_BIT ( int val )` `[inline], [static], [protected]`

Get index of the bit that represents the value val module the size of a chunk, i.e., the position of the corresponding bit within a chunk.

#### Parameters

<i>val</i>	the value w.r.t. the function calculates its position within a chunk of bits
------------	--

#### Returns

position (starting from 0) of the bit corresponding to val.

6.12.2.7 `static constexpr int CudaConcreteDomainBitmap::IDX_CHUNK ( int val )` `[inline], [static], [protected]`

Get index of the chunk of bits containing the bit representing the value given in input.

#### Parameters

<i>max</i>	lower bound used to calculated the index of the bitmap
------------	--

#### Returns

number of int used as bitmaps to represent max

6.12.2.8 `void CudaConcreteDomainBitmap::in_max ( int max )` `[virtual]`

It updates the domain according to max value.

## Parameters

<i>max</i>	domain value.
------------	---------------

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

**6.12.2.9** `void CudaConcreteDomainBitmap::in_min ( int min )` `[virtual]`

It updates the domain according to min value.

## Parameters

<i>min</i>	domain value.
------------	---------------

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

**6.12.2.10** `bool CudaConcreteDomainBitmap::is_singleton ( ) const` `[virtual]`

It checks whether the current domain contains only an element (i.e., it is a singleton).

## Returns

true if the current domain is singleton, false otherwise.

Implements [ConcreteDomain< int >](#).

**6.12.2.11** `static constexpr int CudaConcreteDomainBitmap::NUM_CHUNKS ( int size )` `[inline], [static], [protected]`

Get the number of chunks needed to represent a domain of size values.

## Parameters

<i>size</i>	the size in terms of number of elements of the domain to represent as bitmap.
-------------	---

## Returns

number of chunks needed to represent size value.

**6.12.2.12** `void CudaConcreteDomainBitmap::print ( ) const` `[virtual]`

It prints the current domain representation (its state).

## Note

it prints the content of the object given by "get\_representation ()".

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

**6.12.2.13** `void CudaConcreteDomainBitmap::set_domain ( void *const domain, int rep, int min, int max, int dsz )` `[override], [virtual]`

Sets the internal representation of the domain from a given concrete domain and given lower/upper bounds.

## Parameters

<i>domain</i>	a reference to a given concrete domain.
<i>rep</i>	current internal's domain representation.
<i>min</i>	lower bound to set.
<i>max</i>	upper bound to set.
<i>dsz</i>	domain size to set.

## Note

the client must pass a valid concrete domain's representation.

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

**6.12.2.14** `void CudaConcreteDomainBitmap::shrink ( int min, int max )` `[virtual]`

It updates the domain to have values only within min/max.

## Parameters

<i>min</i>	new lower bound to set for the current domain.
<i>max</i>	new upper bound to set for the current domain.

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

**6.12.2.15** `void CudaConcreteDomainBitmap::subtract ( int value )` `[virtual]`

It subtracts {value} from the current domain.

## Parameters

<i>value</i>	the value to subtract from the current domain.
--------------	--

Implements [ConcreteDomain< int >](#).

Reimplemented in [CudaConcreteBitmapList](#).

## 6.12.3 Member Data Documentation

**6.12.3.1** `constexpr int CudaConcreteDomainBitmap::BITS_IN_BYTE = INT8_C( 8 )` `[static], [protected]`

Macro for the size of a byte in terms of bits.

**6.12.3.2** `constexpr int CudaConcreteDomainBitmap::BITS_IN_CHUNK = sizeof( int ) * BITS_IN_BYTE` `[static], [protected]`

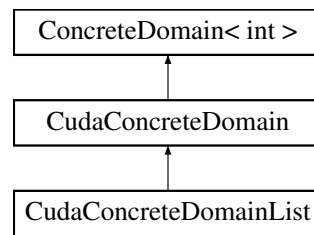
Macro for the size of a chunk in terms of bits.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_bitmap.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_concrete\_bitmap.cpp

## 6.13 CudaConcreteDomainList Class Reference

Inheritance diagram for CudaConcreteDomainList:



### Public Member Functions

- [CudaConcreteDomainList](#) (size\_t [size](#), int min, int max)
- void [set\\_domain](#) (void \*const domain, int rep, int min, int max, int dsz) override
- unsigned int [size](#) () const  
*It returns the current size of the domain.*
- void [shrink](#) (int min, int max)
- void [subtract](#) (int value)
- void [in\\_min](#) (int min)
- void [in\\_max](#) (int max)
- void [add](#) (int value)
- void [add](#) (int min, int max)
- bool [contains](#) (int val) const
- bool [is\\_singleton](#) () const
- int [get\\_singleton](#) () const
- int [get\\_id\\_representation](#) () const override
- void [print](#) () const

### Protected Member Functions

- int [find\\_pair](#) (int val) const
- int [find\\_prev\\_pair](#) (int val) const
- int [find\\_next\\_pair](#) (int val) const

### Protected Attributes

- int [\\_num\\_pairs](#)  
*Number of pairs in the list (list size)*
- int [\\_max\\_allowed\\_pairs](#)  
*Max number of storable pairs in the concrete domain.*
- unsigned int [\\_domain\\_size](#)

### 6.13.1 Constructor & Destructor Documentation

#### 6.13.1.1 CudaConcreteDomainList::CudaConcreteDomainList ( size\_t [size](#), int [min](#), int [max](#) )

Constructor for [CudaConcreteDomainList](#).

## Parameters

<i>size</i>	the size in bytes to allocate for the bitmap.
<i>min</i>	lower bound in {min, max}
<i>max</i>	upper bound in {min, max}

## 6.13.2 Member Function Documentation

6.13.2.1 void CudaConcreteDomainList::add ( int *value* ) [virtual]

It computes union of this domain and {value}.

## Parameters

<i>value</i>	it specifies the value which is being added.
--------------	--

Implements [ConcreteDomain< int >](#).

6.13.2.2 void CudaConcreteDomainList::add ( int *min*, int *max* ) [virtual]

It computes union of this domain and {min, max}.

## Parameters

<i>min</i>	lower bound of the new domain which is being added.
<i>max</i>	upper bound of the new domain which is being added.

Implements [ConcreteDomain< int >](#).

6.13.2.3 bool CudaConcreteDomainList::contains ( int *val* ) const [virtual]

It checks whether the value belongs to the domain or not.

## Parameters

<i>val</i>	to check whether it is in the current domain.
------------	---

## Note

*val* is given w.r.t. the lower bound of 0.

Implements [ConcreteDomain< int >](#).

6.13.2.4 int CudaConcreteDomainList::find\_next\_pair ( int *val* ) const [protected]

Find the index of the first pair with values greater than *val*.

## Parameters

<i>val</i>	to be compared in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair with *val* greater than *val*, -1 if no such pair exists.

6.13.2.5 int CudaConcreteDomainList::find\_pair ( int *val* ) const [protected]

Find the index of the pair containing *val*.

## Parameters

<i>val</i>	to be searched in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair containing *val*, -1 otherwise.

6.13.2.6 `int CudaConcreteDomainList::find_prev_pair ( int val ) const` `[protected]`

Find the index of the last pair with values smaller than *val*.

## Parameters

<i>val</i>	to be compared in the list of pairs.
------------	--------------------------------------

## Returns

the index of the pair with *val* lower than *val*, -1 if no such pair exists.

6.13.2.7 `int CudaConcreteDomainList::get_id_representation ( ) const` `[override],[virtual]`

Returns the current CUDA concrete domain's representation.

## Returns

an integer id indicating the current representation of this domain.

Implements [CudaConcreteDomain](#).

6.13.2.8 `int CudaConcreteDomainList::get_singleton ( ) const` `[virtual]`

It returns the value of type T of the domain if it is a singleton.

## Returns

the value of the singleton element.

## Note

it throws an exception if domain is not singleton.

Implements [ConcreteDomain< int >](#).

6.13.2.9 `void CudaConcreteDomainList::in_max ( int max )` `[virtual]`

It updates the domain according to *max* value.

## Parameters

<i>max</i>	domain value.
------------	---------------

Implements [ConcreteDomain< int >](#).

6.13.2.10 `void CudaConcreteDomainList::in_min ( int min )` `[virtual]`

It updates the domain according to *min* value.



## Parameters

<i>min</i>	domain value.
------------	---------------

Implements [ConcreteDomain< int >](#).

#### 6.13.2.11 bool CudaConcreteDomainList::is\_singleton ( ) const [virtual]

It checks whether the current domain contains only an element (i.e., it is a singleton).

## Returns

true if the current domain is singleton, false otherwise.

Implements [ConcreteDomain< int >](#).

#### 6.13.2.12 void CudaConcreteDomainList::print ( ) const [virtual]

It prints the current domain representation (its state).

## Note

it prints the content of the object given by "get\_representation ()" .

Implements [ConcreteDomain< int >](#).

#### 6.13.2.13 void CudaConcreteDomainList::set\_domain ( void \*const domain, int rep, int min, int max, int dsz ) [override], [virtual]

Sets the internal representation of the domain from a given concrete domain and given lower/upper bounds.

## Parameters

<i>domain</i>	a reference to a given concrete domain.
<i>rep</i>	current internal's domain representation.
<i>min</i>	lower bound to set.
<i>max</i>	upper bound to set.
<i>dsz</i>	domain size to set.

## Note

the client must pass a valid concrete domain's representation.

Reimplemented from [CudaConcreteDomain](#).

#### 6.13.2.14 void CudaConcreteDomainList::shrink ( int min, int max ) [virtual]

It updates the domain to have values only within min/max.

## Parameters

<i>min</i>	new lower bound to set for the current domain.
<i>max</i>	new upper bound to set for the current domain.

Implements [ConcreteDomain< int >](#).

#### 6.13.2.15 void CudaConcreteDomainList::subtract ( int value ) [virtual]

It subtracts {value} from the current domain.

## Parameters

<i>value</i>	the value to subtract from the current domain.
--------------	--

## Note

a value is removed only if it corresponds to a lower/upper bound.

Implements [ConcreteDomain< int >](#).

### 6.13.3 Member Data Documentation

#### 6.13.3.1 unsigned int CudaConcreteDomainList::\_domain\_size [protected]

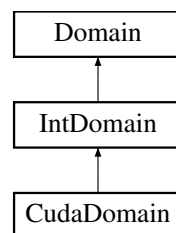
Current domain size, i.e., sum of the elements on each pair of bounds in the list.

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

- /Users/fedecampe/Desktop/NVIDIAOSO-PRJ/NVIDIAOSO/NVIDIAOSO/cuda\_concrete\_list.h
- /Users/fedecampe/Desktop/NVIDIAOSO-PRJ/NVIDIAOSO/NVIDIAOSO/cuda\_concrete\_list.cpp

## 6.14 CudaDomain Class Reference

Inheritance diagram for CudaDomain:



### Public Member Functions

- DomainPtr [clone](#) () const
- void [init\\_domain](#) (int min, int max)
- size\_t [allocated\\_bytes](#) () const
- EventType [get\\_event](#) () const  
*Get event on the current domain.*
- void [reset\\_event](#) ()
- void [set\\_concrete\\_domain](#) (int \*const concrete\_domain)
- int \*const [get\\_concrete\\_domain](#) () const
- size\_t [get\\_size](#) () const
- int [lower\\_bound](#) () const  
*Get the domain's lower bound.*
- int [upper\\_bound](#) () const  
*Get the domain's upper bound.*
- bool [contains](#) (int value) const
- void [set\\_bounds](#) (int min, int max)
- void [shrink](#) (int min, int max)
- bool [set\\_singleton](#) (int val)

- bool [subtract](#) (int n)  
*Subtract the element from the domain (see [int\\_domain.h](#))*
- void [add\\_element](#) (int n)
- void [in\\_min](#) (int min)  
*Increase the lower\_bound to min (see [int\\_domain.h](#))*
- void [in\\_max](#) (int max)  
*Decrease the upper\_bound to max (see [int\\_domain.h](#))*
- void [print](#) () const  
*Print info about domain.*
- void [print\\_domain](#) () const  
*Print internal domain representation.*

### Protected Member Functions

- DomainPtr [clone\\_impl](#) () const  
*Clone method to clone the current object.*
- EventType [int\\_to\\_event](#) () const  
*Convert the current event int to a domain event.*
- void [event\\_to\\_int](#) (EventType evt) const  
*Convert a domain event to the current integer.*
- void [set\\_bit\\_representation](#) ()
- void [set\\_bitlist\\_representation](#) (int num\_list=INT\_BITLIST)
- void [set\\_list\\_representation](#) (int num\_list=INT\_LIST)
- CudaDomainRepresentation [get\\_representation](#) () const  
*Get domain representation (i.e., bitmap, bitmaplist, or list)*
- void [switch\\_list\\_to\\_bitmaplist](#) ()

### Static Protected Member Functions

- static constexpr int [EVT\\_IDX](#) ()
- static constexpr int [REP\\_IDX](#) ()
- static constexpr int [LB\\_IDX](#) ()
- static constexpr int [UB\\_IDX](#) ()
- static constexpr int [DSZ\\_IDX](#) ()
- static constexpr int [BIT\\_IDX](#) ()
- static constexpr int [IDX\\_CHUNK](#) (int val)
- static constexpr int [IDX\\_BIT](#) (int val)
- static int [num\\_chunks](#) (int n)

### Protected Attributes

- CudaConcreteDomainPtr [\\_concrete\\_domain](#)
- int \* [\\_domain](#)
- size\_t [\\_num\\_allocated\\_bytes](#)
- size\_t [\\_num\\_int\\_chunks](#)

## Static Protected Attributes

- static constexpr int **INT\_BITMAP** = 0
- static constexpr int **INT\_BITLIST** = -1
- static constexpr int **INT\_LIST** = 1
- static constexpr int **BITS\_IN\_BYTE** = INT8\_C( 8 )
- static constexpr int **SHARED\_MEM\_KB** = 47
- static constexpr size\_t **MAX\_BYTES\_SIZE** = **SHARED\_MEM\_KB** \* 1024
- static constexpr size\_t **MAX\_STATUS\_SIZE** = 5 \* sizeof( int )
- static constexpr size\_t **MAX\_DOMAIN\_VALUES** = ((**MAX\_BYTES\_SIZE** - **MAX\_STATUS\_SIZE**) / sizeof( int ))

## Additional Inherited Members

### 6.14.1 Member Function Documentation

#### 6.14.1.1 void CudaDomain::add\_element ( int *n* ) [virtual]

Add an element val to the current domain (see [int\\_domain.h](#)).

##### Note

if the element is out of the current bounds, no element will be added, i.e., the domain maintains the current size.

Implements [IntDomain](#).

#### 6.14.1.2 size\_t CudaDomain::allocated\_bytes ( ) const

Get the number of allocated bytes needed for representing the current domain w.r.t. its lower and upper bounds.

##### Returns

the number of allocated bytes.

#### 6.14.1.3 DomainPtr CudaDomain::clone ( ) const [virtual]

Clone the current domain and returns a pointer to it.

##### Returns

a pointer to a domain that has been initialized as a copy (clone) of this domain.

Implements [Domain](#).

#### 6.14.1.4 bool CudaDomain::contains ( int *value* ) const [virtual]

It checks whether the value belongs to the domain or not.

##### Parameters

---

<i>value</i>	to check whether it is in the current domain.
--------------	---

**Returns**

true if value is in this domain, false otherwise

Implements [IntDomain](#).

**6.14.1.5** `static constexpr int CudaDomain::EVT_IDX ( ) [inline],[static],[protected]`

Constants used to retrieve the current domain description. [Domain](#) represented as: | EVT | REP | LB | UB | DSZ || ... BIT ... |. See [system\\_description.h](#).

**6.14.1.6** `int *const CudaDomain::get_concrete_domain ( ) const`

Gets a reference to the current internal representation.

**Returns**

a reference to a (cuda) concrete domain.

**6.14.1.7** `size_t CudaDomain::get_size ( ) const [virtual]`

Get domain size. It returns the current size of the domain, checking whether there are "holes" according to the current representation of the domain (i.e., bitmap or list):

**Returns**

the current domain's size.

Implements [Domain](#).

**6.14.1.8** `static constexpr int CudaDomain::IDX_BIT ( int val ) [inline],[static],[protected]`

Get index of the last int used as bitmap to represent [min, max].

**Parameters**

<i>max</i>	lower bound used to calculate the index of the bitmap
------------	---

**Returns**

number of int used as bitmaps to represent max

**6.14.1.9** `static constexpr int CudaDomain::IDX_CHUNK ( int val ) [inline],[static],[protected]`

Get index of the chunk of bits containing the bit representing the value given in input.

**Parameters**

<i>max</i>	lower bound used to calculated the index of the bitmap
------------	--

**Returns**

number of int used as bitmaps to represent max

#### 6.14.1.10 void CudaDomain::init\_domain ( int *min*, int *max* ) [virtual]

Initializes domain with default values:

- Event: no event;
- Representation: list or bitmap according to [min, max];
- Lower bound: min;
- Upper bound: max;
- Size:  $|\text{max} - \text{min} + 1|$  or MAX\_INT if max = MAN\_INT()/2 and min = MIN\_INT() / 2, etc..

**Note**

It instantiate an array of ints of at most MAX\_BYTES\_SIZE.

**Parameters**

<i>min</i>	lower bound of the domain
<i>max</i>	upper bound of the domain

**Returns**

it fails whenever consistency check on min/max fails (i.e., max < min).

Implements [IntDomain](#).

#### 6.14.1.11 static int CudaDomain::num\_chunks ( int *n* ) [inline], [static], [protected]

Return the number of 32-bit integers needed to represent a set of n domain's values.

**Parameters**

<i>n</i>	number of values to represent as bits
----------	---------------------------------------

**Returns**

number of 32-bit integer chunks needed to represent n values.

#### 6.14.1.12 void CudaDomain::reset\_event ( ) [virtual]

Sets the no event on this domain.

**Note**

No event won't trigger any propagation on this domain.

Implements [Domain](#).

6.14.1.13 `void CudaDomain::set_bit_representation ( )` [protected]

Switch to bit representation of domain. @ It changes only identifier in the REP field.

6.14.1.14 `void CudaDomain::set_bitlist_representation ( int num_list = INT_BITLIST )` [protected]

Switch to bitlist representation of domain.

## Parameters

<i>num_list</i>	the number (positive) of bitlists. @ It changes only identifier in the REP field.
-----------------	---

6.14.1.15 void CudaDomain::set\_bounds ( int *min*, int *max* )

The same as set\_bounds. It shrinks the domain to {min, max}.

## Parameters

<i>min</i>	lower bound
<i>max</i>	upper bound

6.14.1.16 void CudaDomain::set\_concrete\_domain ( int \*const *concrete\_domain* )

Set a concrete domain. It overrides the current concrete domain representation.

## Note

the client must provide a consistent internal domain's representation.

6.14.1.17 void CudaDomain::set\_list\_representation ( int *num\_list* = INT\_LIST ) [protected]

Switch to list representation of domain.

## Parameters

<i>num_list</i>	the number (positive) of bitlists. @ It changes only identifier in the REP field.
-----------------	---

6.14.1.18 bool CudaDomain::set\_singleton ( int *val* ) [virtual]

Set domain as singleton as {val}.

## Parameters

<i>val</i>	the value to set as singleton.
------------	--------------------------------

Implements [IntDomain](#).

6.14.1.19 void CudaDomain::shrink ( int *min*, int *max* ) [virtual]

It specializes the parent method in order to set up the array of (int) values. It instantiates a domain [min, max]. This actually updates the bounds and it performs consistency checking and updating of the domain size.

## Parameters

<i>min</i>	lower bound
<i>max</i>	upper bound

Implements [IntDomain](#).

## 6.14.1.20 void CudaDomain::switch\_list\_to\_bitmaplist ( ) [protected]

Take the current list representation and switch it to a bitmap list representation.



**Note**

it doesn't work from bitmap to bitmap list.

**6.14.2 Member Data Documentation****6.14.2.1 CudaConcreteDomainPtr CudaDomain::\_concrete\_domain [protected]**

Actual domain is represented by an object of type "cuda\_concrete\_domain". This domain can be a either bitmap, a list of bounds, or a bitmap list, depending on the size of the domain. Internal switches between domain representations are performed automatically as soon as the domain's size is reduced to a given threshold.

**Note**

[system\\_description.h](#)

**6.14.2.2 int\* CudaDomain::\_domain [protected]**

**Domain** is the actual bit domain representation. Operations are performed on \_concrete\_domain, status is stored on \_domain. When another class needs this domain's representation, \_domain will be returned.

**6.14.2.3 size\_t CudaDomain::\_num\_allocated\_bytes [protected]**

Total allocated bytes for representing the current domain.

**6.14.2.4 size\_t CudaDomain::\_num\_int\_chunks [protected]**

Total number of bitchunks.

**Note**

it does not consider the first part related to information about domain.

**6.14.2.5 constexpr int CudaDomain::BITS\_IN\_BYTE = INT8\_C(8) [static], [protected]**

Macro to use for declaring the size of a byte in terms of bits.

**6.14.2.6 constexpr size\_t CudaDomain::MAX\_BYTES\_SIZE = SHARED\_MEM\_KB \* 1024 [static], [protected]**

Maximum domain size in terms of bytes.

**Note**

see CUDA specifications. Usually,  $(48 - 1) \text{ kB} = 47 * 1024 = 48128 \text{ Byte}$ .

**6.14.2.7 constexpr size\_t CudaDomain::MAX\_DOMAIN\_VALUES = ((MAX\_BYTES\_SIZE - MAX\_STATUS\_SIZE) / sizeof(int)) [static], [protected]**

Maximum size in terms of storable values. Worst case: list of type {1, 1}, {3, 3}, {5, 5}, ... Number of integers =  $((\text{MAX\_BYTES\_SIZE} - 5 * \text{sizeof}(int)) / \text{sizeof}(int))$

**Note**

see CUDA specifications.

6.14.2.8 `constexpr size_t CudaDomain::MAX_STATUS_SIZE = 5 * sizeof( int )` `[static], [protected]`

Number of Bytes needed for representing the current domain status.

6.14.2.9 `constexpr int CudaDomain::SHARED_MEM_KB = 47` `[static], [protected]`

Shared memory available.

#### Note

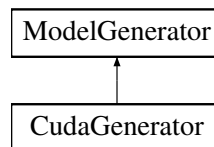
keep 1 kB less than the actual memory available.

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda_domain.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda_domain.cpp`

## 6.15 CudaGenerator Class Reference

Inheritance diagram for CudaGenerator:



### Public Member Functions

- VariablePtr [get\\_variable](#) (TokenPtr)  
*See "model\_generator.h".*
- ConstraintPtr [get\\_constraint](#) (TokenPtr)  
*See "model\_generator.h".*
- SearchEnginePtr [get\\_search\\_engine](#) (TokenPtr)  
*See "model\_generator.h".*
- ConstraintStorePtr [get\\_store](#) ()  
*See "model\_generator.h".*

### Protected Attributes

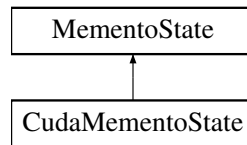
- `std::string _dbg`

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda_model_generator.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda_model_generator.cpp`

## 6.16 CudaMementoState Class Reference

Inheritance diagram for CudaMementoState:



### Public Member Functions

- [CudaMementoState](#) (IntDomainPtr int\_domain)
- void [set\\_memento](#) (IntDomainPtr int\_domain)
- void [print](#) () const override

*Print information about this memento state.*

### 6.16.1 Constructor & Destructor Documentation

#### 6.16.1.1 CudaMementoState::CudaMementoState ( IntDomainPtr int\_domain )

Constructor for Cuda [Memento](#).

Parameters

<i>int_domain</i>	a reference to a int domain from which get the internal domain's representation.
-------------------	--

### 6.16.2 Member Function Documentation

#### 6.16.2.1 void CudaMementoState::set\_memento ( IntDomainPtr int\_domain )

Sets domain's state as new state into the given (int) domain

Parameters

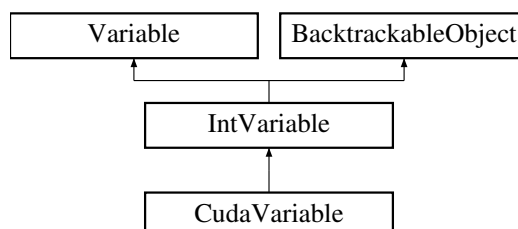
<i>int_domain</i>	a reference to the domain to update.
-------------------	--------------------------------------

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_memento\_state.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_memento\_state.cpp

## 6.17 CudaVariable Class Reference

Inheritance diagram for CudaVariable:



## Public Member Functions

- [CudaVariable](#) ()
- [CudaVariable](#) (int idv)
- virtual [~CudaVariable](#) ()
- Destructor.*
- void [set\\_domain](#) ()
- void [set\\_domain](#) (int lw, int ub)
- void [set\\_domain](#) (std::vector< std::vector< int > > elems)
- void [restore\\_state](#) () override
- void [set\\_state](#) () override
- void [print\\_domain](#) () const override
- Print domain.*
- void [print](#) () const
- print info about the current domain*

## Additional Inherited Members

### 6.17.1 Constructor & Destructor Documentation

#### 6.17.1.1 [CudaVariable::CudaVariable](#) ( )

Base constructor: create a variable with new id. The id is given by a global id generator.

#### 6.17.1.2 [CudaVariable::CudaVariable](#) ( int idv )

One parameter constructor: create a variable with a given id.

##### Parameters

<i>idv</i>	identifier to give to the variable
------------	------------------------------------

### 6.17.2 Member Function Documentation

#### 6.17.2.1 void [CudaVariable::restore\\_state](#) ( ) [override],[virtual]

Restore a state from the current state hold by the [BacktrackableObject](#).

##### Note

override backtrackable object methods.

Implements [BacktrackableObject](#).

#### 6.17.2.2 void [CudaVariable::set\\_domain](#) ( ) [virtual]

Set domain's bounds. If no bounds are provided, an unbounded domain (int) is instantiated. If an array of elements A is provided, the function instantiates a domain D = [min/2 A, max A], deleting all the elements d in D s.t. d does not belong to A.

Implements [IntVariable](#).

#### 6.17.2.3 void [CudaVariable::set\\_domain](#) ( int lw, int ub ) [virtual]

Set domain's bounds. A new domain [lw, ub] is generated.

## Parameters

<i>lw</i>	lower bound
<i>ub</i>	upper bound

Implements [IntVariable](#).

6.17.2.4 `void CudaVariable::set_domain ( std::vector< std::vector< int > > elems ) [virtual]`

Set domain's elements. A domain {d\_1, ..., d\_n} is generated.

## Parameters

<i>elems</i>	vector of vectors (subsets) of domain's elements
--------------	--

**Todo** implement set of sets of elements.

Implements [IntVariable](#).

6.17.2.5 `void CudaVariable::set_state ( ) [override],[virtual]`

Set internal state with other information hold by concrete [BacktrackableObject](#) objects.

## Note

override backtrackable object methods.

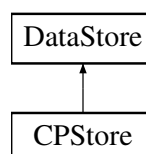
Implements [BacktrackableObject](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_variable.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/cuda\_variable.cpp

## 6.18 DataStore Class Reference

Inheritance diagram for DataStore:



### Public Member Functions

- virtual bool [load\\_model](#) (std::string="")=0
- virtual void [init\\_model](#) ()=0  
*Init model using the information read from files.*
- virtual void [print\\_model\\_info](#) ()=0  
*Print info about the model.*
- virtual [CPModel](#) \* [get\\_model](#) ()  
*Get the instantiated model.*
- virtual void [print\\_model\\_variable\\_info](#) ()
- virtual void [print\\_model\\_domain\\_info](#) ()
- virtual void [print\\_model\\_constraint\\_info](#) ()

## Protected Member Functions

- [DataStore](#) (std::string in\_file)

## Protected Attributes

- bool **\_timer**
- bool **\_verbose**
- std::string **\_dbg**
- std::string **\_in\_file** = ""
- [CPModel](#) \* **\_cp\_model**

*CP Model.*

## 6.18.1 Constructor & Destructor Documentation

### 6.18.1.1 DataStore::DataStore ( std::string in\_file ) [protected]

Constructor.

Parameters

<i>in_file</i>	file path of the model to parse.
----------------	----------------------------------

## 6.18.2 Member Function Documentation

### 6.18.2.1 virtual bool DataStore::load\_model ( std::string = " " ) [pure virtual]

Load model from input file (FlatZinc model).

Note

: the model described as a set of tokens is stored in the [Tokenization](#) class used by the parser. The parser has access to the set of tokens and it manages them in order to retrieve the correct set of tokens to initialize variables, and constraints. See [Parser](#) interface.

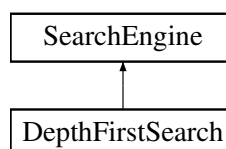
Implemented in [CPStore](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/data\_store.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/data\_store.cpp

## 6.19 DepthFirstSearch Class Reference

Inheritance diagram for DepthFirstSearch:



## Public Member Functions

- void [set\\_debug](#) (bool debug\_on)
- void [set\\_trail\\_debug](#) (bool debug\_on)
- void [set\\_store](#) (ConstraintStorePtr store) override
- void [set\\_heuristic](#) (HeuristicPtr heuristic) override
- void [set\\_solution\\_manager](#) (SolutionManager \*sol\_manager)
- void [set\\_backtrack\\_manager](#) (BacktrackManagerPtr bkt\_manager)
- size\_t [get\\_backtracks](#) () const override
- size\_t [get\\_nodes](#) () const override
- size\_t [get\\_wrong\\_decisions](#) () const override
- void [set\\_solution\\_limit](#) (size\_t num\_sol) override
- void [set\\_timeout\\_limit](#) (double timeout) override
- std::vector< DomainPtr > [get\\_solution](#) () const override
- std::vector< DomainPtr > [get\\_solution](#) (int n\_sol) const override
- bool [label](#) (int var) override
- bool [labeling](#) () override
- void [set\\_backtrack\\_out](#) (size\_t out\_b) override
- void [set\\_nodes\\_out](#) (size\_t out\_n) override
- void [set\\_wrong\\_decisions\\_out](#) (size\_t out\_w) override
- void [print\\_solution](#) () const override
  - Print on standard output last solution found.*
- void [print\\_all\\_solutions](#) () const override
  - Print all solutions found so far.*
- void [print\\_solution](#) (size\_t sol\_idx) const override
- void [print](#) () const override
  - Prints info about the search engine.*

## Protected Member Functions

- virtual void [init\\_search](#) ()
- virtual bool [search\\_out](#) ()

## Protected Attributes

- std::string [\\_dbg](#)
- size\_t [\\_depth](#)
- size\_t [\\_num\\_backtracks](#)
- size\_t [\\_num\\_nodes](#)
- size\_t [\\_num\\_wrong\\_decisions](#)
- bool [\\_debug](#)
  - Specifies if debug option is on.*
- bool [\\_trail\\_debug](#)
  - Specifies if debug and trail debug options are on.*
- bool [\\_search\\_out](#)
  - Specifies if the current search has been terminated.*
- bool [\\_backtrack\\_out\\_on](#)
  - Specifies if backtrack\_out is active.*
- size\_t [\\_backtracks\\_out](#)
  - Limit on the number of backtracks.*
- bool [\\_nodes\\_out\\_on](#)
  - Specifies if nodes\_out is active.*

- [size\\_t \\_nodes\\_out](#)  
*Limit on the number of nodes.*
- [bool \\_wrong\\_out\\_on](#)  
*Specifies if wrong\_out is active.*
- [size\\_t \\_wrong\\_out](#)  
*Limit on the number of wrong decisions.*
- [bool \\_timeout\\_out\\_on](#)  
*Specifies if timeout\_out is active.*
- [double \\_timeout\\_out](#)  
*Timeout value.*
- [ConstraintStorePtr \\_store](#)  
*Reference to the constraint store to use during this search.*
- [HeuristicPtr \\_heuristic](#)  
*Reference to the current heuristic to use during search.*
- [BacktrackManagerPtr \\_backtrack\\_manager](#)  
*Reference to the current backtrack manager.*
- [SolutionManager \\* \\_solution\\_manager](#)  
*Solution manager.*

### Static Protected Attributes

- [static size\\_t \\_search\\_id = 0](#)  
*Id for this search.*

## 6.19.1 Member Function Documentation

### 6.19.1.1 [size\\_t DepthFirstSearch::get\\_backtracks \( \) const](#) [override], [virtual]

Returns the number of backtracks performed by the search.

Returns

the number of backtracks.

Implements [SearchEngine](#).

### 6.19.1.2 [size\\_t DepthFirstSearch::get\\_nodes \( \) const](#) [override], [virtual]

Returns the number of nodes visited by the search.

Returns

the number of visited nodes.

Implements [SearchEngine](#).

### 6.19.1.3 [std::vector< DomainPtr > DepthFirstSearch::get\\_solution \( \) const](#) [override], [virtual]

Return the last solution found if any.

Returns

a vector of variables' domains (pointer to) Each domain is most probably a singleton and together represent a solution.

Implements [SearchEngine](#).



**6.19.1.4** `std::vector< DomainPtr > DepthFirstSearch::get_solution ( int n_sol ) const` `[override],[virtual]`

Return the  $n^{\text{th}}$  solution found if any.

Parameters

<i>n_sol</i>	the solution to get.
--------------	----------------------

Returns

a vector of variables' domains (pointer to) Each domain is most probably a singleton and together represent a solution.

Note

The first solution has index 1.

Implements [SearchEngine](#).

**6.19.1.5** `size_t DepthFirstSearch::get_wrong_decisions ( ) const` `[override],[virtual]`

Returns the number of wrong decisions made during the search process.

Returns

the number of wrong decisions.

Note

a decision is "wrong" depending on the search engine used to explore the search space. Usually, a wrong decision is represented by a leaf of the search tree which has failed.

Implements [SearchEngine](#).

**6.19.1.6** `void DepthFirstSearch::init_search ( )` `[protected],[virtual]`

Initializes the current search (i.e., any parameter used during search, as counters).

**6.19.1.7** `bool DepthFirstSearch::label ( int var )` `[override],[virtual]`

It assigns variables one by one. This function is called recursively.

Parameters

<i>var</i>	the index of the variable (not grounded) to assign.
------------	---

Returns

true if the solution was found.

Implements [SearchEngine](#).

**6.19.1.8** `bool DepthFirstSearch::labeling ( )` `[override],[virtual]`

It performs the actual search. First it sets up the internal items/attributes of search. Then, it calls the labeling function with argument specifying the index of a not grounded variable.

**Returns**

true if a solution was found.

Implements [SearchEngine](#).

**6.19.1.9** void DepthFirstSearch::print\_solution ( size\_t *sol\_idx* ) const [override],[virtual]

Print on standard output a solutions represented by its index.

**Parameters**

<i>sol_idx</i>	the index of the solution to print.
----------------	-------------------------------------

**Note**

first solution has index 1.

Implements [SearchEngine](#).

**6.19.1.10** bool DepthFirstSearch::search\_out ( ) [protected],[virtual]

Tells whether the search has to be terminated due to some limits (e.g., timeout, nodes\_out, etc.).

**Returns**

true is the search has to be terminated, false otherwise.

**6.19.1.11** void DepthFirstSearch::set\_backtrack\_manager ( BacktrackManagerPtr *bkt\_manager* ) [virtual]

Sets a backtrackable manager to this class.

**Parameters**

<i>bkt_manager</i>	a reference to a backtrack manager.
--------------------	-------------------------------------

Implements [SearchEngine](#).

**6.19.1.12** void DepthFirstSearch::set\_backtrack\_out ( size\_t *out\_b* ) [override],[virtual]

Set a maximum number of backtracks to perform during search.

**Parameters**

<i>the</i>	number of backtracks to consider as a limit during the search.
------------	--

Implements [SearchEngine](#).

**6.19.1.13** void DepthFirstSearch::set\_debug ( bool *debug\_on* ) [virtual]

Set debug options.

**Parameters**

<i>debug_on</i>	boolean value indicating if debug should be enabled.
-----------------	--

**Note**

default debug is off.

Implements [SearchEngine](#).

**6.19.1.14** void DepthFirstSearch::set\_heuristic ( HeuristicPtr *heuristic* ) [override],[virtual]

Set the heuristic to use to get the variables and the values every time a node of the search tree is explored.

**Parameters**

<i>a</i>	reference to a heuristic.
----------	---------------------------

Implements [SearchEngine](#).

**6.19.1.15** void DepthFirstSearch::set\_nodes\_out ( size\_t *out\_n* ) [override],[virtual]

Set a maximum number of nodes to visit during search.

**Parameters**

<i>the</i>	number of nodes to visit and to be considered as a limit during the search.
------------	---

Implements [SearchEngine](#).

**6.19.1.16** void DepthFirstSearch::set\_solution\_limit ( size\_t *num\_sol* ) [override],[virtual]

Set maximum number of solutions to be found.

**Parameters**

<i>num_sol</i>	the maximum number of solutions.
----------------	----------------------------------

**Note**

-1 states for "find all solutions".

Implements [SearchEngine](#).

**6.19.1.17** void DepthFirstSearch::set\_solution\_manager ( SolutionManager \* *sol\_manager* ) [virtual]

Set a solution manager for this search engine.

**Parameters**

<i>a</i>	reference to a solution manager.
----------	----------------------------------

Implements [SearchEngine](#).

**6.19.1.18** void DepthFirstSearch::set\_store ( ConstraintStorePtr *store* ) [override],[virtual]

Set a reference to a constraint store. The given store will be used to evaluate the constraints.

## Parameters

<i>a</i>	reference to a constraint store.
----------	----------------------------------

Implements [SearchEngine](#).

**6.19.1.19** void DepthFirstSearch::set\_timeout\_limit ( double *timeout* ) [override],[virtual]

Imposes a timeoutlimit.

## Parameters

<i>timeout</i>	timeout limit.
----------------	----------------

## Note

-1 for no timeout.

Implements [SearchEngine](#).

**6.19.1.20** void DepthFirstSearch::set\_trail\_debug ( bool *debug\_on* ) [virtual]

Set debug with trail option. If enabled it prints debug and trail stack behaviours.

## Parameters

<i>debug_on</i>	boolean value indicating if debug should be enabled.
-----------------	--

Implements [SearchEngine](#).

**6.19.1.21** void DepthFirstSearch::set\_wrong\_decisions\_out ( size\_t *out\_w* ) [override],[virtual]

Set a maximum number of wrong decisions to make before exiting the search phase.

## Parameters

<i>the</i>	number of wrong decisions to set as a limit during the search.
------------	--

Implements [SearchEngine](#).

## 6.19.2 Member Data Documentation

**6.19.2.1** size\_t DepthFirstSearch::\_num\_backtracks [protected]

Stores the number of backtracks during search. A backtrack is a node for which all children have failed.

**6.19.2.2** size\_t DepthFirstSearch::\_num\_nodes [protected]

Stores the number of search nodes explored during search.

**6.19.2.3** size\_t DepthFirstSearch::\_num\_wrong\_decisions [protected]

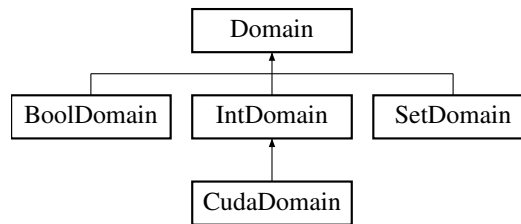
Stores the number of wrong decisions that have been made during search. A wrong decision is represented by a leaf of the search tree which has failed.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/depth\_first\_search.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/depth\_first\_search.cpp

## 6.20 Domain Class Reference

Inheritance diagram for Domain:



### Public Member Functions

- void [set\\_type](#) (DomainType dt)
- DomainType [get\\_type](#) () const
- virtual DomainPtr [clone](#) () const =0
- virtual void [reset\\_event](#) ()=0
- virtual EventType [get\\_event](#) () const =0
- virtual size\_t [get\\_size](#) () const =0
- virtual bool [is\\_empty](#) () const =0
- virtual bool [is\\_singleton](#) () const =0
- virtual bool [is\\_numeric](#) () const =0
- virtual std::string [get\\_string\\_representation](#) () const =0
- virtual void [print](#) () const =0

*Print info about this domain.*

### Static Public Member Functions

- static constexpr int [MIN\\_DOMAIN](#) ()

*Constants for int min/max domain bounds.*

- static constexpr int [MAX\\_DOMAIN](#) ()

*Constants for int min/max domain bounds.*

### Protected Member Functions

- [Domain](#) ()

*Constructor.*

### Protected Attributes

- std::string [\\_dbg](#)

*Debug info string.*

- DomainType [\\_dom\\_type](#)

*Domain type.*

## 6.20.1 Member Function Documentation

### 6.20.1.1 `virtual DomainPtr Domain::clone ( ) const [pure virtual]`

Clone the current domain and returns a pointer to it.

#### Returns

a pointer to a domain that has been initialized as a copy (clone) of this domain.

Implemented in [CudaDomain](#), [SetDomain](#), and [BoolDomain](#).

### 6.20.1.2 `virtual EventType Domain::get_event ( ) const [pure virtual]`

Returns the current event on the domain.

#### Returns

an event described as EventType that represents the current event (state) of this domain.

Implemented in [CudaDomain](#), [SetDomain](#), and [BoolDomain](#).

### 6.20.1.3 `virtual size_t Domain::get_size ( ) const [pure virtual]`

Returns the size of the domain.

#### Returns

the size of this domain.

Implemented in [CudaDomain](#), [SetDomain](#), and [BoolDomain](#).

### 6.20.1.4 `virtual std::string Domain::get_string_representation ( ) const [pure virtual]`

Returns a string description of this domain, i.e., the list of values in the current domain.

#### Returns

a string representing the values in this domain.

Implemented in [SetDomain](#), [BoolDomain](#), and [IntDomain](#).

### 6.20.1.5 `virtual bool Domain::is_empty ( ) const [pure virtual]`

Returns true if the domain is empty.

#### Returns

true if this domain is empty, false otherwise.

Implemented in [SetDomain](#), [BoolDomain](#), and [IntDomain](#).

6.20.1.6 `virtual bool Domain::is_numeric ( ) const [pure virtual]`

Specifies if domain is a finite domain of numeric values (integers).

#### Returns

true if domain contains numeric values (not reals).

Implemented in [SetDomain](#), [BoolDomain](#), and [IntDomain](#).

6.20.1.7 `virtual bool Domain::is_singleton ( ) const [pure virtual]`

Returns true if the domain has only one element.

#### Returns

true if this domain is a singleton, false otherwise.

Implemented in [SetDomain](#), [BoolDomain](#), and [IntDomain](#).

6.20.1.8 `virtual void Domain::reset_event ( ) [pure virtual]`

Sets the no event on this domain.

#### Note

No event won't trigger any propagation on this domain.

Implemented in [CudaDomain](#), [SetDomain](#), and [BoolDomain](#).

6.20.1.9 `void Domain::set_type ( DomainType dt )`

Set domain's type (use `get_type` to get the type).

#### Parameters

<i>dt</i>	domain type of type <code>DomainType</code>
-----------	---

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

- `/Users/fedecampe/Desktop/NVIDIAOSO-PRJ/NVIDIAOSO/NVIDIAOSO/domain.h`
- `/Users/fedecampe/Desktop/NVIDIAOSO-PRJ/NVIDIAOSO/NVIDIAOSO/domain.cpp`

## 6.21 DomainIterator Class Reference

### Public Member Functions

- **DomainIterator** (`IntDomainPtr domain`)
- virtual bool `is_numeric () const`
- virtual int `min_val () const`
- virtual int `max_val () const`
- virtual int `random_val () const`
- virtual `size_t domain_size () const`
- virtual `std::string get_string_representation () const`

## Protected Attributes

- `IntDomainPtr _domain`

### 6.21.1 Member Function Documentation

#### 6.21.1.1 `size_t DomainIterator::domain_size ( ) const [virtual]`

Returns the current domain's size.

Returns

current domain's size.

#### 6.21.1.2 `std::string DomainIterator::get_string_representation ( ) const [virtual]`

Returns a string description of this domain, i.e., the list of values in the current domain.

Returns

a string representing the values in this domain.

#### 6.21.1.3 `bool DomainIterator::is_numeric ( ) const [virtual]`

Checks if the current domain is a numeric domain.

Returns

true if current domain is numeric (i.e., int domain).

#### 6.21.1.4 `int DomainIterator::max_val ( ) const [virtual]`

Returns the current maximal value in domain.

Returns

the maximum value belonging to the domain.

#### 6.21.1.5 `int DomainIterator::min_val ( ) const [virtual]`

Returns the current minimal value in domain.

Returns

the minimum value belonging to the domain.

#### 6.21.1.6 `int DomainIterator::random_val ( ) const [virtual]`

Returns a random value from domain.

Returns

the a random value belonging to the domain.

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

- `/Users/fedecampe/Desktop/NVIDIAIOSO-PRJ/NVIDIAIOSO/NVIDIAIOSO/domain_iterator.h`
- `/Users/fedecampe/Desktop/NVIDIAIOSO-PRJ/NVIDIAIOSO/NVIDIAIOSO/domain_iterator.cpp`



## 6.22 FactoryModelGenerator Class Reference

### Static Public Member Functions

- static [ModelGenerator](#) \* [get\\_generator](#) (GeneratorType gt)  
*Get the right instance of a generator based on the input.*

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/factory\_generator.h

## 6.23 FactoryParser Class Reference

### Static Public Member Functions

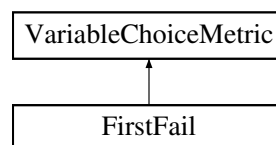
- static [Parser](#) \* [get\\_parser](#) (ParserType pt)  
*Get the right parser based on the input.*

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/factory\_parser.h

## 6.24 FirstFail Class Reference

Inheritance diagram for FirstFail:



### Public Member Functions

- int [compare](#) (double metric, [Variable](#) \*var)
- int [compare](#) ([Variable](#) \*var\_a, [Variable](#) \*var\_b)
- double [metric\\_value](#) ([Variable](#) \*var)  
*Get the metric value for first\_fail.*
- void [print](#) () const  
*Print info.*

### Additional Inherited Members

#### 6.24.1 Member Function Documentation

6.24.1.1 int FirstFail::compare ( double metric, [Variable](#) \* var ) [virtual]

Compare a metric value and a variable. Metric is given by their domain's size.

Implements [VariableChoiceMetric](#).

### 6.24.1.2 `int FirstFail::compare ( Variable * var_a, Variable * var_b ) [virtual]`

Compare variables w.r.t. their metrics. Metric is given by their domain's size.

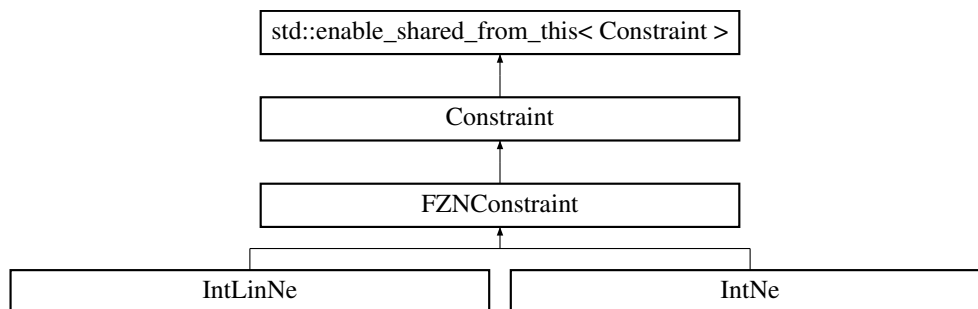
Implements [VariableChoiceMetric](#).

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/first_fail_metric.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/first_fail_metric.cpp`

## 6.25 FZNConstraint Class Reference

Inheritance diagram for FZNConstraint:



### Public Member Functions

- virtual void [setup](#) (std::vector< VariablePtr > vars, std::vector< std::string > args)=0
- void [attach\\_me\\_to\\_vars](#) () override
- void [consistency](#) () override
- bool [satisfied](#) () override
- void [remove\\_constraint](#) ()
- void [print](#) () const override  
*Prints info.*
- void [print\\_semantic](#) () const override  
*Prints the semantic of this constraint.*

### Static Public Member Functions

- static FZNConstraintType [int\\_to\\_type](#) (int number\_id)
- static int [type\\_to\\_int](#) (FZNConstraintType c\_type)
- static int [name\\_to\\_id](#) (std::string c\_name)

### Static Public Attributes

- static const std::string **ARRAY\_BOOL\_AND** = "array\_bool\_and"
- static const std::string **ARRAY\_BOOL\_ELEMENT** = "array\_bool\_element"
- static const std::string **ARRAY\_BOOL\_OR** = "array\_bool\_or"
- static const std::string **ARRAY\_FLOAT\_ELEMENT** = "array\_float\_element"
- static const std::string **ARRAY\_INT\_ELEMENT** = "array\_int\_element"
- static const std::string **ARRAY\_SET\_ELEMENT** = "array\_set\_element"
- static const std::string **ARRAY\_VAR\_BOOL\_ELEMENT** = "array\_var\_bool\_element"

- static const std::string **ARRAY\_VAR\_FLOAT\_ELEMENT** = "array\_var\_float\_element"
- static const std::string **ARRAY\_VAR\_INT\_ELEMENT** = "array\_var\_int\_element"
- static const std::string **ARRAY\_VAR\_SET\_ELEMENT** = "array\_var\_set\_element"
- static const std::string **BOOL2INT** = "bool2int"
- static const std::string **BOOL\_AND** = "bool\_and"
- static const std::string **BOOL\_CLAUSE** = "bool\_clause"
- static const std::string **BOOL\_EQ** = "bool\_eq"
- static const std::string **BOOL\_EQ\_REIF** = "bool\_eq\_reif"
- static const std::string **BOOL\_LE** = "bool\_le"
- static const std::string **BOOL\_LE\_REIF** = "bool\_le\_reif"
- static const std::string **BOOL\_LT** = "bool\_lt"
- static const std::string **BOOL\_LT\_REIF** = "bool\_lt\_reif"
- static const std::string **BOOL\_NOT** = "bool\_not"
- static const std::string **BOOL\_OR** = "bool\_or"
- static const std::string **BOOL\_XOR** = "bool\_xor"
- static const std::string **FLOAT\_ABS** = "float\_abs"
- static const std::string **FLOAT\_ACOS** = "float\_acos"
- static const std::string **FLOAT\_ASIN** = "float\_asin"
- static const std::string **FLOAT\_ATAN** = "float\_atan"
- static const std::string **FLOAT\_COS** = "float\_cos"
- static const std::string **FLOAT\_COSH** = "float\_cosh"
- static const std::string **FLOAT\_EXP** = "float\_exp"
- static const std::string **FLOAT\_LN** = "float\_ln"
- static const std::string **FLOAT\_LOG10** = "float\_log10"
- static const std::string **FLOAT\_LOG2** = "float\_log2"
- static const std::string **FLOAT\_SQRT** = "float\_sqrt"
- static const std::string **FLOAT\_SIN** = "float\_sin"
- static const std::string **FLOAT\_SINH** = "float\_sinh"
- static const std::string **FLOAT\_TAN** = "float\_tan"
- static const std::string **FLOAT\_TANH** = "float\_tanh"
- static const std::string **FLOAT\_EQ** = "float\_eq"
- static const std::string **FLOAT\_EQ\_REIF** = "float\_eq\_reif"
- static const std::string **FLOAT\_LE** = "float\_le"
- static const std::string **FLOAT\_LE\_REIF** = "float\_le\_reif"
- static const std::string **FLOAT\_LIN\_EQ** = "float\_lin\_eq"
- static const std::string **FLOAT\_LIN\_EQ\_REIF** = "float\_lin\_eq\_reif"
- static const std::string **FLOAT\_LIN\_LE** = "float\_lin\_le"
- static const std::string **FLOAT\_LIN\_LE\_REIF** = "float\_lin\_le\_reif"
- static const std::string **FLOAT\_LIN\_LT** = "float\_lin\_lt"
- static const std::string **FLOAT\_LIN\_LT\_REIF** = "float\_lin\_lt\_reif"
- static const std::string **FLOAT\_LIN\_NE** = "float\_lin\_ne"
- static const std::string **FLOAT\_LIN\_NE\_REIF** = "float\_lin\_ne\_reif"
- static const std::string **FLOAT\_LT** = "float\_lt"
- static const std::string **FLOAT\_LT\_REIF** = "float\_lt\_reif"
- static const std::string **FLOAT\_MAX** = "float\_max"
- static const std::string **FLOAT\_MIN** = "float\_min"
- static const std::string **FLOAT\_NE** = "float\_ne"
- static const std::string **FLOAT\_NE\_REIF** = "float\_ne\_reif"
- static const std::string **FLOAT\_PLUS** = "float\_plus"
- static const std::string **INT\_ABS** = "int\_abs"
- static const std::string **INT\_DIV** = "int\_div"
- static const std::string **INT\_EQ** = "int\_eq"
- static const std::string **INT\_EQ\_REIF** = "int\_eq\_reif"
- static const std::string **INT\_LE** = "int\_le"
- static const std::string **INT\_LE\_REIF** = "int\_le\_reif"

- static const std::string **INT\_LIN\_EQ** = "int\_lin\_eq"
- static const std::string **INT\_LIN\_EQ\_REIF** = "int\_lin\_eq\_reif"
- static const std::string **INT\_LIN\_LE** = "int\_lin\_le"
- static const std::string **INT\_LIN\_LE\_REIF** = "int\_lin\_le\_reif"
- static const std::string **INT\_LIN\_NE** = "int\_lin\_ne"
- static const std::string **INT\_LIN\_NE\_REIF** = "int\_lin\_ne\_reif"
- static const std::string **INT\_MAX\_C** = "int\_max"
- static const std::string **INT\_MIN\_C** = "int\_min"
- static const std::string **INT\_MOD** = "int\_mod"
- static const std::string **INT\_NE** = "int\_ne"
- static const std::string **INT\_NE\_REIF** = "int\_ne\_reif"
- static const std::string **INT\_PLUS** = "int\_plus"
- static const std::string **INT\_TIMES** = "int\_times"
- static const std::string **INT2FLOAT** = "int2float"
- static const std::string **SET\_CARD** = "set\_card"
- static const std::string **SET\_DIFF** = "set\_diff"
- static const std::string **SET\_EQ** = "set\_eq"
- static const std::string **SET\_EQ\_REIF** = "set\_eq\_reif"
- static const std::string **SET\_IN** = "set\_in"
- static const std::string **SET\_IN\_REIF** = "set\_in\_reif"
- static const std::string **SET\_INTERSECT** = "set\_intersect"
- static const std::string **SET\_LE** = "set\_le"
- static const std::string **SET\_LT** = "set\_lt"
- static const std::string **SET\_NE** = "set\_ne"
- static const std::string **SET\_NE\_REIF** = "set\_ne\_reif"
- static const std::string **SET\_SUBSET** = "set\_subset"
- static const std::string **SET\_SUBSET\_REIF** = "set\_subset\_reif"
- static const std::string **SET\_SYMDIFF** = "set\_symdiff"
- static const std::string **SET\_UNION** = "set\_union"
- static const std::string **OTHER** = "other"

## Protected Member Functions

- [FZNConstraint](#) (std::string name)

## Protected Attributes

- FZNConstraintType [\\_constraint\\_type](#)  
*FlatZinc constraint type.*
- int [\\_scope\\_size](#)  
*Scope size.*

## 6.25.1 Constructor & Destructor Documentation

### 6.25.1.1 FZNConstraint::FZNConstraint ( std::string name ) [protected]

Base constructor.

Parameters

---

<i>name</i>	the name of the FlatZinc constraint.
<i>vars</i>	the vector of (shared) pointers to the variables in the scope of this constraint.
<i>args</i>	the vector of auxiliary arguments stored as strings needed by this constraint in order to be propagated.

**Note**

[FZNConstraint](#) instantiated with this constructor need to be defined in terms of variables in their scope and, if needed, auxiliary parameters.

**6.25.2 Member Function Documentation****6.25.2.1 void FZNConstraint::attach\_me\_to\_vars ( ) [override],[virtual]**

It attaches this constraint (observer) to the list of the variables in its scope. When a variable changes state, this constraint could be automatically notified (depending on the variable).

Implements [Constraint](#).

**6.25.2.2 void FZNConstraint::consistency ( ) [override],[virtual]**

It is a (most probably incomplete) consistency function which removes the values from variable domains. Only values which do not have any support in a solution space are removed.

Implements [Constraint](#).

Reimplemented in [IntNe](#), and [IntLinNe](#).

**6.25.2.3 FZNConstraintType FZNConstraint::int\_to\_type ( int number\_id ) [static]**

It converts a number\_id name to the correspondent FZNConstraintType type.

**Parameters**

<i>number_id</i>	the number id of the FlatZinc constraint.
------------------	---

**Returns**

the type of the FlatZinc constraint.

**6.25.2.4 int FZNConstraint::name\_to\_id ( std::string c\_name ) [static]**

It converts a string representing the name of a constraint to a unique identifier for the correspondent type of FlatZinc constraint.

**Parameters**

<i>c_name</i>	name of a FlatZinc constraint.
---------------	--------------------------------

**Returns**

the number\_id correspondent to name.

**6.25.2.5 void FZNConstraint::remove\_constraint ( ) [virtual]**

It removes the constraint by removing this constraint from all variables in its scope.

Implements [Constraint](#).

#### 6.25.2.6 `bool FZNConstraint::satisfied ( ) [override],[virtual]`

It checks if the constraint is satisfied.

##### Returns

true if the constraint is for certain satisfied, false otherwise.

##### Note

If this function is incorrectly implemented, a constraint may not be satisfied in a solution.

Implements [Constraint](#).

Reimplemented in [IntNe](#), and [IntLinNe](#).

#### 6.25.2.7 `virtual void FZNConstraint::setup ( std::vector< VariablePtr > vars, std::vector< std::string > args ) [pure virtual]`

It sets the variables and the arguments for this constraint.

##### Parameters

<i>vars</i>	a vector of pointers to the variables in the constraint's scope.
<i>args</i>	a vector of strings representing the auxiliary arguments needed by the constraint in order to ensure consistency.

Implemented in [IntNe](#), and [IntLinNe](#).

#### 6.25.2.8 `int FZNConstraint::type_to_int ( FZNConstraintType c_type ) [static]`

It converts a FZNConstraintType to the correspondent integer type.

##### Parameters

<i>c_type</i>	the type of the FlatZinc constraint.
---------------	--------------------------------------

##### Returns

the number\_id correspondent to c\_type.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_constraint.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_constraint.cpp

## 6.26 FZNConstraintFactory Class Reference

### Static Public Member Functions

- static [FZNConstraint](#) \* [get\\_fzn\\_constraint](#) (std::string c\_name, std::vector< VariablePtr > vars, std::vector< std::string > args)
- static [ConstraintPtr](#) [get\\_fzn\\_constraint\\_shr\\_ptr](#) (std::string c\_name, std::vector< VariablePtr > vars, std::vector< std::string > args)

### 6.26.1 Member Function Documentation

6.26.1.1 `static FZNConstraint* FZNConstraintFactory::get_fzn_constraint ( std::string c_name, std::vector< VariablePtr > vars, std::vector< std::string > args )` `[inline]`, `[static]`

Get the right instance of FlatZinc constraint according to its type described by the input string.

## Parameters

<i>c_name</i>	the FlatZinc name of the constraint to instantiate.
<i>vars</i>	the vector of (shared) pointer to the FD variables in the scope of the constraint to instantiate.
<i>args</i>	the vector of strings representing the auxiliary arguments needed by the constraint to instantiate in order to be propagated.

6.26.1.2 `static ConstraintPtr FZNConstraintFactory::get_fzn_constraint_shr_ptr ( std::string c_name, std::vector< VariablePtr > vars, std::vector< std::string > args ) [inline],[static]`

Get the right instance of FlatZinc constraint according to its type described by the input string.

## Parameters

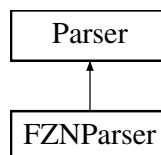
<i>c_name</i>	the FlatZinc name of the constraint to instantiate.
<i>vars</i>	the vector of (shared) pointer to the FD variables in the scope of the constraint to instantiate.
<i>args</i>	the vector of strings representing the auxiliary arguments needed by the constraint to instantiate in order to be propagated.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_constraint\_generator.h

## 6.27 FZNParser Class Reference

Inheritance diagram for FZNParser:



### Public Member Functions

- **FZNParser** (std::string ifile)
- bool `more_variables` () const  
*Ask whether there are more variables to get.*
- bool `more_constraints` () const  
*Ask whether there are more constraints to get.*
- bool `more_search_engines` () const  
*Ask whether there are more search engines to get.*
- TokenPtr `get_variable` ()
- TokenPtr `get_constraint` ()
- TokenPtr `get_search_engine` ()
- TokenPtr `get_next_content` ()  
*Get next (pointer to) token (i.e., FlatZinc element)*
- void `print` () const  
*Print info about the parser.*



## Additional Inherited Members

### 6.27.1 Member Function Documentation

#### 6.27.1.1 TokenPtr FZNParser::get\_constraint ( ) [virtual]

Get a "constraint" token.

##### Returns

token pointer to a "constraint" token.

Implements [Parser](#).

#### 6.27.1.2 TokenPtr FZNParser::get\_next\_content ( ) [virtual]

Get next (pointer to) token (i.e., FlatZinc element)

Set position on file to the most recent position

Implements [Parser](#).

#### 6.27.1.3 TokenPtr FZNParser::get\_search\_engine ( ) [virtual]

Get a "search\_engine" token.

##### Returns

token pointer to a "search\_engine" token.

Implements [Parser](#).

#### 6.27.1.4 TokenPtr FZNParser::get\_variable ( ) [virtual]

Get a "variable" token.

##### Returns

token pointer to a "variable" token.

Implements [Parser](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_parser.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_parser.cpp

## 6.28 FZNSearchFactory Class Reference

### Static Public Member Functions

- static SearchEnginePtr [get\\_fzn\\_search\\_shr\\_ptr](#) (std::vector< [Variable](#) \* > variables, [TokenSol](#) \*search\_tkn)

## 6.28.1 Member Function Documentation

6.28.1.1 `static SearchEnginePtr FZNSearchFactory::get_fzn_search_shr_ptr ( std::vector< Variable * > variables,  
TokenSol * search_tkn ) [inline],[static]`

Get the right instance of FlatZinc search method according to its type described by the input string.

## Parameters

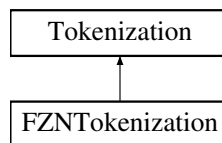
<i>variables</i>	a vector of pointers to all the variables in the model.
<i>search_tkn</i>	reference to a search token in order to define the right instance of search engine.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_search\_generator.h

## 6.29 FZNTokenization Class Reference

Inheritance diagram for FZNTokenization:



### Public Member Functions

- TokenPtr [get\\_token](#) ()

### Additional Inherited Members

#### 6.29.1 Member Function Documentation

##### 6.29.1.1 TokenPtr FZNTokenization::get\_token ( ) [virtual]

Specialized method: It actually gets the right token according to the FlatZinc format. Analysis is performed on "\_c\_token".

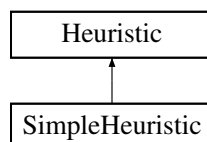
Implements [Tokenization](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_tokenization.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/fzn\_tokenization.cpp

## 6.30 Heuristic Class Reference

Inheritance diagram for Heuristic:



### Public Member Functions

- virtual int [get\\_index](#) () const

- virtual [Variable](#) \* [get\\_choice\\_variable](#) (int idx)=0
- virtual int [get\\_choice\\_value](#) ()=0
- virtual void [print](#) () const =0

*Print info about heuristic.*

## Protected Attributes

- std::string [\\_dbg](#)  
*Debug info.*
- int [\\_current\\_index](#)  
*Current index used to select the next choice variable.*

## 6.30.1 Member Function Documentation

### 6.30.1.1 virtual int Heuristic::get\_choice\_value ( ) [pure virtual]

Returns a value which will represent the next choice point (i.e., the next value to assign to the variable selected by this heuristic).

#### Returns

the value used in the choice point (value)

#### Note

this value is an integer value. If variables are not defined on integer values (e.g., float vars), this method should either be implemented consistently or never used.

Implemented in [SimpleHeuristic](#).

### 6.30.1.2 virtual Variable\* Heuristic::get\_choice\_variable ( int idx ) [pure virtual]

Returns the variable which will represent the next choice point (i.e., the next variable to label).

#### Parameters

<i>idx</i>	the position of the last variable which has been returned by this heuristic and which has not been backtracked upon yet.
------------	--

#### Returns

a reference to the variable to label in the next step according to this heuristic. nullptr is returned if all variables are assigned.

Implemented in [SimpleHeuristic](#).

### 6.30.1.3 int Heuristic::get\_index ( ) const [virtual]

Return the current index (last index used) to select the choice variable.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/heuristic.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/heuristic.cpp

## 6.31 IdGenerator Class Reference

### Public Member Functions

- void [reset\\_int\\_id](#) ()  
*Reset id generator.*
- void [reset\\_str\\_id](#) ()  
*Reset id generator.*
- void [set\\_base\\_offset](#) (int)  
*Set (base) ids (if not already set).*
- void [set\\_base\\_prefix](#) (std::string)  
*Set (base) ids (if not already set)*
- int [get\\_int\\_id](#) ()  
*Get a new unique int id.*
- std::string [get\\_str\\_id](#) ()  
*Get a new unique string id.*
- int [new\\_int\\_id](#) ()  
*Get a new unique int id.*
- std::string [new\\_str\\_id](#) ()  
*Get a new unique string id.*
- int [curr\\_int\\_id](#) ()  
*Get the current id already generated.*
- std::string [curr\\_str\\_id](#) ()  
*Get the current id already generated.*
- void [print\\_int\\_id](#) ()
- void [print\\_str\\_id](#) ()

### Static Public Member Functions

- static [IdGenerator](#) \* [get\\_instance](#) ()  
*Constructor get (static) instance.*

### Protected Member Functions

- [IdGenerator](#) ()
- std::string [n\\_to\\_str](#) (int)  
*Convert numbers to string.*

#### 6.31.1 Constructor & Destructor Documentation

##### 6.31.1.1 [IdGenerator::IdGenerator](#) ( ) `[protected]`

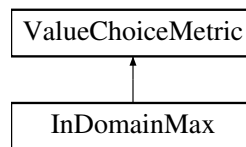
Protected constructor: a client cannot instantiate Singleton directly.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/id\_generator.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/id\_generator.cpp

## 6.32 InDomainMax Class Reference

Inheritance diagram for InDomainMax:



### Public Member Functions

- int [metric\\_value](#) ([Variable](#) \*var)
- void [print](#) () const  
*Print info about this value choice metric.*

### Additional Inherited Members

#### 6.32.1 Member Function Documentation

##### 6.32.1.1 int InDomainMax::metric\_value ( [Variable](#) \* var ) [virtual]

Gets value to assign to var using indomain\_max choice.

#### Parameters

<a href="#">var</a>	the (pointer to) variable for which a value if needed.
---------------------	--

#### Returns

the value to assign to var.

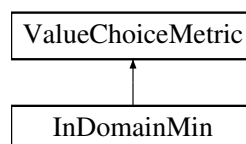
Implements [ValueChoiceMetric](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/indomain\_max\_metric.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/indomain\_max\_metric.cpp

## 6.33 InDomainMin Class Reference

Inheritance diagram for InDomainMin:



### Public Member Functions

- int [metric\\_value](#) ([Variable](#) \*var)
- void [print](#) () const  
*Print info about this value choice metric.*

## Additional Inherited Members

### 6.33.1 Member Function Documentation

#### 6.33.1.1 int InDomainMin::metric\_value ( Variable \* var ) [virtual]

Gets value to assign to var using indomain\_min choice.

##### Parameters

<i>var</i>	the (pointer to) variable for which a value if needed.
------------	--

##### Returns

the value to assign to var.

Implements [ValueChoiceMetric](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/indomain\_min\_metric.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/indomain\_min\_metric.cpp

## 6.34 InputData Class Reference

### Public Member Functions

- bool [verbose](#) () const
- bool [timer](#) () const
- double [timeout](#) () const
- int [max\\_n\\_sol](#) () const
- std::string [get\\_in\\_file](#) () const
- std::string [get\\_out\\_file](#) () const

### Static Public Member Functions

- static [InputData](#) \* [get\\_instance](#) (int argc, char \*argv[])  
*Constructor to get the (static) [InputData](#) instance.*

### Protected Member Functions

- [InputData](#) (int argc, char \*argv[])

### 6.34.1 Constructor & Destructor Documentation

#### 6.34.1.1 InputData::InputData ( int argc, char \* argv[] ) [protected]

Protected constructor: a client cannot instantiate Singleton directly. Exit if the user did not set an input file!

### 6.34.2 Member Function Documentation

#### 6.34.2.1 std::string InputData::get\_in\_file ( ) const

Get input file (path to).

**Returns**

the path where the input file is located.

**6.34.2.2   `std::string InputData::get_out_file ( ) const`**

Get output file (path to). If no path is given, output will be printed on standard output.

**Returns**

the path to the file where the output results should be written.

**6.34.2.3   `int InputData::max_n_sol ( ) const`**

Returns the limit on the number of solution set by the user (default: 1).

**Returns**

the given limit on the number of solutions.

**6.34.2.4   `double InputData::timeout ( ) const`**

Returns the timeout limit set by the user (default: inf).

**Returns**

the timeout limit.

**6.34.2.5   `bool InputData::timer ( ) const`**

Informs about the time option.

**Returns**

true if timer is on.

**6.34.2.6   `bool InputData::verbose ( ) const`**

Informs about the verbose option.

**Returns**

true if verbose is on.

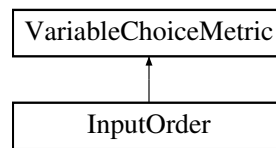
The documentation for this class was generated from the following files:

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/input_data.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/input_data.cpp`



## 6.35 InputOrder Class Reference

Inheritance diagram for InputOrder:



### Public Member Functions

- int [compare](#) (double metric, [Variable](#) \*var)
- int [compare](#) ([Variable](#) \*var\_a, [Variable](#) \*var\_b)
- double [metric\\_value](#) ([Variable](#) \*var)  
*Get the metric value for input\_order.*
- void [print](#) () const  
*Print info.*

### Additional Inherited Members

#### 6.35.1 Member Function Documentation

6.35.1.1 int InputOrder::compare ( double metric, [Variable](#) \* var ) [virtual]

Compare a metric value and a variable. Metric is given by the id of the vars as they have been defined when instantiated.

Implements [VariableChoiceMetric](#).

6.35.1.2 int InputOrder::compare ( [Variable](#) \* var\_a, [Variable](#) \* var\_b ) [virtual]

Compare variables w.r.t. their metrics. Metric is given by the id of the vars as they have been defined when instantiated.

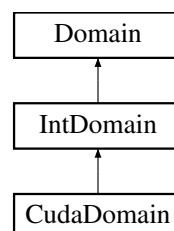
Implements [VariableChoiceMetric](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/input\_order\_metric.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/input\_order\_metric.cpp

## 6.36 IntDomain Class Reference

Inheritance diagram for IntDomain:



## Public Member Functions

- bool [is\\_singleton](#) () const  
*Returns true if the domain has only one element.*
- bool [is\\_empty](#) () const  
*Returns true if the domain is empty.*
- bool [is\\_numeric](#) () const  
*Returns true if this is a numeric finite domain.*
- std::string [get\\_string\\_representation](#) () const  
*Get string rep. of this domain.*
- virtual void [print](#) () const  
*Print base info about int domain.*
- virtual int [lower\\_bound](#) () const =0  
*Get the domain's lower bound.*
- virtual int [upper\\_bound](#) () const =0  
*Get the domain's upper bound.*
- virtual bool [contains](#) (int value) const =0
- virtual void [init\\_domain](#) (int min, int max)=0
- virtual void [shrink](#) (int min, int max)=0
- virtual bool [set\\_singleton](#) (int val)=0
- virtual bool [subtract](#) (int val)=0
- virtual void [add\\_element](#) (int val)=0
- virtual void [in\\_min](#) (int min)=0
- virtual void [in\\_max](#) (int max)=0

## Additional Inherited Members

### 6.36.1 Member Function Documentation

#### 6.36.1.1 virtual void IntDomain::add\_element ( int val ) [pure virtual]

It computes the union of the current domain with the domain represented by the singleton element given in input to the method. If the element is out of [lower\_bound, upper\_bound] it enlarges the domain.

##### Parameters

<i>val</i>	element to add to the current domain.
------------	---------------------------------------

Implemented in [CudaDomain](#).

#### 6.36.1.2 virtual bool IntDomain::contains ( int value ) const [pure virtual]

It checks whether the value belongs to the domain or not.

##### Parameters

<i>value</i>	to check whether it is in the current domain.
--------------	---

##### Returns

true if value is in this domain, false otherwise

Implemented in [CudaDomain](#).

#### 6.36.1.3 virtual void IntDomain::in\_max ( int max ) [pure virtual]

It updates the domain according to the maximum value.

## Parameters

<i>max</i>	domain value.
------------	---------------

Implemented in [CudaDomain](#).

#### 6.36.1.4 virtual void IntDomain::in\_min ( int *min* ) [pure virtual]

It updates the domain according to the minimum value.

## Parameters

<i>min</i>	domain value.
------------	---------------

Implemented in [CudaDomain](#).

#### 6.36.1.5 virtual void IntDomain::init\_domain ( int *min*, int *max* ) [pure virtual]

Initialize domain: this function is used to set up the domain as soon it is created. Classes that derive [IntDomain](#) specilize this method according to their internal representation of domain.

Implemented in [CudaDomain](#).

#### 6.36.1.6 virtual bool IntDomain::set\_singleton ( int *val* ) [pure virtual]

Set domain to the singleton element given in input.

## Parameters

<i>val</i>	the value to set as singleton
------------	-------------------------------

## Returns

true if the domain has been set to singleton, false otherwise.

Implemented in [CudaDomain](#).

#### 6.36.1.7 virtual void IntDomain::shrink ( int *min*, int *max* ) [pure virtual]

Set domain's bounds. It updates the domain to have values only within the interval min..max.

## Parameters

<i>lower</i>	lower bound value
<i>upper</i>	upper bound value

Implemented in [CudaDomain](#).

#### 6.36.1.8 virtual bool IntDomain::subtract ( int *val* ) [pure virtual]

It intersects with the domain which is a complement of the value given as input, i.e., subtract a value from the current domain.

## Parameters

<code>val</code>	the value to subtract from the current domain
------------------	---

#### Returns

true if succeed, false otherwise.

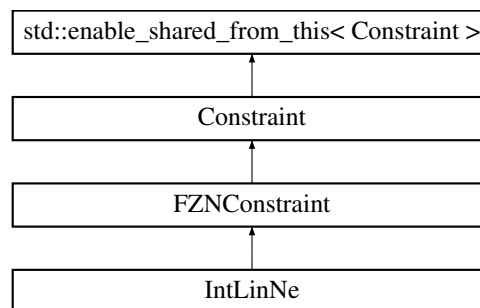
Implemented in [CudaDomain](#).

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int_domain.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int_domain.cpp`

## 6.37 IntLinNe Class Reference

Inheritance diagram for IntLinNe:



### Public Member Functions

- [IntLinNe](#) ()
- [IntLinNe](#) (std::vector< VariablePtr > vars, std::vector< std::string > args)
- void [setup](#) (std::vector< VariablePtr > vars, std::vector< std::string > args) override  
*Setup method, see [fzn\\_constraint.h](#).*
- const std::vector< VariablePtr > [scope](#) () const override
- void [consistency](#) () override  
*It performs domain consistency.*
- bool [satisfied](#) () override  
*It checks if  $x \neq y$ .*
- void [print\\_semantic](#) () const override  
*Prints the semantic of this constraint.*

### Additional Inherited Members

#### 6.37.1 Constructor & Destructor Documentation

##### 6.37.1.1 IntLinNe::IntLinNe ( )

Basic constructor.

#### Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

### 6.37.1.2 IntLinNe::IntLinNe ( std::vector< VariablePtr > vars, std::vector< std::string > args )

Basic constructor.

#### Note

this constructor implicitly calls the setup method to setup variables and arguments for this constraint.

## 6.37.2 Member Function Documentation

### 6.37.2.1 void IntLinNe::consistency ( ) [override],[virtual]

It performs domain consistency.

This function propagates only when there is just variables that is not still assigned. Otherwise it returns without any check.

Reimplemented from [FZNConstraint](#).

### 6.37.2.2 const std::vector< VariablePtr > IntLinNe::scope ( ) const [override],[virtual]

It returns the vector of (shared) pointers of all the variables involved in a given constraint (i.e., its scope).

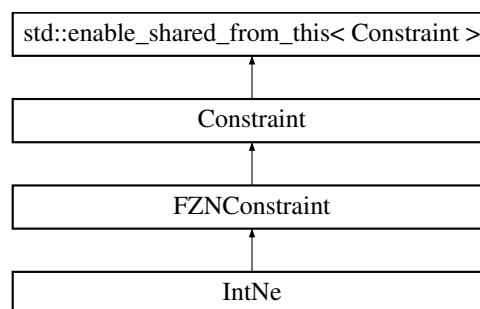
Implements [Constraint](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_lin\_ne.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_lin\_ne.cpp

## 6.38 IntNe Class Reference

Inheritance diagram for IntNe:



### Public Member Functions

- [IntNe](#) ( )
  - [IntNe](#) (std::vector< VariablePtr > vars, std::vector< std::string > args)
  - [IntNe](#) (int x, int y)
  - [IntNe](#) (IntVariablePtr x, int y)
  - [IntNe](#) (int x, IntVariablePtr y)
  - [IntNe](#) (IntVariablePtr x, IntVariablePtr y)
  - void [setup](#) (std::vector< VariablePtr > vars, std::vector< std::string > args) override
- Setup method, see [fzn\\_constraint.h](#).*

- `const std::vector< VariablePtr > scope ()` const override
- `void consistency ()` override  
*It performs domain consistency.*
- `bool satisfied ()` override  
*It checks if  $x \neq y$ .*
- `void print_semantic ()` const override  
*Prints the semantic of this constraint.*

## Additional Inherited Members

### 6.38.1 Constructor & Destructor Documentation

#### 6.38.1.1 `IntNe::IntNe ( )`

Basic constructor.

##### Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

#### 6.38.1.2 `IntNe::IntNe ( std::vector< VariablePtr > vars, std::vector< std::string > args )`

Basic constructor.

##### Note

this constructor implicitly calls the setup method to setup variables and arguments for this constraint.

#### 6.38.1.3 `IntNe::IntNe ( int x, int y )`

Basic constructor: it checks if  $x \neq y$ .

##### Parameters

<code>x</code>	an integer value.
<code>y</code>	an integer value.

#### 6.38.1.4 `IntNe::IntNe ( IntVariablePtr x, int y )`

Constructor.

##### Parameters

<code>x</code>	(pointer to) a FD variable.
<code>y</code>	an integer value.

##### Note

It subtracts the value `y` from the domain of the variable `x` if `x` has a domain defined on integers.

#### 6.38.1.5 `IntNe::IntNe ( int x, IntVariablePtr y )`

Constructor.

## Parameters

<i>x</i>	an integer value.
<i>y</i>	(pointer to) a FD variable.

## Note

It subtracts the value *x* from the domain of the variable *y* if *y* has a domain defined on integers.

6.38.1.6 IntNe::IntNe ( IntVariablePtr *x*, IntVariablePtr *y* )

Constructor.

## Parameters

<i>x</i>	(pointer to) a FD variable.
<i>y</i>	(pointer to) a FD variable.

## 6.38.2 Member Function Documentation

## 6.38.2.1 const std::vector&lt; VariablePtr &gt; IntNe::scope ( ) const [override],[virtual]

It returns the vector of (shared) pointers of all the variables involved in a given constraint (i.e., its scope).

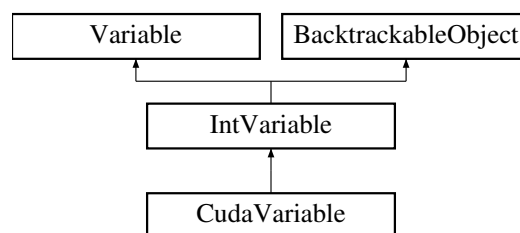
Implements [Constraint](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_ne.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_ne.cpp

## 6.39 IntVariable Class Reference

Inheritance diagram for IntVariable:



## Public Member Functions

- virtual void [set\\_domain](#) ()=0
- virtual void [set\\_domain](#) (int lw, int ub)=0
- virtual void [set\\_domain](#) (std::vector< std::vector< int > > elems)=0
- virtual void [set\\_backtrack\\_manager](#) (BacktrackManagerPtr bkt\_manager)
- EventType [get\\_event](#) () const  
*Get event on this domain.*
- void [reset\\_event](#) ()  
*Reset default event on this domain.*

- void [set\\_domain\\_type](#) (DomainType dt)
- size\_t [get\\_size](#) () const
- bool [is\\_singleton](#) () const
- bool [is\\_empty](#) () const
- virtual int [min](#) () const
- virtual int [max](#) () const
- virtual void [shrink](#) (int [min](#), int [max](#))
- virtual bool [subtract](#) (int val)
- virtual void [in\\_min](#) (int [min](#))
- virtual void [in\\_max](#) (int [max](#))
- void [set\\_backtrackable\\_id](#) () override
- void [print\\_domain](#) () const override
 

*Print domain.*
- virtual void [print](#) () const
 

*print info about the current domain*

## Protected Member Functions

- **IntVariable** (int idv)
- virtual void [notify\\_backtrack\\_manager](#) ()
- virtual void [notify\\_observers](#) ()

## Protected Attributes

- IntDomainPtr [\\_domain\\_ptr](#)
- BacktrackManagerPtr [\\_backtrack\\_manager](#)

## Additional Inherited Members

### 6.39.1 Member Function Documentation

#### 6.39.1.1 size\_t IntVariable::get\_size ( ) const [virtual]

It returns the size of the current domain.

#### Returns

the size of the current variable's domain.

Implements [Variable](#).

#### 6.39.1.2 void IntVariable::in\_max ( int *max* ) [virtual]

It updates the domain according to the maximum value.

#### Parameters

<i>max</i>	domain value.
------------	---------------

#### 6.39.1.3 void IntVariable::in\_min ( int *min* ) [virtual]

It updates the domain according to the minimum value.



## Parameters

<i>min</i>	domain value.
------------	---------------

**6.39.1.4** `bool IntVariable::is_empty ( ) const [virtual]`

It checks if the domain is empty.

## Returns

true if variable domain is empty. false otherwise.

Implements [Variable](#).

**6.39.1.5** `bool IntVariable::is_singleton ( ) const [virtual]`

It checks if the domain contains only one value.

## Returns

true if the the variable's domain is a singleton, false otherwise.

Implements [Variable](#).

**6.39.1.6** `int IntVariable::max ( ) const [virtual]`

It returns the current maximal value in the domain of this variable.

## Returns

the maximum value belonging to the domain.

## Note

the same value can be obtained by using the domain iterator.

**6.39.1.7** `int IntVariable::min ( ) const [virtual]`

It returns the current minimal value in the domain of this variable.

## Returns

the minimum value belonging to the domain.

## Note

the same value can be obtained by using the domain iterator.

**6.39.1.8** `void IntVariable::notify_backtrack_manager ( ) [protected],[virtual]`

Notifies the backtrack manager that a change happened on this variable, so the manager can manage this back-trackable object.

**6.39.1.9** `void IntVariable::notify_observers ( )` `[protected],[virtual]`

Notifies every listener which is observing any change on this variable.

**Note**

usually the store and the backtrack manager will be notified on changes on this variable.

**6.39.1.10** `void IntVariable::set_backtrack_manager ( BacktrackManagerPtr bkt_manager )` `[virtual]`

Set a backtrack manager for this backtrackable object.

**Parameters**

<i>bkt_manager</i>	a reference to the backtrack manager that will manage this backtrackable object.
--------------------	--

**6.39.1.11** `void IntVariable::set_backtrackable_id ( )` `[override],[virtual]`

Set unique id for this backtrackable object.

**Note**

the (unique) variable id is used also for the id of the backtrackable object.  
override backtrackable object methods.

Implements [BacktrackableObject](#).

**6.39.1.12** `virtual void IntVariable::set_domain ( )` `[pure virtual]`

Set domain's bounds. If no bounds are provided, an unbounded domain (int) is instantiated. If an array of elements A is provided, the function instantiates a domain D = [min A, max A], deleting all the elements d in D s.t. d does not belong to A.

Implemented in [CudaVariable](#).

**6.39.1.13** `virtual void IntVariable::set_domain ( int lw, int ub )` `[pure virtual]`

Set domain's bounds. A new domain [lw, ub] is generated.

**Parameters**

<i>lw</i>	lower bound
<i>ub</i>	upper bound

Implemented in [CudaVariable](#).

**6.39.1.14** `virtual void IntVariable::set_domain ( std::vector< std::vector< int > > elems )` `[pure virtual]`

Set domain's elements. A domain {d\_1, ..., d\_n} is generated.

**Parameters**

<i>elems</i>	vector of vectors (subsets) of domain's elements
--------------	--

**Todo** implement set of sets of elements.

Implemented in [CudaVariable](#).

**6.39.1.15** void IntVariable::set\_domain\_type ( DomainType *dt* ) [virtual]

Set domain according to the specific variable implementation.

**Note**

: different types of variable

**Parameters**

<i>dt</i>	domain type of type DomainType to set to the current variable
-----------	---

Implements [Variable](#).

**6.39.1.16** void IntVariable::shrink ( int *min*, int *max* ) [virtual]

Set domain's bounds. It updates the domain to have values only within the interval min..max.

**Note**

it does not update `_lower_bound` and `_upper_bound` here for efficiency reasons.

**Parameters**

<i>lower</i>	lower bound value
<i>upper</i>	upper bound value

**6.39.1.17** bool IntVariable::subtract ( int *val* ) [virtual]

It intersects with the domain which is a complement of the value given as input, i.e., subtract a value from the current domain.

**Parameters**

<i>val</i>	the value to subtract from the current domain
------------	---

**Returns**

true if succeed, false otherwise.

**6.39.2 Member Data Documentation****6.39.2.1** BacktrackManagerPtr IntVariable::\_backtrack\_manager [protected]

Reference to the backtrack manager that will manage the state of this [BacktrackableObject](#). This manager will be notified every time this variable changes its internal state.

**6.39.2.2** IntDomainPtr IntVariable::\_domain\_ptr [protected]

Reference to the domain of the variable. [IntDomain](#) for [IntVariable](#)

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_variable.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/int\_variable.cpp

## 6.40 Logger Class Reference

### Public Member Functions

- void **set\_out\_file** (std::string)
- void **set\_verbose** (bool)
- void **message** (std::string)  
*Print message on stdout or file (print\_message force printing)*
- void **print\_message** (std::string)
- void **log** (std::string)  
*Print log on stdout or file.*
- void **oflog** (std::string)
- void **error** (std::string)  
*Print error message on cerr (optional: **FILE** and **LINE**)*
- void **error** (std::string, const char \*)
- void **error** (std::string, const char \*, const int)

### Static Public Member Functions

- static **Logger** \* **get\_instance** (std::string log\_file="")  
*Constructor get (static) instance.*

### Protected Member Functions

- **Logger** (std::string="")

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/logger.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/logger.cpp

## 6.41 Memento Class Reference

### Protected Member Functions

- virtual void **set\_state** (MementoState \*state)
- virtual MementoState \* **get\_state** ()
- **Memento** ()

*Protected constructor.*

### Protected Attributes

- MementoState \* **\_memento\_state**

### Friends

- class **BacktrackableObject**

### 6.41.1 Member Function Documentation

6.41.1.1 `virtual MementoState* Memento::get_state ( ) [inline],[protected],[virtual]`

Get the current state saved as memento.

Returns

the current state/memento.

6.41.1.2 `virtual void Memento::set_state ( MementoState * state ) [inline],[protected],[virtual]`

Set a state as a memento object.

Parameters

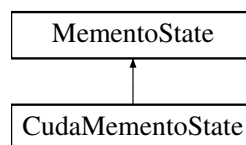
<code>state</code>	the current state representing a mememnto object.
--------------------	---

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/memento.h

## 6.42 MementoState Class Reference

Inheritance diagram for MementoState:



### Public Member Functions

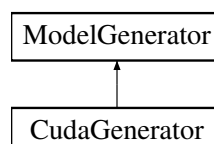
- virtual void `print` () const =0  
*Print information about this memento state.*

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/memento\_state.h

## 6.43 ModelGenerator Class Reference

Inheritance diagram for ModelGenerator:



## Public Member Functions

- virtual VariablePtr [get\\_variable](#) (TokenPtr)=0
- virtual ConstraintPtr [get\\_constraint](#) (TokenPtr)=0
- virtual SearchEnginePtr [get\\_search\\_engine](#) (TokenPtr)=0
- virtual ConstraintStorePtr [get\\_store](#) ()=0

### 6.43.1 Member Function Documentation

#### 6.43.1.1 virtual ConstraintPtr ModelGenerator::get\_constraint ( TokenPtr ) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

##### Parameters

<i>TokenPtr</i>	pointer to the token describing a constraint. If the token does not correspond to the object to instantiate, it returns nullptr.
-----------------	--

Implemented in [CudaGenerator](#).

#### 6.43.1.2 virtual SearchEnginePtr ModelGenerator::get\_search\_engine ( TokenPtr ) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

##### Parameters

<i>TokenPtr</i>	pointer to the token describing a search engine. If the token does not correspond to the object to instantiate, it returns nullptr.
-----------------	---

Implemented in [CudaGenerator](#).

#### 6.43.1.3 virtual ConstraintStorePtr ModelGenerator::get\_store ( ) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

##### Parameters

<i>TokenPtr</i>	pointer to the token describing a search engine. If the token does not correspond to the object to instantiate, it returns nullptr.
-----------------	---

Implemented in [CudaGenerator](#).

#### 6.43.1.4 virtual VariablePtr ModelGenerator::get\_variable ( TokenPtr ) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

##### Parameters

<i>TokenPtr</i>	pointer to the token describing a variable. If the token does not correspond to the object to instantiate, it returns nullptr.
-----------------	--

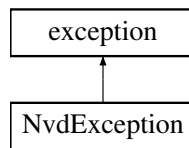
Implemented in [CudaGenerator](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/model\_generator.h

## 6.44 NvdException Class Reference

Inheritance diagram for NvdException:



### Public Member Functions

- [NvdException](#) (const char \*msg="")
- [NvdException](#) (const char \*msg, const char \*file)
- [NvdException](#) (const char \*msg, const char \*file, int line)
- virtual const char \* [what](#) () const noexcept

### Protected Attributes

- int [\\_expt\\_line](#)  
*Code line where the exception was thrown.*
- std::string [\\_expt\\_file](#)  
*Name of the file where the exception was thrown.*
- std::string [\\_expt\\_message](#)  
*Exception message.*

### 6.44.1 Constructor & Destructor Documentation

#### 6.44.1.1 NvdException::NvdException ( const char \* *msg* = " " )

Constructor.

Parameters

<i>msg</i>	the message related to the exception.
------------	---------------------------------------

#### 6.44.1.2 NvdException::NvdException ( const char \* *msg*, const char \* *file* )

Constructor.

Parameters

<i>msg</i>	the message related to the exception.
<i>file</i>	where the exception has been raised.

#### 6.44.1.3 NvdException::NvdException ( const char \* *msg*, const char \* *file*, int *line* )

Constructor.

## Parameters

<i>msg</i>	the message related to the exception.
<i>file</i>	where the exception has been raised.
<i>line</i>	of code where the exception has been raised.

## 6.44.2 Member Function Documentation

### 6.44.2.1 `const char * NvdException::what ( ) const` `[virtual],[noexcept]`

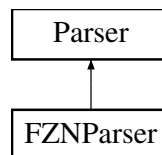
Overwrite the what method to print other information about the exception.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/nvd\_exception.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/nvd\_exception.cpp

## 6.45 Parser Class Reference

Inheritance diagram for Parser:



### Public Member Functions

- void `set_input` (std::string)  
*Set input.*
- void `add_delimiter` (std::string)  
*Add delimiter to tokenizer.*
- int `get_current_line` ()  
*Get current (parsed) line.*
- bool `is_failed` () const  
*Check whether the parser has failed.*
- virtual bool `more_tokens` ()
- virtual void `open` ()
- virtual void `close` ()
- virtual std::string `get_next_token` ()
- virtual bool `more_variables` () const =0
- virtual bool `more_constraints` () const =0
- virtual bool `more_search_engines` () const =0
- virtual TokenPtr `get_variable` ()=0
- virtual TokenPtr `get_constraint` ()=0
- virtual TokenPtr `get_search_engine` ()=0
- virtual TokenPtr `get_next_content` ()=0
- virtual void `print` () const =0  
*Print info.*



## Protected Member Functions

- [Parser](#) ()  
*Constructor.*
- **Parser** (std::string)

## Protected Attributes

- [Tokenization](#) \* [\\_tokenizer](#)  
*Tokenizer: it tokenizes lines read from the input file.*
- std::ifstream \* [\\_if\\_stream](#)  
*Input stream (from file)*
- std::string [\\_input\\_path](#)
- std::string [\\_dbg](#)
- bool [\\_open\\_file](#)
- bool [\\_open\\_first\\_time](#)
- bool [\\_more\\_tokens](#)
- bool [\\_new\\_line](#)
- bool [\\_failure](#)
- int [\\_current\\_line](#)  
*Number of lines read so far.*
- std::string [\\_delimiters](#)  
*Delimiter to use to tokenize words.*
- std::streampos [\\_curr\\_pos](#)  
*Other variables needed to move into the file.*
- std::map< size\_t, TokenPtr > [\\_map\\_tokens](#)  
*Pointers to all tokens parsed so far.*

### 6.45.1 Member Function Documentation

#### 6.45.1.1 void Parser::close ( ) [virtual]

Close the file.

#### Note

: alternating [open\(\)](#) and [close\(\)](#) the client can decided how much text has to be parsed.

#### 6.45.1.2 virtual TokenPtr Parser::get\_next\_content ( ) [pure virtual]

Give next [Token](#). A [Token](#) is built from a (string) token and represents a semantic object read from the FlatZinc model given in input. It holds other useful info related to the (string) token itself, e.g., line where the token has been found. If this function is call and no other [Token](#) is available it returns nullptr.

Implemented in [FZNParser](#).

#### 6.45.1.3 std::string Parser::get\_next\_token ( ) [virtual]

Get next token. This function returns a string corresponding to the token parsed according to the internal state of the object (i.e., pointer in the text file).

#### 6.45.1.4 virtual TokenPtr Parser::get\_variable ( ) [pure virtual]

Get methods: get variables, constraints, and the search engine. They increment the counter of available tokens. The tokens are returned in order w.r.t. their variables.

Implemented in [FZNParser](#).

#### 6.45.1.5 bool Parser::more\_tokens ( ) [virtual]

Check if the internal status has more tokens to give back to the client.

#### 6.45.1.6 virtual bool Parser::more\_variables ( ) const [pure virtual]

Get methods: more tokens of the same related type (i.e., variables, constraints, and search engine). These methods should be used together with the "get" methods.

Implemented in [FZNParser](#).

#### 6.45.1.7 void Parser::open ( ) [virtual]

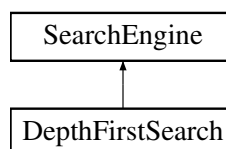
Open the file. The file is open (if not already open) and the pointer is placed on the last position read. If the file is open for the first time, the pointer is placed on the first position.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/parser.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/parser.cpp

## 6.46 SearchEngine Class Reference

Inheritance diagram for SearchEngine:



### Public Member Functions

- virtual void [set\\_debug](#) (bool debug\_on)=0
- virtual void [set\\_trail\\_debug](#) (bool debug\_on)=0
- virtual void [set\\_store](#) (ConstraintStorePtr store)=0
- virtual void [set\\_heuristic](#) (HeuristicPtr heuristic)=0
- virtual void [set\\_solution\\_manager](#) (SolutionManager \*sol\_manager)=0
- virtual void [set\\_backtrack\\_manager](#) (BacktrackManagerPtr bkt\_manager)=0
- virtual size\_t [get\\_backtracks](#) ( ) const =0
- virtual size\_t [get\\_nodes](#) ( ) const =0
- virtual size\_t [get\\_wrong\\_decisions](#) ( ) const =0
- virtual void [set\\_solution\\_limit](#) (size\_t num\_sol)=0
- virtual void [set\\_timeout\\_limit](#) (double timeout)=0
- virtual void [print\\_solution](#) ( ) const =0

*Print on standard output last solution found.*

- virtual void [print\\_all\\_solutions](#) () const =0  
*Print all solutions found so far.*
- virtual void [print\\_solution](#) (size\_t sol\_idx) const =0
- virtual std::vector< DomainPtr > [get\\_solution](#) () const =0
- virtual std::vector< DomainPtr > [get\\_solution](#) (int n\_sol) const =0
- virtual bool [label](#) (int var)=0
- virtual bool [labeling](#) ()=0
- virtual void [set\\_backtrack\\_out](#) (size\_t out\_b)=0
- virtual void [set\\_nodes\\_out](#) (size\_t out\_n)=0
- virtual void [set\\_wrong\\_decisions\\_out](#) (size\_t out\_w)=0
- virtual void [print](#) () const =0  
*Prints info about the search engine.*

## 6.46.1 Member Function Documentation

### 6.46.1.1 virtual size\_t SearchEngine::get\_backtracks ( ) const [pure virtual]

Returns the number of backtracks performed by the search.

#### Returns

the number of backtracks.

Implemented in [DepthFirstSearch](#).

### 6.46.1.2 virtual size\_t SearchEngine::get\_nodes ( ) const [pure virtual]

Returns the number of nodes visited by the search.

#### Returns

the number of visited nodes.

Implemented in [DepthFirstSearch](#).

### 6.46.1.3 virtual std::vector<DomainPtr> SearchEngine::get\_solution ( ) const [pure virtual]

Return the last solution found if any.

#### Returns

a vector of variables' domains (pointer to) Each domain is most probably a singleton and together represent a solution.

Implemented in [DepthFirstSearch](#).

### 6.46.1.4 virtual std::vector<DomainPtr> SearchEngine::get\_solution ( int n\_sol ) const [pure virtual]

Return the  $n^{\text{th}}$  solution found if any.

**Parameters**

<i>n_sol</i>	the solution to get.
--------------	----------------------

**Returns**

a vector of variables' domains (pointer to) Each domain is most probably a singleton and together represent a solution.

**Note**

The first solution has index 1.

Implemented in [DepthFirstSearch](#).

**6.46.1.5** `virtual size_t SearchEngine::get_wrong_decisions ( ) const` `[pure virtual]`

Returns the number of wrong decisions made during the search process.

**Returns**

the number of wrong decisions.

**Note**

a decision is "wrong" depending on the search engine used to explore the search space. Usually, a wrong decision is represented by a leaf of the search tree which has failed.

Implemented in [DepthFirstSearch](#).

**6.46.1.6** `virtual bool SearchEngine::label ( int var )` `[pure virtual]`

It assigns variables one by one. This function is called recursively.

**Parameters**

<i>var</i>	the index of the variable (not grounded) to assign.
------------	---

**Returns**

true if the solution was found.

Implemented in [DepthFirstSearch](#).

**6.46.1.7** `virtual bool SearchEngine::labeling ( )` `[pure virtual]`

It performs the actual search. First it sets up the internal items/attributes of search. Then, it calls the labeling function with argument specifying the index of a not grounded variable.

**Returns**

true if a solution was found.

Implemented in [DepthFirstSearch](#).

**6.46.1.8** `virtual void SearchEngine::print_solution ( size_t sol_idx ) const` `[pure virtual]`

Print on standard output a solutions represented by its index.

## Parameters

<i>sol_idx</i>	the index of the solution to print.
----------------	-------------------------------------

## Note

first solution has index 1.

Implemented in [DepthFirstSearch](#).

6.46.1.9 `virtual void SearchEngine::set_backtrack_manager ( BacktrackManagerPtr bkt_manager ) [pure virtual]`

Sets a backtrackable manager to this class.

## Parameters

<i>bkt_manager</i>	a reference to a backtrack manager.
--------------------	-------------------------------------

Implemented in [DepthFirstSearch](#).

6.46.1.10 `virtual void SearchEngine::set_backtrack_out ( size_t out_b ) [pure virtual]`

Set a maximum number of backtracks to perform during search.

## Parameters

<i>the</i>	number of backtracks to consider as a limit during the search.
------------	--

Implemented in [DepthFirstSearch](#).

6.46.1.11 `virtual void SearchEngine::set_debug ( bool debug_on ) [pure virtual]`

Set debug option.

## Parameters

<i>debug_on</i>	boolean value indicating if debug should be enabled.
-----------------	--

Implemented in [DepthFirstSearch](#).

6.46.1.12 `virtual void SearchEngine::set_heuristic ( HeuristicPtr heuristic ) [pure virtual]`

Set the heuristic to use to get the variables and the values every time a node of the search tree is explored.

## Parameters

<i>a</i>	reference to a heuristic.
----------	---------------------------

Implemented in [DepthFirstSearch](#).

6.46.1.13 `virtual void SearchEngine::set_nodes_out ( size_t out_n ) [pure virtual]`

Set a maximum number of nodes to visit during search.

## Parameters

<i>the</i>	number of nodes to visit and to be considered as a limit during the search.
------------	---

Implemented in [DepthFirstSearch](#).

6.46.1.14 `virtual void SearchEngine::set_solution_limit ( size_t num_sol ) [pure virtual]`

Set maximum number of solutions to be found.

Parameters

<i>num_sol</i>	the maximum number of solutions.
----------------	----------------------------------

Note

-1 for finding all solutions.

Implemented in [DepthFirstSearch](#).

6.46.1.15 `virtual void SearchEngine::set_solution_manager ( SolutionManager * sol_manager ) [pure virtual]`

Set a solution manager for this search engine.

Parameters

<i>a</i>	reference to a solution manager.
----------	----------------------------------

Implemented in [DepthFirstSearch](#).

6.46.1.16 `virtual void SearchEngine::set_store ( ConstraintStorePtr store ) [pure virtual]`

Set a reference to a constraint store. The given store will be used to evaluate the constraints.

Parameters

<i>a</i>	reference to a constraint store.
----------	----------------------------------

Implemented in [DepthFirstSearch](#).

6.46.1.17 `virtual void SearchEngine::set_timeout_limit ( double timeout ) [pure virtual]`

Imposes a timeout limit.

Parameters

<i>timeout</i>	timeout limit.
----------------	----------------

Note

-1 for no timeout.

Implemented in [DepthFirstSearch](#).

6.46.1.18 `virtual void SearchEngine::set_trail_debug ( bool debug_on ) [pure virtual]`

Set debug with trail option. If enabled it prints debug and trail stack behaviours.

## Parameters

<i>debug_on</i>	boolean value indicating if debug should be enabled.
-----------------	--

Implemented in [DepthFirstSearch](#).

6.46.1.19 virtual void `SearchEngine::set_wrong_decisions_out ( size_t out_w )` [pure virtual]

Set a maximum number of wrong decisions to make before exiting the search phase.

## Parameters

<i>the</i>	number of wrong decisions to set as a limit during the search.
------------	--

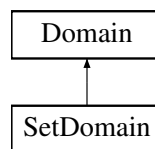
Implemented in [DepthFirstSearch](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/search\_engine.h

## 6.47 SetDomain Class Reference

Inheritance diagram for SetDomain:



### Public Member Functions

- virtual void [set\\_values](#) (std::vector< int > elems)
- virtual std::vector< int > [get\\_values](#) () const
- DomainPtr [clone](#) () const  
*Clone the current domain and returns a pointer to it.*
- EventType [get\\_event](#) () const
- void [reset\\_event](#) ()
- size\_t [get\\_size](#) () const  
*Returns the size of the domain.*
- bool [is\\_empty](#) () const  
*Returns true if the domain is empty.*
- bool [is\\_singleton](#) () const  
*Returns true if the domain has only one element.*
- bool [is\\_numeric](#) () const  
*Returns true if this is a numeric finite domain.*
- std::string [get\\_string\\_representation](#) () const  
*Get string rep. of this domain.*
- void [print](#) () const  
*Print info about the domain.*

### Protected Member Functions

- DomainPtr [clone\\_impl](#) () const

## Protected Attributes

- `std::vector< int > _d_elements`

## Additional Inherited Members

### 6.47.1 Member Function Documentation

#### 6.47.1.1 EventType SetDomain::get\_event ( ) const [virtual]

Get event on this domain

**Todo** implement this function

Implements [Domain](#).

#### 6.47.1.2 std::vector< int > SetDomain::get\_values ( ) const [virtual]

Get a vector containing the current values contained in the domain.

Returns

the current elements in the domain

#### 6.47.1.3 void SetDomain::reset\_event ( ) [virtual]

Sets the no event on this domain.

Note

No event won't trigger any propagation on this domain.

Implements [Domain](#).

#### 6.47.1.4 void SetDomain::set\_values ( std::vector< int > elems ) [virtual]

Set bounds and perform some consistency checking. It throws "no solutions" if consistency checking fails.

Parameters

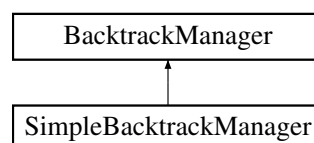
<i>elems</i>	vector of domain's elements
--------------	-----------------------------

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/set\_domain.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/set\_domain.cpp

## 6.48 SimpleBacktrackManager Class Reference

Inheritance diagram for SimpleBacktrackManager:





## Public Member Functions

- void [attach\\_backtracable](#) ([BacktrackableObject](#) \*bkt\_obj)
- void [detach\\_backtracable](#) (size\_t bkt\_id)
- size\_t [get\\_level](#) () const
- void [add\\_changed](#) (size\_t idx)
- void [set\\_level](#) (size\_t lvl) override
- void [force\\_storage](#) () override
- void [remove\\_level](#) (size\_t lvl) override
- void [remove\\_until\\_level](#) (size\_t lvl) override
- size\_t [number\\_backtracable](#) () const override
- size\_t [number\\_changed\\_backtracable](#) () const override
- void [print](#) () const override

*Print information about this simple backtrack manager.*

## Protected Attributes

- std::string [\\_dbg](#)  
*Debug info.*
- size\_t [\\_current\\_level](#)  
*Current active level in the manager.*
- std::map< size\_t, [BacktrackableObject](#) \* > [\\_backtrackable\\_objects](#)
- std::set< size\_t > [\\_changed\\_backtrackables](#)
- std::stack< std::pair< size\_t, [Memento](#) \* > > > [\\_trail\\_stack](#)
- std::stack< std::vector< size\_t > > [\\_trail\\_stack\\_info](#)

### 6.48.1 Member Function Documentation

#### 6.48.1.1 void SimpleBacktrackManager::add\_changed ( size\_t idx ) [virtual]

Informs the manager that a given backtrackable object has changed at a given level.

##### Parameters

<i>idx</i>	the (unique) id of the backtrackable object which is changed.
------------	---

##### Note

only object already registered with this manager can be restored later.

Implements [BacktrackManager](#).

#### 6.48.1.2 void SimpleBacktrackManager::attach\_backtracable ( BacktrackableObject \* bkt\_obj ) [virtual]

Register a backtrackable object to this manager using the unique id of the backtrackable object.

##### Parameters

<i>bkt_obj</i>	a reference to a backtrackable object.
----------------	--

Implements [BacktrackManager](#).

**6.48.1.3** `void SimpleBacktrackManager::detach_backtracable ( size_t bkt_id ) [virtual]`

Detaches a backtrackable object from this manager, so its state won't be restored anymore.

Parameters

<i>bkt_id</i>	the id of the backtrackable object to detach.
---------------	---

Implements [BacktrackManager](#).

**6.48.1.4** `void SimpleBacktrackManager::force_storage ( ) [override],[virtual]`

Forces the storage of all the backtrackable objects attached to this manager (at next `set_level` call), no matter if a backtrackable object has been modified or not.

Implements [BacktrackManager](#).

**6.48.1.5** `size_t SimpleBacktrackManager::get_level ( ) const [virtual]`

Get the current active level.

Returns

current active level in the manager.

Implements [BacktrackManager](#).

**6.48.1.6** `size_t SimpleBacktrackManager::number_backtracable ( ) const [override],[virtual]`

Returns the number of backtrackable objects attached to this backtrack manager.

Returns

number of objects attached to this manager.

Implements [BacktrackManager](#).

**6.48.1.7** `size_t SimpleBacktrackManager::number_changed_backtracable ( ) const [override],[virtual]`

Returns the number of changed backtrackable objects from last call to `set_level` in this backtrack manager.

Returns

number of changed objects.

Implements [BacktrackManager](#).

**6.48.1.8** `void SimpleBacktrackManager::remove_level ( size_t lvl ) [override],[virtual]`

Removes a level. It performs a backtrack from that level.

## Parameters

<i>lvl</i>	the level which is being removed.
------------	-----------------------------------

Implements [BacktrackManager](#).

**6.48.1.9** void SimpleBacktrackManager::remove\_until\_level ( size\_t *lvl* ) [override],[virtual]

Removes all levels until the one given as input. It performs backtrack until the level given as input.

## Parameters

<i>lvl</i>	the level to backtrack to.
------------	----------------------------

Implements [BacktrackManager](#).

**6.48.1.10** void SimpleBacktrackManager::set\_level ( size\_t *lvl* ) [override],[virtual]

Specifies the level which should become the active one in the manager.

## Parameters

<i>lvl</i>	the active level at which the changes will be recorded.
------------	---

Implements [BacktrackManager](#).

## 6.48.2 Member Data Documentation

**6.48.2.1** std::map< size\_t, BacktrackableObject \* > SimpleBacktrackManager::\_backtrackable\_objects  
[protected]

Ordered list of backtrackable objects that are subjects of this [BacktrackManager](#) observer.

**6.48.2.2** std::set< size\_t > SimpleBacktrackManager::\_changed\_backtrackables [protected]

Set of changed backtrackable objects. When the set\_level method is called, the objects in this list will be considered for saving their memento objects (i.e., their state).

**6.48.2.3** std::stack< std::pair < size\_t, std::vector< std::pair < size\_t, Memento \* > > > > SimpleBacktrackManager::\_trail\_stack [protected]

Stack of list of Mementos to restore when the method remove\_level is invoked. The states of the backtrackable objects will be re-stored from here. Each object in the trail stack is a pair where the first element represents the level in which the second element (pairs of backtrackable object and memento objects) are stored. For example, at a given level: < level, [ (id\_1, Memento\_1), (id\_2, Memento\_2), ... ] >

**6.48.2.4** std::stack< std::vector< size\_t > > SimpleBacktrackManager::\_trail\_stack\_info [protected]

Stack used to store auxiliary information for each level of the trail stack. Using this stack the backtrack process can be speeded up re-setting only the most memento of each backtrackable object.

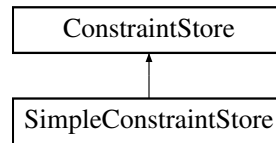
**Todo** implement this functionality.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_backtrack\_manager.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_backtrack\_manager.cpp

## 6.49 SimpleConstraintStore Class Reference

Inheritance diagram for SimpleConstraintStore:



### Public Member Functions

- [SimpleConstraintStore](#) ()
- void [fail](#) ()
- void [sat\\_check](#) (bool sat\_check=true)
- void [add\\_changed](#) (std::vector< size\_t > &c\_id, EventType event)
- void [impose](#) (ConstraintPtr c)
- bool [consistency](#) ()
- [Constraint](#) \*const [getConstraint](#) ()
- void [clear\\_queue](#) ()
- size\_t [num\\_constraints](#) () const
- size\_t [num\\_constraints\\_to\\_reevaluate](#) () const
- void [print](#) () const

*Print information about this simple constraint store.*

### Protected Member Functions

- virtual void [handle\\_failure](#) ()  
*Handle a failure state.*
- virtual void [add\\_changed](#) (size\_t c\_id, EventType event)  
*Add a single constraint for re-evaluation.*

### Protected Attributes

- std::string [\\_dbg](#)  
*Debug info.*
- std::map< size\_t, ConstraintPtr > [\\_lookup\\_table](#)
- std::set< size\_t > [\\_constraint\\_queue](#)
- [Constraint](#) \* [\\_constraint\\_to\\_reevaluate](#)  
*Current constraint to reevaluate.*
- size\_t [\\_constraint\\_queue\\_size](#)  
*Number of constraints in the \_constraint\_queue.*
- size\_t [\\_number\\_of\\_constraints](#)  
*Number of constraints imposed into the store.*
- bool [\\_satisfiability\\_check](#)
- bool [\\_failure](#)

### 6.49.1 Constructor & Destructor Documentation

#### 6.49.1.1 SimpleConstraintStore::SimpleConstraintStore ( )

Default constructor. It initializes the internal data structures of this constraint store.

## 6.49.2 Member Function Documentation

**6.49.2.1** `void SimpleConstraintStore::add_changed ( std::vector< size_t > & c_id, EventType event )` [virtual]

It adds the constraints given in input to the queue of constraint to re-evaluate.

Parameters

<i>c_id</i>	the vector of constraints ids to re-evaluate.
<i>event</i>	the event that has triggered the re-evaluation of the given list of constraints.

Note

only constraints that have been previously attached/imposed to this constraint store will be re-evaluated.

Implements [ConstraintStore](#).

**6.49.2.2** `void SimpleConstraintStore::clear_queue ( )` [virtual]

Clears the queue of constraints to re-evaluate. It can be used when implementing different scheme of constraint propagation.

Implements [ConstraintStore](#).

**6.49.2.3** `bool SimpleConstraintStore::consistency ( )` [virtual]

Computes the consistency function. This function propagates the constraints that are in the constraint queue until the queue is empty.

Returns

true if all propagate constraints are consistent, false otherwise.

Implements [ConstraintStore](#).

**6.49.2.4** `void SimpleConstraintStore::fail ( )` [virtual]

Informs the constraint store that something bad happened somewhere else. This forces the store to clean up everything and exit as soon as possible without re-evaluating any constraint.

Implements [ConstraintStore](#).

**6.49.2.5** `Constraint *const SimpleConstraintStore::getConstraint ( )` [virtual]

Returns a constraint that is scheduled for re-evaluation. The basic implementation is first-in-first-out. The constraint is hence remove from the constraint queue, since it is assumed that it will be re-evaluated right away.

Returns

a const pointer to a constraint to re-evaluate.

Implements [ConstraintStore](#).

**6.49.2.6** `void SimpleConstraintStore::impose ( ConstraintPtr c )` [virtual]

Imposes a constraint to the store. The constraint is added to the list of constraints in this constraint store as well as to the queue of constraint to re-evaluate next call to consistency. Most probably this function is called every time a new constraint is instantiated.

## Parameters

<i>c</i>	the constraint to impose in this constraint store.
----------	--

## Note

if *c* is already in the list of constraints in this constraint store, it won't be added again nor re-evaluated.

Implements [ConstraintStore](#).

**6.49.2.7** `size_t SimpleConstraintStore::num_constraints ( ) const` [virtual]

Returns the total number of constraints in this constraint store.

Implements [ConstraintStore](#).

**6.49.2.8** `size_t SimpleConstraintStore::num_constraints_to_reevaluate ( ) const` [virtual]

Returns the number of constraints to re-evaluate.

## Returns

number of constraints to re-evaluate.

Implements [ConstraintStore](#).

**6.49.2.9** `void SimpleConstraintStore::sat_check ( bool sat_check =true )` [virtual]

Sets the satisfiability check during constraint propagation. This check increases the time spent for consistency but reduces the total execution time.

## Parameters

<i>sat_check</i>	boolean value representing whether or not the satisfiability check should be performed (default: true).
------------------	---

Implements [ConstraintStore](#).

### 6.49.3 Member Data Documentation

**6.49.3.1** `std::set< size_t > SimpleConstraintStore::_constraint_queue` [protected]

Stores the constraints for which reevaluation is needed. It represents the `constraint_queue`. It does not register constraints that are already in the constraint queue.

## Note

there is only a queue in this simple constraint store. Other implementations may consider to use multiple constraint queue (e.g., one for each domains'event).

**6.49.3.2** `bool SimpleConstraintStore::_failure` [protected]

Keeps track whether some failure happened during some operations on this constraint store.

6.49.3.3 `std::map< size_t, ConstraintPtr > SimpleConstraintStore::_lookup_table` [protected]

Mapping between constraints' ids and constraints' pointer. Any new constraint imposed into the store is stored here.

6.49.3.4 `bool SimpleConstraintStore::_satisfiability_check` [protected]

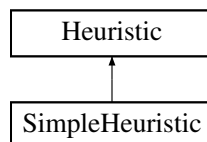
States whether the satisfiability check should be performed or not (default: true).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_constraint\_store.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_constraint\_store.cpp

## 6.50 SimpleHeuristic Class Reference

Inheritance diagram for SimpleHeuristic:



### Public Member Functions

- `SimpleHeuristic` (`std::vector< Variable * > vars`, `VariableChoiceMetric * var_cm`, `ValueChoiceMetric * val_cm`)
- `Variable * get_choice_variable` (`int idx`)
- `int get_choice_value` ()
- `void print` () const

*Print info about this heuristic.*

### Protected Attributes

- `std::vector< Variable * > _fd_variables`
- `VariableChoiceMetric * _variable_metric`
- `ValueChoiceMetric * _value_metric`

### 6.50.1 Constructor & Destructor Documentation

6.50.1.1 `SimpleHeuristic::SimpleHeuristic ( std::vector< Variable * > vars, VariableChoiceMetric * var_cm, ValueChoiceMetric * val_cm )`

Constructor, defines a new simple heuristic given the metrics for selecting the next variable to label and the value to assign to such variable.

Parameters

<code>vars</code>	a vector of pointer to variables to label.
-------------------	--

<i>var_cm</i>	the variable metric used to select the next variable to label.
<i>val_cm</i>	the value metric used to select the next value to assign to the selected variable.

**Note**

if the variable metric is a nullptr, the next variable to label is the first non-ground variable.

**6.50.2 Member Function Documentation****6.50.2.1** `int SimpleHeuristic::get_choice_value ( )` `[virtual]`

Returns the next value to assign to the variable selected by this heuristic.

**Returns**

the value to assign to the selected variable.

Implements [Heuristic](#).

**6.50.2.2** `Variable * SimpleHeuristic::get_choice_variable ( int idx )` `[virtual]`

Gets next variable to label according to the [VariableChoiceMetric](#).

**Parameters**

<i>idx</i>	the index of the last variable returned by this heuristic.
------------	--

**Returns**

a pointer to the next variable to label.

Implements [Heuristic](#).

**6.50.3 Member Data Documentation****6.50.3.1** `std::vector< Variable* > SimpleHeuristic::_fd_variables` `[protected]`

The array of (pointers to) variables used to store the references and hence to select the next variable to label according to the heuristic parameter specified as input.

**6.50.3.2** `ValueChoiceMetric* SimpleHeuristic::_value_metric` `[protected]`

The metric used to select the next value to assign to the selected variable.

**6.50.3.3** `VariableChoiceMetric* SimpleHeuristic::_variable_metric` `[protected]`

The metric used to select the next variable to label.

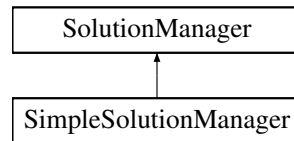
The documentation for this class was generated from the following files:

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple_heuristic.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple_heuristic.cpp`



## 6.51 SimpleSolutionManager Class Reference

Inheritance diagram for SimpleSolutionManager:



### Public Member Functions

- [SimpleSolutionManager](#) ()  
*Basic constructor.*
- [SimpleSolutionManager](#) (std::vector< [Variable](#) \* > &vars)
- void [set\\_variables](#) (std::vector< [Variable](#) \* > &vars) override
- void [print\\_solution](#) () override
- size\_t [number\\_of\\_solutions](#) () override
- std::string [get\\_solution](#) () const override
- std::string [get\\_solution](#) (size\_t sol\_idx) const override
- std::vector< std::string > [get\\_all\\_solutions](#) () const override
- void [set\\_solution\\_limit](#) (int n\_sol) override
- bool [notify](#) () override
- void [print\\_variables](#) () override  
*Print current variables' domains.*
- void [print](#) () const override  
*Print information about this simple solution manager.*

### Protected Attributes

- bool [\\_find\\_all\\_solutions](#)  
*States wheter all solutions must be find or not.*
- size\_t [\\_max\\_number\\_of\\_solutions](#)
- size\_t [\\_number\\_of\\_solutions](#)  
*Stores the number of solutions found so far.*
- std::map< int, [Variable](#) \* > [\\_variables](#)
- std::vector< std::string > [\\_solution\\_strings](#)

#### 6.51.1 Constructor & Destructor Documentation

##### 6.51.1.1 SimpleSolutionManager::SimpleSolutionManager ( std::vector< [Variable](#) \* > &vars )

Constructor. It creates a new simple solution manager attached to the given list of variables.

Parameters

<a href="#">vars</a>	a vector of references to variables.
----------------------	--------------------------------------

## 6.51.2 Member Function Documentation

**6.51.2.1** `std::vector< std::string > SimpleSolutionManager::get_all_solutions ( ) const` `[override],[virtual]`

Get the all solutions found so far.

Returns

a vector of strings representing all solutions found so far.

Implements [SolutionManager](#).

**6.51.2.2** `std::string SimpleSolutionManager::get_solution ( ) const` `[override],[virtual]`

Get the last solution found.

Returns

a string representing the last solution found.

Implements [SolutionManager](#).

**6.51.2.3** `std::string SimpleSolutionManager::get_solution ( size_t sol_idx ) const` `[override],[virtual]`

Get the solution identified by its index.

Parameters

<i>sol_idx</i>	the index of the required solution.
----------------	-------------------------------------

Returns

a string representing the required solution.

Note

first solution has index 1.

Implements [SolutionManager](#).

**6.51.2.4** `bool SimpleSolutionManager::notify ( )` `[override],[virtual]`

Increases the number of solutions found so far and computes the current solution (also storing it). States whether another solution is required by this solution manager in order to reach the total number of solutions.

Returns

true if no more solutions are required, false otherwise.

Implements [SolutionManager](#).

**6.51.2.5** `size_t SimpleSolutionManager::number_of_solutions ( )` `[override],[virtual]`

Returns the number of solutions found so far.

Returns

the number of solutions.

Implements [SolutionManager](#).

6.51.2.6 `void SimpleSolutionManager::print_solution ( ) [override],[virtual]`

Prints on standard output the last solution found.

**Note**

a solution is represented by the current values assigned to the variables attached to this solution manager.

Implements [SolutionManager](#).

6.51.2.7 `void SimpleSolutionManager::set_solution_limit ( int n_sol ) [override],[virtual]`

Sets a maximum number of solutions.

**Parameters**

<i>n_sol</i>	the number of solutions to compute.
--------------	-------------------------------------

**Note**

-1 stands for "find all solutions".

Implements [SolutionManager](#).

6.51.2.8 `void SimpleSolutionManager::set_variables ( std::vector< Variable * > & vars ) [override],[virtual]`

Set the list of variables for which a solution is required.

**Parameters**

<i>vars</i>	a vector of references to variables.
-------------	--------------------------------------

Implements [SolutionManager](#).

## 6.51.3 Member Data Documentation

6.51.3.1 `size_t SimpleSolutionManager::_max_number_of_solutions [protected]`

Stores the maximum number of solutions handled by this solution manager.

**Note**

default value is 1;  
if it is set to -1, all solutions are handled.

6.51.3.2 `std::vector< std::string > SimpleSolutionManager::_solution_strings [protected]`

Store the string representations of the solutions found so far.

6.51.3.3 `std::map< int, Variable * > SimpleSolutionManager::_variables [protected]`

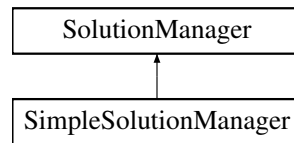
Stores the ordered list of variables that represent a solution. The order is given by variables' ids.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_solution\_manager.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/simple\_solution\_manager.cpp

## 6.52 SolutionManager Class Reference

Inheritance diagram for SolutionManager:



### Public Member Functions

- virtual void [set\\_variables](#) (std::vector< [Variable](#) \* > &vars)=0
- virtual void [print\\_solution](#) ()=0
- virtual size\_t [number\\_of\\_solutions](#) ()=0
- virtual std::string [get\\_solution](#) () const =0
- virtual std::string [get\\_solution](#) (size\_t sol\_idx) const =0
- virtual std::vector< std::string > [get\\_all\\_solutions](#) () const =0
- virtual void [set\\_solution\\_limit](#) (int n\_sol)=0
- virtual bool [notify](#) ()=0
- virtual void [print\\_variables](#) ()=0  
*Print current variables' domains.*
- virtual void [print](#) () const =0  
*Print information about this solution manager.*

### 6.52.1 Member Function Documentation

**6.52.1.1** virtual std::vector< std::string > [SolutionManager::get\\_all\\_solutions](#) ( ) const [pure virtual]

Get the all solutions found so far.

#### Returns

a vector of strings representing all solutions found so far.

Implemented in [SimpleSolutionManager](#).

**6.52.1.2** virtual std::string [SolutionManager::get\\_solution](#) ( ) const [pure virtual]

Get the last solution found.

#### Returns

a string representing the last solution found.

Implemented in [SimpleSolutionManager](#).

**6.52.1.3** virtual std::string [SolutionManager::get\\_solution](#) ( size\_t sol\_idx ) const [pure virtual]

Get the solution identified by its index.

## Parameters

<i>sol_idx</i>	the index of the required solution.
----------------	-------------------------------------

## Returns

a string representing the required solution.

## Note

first solution has index 1.

Implemented in [SimpleSolutionManager](#).

#### 6.52.1.4 virtual bool SolutionManager::notify ( ) [pure virtual]

Increases the number of solutions found so far and computes the current solution (also storing it). States whether another solution is required by this solution manager in order to reach the total number of solutions.

## Returns

true no more solutions are required, false otherwise.

Implemented in [SimpleSolutionManager](#).

#### 6.52.1.5 virtual size\_t SolutionManager::number\_of\_solutions ( ) [pure virtual]

Returns the number of solutions found so far.

## Returns

the number of solutions.

Implemented in [SimpleSolutionManager](#).

#### 6.52.1.6 virtual void SolutionManager::print\_solution ( ) [pure virtual]

Prints the last solution found on standard output.

## Note

a solution is represented by the current values assigned to the variables attached to this solution manager.

Implemented in [SimpleSolutionManager](#).

#### 6.52.1.7 virtual void SolutionManager::set\_solution\_limit ( int *n\_sol* ) [pure virtual]

Sets a maximum number of solutions.

## Parameters

<i>n_sol</i>	the number of solutions to compute.
--------------	-------------------------------------

## Note

-1 stands for "find all solutions".

Implemented in [SimpleSolutionManager](#).

6.52.1.8 `virtual void SolutionManager::set_variables ( std::vector< Variable * > & vars )` [pure virtual]

Set the list of variables for which a solution is required.

## Parameters

<i>vars</i>	a vector of references to variables.
-------------	--------------------------------------

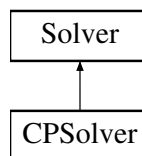
Implemented in [SimpleSolutionManager](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/solution\_manager.h

## 6.53 Solver Class Reference

Inheritance diagram for Solver:



### Public Member Functions

- virtual void [add\\_model](#) ([CPModel](#) \*model)=0
- virtual void [remove\\_model](#) (int model\_idx)=0
- virtual [CPModel](#) \* [get\\_model](#) (int model\_idx) const =0
- virtual void [customize](#) (const [InputData](#) &i\_data, int model\_idx=0)=0
- virtual void [run](#) ()=0
- virtual void [run](#) (int model\_idx)=0
- virtual int [num\\_models](#) () const =0
- virtual int [num\\_solved\\_models](#) () const =0
- virtual int [sat\\_models](#) () const =0
- virtual int [unsat\\_models](#) () const =0
- virtual void [print](#) () const =0

*Print information about this solver.*

### 6.53.1 Member Function Documentation

#### 6.53.1.1 virtual void Solver::add\_model ( [CPModel](#) \* *model* ) [pure virtual]

Add a model to the solver.

## Parameters

<i>model</i>	the reference to the (CP) model to add to the solver.
--------------	---

## Note

a solver can hold several models and decide both the model to run and the order in which run each model.

Implemented in [CPSolver](#).

#### 6.53.1.2 virtual void Solver::customize ( const [InputData](#) & *i\_data*, int *model\_idx* = 0 ) [pure virtual]

Further customizes a given model (identified by its index) with user options.

## Parameters

<i>i_data</i>	a reference to a <code>input_data</code> class where options are retrieved.
<i>model_idx</i>	the index of the model to customize (default: 0, i.e., first model).

Implemented in [CPSolver](#).

6.53.1.3 `virtual CPMModel* Solver::get_model ( int model_idx ) const` [pure virtual]

Returns a reference to model.

## Parameters

<i>model_idx</i>	the index of the model to return ( <code>model_idx = 0</code> means first model).
------------------	---

Implemented in [CPSolver](#).

6.53.1.4 `virtual int Solver::num_models ( ) const` [pure virtual]

Returns the number of models that are managed by this solver.

## Returns

the number of models managed by this solver.

Implemented in [CPSolver](#).

6.53.1.5 `virtual int Solver::num_solved_models ( ) const` [pure virtual]

Returns the current number of runned models.

## Returns

the number of models for which the run function has been called.

Implemented in [CPSolver](#).

6.53.1.6 `virtual void Solver::remove_model ( int model_idx )` [pure virtual]

Removes a model actually destroying it.

## Parameters

<i>model_idx</i>	the index of the model to destroy, ( <code>model_idx = 0</code> means first model).
------------------	---

Implemented in [CPSolver](#).

6.53.1.7 `virtual void Solver::run ( )` [pure virtual]

It runs the solver in order to find a solution, the best solutions or other solutions for all the models given to the solver.

Implemented in [CPSolver](#).

6.53.1.8 `virtual void Solver::run ( int model_idx )` [pure virtual]

It runs the solver in order to find a solution, the best solutions or other solutions for the model specified by its index.



## Parameters

<code>model_idx</code>	the index of the model to solve ( <code>model_idx = 0</code> means first model).
------------------------	--

Implemented in [CPSolver](#).

#### 6.53.1.9 virtual int Solver::sat\_models ( ) const [pure virtual]

Returns the number of models for which a solution has been found (out of the number of solved models).

## Returns

the number of models for which a solution has been found.

Implemented in [CPSolver](#).

#### 6.53.1.10 virtual int Solver::unsat\_models ( ) const [pure virtual]

Returns the number of unsatisfiable models, i.e., the number of models with no solutions among those that have been solved so far.

## Returns

the number of unsatisfiable models.

Implemented in [CPSolver](#).

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/solver.h`

## 6.54 Statistics Class Reference

### Public Member Functions

- void [set\\_timer](#) ()  
*Set timer (starts "watching" the running time)*
- void [set\\_timer](#) (int tt)
- void [stopwatch](#) (int tt=T\_GENERAL)
- void [stopwatch\\_and\\_add](#) (int tt=T\_GENERAL)
- double [get\\_timer](#) (int tt=T\_GENERAL)
- virtual void [print](#) () const  
*Print info about statistics on the program.*

### Static Public Member Functions

- static [Statistics](#) \* [get\\_instance](#) ()  
*Get (static) instance (singleton) of [Statistics](#).*

### Static Public Attributes

- static constexpr int **T\_GENERAL** = 0
- static constexpr int **T\_SEARCH** = 1
- static constexpr int **T\_FIRST\_SOL** = 2
- static constexpr int **T\_PREPROCESS** = 3
- static constexpr int **T\_FILTERING** = 4
- static constexpr int **T\_ALL** = 5

## Protected Attributes

- `std::string _dbg`  
*Debug string info.*
- `timeval _time_stats`
- `double _time_start`
- `double _time [MAX_T_TYPE]`  
*Computational times are recorded here.*
- `bool _stop_watch [MAX_T_TYPE]`  
*States if a watching has been stopped for a given computation.*

## Static Protected Attributes

- `static constexpr double USEC = 1000000.0`  
*USEC unit.*
- `static constexpr int MAX_T_TYPE = 10`  
*Max size of the array of times.*

### 6.54.1 Member Function Documentation

#### 6.54.1.1 `double Statistics::get_timer ( int tt = T_GENERAL )`

Get the value of the running time in seconds.

##### Parameters

<code>tt</code>	describes which kind of computation time must be returned,
-----------------	--

##### Returns

the computational time related to `tt` in seconds.

#### 6.54.1.2 `void Statistics::set_timer ( int tt )`

Set timer for a given computation which will be observed.

##### Parameters

<code>tt</code>	describes which kind of computation will be observed.
-----------------	---

#### 6.54.1.3 `void Statistics::stopwatch ( int tt = T_GENERAL )`

Stop watching the running time.

##### Parameters

<code>tt</code>	describes which kind of computation has been observed.
-----------------	--

#### 6.54.1.4 `void Statistics::stopwatch_and_add ( int tt = T_GENERAL )`

Stop watching the running time and add the time to the previous times watched for `tt`.

## Parameters

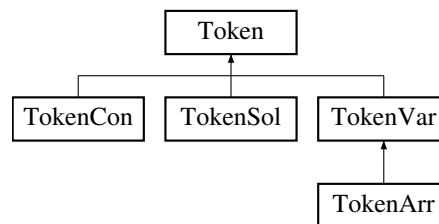
<i>tt</i>	describes which kind of computation has been observed.
-----------	--

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/statistics.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/statistics.cpp

## 6.55 Token Class Reference

Inheritance diagram for Token:



### Public Member Functions

- **Token** (TokenType)
- int **get\_id** () const
- void **set\_type** (TokenType)
- TokenType **get\_type** () const
- virtual void **print** () const  
*Print info about the token.*

### Protected Attributes

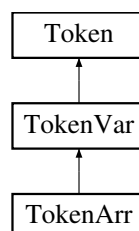
- std::string **\_dbg**
- TokenType **\_tkn\_type**

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token.cpp

## 6.56 TokenArr Class Reference

Inheritance diagram for TokenArr:



## Public Member Functions

- void **set\_size\_arr** (int)
- int **get\_size\_arr** () const
- void **set\_array\_bounds** (int lw, int up)
- int **get\_lw\_bound** () const
- int **get\_up\_bound** () const
- int **get\_lower\_var** () const
- int **get\_upper\_var** () const
- bool **is\_var\_in** (int var) const
- bool **is\_var\_in** (std::string) const
- void **set\_output\_arr** ()  
*Identifies the current variable array as a support variable array.*
- bool **is\_output\_arr** () const
- void **set\_support\_elements** (std::string elem\_str)  
*Set a string representing the elements of a support array.*
- std::string **get\_support\_elements** () const  
*Returns a string describing the elements of a support array.*
- void **print** () const  
*Print info methods.*

## Additional Inherited Members

### 6.56.1 Member Function Documentation

#### 6.56.1.1 int TokenArr::get\_lower\_var ( ) const

Variables (idx) within the array. The index is given w.r.t. the global index of parsed tokens so far.

##### Returns

the lower idx of variable within the array

#### 6.56.1.2 int TokenArr::get\_upper\_var ( ) const

Variables (idx) within the array. The index is given w.r.t. the global index of parsed tokens so far.

##### Returns

the higher idx of variable within the array

#### 6.56.1.3 bool TokenArr::is\_var\_in ( int var ) const

Check whether a given variable (idx) is indexed by the array (i.e., is within the array).

##### Note

: check is performed w.r.t. both the variable string identifier (e.g., a[i]) and its global id.

## Parameters

<i>var</i>	the variable to check membership
------------	----------------------------------

## Returns

true if *var* is in the current array, false otherwise

6.56.1.4 void TokenArr::set\_array\_bounds ( int *lw*, int *up* )

Array set and info. For example, array [1..30] of ... `get_lw_bound -> 1` `get_lw_bound -> 30` It sets the bounds of the array.

## Parameters

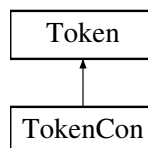
<i>lw</i>	lower bound
<i>up</i>	upper bound

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_arr.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_arr.cpp

## 6.57 TokenCon Class Reference

Inheritance diagram for TokenCon:



### Public Member Functions

- void [set\\_con\\_id](#) (std::string)
  - Set method constraint id (i.e., constraint's name).*
- std::string [get\\_con\\_id](#) () const
  - Get the string representing the constraint's name.*
- void [add\\_expr](#) (std::string str)
- int [get\\_num\\_expr](#) () const
  - Get the number of parameters needed by the constraint.*
- std::string [get\\_expr](#) (int) const
- const std::vector< std::string > [get\\_expr\\_array](#) ()
- const std::vector< std::string > [get\\_expr\\_elements\\_array](#) ()
- const std::vector< std::string > [get\\_expr\\_var\\_elements\\_array](#) ()
- const std::vector< std::string > [get\\_expr\\_not\\_var\\_elements\\_array](#) ()
- virtual void [print](#) () const
  - Print info methods.*

## Protected Attributes

- `std::string _con_id`  
*Info about the constraint.*
- `std::vector< std::string > _exprs`  
*Parameters involved in the constraint.*

### 6.57.1 Member Function Documentation

#### 6.57.1.1 `void TokenCon::add_expr ( std::string str )`

Add expression (parameters) to the token that identifies the parsed constraint. For example, constraint `int_↵ne(magic[1], magic[2])` expression = "magic[1]" and "magic[2]"

##### Parameters

<i>str</i>	string representing the expression.
------------	-------------------------------------

#### 6.57.1.2 `std::string TokenCon::get_expr ( int idx ) const`

Get the string represeting the ith expression that defines the constraint.

##### Parameters

<i>idx</i>	index of the expression to return
------------	-----------------------------------

##### Returns

return the idx<sup>th</sup> expression

#### 6.57.1.3 `const std::vector< std::string > TokenCon::get_expr_array ( )`

Return an array containing all the (string) expressions that define the current constraint.

##### Returns

a vector of strings representing the expressions defining this constraint.

#### 6.57.1.4 `const std::vector< std::string > TokenCon::get_expr_elements_array ( )`

Return an array containing all the (string) elements of each expression that define the current constraint.

##### Returns

a vector of strings representing the elements of each expression that defines this constraint.

##### Note

the strings in output preserves the order as found in the original string token.

#### 6.57.1.5 `const std::vector< std::string > TokenCon::get_expr_not_var_elements_array ( )`

Return an array containing all the (string) "non variable" elements of each expression that define the current constraint.

##### Returns

a vector of strings representing the "non variable" elements of each expression that defines this constraint.

##### Note

the strings in output preserves the order as found in the original string token.

#### 6.57.1.6 `const std::vector< std::string > TokenCon::get_expr_var_elements_array ( )`

Return an array containing all the (string) "variable" elements of each expression that define the current constraint.

##### Returns

a vector of strings representing the "variable" elements of each expression that defines this constraint.

##### Note

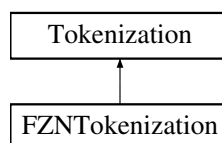
the strings in output preserves the order as found in the original string token.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_con.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_con.cpp

## 6.58 Tokenization Class Reference

Inheritance diagram for Tokenization:



### Public Member Functions

- void **add\_delimiter** (std::string)
- void **set\_delimiter** (std::string)
- void **add\_white\_spaces** (std::string)
- void **set\_white\_spaces** (std::string)
- void **set\_new\_tokenizer** (std::string line)
- bool **find\_new\_line** ()  
*Informs whether a new line has been found.*
- bool **is\_failed** () const  
*Check whether the tokenizer has failed.*
- bool **need\_line** ()

*Asks whether the tokenizer has finished all the tokens.*

- void `add_comment_symb` (char)

*Set preferences.*

- void `add_comment_symb` (std::string)
- virtual TokenPtr `get_token` ()=0

*Get the string correspondent to the (filtered) token.*

## Protected Member Functions

- virtual bool `avoid_char` (char)

*It states whether the current char has to be skipped or not.*

- virtual bool `skip_line` ()

*It states whether \_c\_token or the a line have to be skipped or not.*

- virtual bool `skip_line` (std::string)
- virtual bool `set_new_line` ()
- virtual void `clear_line` ()
- virtual TokenPtr `analyze_token` ()=0

## Protected Attributes

- std::string `_dbg`
- std::string `DELIMITERS` = "\t\r\n "
- std::string `WHITESPACE` = " \t"
- std::string `_comment_lines`
- bool `_new_line`
- bool `_need_line`
- bool `_failed`
- char \* `_c_token`

*Token returned by strtok.*

- char \* `_parsed_line`

*Parsed line.*

## 6.58.1 Member Function Documentation

### 6.58.1.1 virtual TokenPtr Tokenization::analyze\_token ( ) [protected],[pure virtual]

Analyze token: this function acts like a filter. It analyzes \_c\_token and returns a string corresponding to the token cleaned from useless chars.

### 6.58.1.2 void Tokenization::clear\_line ( ) [protected],[virtual]

It "clears" the text line by removing possible initial white spaces from line. Different heuristics may be used here.

### 6.58.1.3 bool Tokenization::set\_new\_line ( ) [protected],[virtual]

It states whether a new line has been found. Different heuristics may be used here.

### 6.58.1.4 void Tokenization::set\_new\_tokenizer ( std::string line )

Prepare a new tokenizer (i.e., string for strtok).



## Parameters

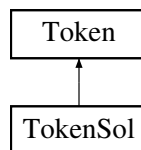
<i>line</i>	the string to tokenize.
-------------	-------------------------

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/tokenization.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/tokenization.cpp

## 6.59 TokenSol Class Reference

Inheritance diagram for TokenSol:



### Public Member Functions

- void **set\_var\_goal** (std::string)
- void **set\_solve\_goal** (std::string)
- void **set\_solve\_params** (std::string)
- void **set\_label\_choice** (std::string)
- void **set\_search\_choice** (std::string)
- void **set\_variable\_choice** (std::string)
- void **set\_assignment\_choice** (std::string)
- void **set\_strategy\_choice** (std::string)
- void **set\_var\_to\_label** (std::string)  
*Set the (string) identifier of a variable to label.*
- std::string **get\_var\_goal** () const  
*Var goal to optimize (if any).*
- std::string **get\_solve\_goal** () const  
*Solve goal: satisfy, minimize, maximize.*
- std::string **get\_search\_choice** () const  
*int\_search, bool\_search, set\_search (if any).*
- std::string **get\_label\_choice** () const  
*Variables to be assigned (if any).*
- std::string **get\_variable\_choice** () const  
*input\_order, first\_fail, etc, (if any).*
- std::string **get\_assignment\_choice** () const  
*indomain\_min, indomain\_max, etc, (if any).*
- std::string **get\_strategy\_choice** () const  
*complete, lns, etc, (if any).*
- int **num\_var\_to\_label** () const
- const std::vector< std::string > **get\_var\_to\_label** () const
- std::string **get\_var\_to\_label** (int idx) const
- virtual void **print** () const  
*Print info methods.*

## Protected Attributes

- `std::string _var_goal`
- `std::string _solve_goal`
- `std::string _search_choice`
- `std::string _label_choice`
- `std::string _variable_choice`
- `std::string _assignment_choice`
- `std::string _strategy_choice`
- `std::vector< std::string > _var_to_label`

## 6.59.1 Member Function Documentation

### 6.59.1.1 `const vector< std::string > TokenSol::get_var_to_label ( ) const`

Identifiers of the variables to label.

#### Returns

a vector of string identifiers of the variable to label during the search phase.

### 6.59.1.2 `string TokenSol::get_var_to_label ( int idx ) const`

Get the string corresponding to the *ith* variable to label.

#### Parameters

<i>idx</i>	the index of the variable to label.
------------	-------------------------------------

#### Returns

the string identifier of the *idx*<sup>th</sup> variable to label.

### 6.59.1.3 `int TokenSol::num_var_to_label ( ) const`

Number of variables to label if specified by the model.

#### Returns

the number of variables to label.

## 6.59.2 Member Data Documentation

### 6.59.2.1 `std::vector< std::string > TokenSol::_var_to_label` `[protected]`

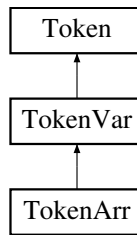
Vector of strings corresponding to the variables to label during the search phase.

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

- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token_sol.h`
- `/Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token_sol.cpp`

## 6.60 TokenVar Class Reference

Inheritance diagram for TokenVar:



### Public Member Functions

- void [set\\_var\\_id](#) (std::string str)
- std::string [get\\_var\\_id](#) () const  
*Get the string id of the current variable.*
- void [set\\_objective\\_var](#) ()  
*Identifies the current variable as an objective variable.*
- bool [is\\_objective\\_var](#) () const
- void [set\\_support\\_var](#) ()  
*Identifies the current variable as a support variable.*
- bool [is\\_support\\_var](#) () const
- void [set\\_var\\_dom\\_type](#) (VarDomainType vdt)
- VarDomainType [get\\_var\\_dom\\_type](#) () const
- void [set\\_boolean\\_domain](#) ()  
*Specifies a boolean domain for the variable.*
- void [set\\_float\\_domain](#) ()  
*Specifies a float domain for the variable.*
- void [set\\_int\\_domain](#) ()  
*Specifies an integer domain for the variable.*
- void [set\\_range\\_domain](#) (std::string str)
- void [set\\_range\\_domain](#) (int lw, int ub)
- int [get\\_lw\\_bound\\_domain](#) () const
- int [get\\_up\\_bound\\_domain](#) () const
- void [set\\_subset\\_domain](#) (std::string str)
- void [set\\_subset\\_domain](#) ()
- void [set\\_subset\\_domain](#) (const std::vector< int > &elems)
- void [set\\_subset\\_domain](#) (const std::vector< std::vector< int > > &elems)
- void [set\\_subset\\_domain](#) (const std::pair< int, int > &range)
- const std::vector< std::vector< int > > [get\\_subset\\_domain](#) ()
- virtual void [print](#) () const  
*Print info methods.*

### Protected Member Functions

- std::pair< int, int > [get\\_range](#) (std::string str) const
- std::vector< int > [get\\_subset](#) (std::string str) const

## Protected Attributes

- std::string **\_var\_id**
- bool **\_objective\_var**
- bool **\_support\_var**
- VarDomainType **\_var\_dom\_type**
- int **\_lw\_bound**
- int **\_up\_bound**
- std::vector< std::vector< int > > **\_subset\_domain**

## 6.60.1 Member Function Documentation

### 6.60.1.1 pair< int, int > TokenVar::get\_range ( std::string *str* ) const [protected]

Get a pair <x1, x2> from a string of type "*\*x1..x2\**".

#### Parameters

<i>str</i>	string to parse
------------	-----------------

#### Returns

a pair representing the range expressed with *str*

### 6.60.1.2 vector< int > TokenVar::get\_subset ( std::string *str* ) const [protected]

Get a vector of elements from a string of type "*\*{x1, x2, ...xk}\**".

#### Parameters

<i>str</i>	string to parse
------------	-----------------

#### Returns

a pair representing the range expressed with *str*

### 6.60.1.3 const vector< vector< int > > TokenVar::get\_subset\_domain ( )

Get the set of subsets of values for a var set type.

#### Returns

a vector of vectors of values representing the subsets of the var set type domain.

### 6.60.1.4 void TokenVar::set\_range\_domain ( std::string *str* )

Specifies a range domain for the variable with a given a string of type "*\*x1..x2\**".

### 6.60.1.5 void TokenVar::set\_range\_domain ( int *lw*, int *ub* )

Specifies a range domain for the variable with a given lower and upper bound.

## Parameters

<i>lw</i>	lower bound
<i>ub</i>	upper bound

6.60.1.6 void TokenVar::set\_subset\_domain ( std::string *str* )

Call the right subset function, parsing the string given in input.

## 6.60.1.7 void TokenVar::set\_subset\_domain ( )

Specifies a set of int domain.

## Note

set of int;

6.60.1.8 void TokenVar::set\_subset\_domain ( const std::vector< int > & *elems* )

Specifies a subsets of set domain for the variable with the given vector of elements.

## Parameters

<i>elems</i>	vector of elements
--------------	--------------------

## Note

set of {x1, x2, ...xk}

6.60.1.9 void TokenVar::set\_subset\_domain ( const std::vector< std::vector< int > > & *elems* )

Specifies a subsets of set domain for the variable with the given vector of elements.

## Parameters

<i>elems</i>	vector of vectors of elements
--------------	-------------------------------

## Note

set as {{x1, x2, ...xk}, ...}

6.60.1.10 void TokenVar::set\_subset\_domain ( const std::pair< int, int > & *range* )

Specifies a set of ints in range domain for the variable with the given range.

## Parameters

<i>range</i>	pair of int elements for range
--------------	--------------------------------

## Note

set of x1..x2

6.60.1.11 void TokenVar::set\_var\_dom\_type ( VarDomainType *vdt* )

Set the type of the current (token) variable.

## Parameters

<i>vdt</i>	the variable domain type of type VarDomainType.
------------	---

## 6.60.1.12 void TokenVar::set\_var\_id ( std::string str )

Set the (string) identifier of the variable represented as a token. The id is retrieved using the [get\\_var\\_id\(\)](#) method.

## Parameters

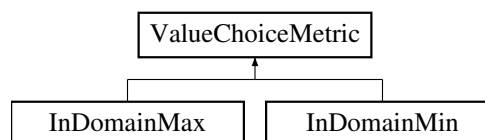
<i>str</i>	the string identifier of the variable.
------------	--

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_var.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/token\_var.cpp

## 6.61 ValueChoiceMetric Class Reference

Inheritance diagram for ValueChoiceMetric:



### Public Member Functions

- virtual ValueChoiceMetricType [metric\\_type](#) () const
- virtual int [metric\\_value](#) (Variable \*var)=0
- virtual void [print](#) () const =0

*Print info about this value choice metric.*

### Protected Attributes

- std::string [\\_dbg](#)  
*Debug string.*
- ValueChoiceMetricType [\\_metric\\_type](#)  
*Value choice metric type.*

### 6.61.1 Member Function Documentation

#### 6.61.1.1 ValueChoiceMetricType ValueChoiceMetric::metric\_type ( ) const [virtual]

Get the type of metric for this value choice metric.

#### Returns

the metric type of this value choice metric.

6.61.1.2 `virtual int ValueChoiceMetric::metric_value ( Variable * var ) [pure virtual]`

Returns the value within a variable's domain which should be used to label the current variable.

**Parameters**

<code>var</code>	(pointer to) the variable for which value for assignment is given.
------------------	--

**Returns**

the value to assign to the given variable.

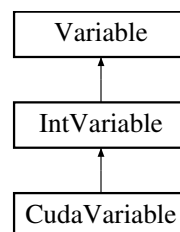
Implemented in [InDomainMax](#), and [InDomainMin](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/value\_choice\_metric.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/value\_choice\_metric.cpp

## 6.62 Variable Class Reference

Inheritance diagram for Variable:

**Public Member Functions**

- int [get\\_id](#) () const  
*Get integer id of this variable.*
- void [set\\_str\\_id](#) (std::string str)
- std::string [get\\_str\\_id](#) () const
- void [set\\_type](#) (VariableType vt)  
*Set the type of variable (i.e., FD\_VARIABLE, SUP\_VARIABLE, etc.)*
- VariableType [get\\_type](#) () const  
*Get the type of variable (i.e., FD\_VARIABLE, SUP\_VARIABLE, etc.)*
- virtual EventType [get\\_event](#) () const =0  
*Get the event happened on this domain.*
- virtual void [reset\\_event](#) ()=0  
*Reset default event on this domain.*
- virtual void [set\\_domain\\_type](#) (DomainType dt)=0
- virtual size\_t [get\\_size](#) () const =0
- virtual bool [is\\_singleton](#) () const =0
- virtual bool [is\\_empty](#) () const =0
- virtual void [print\\_domain](#) () const =0  
*Print domain.*
- virtual void [attach\\_store](#) (ConstraintStorePtr store)
- virtual void [attach\\_constraint](#) (ConstraintPtr c)
- virtual void [detach\\_constraint](#) (ConstraintPtr c)
- virtual void [detach\\_constraint](#) (size\_t c\_id)
- virtual void [notify\\_constraint](#) ()
- virtual void [notify\\_store](#) ()



- virtual size\_t [size\\_constraints](#) ()
- virtual size\_t [size\\_constraints\\_original](#) () const
- virtual void [print](#) () const

*Print info about the variable.*

## Public Attributes

- [DomainIterator](#) \* [domain\\_iterator](#)

## Protected Member Functions

- virtual bool [is\\_attached](#) (size\_t c\_id)
- [Variable](#) ()
- [Variable](#) (int v\_id)

## Protected Attributes

- std::string [\\_dbg](#)
- ConstraintStorePtr [\\_constraint\\_store](#)
- int [\\_id](#)
- std::string [\\_str\\_id](#)
- VariableType [\\_var\\_type](#)
- size\_t [\\_number\\_of\\_constraints](#)  
*Total number of observers.*
- std::map< EventType,  
std::vector< ConstraintPtr > > [\\_attached\\_constraints](#)
- std::list< size\_t > [\\_detach\\_constraints](#)

## 6.62.1 Constructor & Destructor Documentation

### 6.62.1.1 `Variable::Variable ( )` [protected]

Base constructor.

#### Note

a global unique id is assigned to this variable.

### 6.62.1.2 `Variable::Variable ( int v_id )` [protected]

Base constructor.

#### Parameters

<code>v_id</code>	the id to assign to this variable.
-------------------	------------------------------------

## 6.62.2 Member Function Documentation

### 6.62.2.1 `void Variable::attach_constraint ( ConstraintPtr c )` [virtual]

It registers constraint with this variable, so always when this variable is changed the constraint is reevaluated/notified.

## Parameters

<i>c</i>	the (pointer to) the constraint which is added to this variable.
----------	--

6.62.2.2 `void Variable::attach_store ( ConstraintStorePtr store ) [virtual]`

Set a constraint store as current constraint store for this variable. The store will be notified when this variable will change its internal state.

## Parameters

<i>store</i>	the constraint store to attach to this variable.
--------------	--

6.62.2.3 `void Variable::detach_constraint ( ConstraintPtr c ) [virtual]`

It detaches constraint from this variable, so change in variable will not cause constraint reevaluation.

## Parameters

<i>c</i>	the (pointer to) the constraint which is detached from this variable.
----------	---

## Note

If *c* appears only to be attached to this variable, this method actually destroys the constraint *c*. The client must be care of storing *c* somewhere else in order to restore the state (e.g. for backtrack actions).

6.62.2.4 `void Variable::detach_constraint ( size_t c_id ) [virtual]`

It detaches constraint from this variable, so change in variable will not cause constraint reevaluation.

## Parameters

<i>c</i>	the id of the constraint which is detached from this variable.
----------	--

## Note

If *c* appears only to be attached to this variable, this method actually destroys the constraint *c*. The client must be care of storing *c* somewhere else in order to restore the state (e.g. for backtrack actions).

6.62.2.5 `virtual size_t Variable::get_size ( ) const [pure virtual]`

It returns the size of the current domain.

## Returns

the size of the current variable's domain.

Implemented in [IntVariable](#).

6.62.2.6 `string Variable::get_str_id ( ) const`

Get the string id of this variable.

## Returns

a string representing the id of this variable.

6.62.2.7 `bool Variable::is_attached ( size_t c_id )` `[protected]`, `[virtual]`

It checks whether a given id belongs to the list of detached constraints.

## Parameters

<i>c_id</i>	the id of the constraint to check if it is detached or not.
-------------	---

## Returns

true if *c\_id* is attached, i.e., it does not belong to the list of detached constraints.

#### 6.62.2.8 `bool Variable::is_empty ( ) const` [pure virtual]

It checks if the domain is empty.

## Returns

true if variable domain is empty. false otherwise.

Implemented in [IntVariable](#).

#### 6.62.2.9 `virtual bool Variable::is_singleton ( ) const` [pure virtual]

It checks if the domain contains only one value.

## Returns

true if the the variable's domain is a singleton, false otherwise.

Implemented in [IntVariable](#).

#### 6.62.2.10 `void Variable::notify_constraint ( )` [virtual]

It notifies all the constraints attached to this variables that a change has been done on this very variable.

#### 6.62.2.11 `void Variable::notify_store ( )` [virtual]

It notifies the current store attached to this variable that a change has been done on this very variable. It actually checks which constraint should be reevaluated according to the event happened on the domain.

#### 6.62.2.12 `virtual void Variable::set_domain_type ( DomainType dt )` [pure virtual]

Set domain according to the specific variable implementation.

## Note

: different types of variable

## Parameters

<i>dt</i>	domain type of type <code>DomainType</code> to set to the current variable
-----------	--

Implemented in [IntVariable](#).

#### 6.62.2.13 `void Variable::set_str_id ( std::string str )`

Set the (string) id of the variable.

## Parameters

<i>str</i>	the string to set as variable's identifier.
------------	---

**6.62.2.14** `size_t Variable::size_constraints ( ) [virtual]`

It returns the current number of constraints attached to this variable and that are not yet satisfied.

## Returns

number of constraints attached to the variable not yet satisfied.

## Note

use this method to implement some heuristics (e.g., min conflict heuristic).

**6.62.2.15** `size_t Variable::size_constraints_original ( ) const [virtual]`

It returns the current number of constraints attached to this variable (either satisfied or not satisfied yet).

## Returns

number of constraints attached to the variable.

**6.62.3 Member Data Documentation****6.62.3.1** `std::map< EventType, std::vector< ConstraintPtr > > Variable::_attached_constraints [protected]`

List of constraints attached to this variable. These constraints are organized by the type of event they are triggered by.

**6.62.3.2** `ConstraintStorePtr Variable::_constraint_store [protected]`

The constraint store on which this variable operates (i.e., constraint store to notify).

**6.62.3.3** `std::list< size_t > Variable::_detach_constraints [protected]`

List of ids of detached constraints from this variable. These ids (i.e., constraints' ids) will be used to restore the variable's state during search.

## Note

`|_observer| + |_detach_observers| = _number_of_observers.`

**6.62.3.4** `DomainIterator* Variable::domain_iterator`

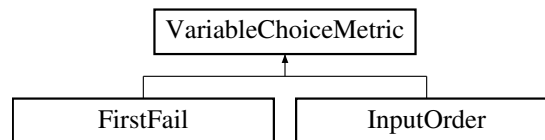
Iterator to use to get domain's elements from the current variable's domain. Domains should be accessed only through this iterator.

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/variable.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/variable.cpp

## 6.63 VariableChoiceMetric Class Reference

Inheritance diagram for VariableChoiceMetric:



### Public Member Functions

- virtual VariableChoiceMetricType [metric\\_type](#) () const
- virtual int [compare](#) (double metric, [Variable](#) \*var)=0
- virtual int [compare](#) ([Variable](#) \*var\_a, [Variable](#) \*var\_b)=0
- virtual double [metric\\_value](#) ([Variable](#) \*var)=0
- virtual void [print](#) () const =0

*Print info about this variable choice metric.*

### Protected Attributes

- std::string [\\_dbg](#)  
*Debug info.*
- VariableChoiceMetricType [\\_metric\\_type](#)

### 6.63.1 Member Function Documentation

**6.63.1.1** virtual int VariableChoiceMetric::compare ( double *metric*, [Variable](#) \* *var* ) [pure virtual]

Compares the metric value with a given variable.

Parameters

<i>metric</i>	the (metric) value to compare with.
<i>var</i>	the (pointer to) variable to compare with the metric value.

Returns

1 if metric is larger than variable 0 if metric is equal to variable -1 if metric is smaller than variable

Implemented in [InputOrder](#), and [FirstFail](#).

**6.63.1.2** virtual int VariableChoiceMetric::compare ( [Variable](#) \* *var\_a*, [Variable](#) \* *var\_b* ) [pure virtual]

Compares the metric value of var\_a with the metric value of var\_b.

Parameters

<i>var_a</i>	the (pointer to) variable to compare with the metric value of var_b.
<i>var_b</i>	the (pointer to) variable to compare with the metric value of var_a.

Returns

1 if var\_a is larger than var\_b 0 if var\_a is equal to var\_b -1 if var\_a is smaller than var\_b

Implemented in [InputOrder](#), and [FirstFail](#).

### 6.63.1.3 VariableChoiceMetricType VariableChoiceMetric::metric\_type ( ) const [virtual]

Get the type of metric for this variable choice metric.

#### Returns

the metric type of this variable choice metric.

### 6.63.1.4 virtual double VariableChoiceMetric::metric\_value ( Variable \* var ) [pure virtual]

Returns the value of the metric of a given variable.

#### Parameters

<i>var</i>	the variable for which the metric is required.
------------	--

#### Returns

the value of the metric.

Implemented in [InputOrder](#), and [FirstFail](#).

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

- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/variable\_choice\_metric.h
- /Users/fedecampe/Desktop/NVIDIOSO-PRJ/NVIDIOSO/NVIDIOSO/variable\_choice\_metric.cpp

# Index

- arguments
  - Constraint, [22](#)
- clone
  - Domain, [74](#)
- close
  - Parser, [109](#)
- consistency
  - Constraint, [22](#)
- Constraint, [20](#)
  - arguments, [22](#)
  - consistency, [22](#)
  - Constraint, [22](#)
  - decompose, [22](#)
  - events, [23](#)
  - satisfied, [24](#)
  - scope, [24](#)
  - update, [24](#)
- customize
  - Solver, [131](#)
- decompose
  - Constraint, [22](#)
- Domain, [73](#)
  - clone, [74](#)
- events
  - Constraint, [23](#)
- Heuristic, [87](#)
- Logger, [104](#)
- Memento, [104](#)
- open
  - Parser, [110](#)
- Parser, [108](#)
  - close, [109](#)
  - open, [110](#)
- run
  - Solver, [132](#)
- satisfied
  - Constraint, [24](#)
- scope
  - Constraint, [24](#)
- Solver, [131](#)
  - customize, [131](#)
  - run, [132](#)
  - Statistics, [133](#)
  - stopwatch, [134](#)
- stopwatch
  - Statistics, [134](#)
- Token, [135](#)
- Tokenization, [139](#)
- update
  - Constraint, [24](#)
- Variable, [148](#)
  - Variable, [149](#)