iNVIDIOSO1.0

Generated by Doxygen 1.8.7

Sat Jul 18 2015 12:42:31

Contents

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| <= max_vector or not:

- Bitmap: if |D| <= max_vector;
- List of bounds: otherwise.

By default, max_vector is equal to 256. This value can be redefined via and 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 0, 1, 2, ... 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 <= 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 ($<$=) VECTOR_MAX bits representing REP pairs of bounds. The first part of BIT is used to store REP pairs $<$LB, UB$>$. 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$>$.
  - REP = 0: there are UB - LB + 1 <= 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}, without any pair in BIT. 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 int $*$ bit_per_int) / B) / kB.
- REP $>$ 0: 45 kB = 11520 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 $<$ VECT↩ OR_MAX.
When a domain (or subsets of it) is (are) represented using a bitmap, the values are stored from left to right in chunks of 32 bits (considering a 32bit representation for an unsigned int), where the most significan bit is in the leftmost position of the chuck, i.e., it is the 31th bit. For example, the domain {0, 63} is store as |31...0|32...63|. The chunk is easily retrieved computing num / 32, while the position within each chunk can be retrieved by num % 32.

Chapter 2

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 FactoryCStore::get_cstore (bool on_device=false, int type=0)
propagation 1 block per variable

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 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

Class Documentation

5.1 AntiFirstFail Class Reference

Inheritance diagram for AntiFirstFail:

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

5.1.1 Member Function Documentation

5.1.1.1 int AntiFirstFail::compare (double *metric,* Variable * *var*) [virtual]

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

Implements VariableChoiceMetric.

5.1.1.2 int AntiFirstFail::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

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

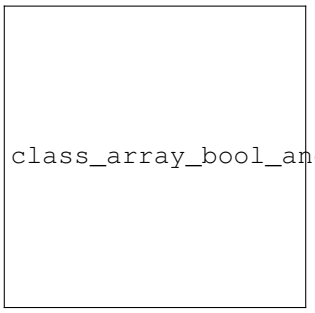
Implements VariableChoiceMetric.

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

- src/anti_first_fail_metric.h
- src/anti_first_fail_metric.cpp

5.2 ArrayBoolAnd Class Reference

Inheritance diagram for ArrayBoolAnd:



Public Member Functions

- ArrayBoolAnd ()
- ArrayBoolAnd (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.2.1 Constructor & Destructor Documentation

5.2.1.1 ArrayBoolAnd::ArrayBoolAnd ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.2.1.2 ArrayBoolAnd::ArrayBoolAnd (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.

5.2.2 Member Function Documentation

5.2.2.1 const std::vector< VariablePtr > ArrayBoolAnd::scope ( ) const [override], [virtual]

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

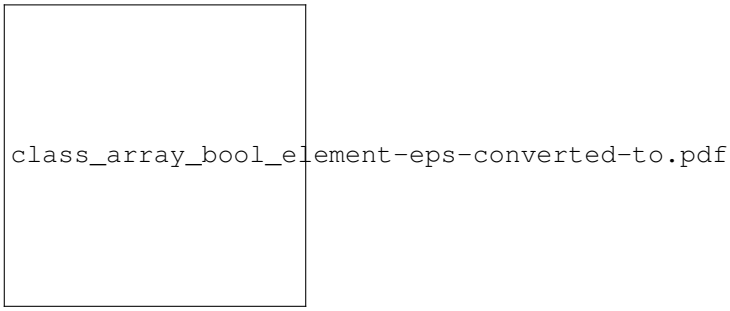
Implements Constraint.

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

- src/array_bool_and.h
- src/array_bool_and.cpp

5.3 ArrayBoolElement Class Reference

Inheritance diagram for ArrayBoolElement:



Public Member Functions

- ArrayBoolElement ()
- ArrayBoolElement (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.3.1 Constructor & Destructor Documentation

5.3.1.1 ArrayBoolElement::ArrayBoolElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.3.1.2 ArrayBoolElement::ArrayBoolElement (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.

5.3.2 Member Function Documentation

5.3.2.1 const std::vector< VariablePtr > ArrayBoolElement::scope ( ) const [override], [virtual]

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

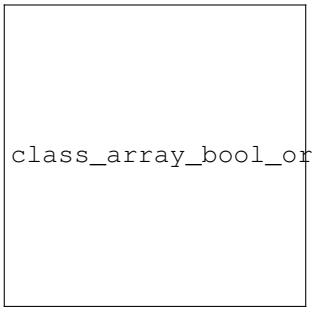
Implements Constraint.

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

- src/array_bool_element.h
- src/array_bool_element.cpp

5.4 ArrayBoolOr Class Reference

Inheritance diagram for ArrayBoolOr:



Public Member Functions

- ArrayBoolOr ()
- ArrayBoolOr (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.4.1 Constructor & Destructor Documentation

5.4.1.1 ArrayBoolOr::ArrayBoolOr ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.4.1.2 ArrayBoolOr::ArrayBoolOr (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.

5.4.2 Member Function Documentation

5.4.2.1 const std::vector< VariablePtr > ArrayBoolOr::scope ( ) const [override], [virtual]

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

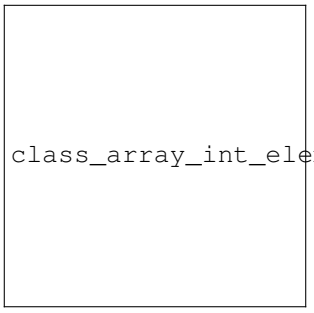
Implements Constraint.

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

- src/array_bool_or.h
- src/array_bool_or.cpp

5.5 ArrayIntElement Class Reference

Inheritance diagram for ArrayIntElement:



Public Member Functions

- ArrayIntElement ()
- ArrayIntElement (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.5.1 Constructor & Destructor Documentation

5.5.1.1 ArrayIntElement::ArrayIntElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.5.1.2 ArrayIntElement::ArrayIntElement (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.

5.5.2 Member Function Documentation

5.5.2.1 const std::vector< VariablePtr > ArrayIntElement::scope ( ) const [override], [virtual]

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

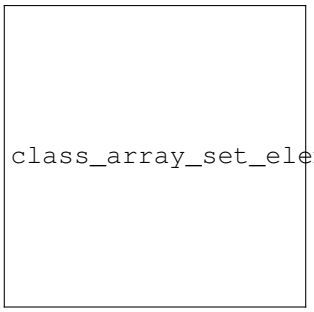
Implements Constraint.

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

- src/array_int_element.h
- src/array_int_element.cpp

5.6 ArraySetElement Class Reference

Inheritance diagram for ArraySetElement:



Public Member Functions

- ArraySetElement ()
- ArraySetElement (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.6.1 Constructor & Destructor Documentation

5.6.1.1 ArraySetElement::ArraySetElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.6.1.2 ArraySetElement::ArraySetElement (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.

5.6.2 Member Function Documentation

5.6.2.1 const std::vector< VariablePtr > ArraySetElement::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:

- src/array_set_element.h
- src/array_set_element.cpp

5.7 ArrayVarBoolElement Class Reference

Inheritance diagram for ArrayVarBoolElement:

Public Member Functions

- ArrayVarBoolElement ()
- ArrayVarBoolElement (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.7.1 Constructor & Destructor Documentation

5.7.1.1 ArrayVarBoolElement::ArrayVarBoolElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.7.1.2 ArrayVarBoolElement::ArrayVarBoolElement (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.

5.7.2 Member Function Documentation

5.7.2.1 const std::vector< VariablePtr > ArrayVarBoolElement::scope ( ) const [override], [virtual]

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

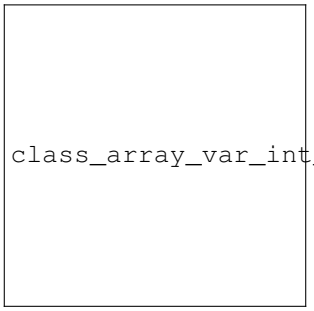
Implements Constraint.

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

- src/array_var_bool_element.h
- src/array_var_bool_element.cpp

5.8 ArrayVarIntElement Class Reference

Inheritance diagram for ArrayVarIntElement:



Public Member Functions

- ArrayVarIntElement ()
- ArrayVarIntElement (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.8.1 Constructor & Destructor Documentation

5.8.1.1 ArrayVarIntElement::ArrayVarIntElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.8.1.2 ArrayVarIntElement::ArrayVarIntElement (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.

5.8.2 Member Function Documentation

5.8.2.1 const std::vector< VariablePtr > ArrayVarIntElement::scope ( ) const [override], [virtual]

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

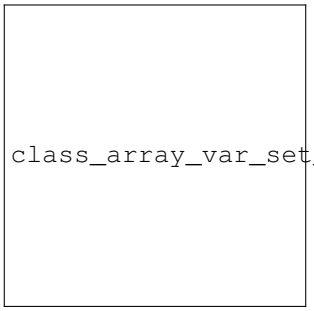
Implements Constraint.

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

- src/array_var_int_element.h
- src/array_var_int_element.cpp

5.9 ArrayVarSetElement Class Reference

Inheritance diagram for ArrayVarSetElement:



Public Member Functions

- ArrayVarSetElement ()
- ArrayVarSetElement (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.9.1 Constructor & Destructor Documentation

5.9.1.1 ArrayVarSetElement::ArrayVarSetElement ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.9.1.2 ArrayVarSetElement::ArrayVarSetElement (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.

5.9.2 Member Function Documentation

5.9.2.1 const std::vector< VariablePtr > ArrayVarSetElement::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:

- src/array_var_set_element.h
- src/array_var_set_element.cpp

5.10 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.

5.10.1 Member Function Documentation

5.10.1.1 virtual Memento* BacktrackableObject::create_memento ( ) [inline], [virtual]

Create a new memento object (state).

Returns

a reference to a new memento.

5.10.1.2 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.

5.10.1.3 virtual void BacktrackableObject::restore_state ( ) [pure virtual]

Restore a state from the current state hold by the BacktrackableObject.

Implemented in CudaVariable.

5.10.1.4 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.

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

Set a memento as current state.

Parameters

m	the memento to set as current state.

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

Set the current state of this backtrackable object.

Parameters

state	the current state to set.

5.10.1.7 virtual void BacktrackableObject::set_state ( ) [pure virtual]

Set internal state with other information hold by concrete BacktrackableObject objects.

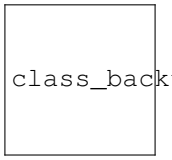
Implemented in CudaVariable.

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

- src/backtrackable_object.h

5.11 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.

5.11.1 Member Function Documentation

5.11.1.1 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

idx	the (unique) id of the backtrackable object which is changed.

Implemented in SimpleBacktrackManager.

5.11.1.2 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

bkt_obj	a reference to a backtrackable object.

Implemented in SimpleBacktrackManager.

5.11.1.3 virtual void BacktrackManager::detach_backtracable (size_t *bkt_id*) [pure virtual]

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

Parameters

bkt_id	the id of the backtrackable object to detach.

Implemented in SimpleBacktrackManager.

5.11.1.4 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.

5.11.1.5 virtual size_t BacktrackManager::get_level ( ) const [pure virtual]

Get the current active level.

Returns

current active level in the manager.

Implemented in SimpleBacktrackManager.

5.11.1.6 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.

5.11.1.7 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.

5.11.1.8 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.

5.11.1.9 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

lvl	the level to backtrack to.

Implemented in SimpleBacktrackManager.

5.11.1.10 virtual void BacktrackManager::set_level (size_t *lvl*) [pure virtual]

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

Parameters

lvl	the active level at which the changes will be recorded.

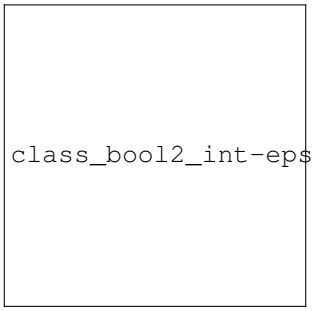
Implemented in SimpleBacktrackManager.

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

- src/backtrack_manager.h

5.12 Bool2Int Class Reference

Inheritance diagram for Bool2Int:



Public Member Functions

- Bool2Int ()
- Bool2Int (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.12.1 Constructor & Destructor Documentation

5.12.1.1 Bool2Int::Bool2Int ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.12.1.2 Bool2Int::Bool2Int (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.

5.12.2 Member Function Documentation

5.12.2.1 const std::vector< VariablePtr > Bool2Int::scope ( ) const [override], [virtual]

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

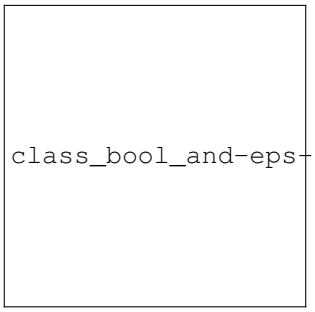
Implements Constraint.

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

- src/bool_2_int.h
- src/bool_2_int.cpp

5.13 BoolAnd Class Reference

Inheritance diagram for BoolAnd:



Public Member Functions

- BoolAnd ()
- BoolAnd (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.13.1 Constructor & Destructor Documentation

5.13.1.1 BoolAnd::BoolAnd ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.13.1.2 BoolAnd::BoolAnd (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.

5.13.2 Member Function Documentation

5.13.2.1 const std::vector< VariablePtr > BoolAnd::scope ( ) const [override], [virtual]

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

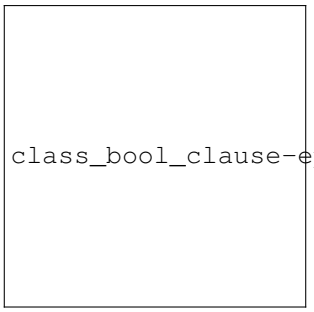
Implements Constraint.

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

- src/bool_and.h
- src/bool_and.cpp

5.14 BoolClause Class Reference

Inheritance diagram for BoolClause:



Public Member Functions

- BoolClause ()
- BoolClause (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.14.1 Constructor & Destructor Documentation

5.14.1.1 BoolClause::BoolClause ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.14.1.2 BoolClause::BoolClause (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.

5.14.2 Member Function Documentation

5.14.2.1 const std::vector< VariablePtr > BoolClause::scope ( ) const [override], [virtual]

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

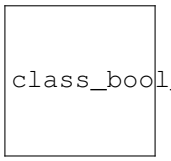
Implements Constraint.

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

- src/bool_clause.h
- src/bool_clause.cpp

5.15 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

5.15.1 Member Function Documentation

5.15.1.1 EventType BoolDomain::get_event ( ) const [virtual]

Get event on this domain

Todo implement this function

Implements Domain.

5.15.1.2 void BoolDomain::reset_event ( ) [virtual]

Sets the no event on this domain.

Note

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

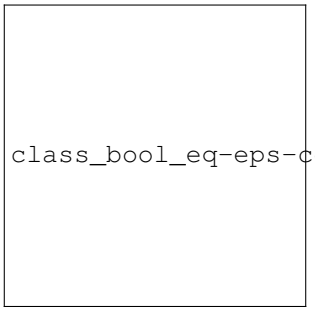
Implements Domain.

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

- src/bool_domain.h
- src/bool_domain.cpp

5.16 BoolEq Class Reference

Inheritance diagram for BoolEq:



Public Member Functions

- BoolEq ()
- BoolEq (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.16.1 Constructor & Destructor Documentation

5.16.1.1 BoolEq::BoolEq ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.16.1.2 BoolEq::BoolEq (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.

5.16.2 Member Function Documentation

5.16.2.1 const std::vector< VariablePtr > BoolEq::scope ( ) const [override], [virtual]

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

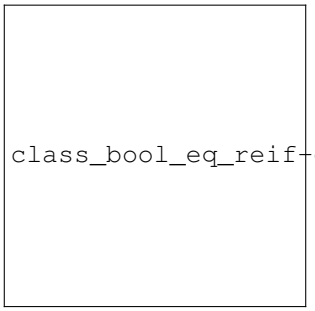
Implements Constraint.

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

- src/bool_eq.h
- src/bool_eq.cpp

5.17 BoolEqReif Class Reference

Inheritance diagram for BoolEqReif:



Public Member Functions

- BoolEqReif ()
- BoolEqReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.17.1 Constructor & Destructor Documentation

5.17.1.1 BoolEqReif::BoolEqReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.17.1.2 BoolEqReif::BoolEqReif (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.

5.17.2 Member Function Documentation

5.17.2.1 const std::vector< VariablePtr > BoolEqReif::scope ( ) const [override], [virtual]

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

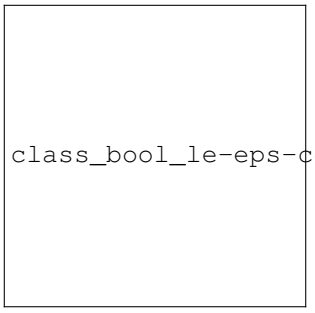
Implements Constraint.

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

- src/bool_eq_reif.h
- src/bool_eq_reif.cpp

5.18 BoolLe Class Reference

Inheritance diagram for BoolLe:



Public Member Functions

- BoolLe ()
- BoolLe (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.18.1 Constructor & Destructor Documentation

5.18.1.1 BoolLe::BoolLe ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.18.1.2 BoolLe::BoolLe (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.

5.18.2 Member Function Documentation

5.18.2.1 const std::vector< VariablePtr > BoolLe::scope ( ) const [override], [virtual]

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

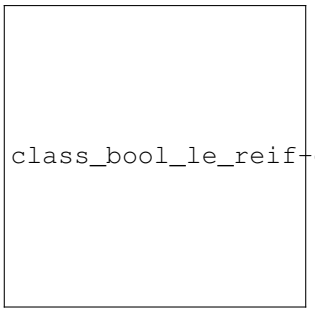
Implements Constraint.

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

- src/bool_le.h
- src/bool_le.cpp

5.19 BoolLeReif Class Reference

Inheritance diagram for BoolLeReif:



Public Member Functions

- BoolLeReif ()
- BoolLeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.19.1 Constructor & Destructor Documentation

5.19.1.1 BoolLeReif::BoolLeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.19.1.2 BoolLeReif::BoolLeReif (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.

5.19.2 Member Function Documentation

5.19.2.1 const std::vector< VariablePtr > BoolLeReif::scope ( ) const [override], [virtual]

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

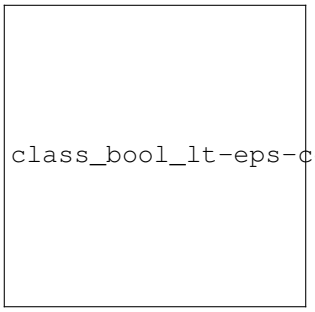
Implements Constraint.

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

- src/bool_le_reif.h
- src/bool_le_reif.cpp

5.20 BoolLt Class Reference

Inheritance diagram for BoolLt:



Public Member Functions

- BoolLt ()
- BoolLt (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.20.1 Constructor & Destructor Documentation

5.20.1.1 BoolLt::BoolLt ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.20.1.2 BoolLt::BoolLt (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.

5.20.2 Member Function Documentation

5.20.2.1 const std::vector< VariablePtr > BoolLt::scope ( ) const [override], [virtual]

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

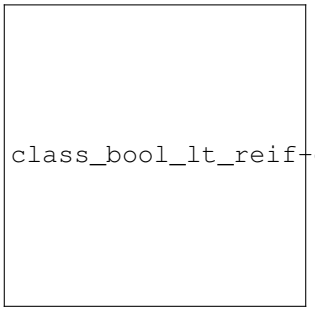
Implements Constraint.

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

- src/bool_lt.h
- src/bool_lt.cpp

5.21 BoolLtReif Class Reference

Inheritance diagram for BoolLtReif:



Public Member Functions

- BoolLtReif ()
- BoolLtReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.21.1 Constructor & Destructor Documentation

5.21.1.1 BoolLtReif::BoolLtReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.21.1.2 BoolLtReif::BoolLtReif (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.

5.21.2 Member Function Documentation

5.21.2.1 const std::vector< VariablePtr > BoolLtReif::scope ( ) const [override], [virtual]

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

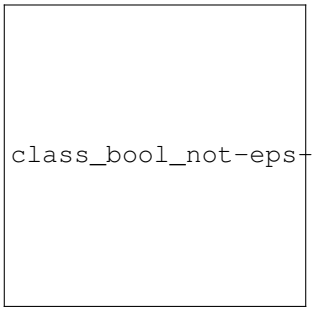
Implements Constraint.

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

- src/bool_lt_reif.h
- src/bool_lt_reif.cpp

5.22 BoolNot Class Reference

Inheritance diagram for BoolNot:



Public Member Functions

- BoolNot ()
- BoolNot (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.22.1 Constructor & Destructor Documentation

5.22.1.1 BoolNot::BoolNot ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.22.1.2 BoolNot::BoolNot (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.

5.22.2 Member Function Documentation

5.22.2.1 const std::vector< VariablePtr > BoolNot::scope ( ) const [override], [virtual]

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

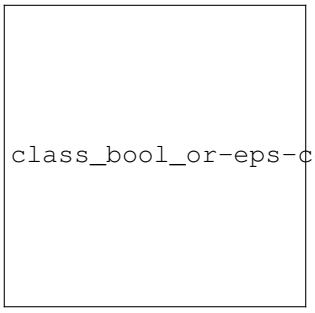
Implements Constraint.

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

- src/bool_not.h
- src/bool_not.cpp

5.23 BoolOr Class Reference

Inheritance diagram for BoolOr:



Public Member Functions

- BoolOr ()
- BoolOr (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.23.1 Constructor & Destructor Documentation

5.23.1.1 BoolOr::BoolOr ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.23.1.2 BoolOr::BoolOr (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.

5.23.2 Member Function Documentation

5.23.2.1 const std::vector< VariablePtr > BoolOr::scope ( ) const [override], [virtual]

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

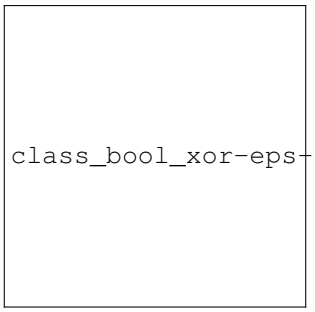
Implements Constraint.

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

- src/bool_or.h
- src/bool_or.cpp

5.24 BoolXor Class Reference

Inheritance diagram for BoolXor:



Public Member Functions

- BoolXor ()
- BoolXor (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.24.1 Constructor & Destructor Documentation

5.24.1.1 BoolXor::BoolXor ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.24.1.2 BoolXor::BoolXor (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.

5.24.2 Member Function Documentation

5.24.2.1 const std::vector< VariablePtr > BoolXor::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:

- src/bool_xor.h
- src/bool_xor.cpp

5.25 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

5.25.1 Member Function Documentation

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

It computes union of this domain and {value}.

Parameters

value	it specifies the value which is being added.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

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

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

Parameters

min	lower bound of the new domain which is being added.
max	upper bound of the new domain which is being added.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

5.25.1.3 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

value	to check whether it is in the current domain.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

5.25.1.4 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.

5.25.1.5 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 invokation of the method and a non-singleton domain. For example, throw an exception or returning the lower bound.

Implemented in CudaConcreteDomainBitmap, and CudaConcreteDomainList.

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

It updates the domain according to the maximum value.

Parameters

max	domain value.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

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

It updates the domain according to the minimum value.

Parameters

min	domain value.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

5.25.1.8 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.

5.25.1.9 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.

5.25.1.10 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.

5.25.1.11 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

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

Note

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

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, CudaConcreteDomain, and CudaConcreteDomainList.

5.25.1.12 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

min	new lower bound to set for the current domain.
max	new upper bound to set for the current domain.

Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

5.25.1.13 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.

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

It substracts {value} from the current domain.

Parameters

value	the value to subtract from the current domain.

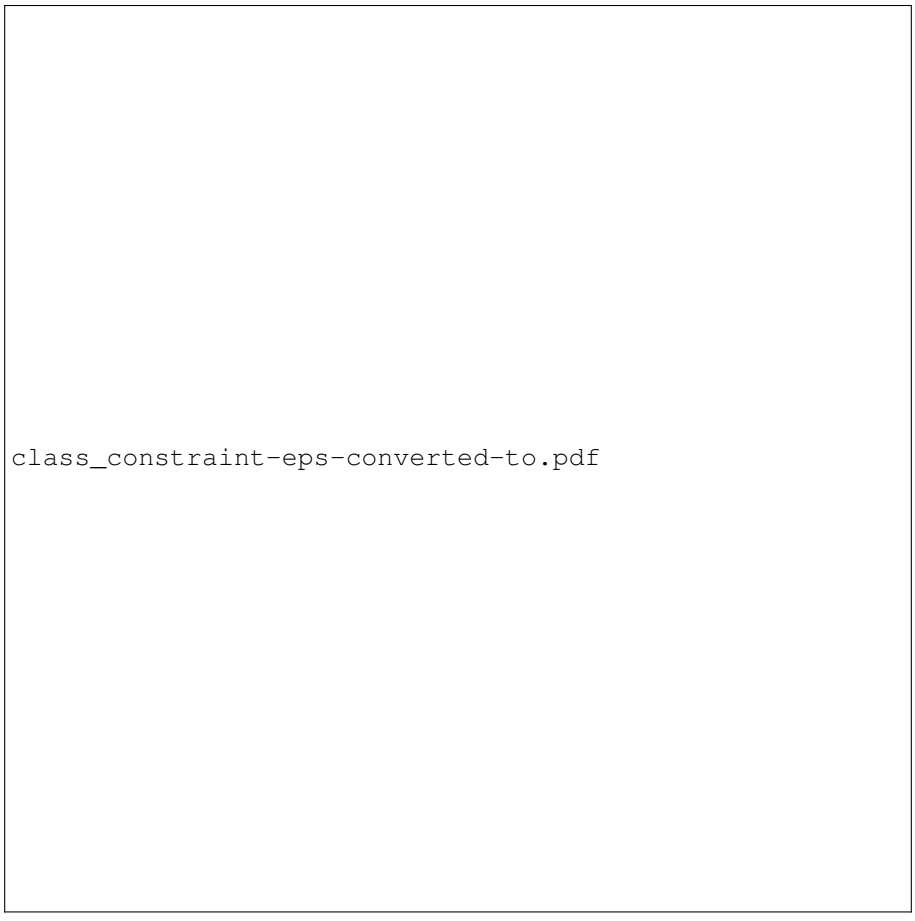
Implemented in CudaConcreteBitmapList, CudaConcreteDomainBitmap, and CudaConcreteDomainList.

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

- src/concrete_domain.h

5.26 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

5.26.1 Constructor & Destructor Documentation

5.26.1.1 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.

5.26.2 Member Function Documentation

5.26.2.1 const std::vector< int > & Constraint::arguments ( ) const

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

5.26.2.2 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.

5.26.2.3 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.

5.26.2.4 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

event	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.

5.26.2.5 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, IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, ArrayBoolAnd, ArrayBoolElement, ArrayBoolOr, ArrayIntElement, ArraySetElement, ArrayVarBoolElement, ArrayVarIntElement, ArrayVarSetElement, Bool2Int, BoolAnd, BoolClause, BoolEq, BoolEqReif, BoolLe, BoolLeReif, BoolLt, BoolLtReif, BoolNot, BoolOr, BoolXor, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubsetReif, SetSymDiff, and SetUnion.

5.26.2.6 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.

5.26.2.7 void Constraint::decrease_weight (int *weight* = 1)

Decrease current weight.

Parameters

weight	the weight to decrease from the current weight (default: 1).

5.26.2.8 const std::vector< EventType > & Constraint::events ( ) const

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

5.26.2.9 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.

5.26.2.10 int Constraint::get_number_id ( ) const

Get number id of this constraint.

Note

same type of constraints have same number_id.

5.26.2.11 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.

5.26.2.12 ConstraintPtr Constraint::get_this_shared_ptr ( ) [protected], [virtual]

Create a shared pointer from this instance.

Returns

a shared pointer to Constraint object.

5.26.2.13 void Constraint::increase_weight (int *weight* = 1)

Increse current weight.

Parameters

weight	the weight to add to the current weight (default: 1).

5.26.2.14 virtual void Constraint::remove_constraint ( ) [pure virtual]

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

Implemented in FZNConstraint.

5.26.2.15 virtual bool Constraint::satisfied ( ) [pure virtual]

It checks if the constraint is satisfied.

Returns

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

Note

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

Implemented in FZNConstraint, IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, ArrayBoolAnd, ArrayBoolElement, ArrayBoolOr, ArrayIntElement, ArraySetElement, ArrayVarBoolElement, ArrayVarIntElement, ArrayVarSetElement, Bool2Int, BoolAnd, BoolClause, BoolEq, BoolEqReif, BoolLe, BoolLeReif, BoolLt, BoolLtReif, BoolNot, BoolOr, BoolXor, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubsetReif, SetSymDiff, and SetUnion.

5.26.2.16 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 IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, ArrayBoolAnd, ArrayBoolElement, ArrayBoolOr, ArrayIntElement, ArraySetElement, ArrayVarBoolElement, ArrayVarIntElement, ArrayVarSetElement, Bool2Int, BoolAnd, BoolClause, BoolEq, BoolEqReif, BoolLe, BoolLeReif, BoolLt, BoolLtReif, BoolNot, BoolOr, BoolXor, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubsetReif, SetSymDiff, and SetUnion.

5.26.2.17 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.

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

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

Parameters

event	the event that will trigger the re-evaluation of this constriant.

Note

default: CHANGE_EVT.
different constraints should specilize this method with the appropriate list of events.

5.26.2.19 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.

5.26.2.20 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

e	an object of type Event that specifies the event that triggered the update.

5.26.3 Member Data Documentation

5.26.3.1 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.

5.26.3.2 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.

5.26.3.3 int Constraint::_number_id [protected]

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

5.26.3.4 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.

5.26.3.5 std::vector<EventType> Constraint::_trigger_events [protected]

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

Note

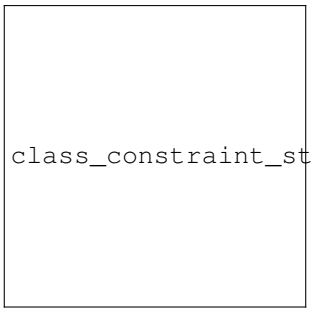
see domain.h for the list of events of type "EventType".

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

- src/constraint.h
- src/constraint.cpp

5.27 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 con_check (bool con_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 * getConstraint ()=0
- virtual void clear_queue ()=0
- virtual size_t num_constraints () const =0
- virtual size_t num_constraints_to_reevaluate () const =0
- virtual size_t num_propagations () const =0
- virtual void print () const =0

  Print information about this constraint store.

5.27.1 Member Function Documentation

5.27.1.1 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

c_id	the vector of constraints ids to re-evaluate.
event	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.

5.27.1.2 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.

5.27.1.3 virtual void ConstraintStore::con_check (bool *con_check* = true) [pure virtual]

Sets constraint propagation. Thic set increases the time spent during propagation but reduces the total exectuion time.

Parameters

con_check	boolean value representing whether or not the satisfiability check should be performed (default: true).

Implemented in SimpleConstraintStore.

5.27.1.4 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, and CudaSimpleConstraintStore.

5.27.1.5 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.

5.27.1.6 virtual Constraint* 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.

5.27.1.7 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

c	the constraint to impose in this constraint store.

Implemented in SimpleConstraintStore.

5.27.1.8 virtual size_t ConstraintStore::num_constraints ( ) const [pure virtual]

Returns the total number of constraints in this constraint store.

Implemented in SimpleConstraintStore.

5.27.1.9 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.

5.27.1.10 virtual size_t ConstraintStore::num_propagations ( ) const [pure virtual]

Returns the total number of propagations performed by this constraint store so far.

Implemented in SimpleConstraintStore.

5.27.1.11 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

sat_check	boolean value representing whether or not the satisfiability check should be performed (default: true).

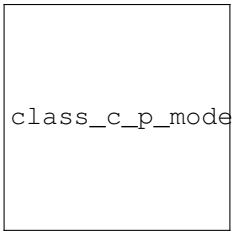
Implemented in SimpleConstraintStore.

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

- src/constraint_store.h

5.28 CPModel Class Reference

Inheritance diagram for CPModel:



Public Member Functions

- virtual int get_id () const
- virtual size_t num_variables () const

  Return the current number of variabes in the model.
- virtual size_t num_constraints () const

  Return the current number of constraints in the model.
- 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 finalize ()
- 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.

5.28.1 Member Function Documentation

5.28.1.1 void CPModel::add_constraint (ConstraintPtr *ptr*) [virtual]

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

Parameters

ptr	pointer to the constraint to add to the model

5.28.1.2 void CPModel::add_constraint_store (ConstraintStorePtr *store*) [virtual]

Add a constraint store to the model.

Parameters

store	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.

5.28.1.3 void CPModel::add_search_engine (SearchEnginePtr *ptr*) [virtual]

Add a search engine to the model.

Parameters

ptr	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.

5.28.1.4 void CPModel::add_variable (VariablePtr *ptr*) [virtual]

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

Parameters

ptr	pointer to the variable to add to the model

5.28.1.5 void CPModel::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.

5.28.1.6 void CPModel::create_constraint_graph ( ) [virtual]

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

5.28.1.7 void CPModel::finalize ( ) [virtual]

Finalizes the model.

Note

This is an auxiliary method needed by some derived classes in order to finalize the model on different architectures.

Reimplemented in CudaCPModel.

5.28.1.8 int CPModel::get_id ( ) const [virtual]

Get the (unique) id of this model.

Returns

the model's id.

5.28.1.9 SearchEnginePtr CPModel::get_search_engine ( ) [virtual]

Gets the search engine in order to run it.

Returns

a reference to the search engine in this model.

5.28.1.10 void CPModel::init_constraint_store ( ) [virtual]

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

5.28.1.11 void CPModel::set_solutions_limit (size_t *sol_limit*) [virtual]

Imposes a limit on the number of solutions.

Parameters

sol_limit	the maximum number of solutions for this model.

Note

-1 means find all solutions.

5.28.1.12 void CPModel::set_timeout_limit (double *timeout*) [virtual]

Imposes a timeoutlimit.

Parameters

timeout	timeout limit.

Note

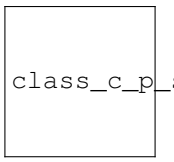
-1 means no timeout.

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

- src/cp_model.h
- src/cp_model.cpp

5.29 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.

5.29.1 Constructor & Destructor Documentation

5.29.1.1 CPSolver::CPSolver (CPModel * *model*)

Constructor.

Parameters

model	a model to add to this CPSolver.

5.29.2 Member Function Documentation

5.29.2.1 void CPSolver::add_model (CPModel * *model*) [override], [virtual]

Add a model to the solver.

Parameters

model	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.

5.29.2.2 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_data	a reference to a input_data class where options are retrieved.
model_idx	the index of the model to customize (default: 0, i.e., first model).

Implements Solver.

5.29.2.3 CPModel * CPSolver::get_model (int *model_idx*) const [override], [virtual]

Returns a reference to model.

Parameters

model_idx	the index of the model to return.

Implements Solver.

5.29.2.4 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.

5.29.2.5 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.

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

Removes a model actually destroying it.

Parameters

model_idx	the index of the model to destroy.

Implements Solver.

5.29.2.7 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.

5.29.2.8 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

model_idx	the index of the model to solve.

Implements Solver.

5.29.2.9 void CPSolver::run_model (CPModel * *model*) [protected]

It actually runs a CP Model.

Parameters

a	reference to a CP Model.

5.29.2.10 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.

5.29.2.11 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.

5.29.3 Member Data Documentation

5.29.3.1 std::vector< CPModel * > CPSolver::_models [protected]

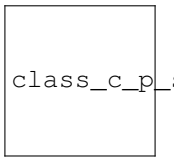
CP models to be considered by this CPSolver. The solver may decide which model to solve and in which order solve it.

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

- src/cp_solver.h
- src/cp_solver.cpp

5.30 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

5.30.1 Member Function Documentation

5.30.1.1 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.

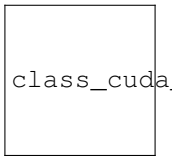
Implements DataStore.

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

- src/cp_store.h
- src/cp_store.cpp

5.31 CudaArrayBoolAnd Class Reference

Inheritance diagram for CudaArrayBoolAnd:



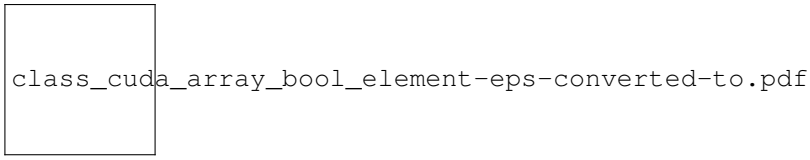
Additional Inherited Members

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

- src/cuda_array_bool_and.h

5.32 CudaArrayBoolElement Class Reference

Inheritance diagram for CudaArrayBoolElement:



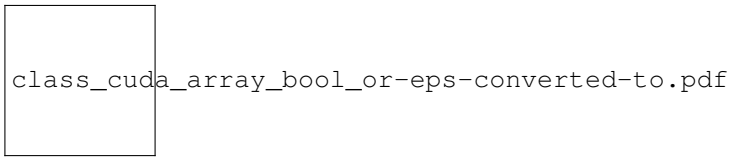
Additional Inherited Members

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

- src/cuda_array_bool_element.h

5.33 CudaArrayBoolOr Class Reference

Inheritance diagram for CudaArrayBoolOr:



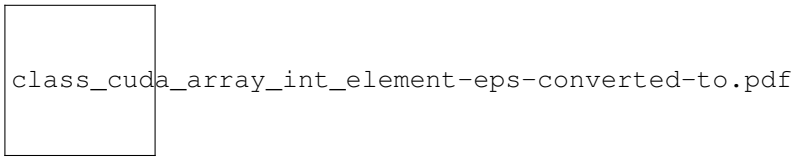
Additional Inherited Members

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

- src/cuda_array_bool_or.h

5.34 CudaArrayIntElement Class Reference

Inheritance diagram for CudaArrayIntElement:



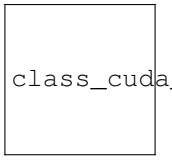
Additional Inherited Members

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

- src/cuda_array_int_element.h

5.35 CudaArraySetElement Class Reference

Inheritance diagram for CudaArraySetElement:



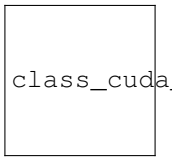
Additional Inherited Members

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

- src/cuda_array_set_element.h

5.36 CudaArrayVarBoolElement Class Reference

Inheritance diagram for CudaArrayVarBoolElement:



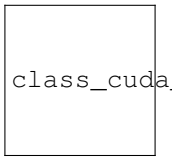
Additional Inherited Members

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

- src/cuda_array_var_bool_element.h

5.37 CudaArrayVarIntElement Class Reference

Inheritance diagram for CudaArrayVarIntElement:



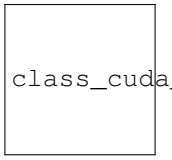
Additional Inherited Members

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

- src/cuda_array_var_int_element.h

5.38 CudaArrayVarSetElement Class Reference

Inheritance diagram for CudaArrayVarSetElement:



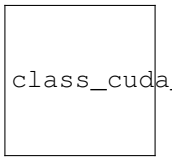
Additional Inherited Members

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

- src/cuda_array_var_set_element.h

5.39 CudaBool2Int Class Reference

Inheritance diagram for CudaBool2Int:



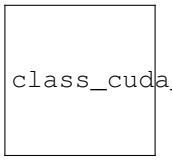
Additional Inherited Members

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

- src/cuda_bool_2_int.h

5.40 CudaBoolAnd Class Reference

Inheritance diagram for CudaBoolAnd:



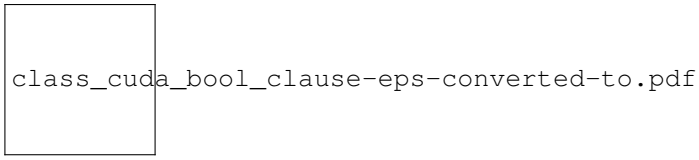
Additional Inherited Members

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

- src/cuda_bool_and.h

5.41 CudaBoolClause Class Reference

Inheritance diagram for CudaBoolClause:



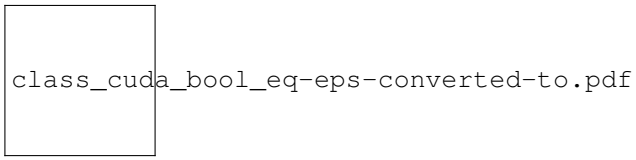
Additional Inherited Members

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

- src/cuda_bool_clause.h

5.42 CudaBoolEq Class Reference

Inheritance diagram for CudaBoolEq:



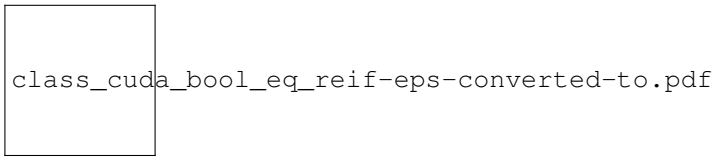
Additional Inherited Members

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

- src/cuda_bool_eq.h

5.43 CudaBoolEqReif Class Reference

Inheritance diagram for CudaBoolEqReif:



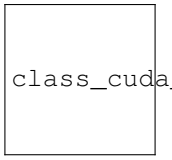
Additional Inherited Members

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

- src/cuda_bool_eq_reif.h

5.44 CudaBoolLe Class Reference

Inheritance diagram for CudaBoolLe:



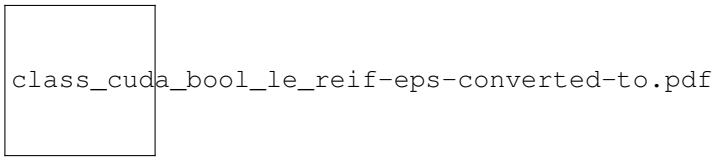
Additional Inherited Members

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

- src/cuda_bool_le.h

5.45 CudaBoolLeReif Class Reference

Inheritance diagram for CudaBoolLeReif:



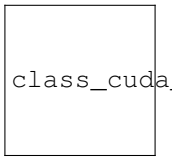
Additional Inherited Members

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

- src/cuda_bool_le_reif.h

5.46 CudaBoolLt Class Reference

Inheritance diagram for CudaBoolLt:



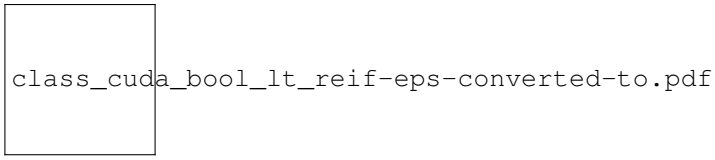
Additional Inherited Members

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

- src/cuda_bool_lt.h

5.47 CudaBoolLtReif Class Reference

Inheritance diagram for CudaBoolLtReif:



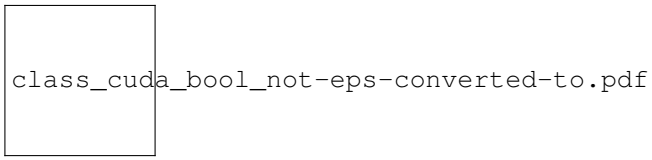
Additional Inherited Members

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

- src/cuda_bool_lt_reif.h

5.48 CudaBoolNot Class Reference

Inheritance diagram for CudaBoolNot:



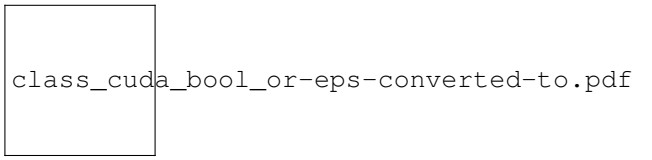
Additional Inherited Members

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

- src/cuda_bool_not.h

5.49 CudaBoolOr Class Reference

Inheritance diagram for CudaBoolOr:



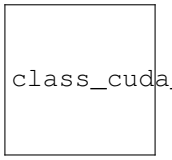
Additional Inherited Members

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

- src/cuda_bool_or.h

5.50 CudaBoolXor Class Reference

Inheritance diagram for CudaBoolXor:



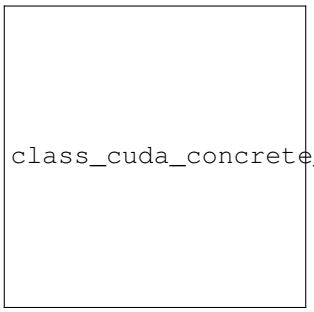
Additional Inherited Members

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

- src/cuda_bool_xor.h

5.51 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

5.51.1 Constructor & Destructor Documentation

5.51.1.1 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

size	the number of bytes to allocate.
pairs	the SORTED list of pairs to allocate.

5.51.2 Member Function Documentation

5.51.2.1 void CudaConcreteBitmapList::add (int *value*) [virtual]

It computes union of this domain and {value}.

Parameters

value	it specifies the value which is being added.

Reimplemented from CudaConcreteDomainBitmap.

5.51.2.2 void CudaConcreteBitmapList::add (int *min*, int *max*) [virtual]

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

Parameters

min	lower bound of the new domain which is being added.
max	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.

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

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

Parameters

val	to check whether it is in the current domain.

Note

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

Reimplemented from CudaConcreteDomainBitmap.

5.51.2.4 int CudaConcreteBitmapList::find_next_pair (int *val*) const [protected]

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

Parameters

val	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.

5.51.2.5 int CudaConcreteBitmapList::find_pair (int *val*) const [protected]

Find the index of the pair containing val.

Parameters

val	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.

5.51.2.6 int CudaConcreteBitmapList::find_prev_pair (int *val*) const [protected]

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

Parameters

val	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.

5.51.2.7 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.

5.51.2.8 void CudaConcreteBitmapList::in_max (int *max*) [virtual]

It updates the domain according to max value.

Parameters

max	domain value.

Reimplemented from CudaConcreteDomainBitmap.

5.51.2.9 void CudaConcreteBitmapList::in_min (int *min*) [virtual]

It updates the domain according to min value.

Parameters

min	domain value.

Reimplemented from CudaConcreteDomainBitmap.

5.51.2.10 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.

5.51.2.11 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

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

Note

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

Reimplemented from CudaConcreteDomainBitmap.

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

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

Parameters

min	new lower bound to set for the current domain.
max	new upper bound to set for the current domain.

Reimplemented from CudaConcreteDomainBitmap.

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

It substracts {value} from the current domain.

Parameters

value	the value to subtract from the current domain.

Reimplemented from CudaConcreteDomainBitmap.

5.51.3 Member Data Documentation

5.51.3.1 unsigned int CudaConcreteBitmapList::_domain_size [protected]

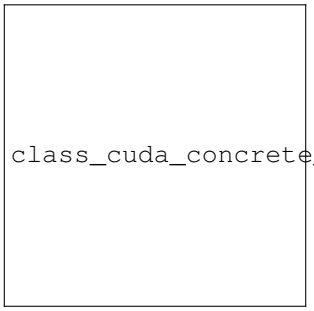
Current domain size, i.e., sum of the elements on each bitmap.

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

- src/cuda_concrete_bitmaplist.h
- src/cuda_concrete_bitmaplist.cpp

5.52 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

5.52.1 Constructor & Destructor Documentation

5.52.1.1 CudaConcreteDomain::CudaConcreteDomain (size_t *size*) [protected]

Constructor for CudaConcreteDomain. It instantiates a new object and allocate size bytes for the array of integers

Parameters

size	the number of bytes to allocate.

Note

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

5.52.2 Member Function Documentation

5.52.2.1 size_t CudaConcreteDomain::allocated_bytes ( ) const

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

5.52.2.2 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.

5.52.2.3 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.

5.52.2.4 int CudaConcreteDomain::get_num_chunks ( ) const

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

5.52.2.5 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 >.

5.52.2.6 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 >.

5.52.2.7 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

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

Note

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

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteDomainList.

5.52.2.8 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.

5.52.3 Member Data Documentation

5.52.3.1 int* CudaConcreteDomain::_concrete_domain [protected]

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

Note

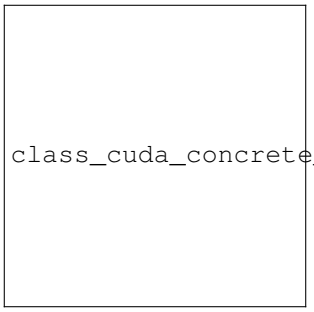
actual internal representation of domain.

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

- src/cuda_concrete_domain.h
- src/cuda_concrete_domain.cpp

5.53 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

5.53.1 Constructor & Destructor Documentation

5.53.1.1 CudaConcreteDomainBitmap::CudaConcreteDomainBitmap (size_t *size*)

Constructor for CudaConcreteDomainBitmap.

Parameters

size	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).

5.53.1.2 CudaConcreteDomainBitmap::CudaConcreteDomainBitmap (size_t *size*, int *min*, int *max*)

Constructor for CudaConcreteDomainBitmap.

Parameters

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

5.53.2 Member Function Documentation

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

It computes union of this domain and {value}.

Parameters

value	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.

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

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

Parameters

min	lower bound of the new domain which is being added.
max	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.

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

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

Parameters

value	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.

5.53.2.4 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.

5.53.2.5 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 >.

5.53.2.6 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 chuck, i.e., the position of the corresponding bit within a chunk.

Parameters

val	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.

5.53.2.7 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

val	is the (integer) value for which the chunk is needed

Returns

number of int used as bitmaps to represent max

5.53.2.8 void CudaConcreteDomainBitmap::in_max (int *max*) [virtual]

It updates the domain according to max value.

Parameters

max	domain value.

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteBitmapList.

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

It updates the domain according to min value.

Parameters

min	domain value.

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteBitmapList.

5.53.2.10 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 >.

5.53.2.11 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

size	the size in terms of number of elements of the domain to represent as bitmap.

Returns

number of chunks needed to represent size valus.

5.53.2.12 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.

5.53.2.13 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

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

Note

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

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteBitmapList.

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

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

Parameters

min	new lower bound to set for the current domain.
max	new upper bound to set for the current domain.

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteBitmapList.

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

It substracts {value} from the current domain.

Parameters

value	the value to subtract from the current domain.

Implements ConcreteDomain< int >.

Reimplemented in CudaConcreteBitmapList.

5.53.3 Member Data Documentation

5.53.3.1 constexpr int CudaConcreteDomainBitmap::BITS_IN_BYTE = INT8_C(8) [static], [protected]

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

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

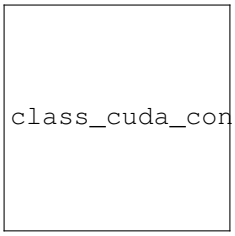
Macro for the size of a chunk in terms of bits.

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

- src/cuda_concrete_bitmap.h
- src/cuda_concrete_bitmap.cpp

5.54 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

5.54.1 Constructor & Destructor Documentation

5.54.1.1 CudaConcreteDomainList::CudaConcreteDomainList (size_t *size*, int *min*, int *max*)

Constructor for CudaConcreteDomainList.

Parameters

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

5.54.2 Member Function Documentation

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

It computes union of this domain and {value}.

Parameters

value	it specifies the value which is being added.

Implements ConcreteDomain< int >.

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

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

Parameters

min	lower bound of the new domain which is being added.
max	upper bound of the new domain which is being added.

Implements ConcreteDomain< int >.

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

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

Parameters

val	to check whether it is in the current domain.

Note

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

Implements ConcreteDomain< int >.

5.54.2.4 int CudaConcreteDomainList::find_next_pair (int *val*) const [protected]

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

Parameters

val	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.

5.54.2.5 int CudaConcreteDomainList::find_pair (int *val*) const [protected]

Find the index of the pair containing val.

Parameters

val	to be searched in the list of pairs.

Returns

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

5.54.2.6 int CudaConcreteDomainList::find_prev_pair (int *val*) const [protected]

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

Parameters

val	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.

5.54.2.7 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.

5.54.2.8 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 >.

5.54.2.9 void CudaConcreteDomainList::in_max (int *max*) [virtual]

It updates the domain according to max value.

Parameters

max	domain value.

Implements ConcreteDomain< int >.

5.54.2.10 void CudaConcreteDomainList::in_min (int *min*) [virtual]

It updates the domain according to min value.

Parameters

min	domain value.

Implements ConcreteDomain< int >.

5.54.2.11 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 >.

5.54.2.12 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 >.

5.54.2.13 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

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

Note

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

Reimplemented from CudaConcreteDomain.

5.54.2.14 void CudaConcreteDomainList::shrink (int *min*, int *max*) [virtual]

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

Parameters

min	new lower bound to set for the current domain.
max	new upper bound to set for the current domain.

Implements ConcreteDomain< int >.

5.54.2.15 void CudaConcreteDomainList::subtract (int *value*) [virtual]

It substracts {value} from the current domain.

Parameters

value	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 >.

5.54.3 Member Data Documentation

5.54.3.1 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:

- src/cuda_concrete_list.h
- src/cuda_concrete_list.cpp

5.55 CudaConstraint Class Reference

Inheritance diagram for CudaConstraint:

Protected Attributes

- size_t _unique_id

  Unique global identifier for a given constraint.
- int _scope_size

  Scope size.
- int * _vars
- int _args_size

  Number of arguments.
- int * _args
- uint ** _status
- uint ** _temp_status

  Temporary status used for copy on shared memory.

5.55.1 Member Data Documentation

5.55.1.1 int* CudaConstraint::_args [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.

5.55.1.2 uint** CudaConstraint::_status [protected]

Array of pointers to the domains of the variables involved in this constraint.

5.55.1.3 int* CudaConstraint::_vars [protected]

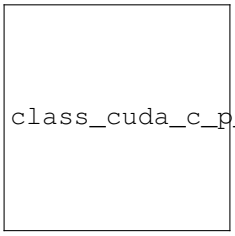
It represents the array of pointers to the domains of the variables in the scope of this constraint.

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

- src/cuda_constraint.h

5.56 CudaCPModel Class Reference

Inheritance diagram for CudaCPModel:



Public Member Functions

- uint *const get_dev_domain_states_ptr () const

  Get function for domain states.
- int *const get_dev_domain_index_ptr () const

  Get function for domain states indeces.
- void finalize ()
- virtual bool upload_device_state ()
- virtual bool download_device_state ()

Public Attributes

- std::unordered_map< size_t, size_t > constraint_mapping_h_d

  Mapping between constraint ids and the constraints on device.

Protected Member Functions

- virtual bool alloc_variables ()

  Allocate domains on device.
- virtual bool alloc_constraints ()

  Allocate constraints on device.

Protected Attributes

- std::string **_dbg**
- uint * _h_domain_states

  Domain state on host.
- uint * _d_domain_states

  Domain state on device.

- size_t _domain_state_size

  Size of (num. of bytes) all domains.
- int * _d_domain_index

  Domain (begin) index.
- int * d_constraint_description
- std::map< int, int > _cuda_var_lookup

  Map from var id on host to var id on device.

5.56.1 Member Function Documentation

5.56.1.1 bool CudaCPModel::download_device_state ( ) [virtual]

Move the current state (set of domains) from device to host.

Returns

true if the dowload has been completed successfully AND no empty domains are present. False otherwise.

Note

update all variables into host.

Reimplemented in CudaCPModelSimple.

5.56.1.2 void CudaCPModel::finalize ( ) [virtual]

Finalizes the model. This method actually allocates the structures on the device.

Reimplemented from CPModel.

5.56.1.3 bool CudaCPModel::upload_device_state ( ) [virtual]

Move the current state (set of domains) from host to device.

Returns

true if the upload has been completed successfully. False otherwise.

Note

update all variables into device.

Reimplemented in CudaCPModelSimple.

5.56.2 Member Data Documentation

5.56.2.1 int* CudaCPModel::d_constraint_description [protected]

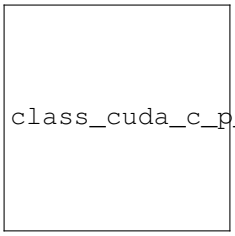
Information related to constraints: 1 - Type of constraint 2 - constraint's id 3 - scope size 4 - number of auxiliary arguments 5 - list of variables ids 6 - list of auxiliary arguments

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

- src/cuda_cp_model.h
- src/cuda_cp_model.cpp

5.57 CudaCPModelSimple Class Reference

Inheritance diagram for CudaCPModelSimple:



Public Member Functions

- bool upload_device_state () override
- bool download_device_state () override

Protected Member Functions

- bool alloc_variables ()

  Allocate domains on device.

Protected Attributes

- std::unordered_set< int > _bool_var_lookup

  Map for Boolean variables.

Additional Inherited Members

5.57.1 Member Function Documentation

5.57.1.1 bool CudaCPModelSimple::download_device_state ( ) [override], [virtual]

Move the current state (set of domains) from device to host.

Returns

true if the dowload has been completed successfully AND no empty domains are present. False otherwise.

Note

update all variables into host.
change domain representation to a domain representation used on Host

Reimplemented from CudaCPModel.

5.57.1.2 bool CudaCPModelSimple::upload_device_state ( ) [override], [virtual]

Move the current state (set of domains) from host to device.

Returns

true if the upload has been completed successfully, False otherwise.

Note

update all variables into device.
change domain representation to a domain representation used on Device

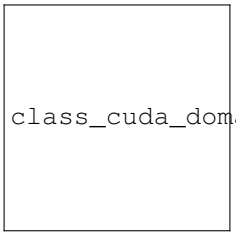
Reimplemented from CudaCPModel.

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

- src/cuda_cp_model_simple.h
- src/cuda_cp_model_simple.cpp

5.58 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_domain_status (void *concrete_domain)
- size_t get_domain_size () const
- const void * get_domain_status () const
- const int * 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)
- CudaDomainRepresenation 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

5.58.1 Member Function Documentation

5.58.1.1 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 mantains the current size.

Implements IntDomain.

5.58.1.2 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.

5.58.1.3 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.

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

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

Parameters

value	to check whether it is in the current domain.

Returns

true if value is in this domain, false othewise

Implements IntDomain.

5.58.1.5 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.

5.58.1.6 const int * CudaDomain::get_concrete_domain ( ) const

Gets a reference to the current internal representation.

Returns

a reference to a (cuda) concrete domain.

5.58.1.7 size_t CudaDomain::get_domain_size ( ) const [virtual]

Get the size if the current domain (internal representation).

Returns

number of bytes of the internal domain representaion.

Reimplemented from IntDomain.

5.58.1.8 const void * CudaDomain::get_domain_status ( ) const [virtual]

Get a pointer to the area of memory representing the current internal representation of this domain.

Returns

const void pointer to the current domain (internal representation)

Reimplemented from IntDomain.

5.58.1.9 size_t CudaDomain::get_size ( ) const [virtual]

Get domain size. It returns the currenst 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.

5.58.1.10 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

max	lower bound used to calculated the index of the bitmap

Returns

number of int used as bitmaps to represent max

5.58.1.11 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

max	lower bound used to calculated the index of the bitmap

Returns

number of int used as bitmaps to represent max

5.58.1.12 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: $|max - min + 1|$ 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

min	lower bound of the domain
max	upper bound of the domain

Returns

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

Implements IntDomain.

5.58.1.13 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

n	number of values to represent as bits

Returns

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

5.58.1.14 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.

5.58.1.15 void CudaDomain::set_bit_representation ( ) [protected]

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

5.58.1.16 void CudaDomain::set_bitlist_representation (int *num_list* = INT_BITLIST) [protected]

Switch to bitlist representation of domain.

Parameters

num_list	the number (positive) of bitlists. @ It changes only identifier in the REP field.

5.58.1.17 void CudaDomain::set_bounds (int *min*, int *max*)

The same as set_bounds. It shrinks the domain to {min, max}.

Parameters

min	lower bound
max	upper bound

5.58.1.18 void CudaDomain::set_domain_status (void * *concrete_domain*) [virtual]

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

Note

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

Reimplemented from IntDomain.

5.58.1.19 void CudaDomain::set_list_representation (int *num_list* = INT_LIST) [protected]

Switch to list representation of domain.

Parameters

num_list	the number (positive) of bitlists. @ It changes only identifier in the REP field.

5.58.1.20 bool CudaDomain::set_singleton (int *val*) [virtual]

Set domain as singleton as {val}.

Parameters

val	the value to set as singleton.

Implements IntDomain.

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

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

Parameters

min	lower bound
max	upper bound

Implements IntDomain.

5.58.1.22 void CudaDomain::switch_list_to_bitmaplist ( ) [protected]

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

Note

it doesn't work from bitmap to bitmap list.

5.58.2 Member Data Documentation

5.58.2.1 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

5.58.2.2 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.

5.58.2.3 size_t CudaDomain::_num_allocated_bytes [protected]

Total allocated bytes for representing the current domain.

5.58.2.4 size_t CudaDomain::_num_int_chunks [protected]

Total number of bitchunks.

Note

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

5.58.2.5 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.

5.58.2.6 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) kB = 47 * 1024 = 48128 Byte.

5.58.2.7 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 = ((MAX_BYTES_SIZE - 5 ∗ sizeof(int)) / sizeof(int))

Note

see CUDA specifications.

5.58.2.8 constexpr size_t CudaDomain::MAX_STATUS_SIZE = 5 ∗ sizeof(int) [static], [protected]

Number of Bytes needed for representing the current domain status.

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

Shared memory available.

Note

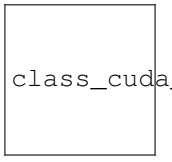
keep 1 kB less than the actual memory available.

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

- src/cuda_domain.h
- src/cuda_domain.cpp

5.59 CudaGenerator Class Reference

Inheritance diagram for CudaGenerator:



Public Member Functions

- VariablePtr get_variable (UTokenPtr)

  See "model_generator.h".
- ConstraintPtr get_constraint (UTokenPtr)

  See "model_generator.h".
- SearchEnginePtr get_search_engine (UTokenPtr)

  See "model_generator.h".
- ConstraintStorePtr get_store ()

  See "model_generator.h".

Protected Attributes

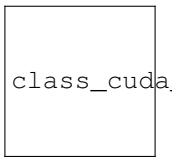
- std::string **_dbg**

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

- src/cuda_model_generator.h
- src/cuda_model_generator.cpp

5.60 CudaIntAbs Class Reference

Inheritance diagram for CudaIntAbs:



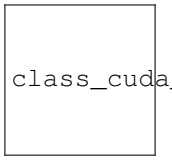
Additional Inherited Members

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

- src/cuda_int_abs.h

5.61 CudaIntDiv Class Reference

Inheritance diagram for CudaIntDiv:



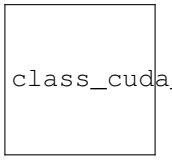
Additional Inherited Members

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

- src/cuda_int_div.h

5.62 CudaIntEq Class Reference

Inheritance diagram for CudaIntEq:



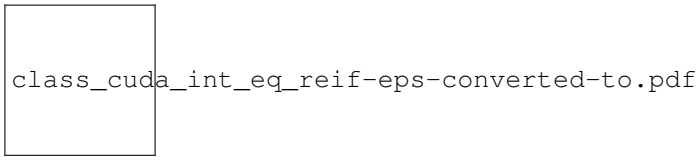
Additional Inherited Members

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

- src/cuda_int_eq.h

5.63 CudaIntEqReif Class Reference

Inheritance diagram for CudaIntEqReif:



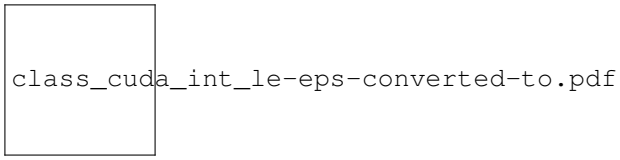
Additional Inherited Members

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

- src/cuda_int_eq_reif.h

5.64 CudaIntLe Class Reference

Inheritance diagram for CudaIntLe:



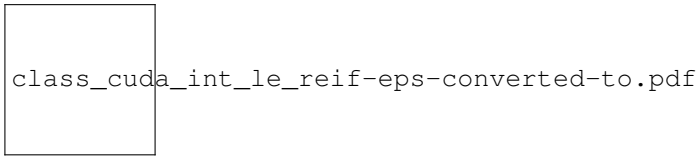
Additional Inherited Members

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

- src/cuda_int_le.h

5.65 CudaIntLeReif Class Reference

Inheritance diagram for CudaIntLeReif:



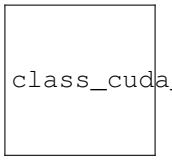
Additional Inherited Members

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

- src/cuda_int_le_reif.h

5.66 CudaIntLinEq Class Reference

Inheritance diagram for CudaIntLinEq:



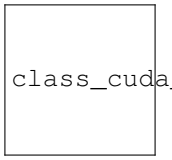
Additional Inherited Members

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

- src/cuda_int_lin_eq.h

5.67 CudaIntLinEqReif Class Reference

Inheritance diagram for CudaIntLinEqReif:



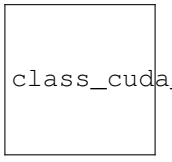
Additional Inherited Members

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

- src/cuda_int_lin_eq_reif.h

5.68 CudaIntLinLe Class Reference

Inheritance diagram for CudaIntLinLe:



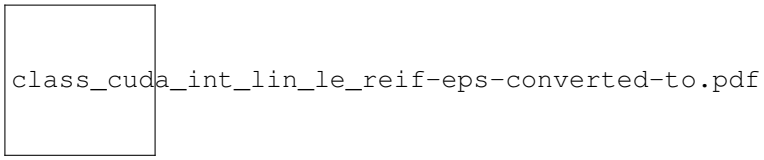
Additional Inherited Members

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

- src/cuda_int_lin_le.h

5.69 CudaIntLinLeReif Class Reference

Inheritance diagram for CudaIntLinLeReif:



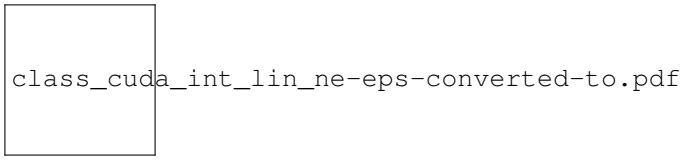
Additional Inherited Members

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

- src/cuda_int_lin_le_reif.h

5.70 CudaIntLinNe Class Reference

Inheritance diagram for CudaIntLinNe:



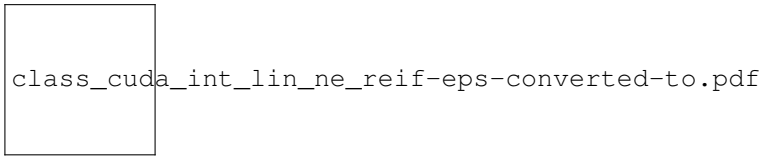
Additional Inherited Members

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

- src/cuda_int_lin_ne.h

5.71 CudaIntLinNeReif Class Reference

Inheritance diagram for CudaIntLinNeReif:



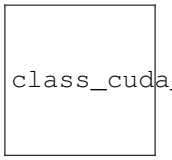
Additional Inherited Members

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

- src/cuda_int_lin_ne_reif.h

5.72 CudaIntLt Class Reference

Inheritance diagram for CudaIntLt:



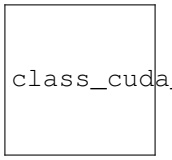
Additional Inherited Members

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

- src/cuda_int_lt.h

5.73 CudaIntLtReif Class Reference

Inheritance diagram for CudaIntLtReif:



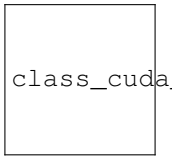
Additional Inherited Members

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

- src/cuda_int_lt_reif.h

5.74 CudaIntMaxC Class Reference

Inheritance diagram for CudaIntMaxC:



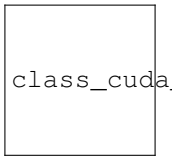
Additional Inherited Members

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

- src/cuda_int_max_c.h

5.75 CudaIntMinC Class Reference

Inheritance diagram for CudaIntMinC:



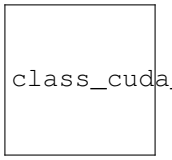
Additional Inherited Members

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

- src/cuda_int_min_c.h

5.76 CudaIntMod Class Reference

Inheritance diagram for CudaIntMod:



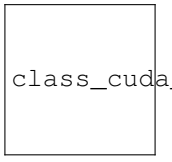
Additional Inherited Members

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

- src/cuda_int_mod.h

5.77 CudaIntNe Class Reference

Inheritance diagram for CudaIntNe:



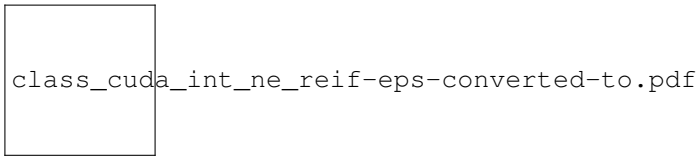
Additional Inherited Members

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

- src/cuda_int_ne.h

5.78 CudaIntNeReif Class Reference

Inheritance diagram for CudaIntNeReif:



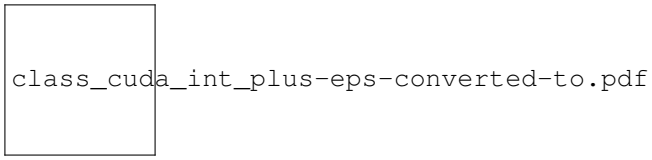
Additional Inherited Members

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

- src/cuda_int_ne_reif.h

5.79 CudaIntPlus Class Reference

Inheritance diagram for CudaIntPlus:



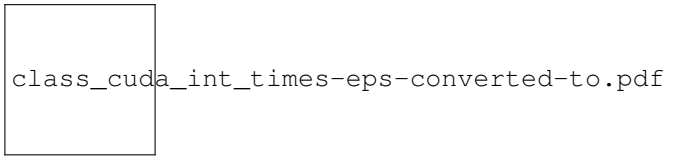
Additional Inherited Members

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

- src/cuda_int_plus.h

5.80 CudaIntTimes Class Reference

Inheritance diagram for CudaIntTimes:



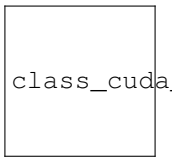
Additional Inherited Members

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

- src/cuda_int_times.h

5.81 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.

5.81.1 Constructor & Destructor Documentation

5.81.1.1 CudaMementoState::CudaMementoState (IntDomainPtr *int_domain*)

Constructor for Cuda Memento.

Parameters

int_domain	a reference to a int domain from which get the internal domain's representation.

5.81.2 Member Function Documentation

5.81.2.1 void CudaMementoState::set_memento (IntDomainPtr *int_domain*)

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

Parameters

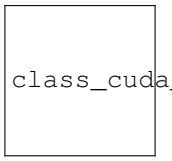
int_domain	a reference to the domain to update.

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

- src/cuda_memento_state.h
- src/cuda_memento_state.cpp

5.82 CudaSetCard Class Reference

Inheritance diagram for CudaSetCard:



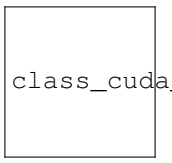
Additional Inherited Members

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

- src/cuda_set_card.h

5.83 CudaSetDiff Class Reference

Inheritance diagram for CudaSetDiff:



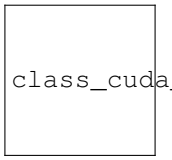
Additional Inherited Members

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

- src/cuda_set_diff.h

5.84 CudaSetEq Class Reference

Inheritance diagram for CudaSetEq:



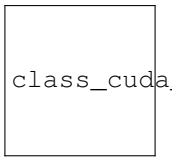
Additional Inherited Members

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

- src/cuda_set_eq.h

5.85 CudaSetEqReif Class Reference

Inheritance diagram for CudaSetEqReif:



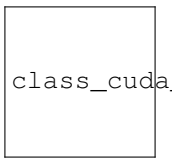
Additional Inherited Members

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

- src/cuda_set_eq_reif.h

5.86 CudaSetIn Class Reference

Inheritance diagram for CudaSetIn:



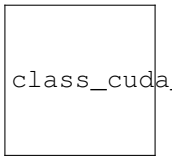
Additional Inherited Members

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

- src/cuda_set_in.h

5.87 CudaSetInReif Class Reference

Inheritance diagram for CudaSetInReif:



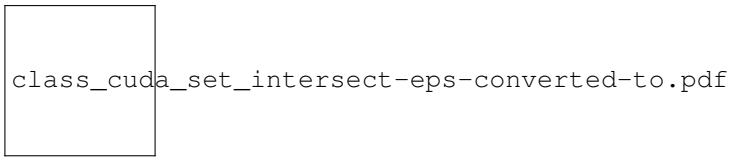
Additional Inherited Members

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

- src/cuda_set_in_reif.h

5.88 CudaSetIntersect Class Reference

Inheritance diagram for CudaSetIntersect:



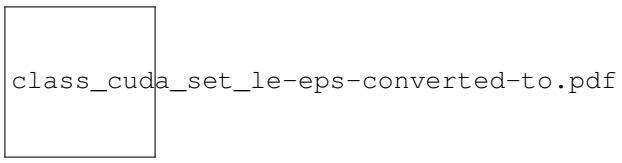
Additional Inherited Members

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

- src/cuda_set_intersect.h

5.89 CudaSetLe Class Reference

Inheritance diagram for CudaSetLe:



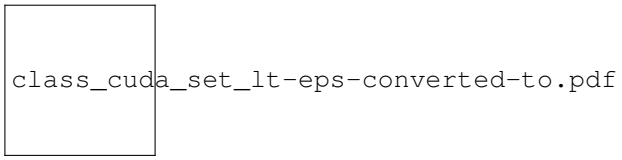
Additional Inherited Members

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

- src/cuda_set_le.h

5.90 CudaSetLt Class Reference

Inheritance diagram for CudaSetLt:



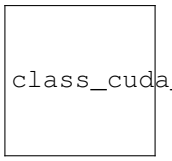
Additional Inherited Members

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

- src/cuda_set_lt.h

5.91 CudaSetNe Class Reference

Inheritance diagram for CudaSetNe:



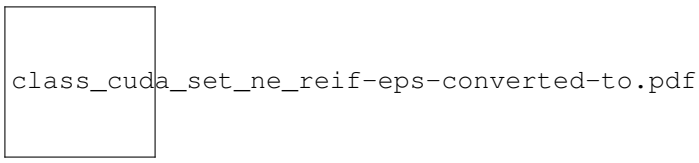
Additional Inherited Members

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

- src/cuda_set_ne.h

5.92 CudaSetNeReif Class Reference

Inheritance diagram for CudaSetNeReif:



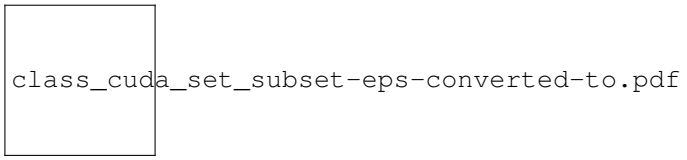
Additional Inherited Members

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

- src/cuda_set_ne_reif.h

5.93 CudaSetSubset Class Reference

Inheritance diagram for CudaSetSubset:



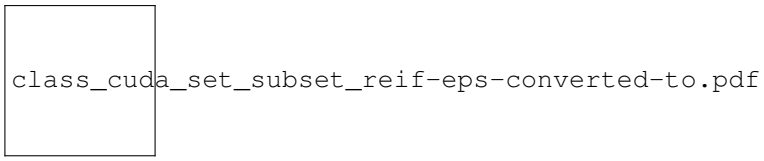
Additional Inherited Members

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

- src/cuda_set_subset.h

5.94 CudaSetSubsetReif Class Reference

Inheritance diagram for CudaSetSubsetReif:



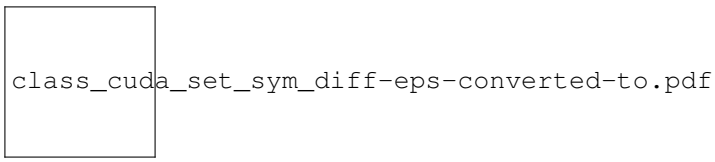
Additional Inherited Members

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

- src/cuda_set_subset_reif.h

5.95 CudaSetSymDiff Class Reference

Inheritance diagram for CudaSetSymDiff:



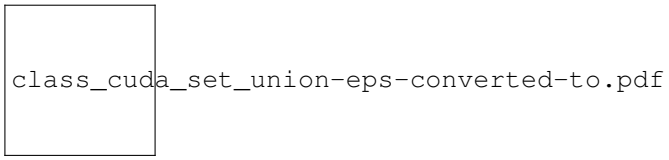
Additional Inherited Members

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

- src/cuda_set_sym_diff.h

5.96 CudaSetUnion Class Reference

Inheritance diagram for CudaSetUnion:



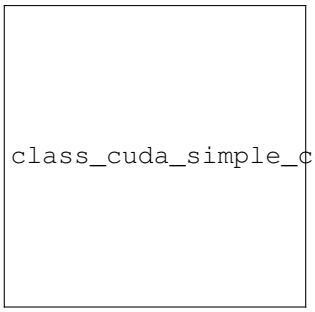
Additional Inherited Members

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

- src/cuda_set_union.h

5.97 CudaSimpleConstraintStore Class Reference

Inheritance diagram for CudaSimpleConstraintStore:



Public Member Functions

- CudaSimpleConstraintStore ()
- void set_prop_loop_out (int loop_out=1)
- int get_prop_loop_out () const

  Get the number of times the propagation loop is invoked.
- bool consistency () override
- virtual void finalize (CudaCPModel *ptr)

Protected Member Functions

- bool move_states_to_device ()
- bool move_states_from_device ()
- void move_queue_to_device ()

  Copy the current queue of constraints on device.
- virtual void dev_consistency ()

  Invoke the kernel which performs consistency on device.

Protected Attributes

- size_t * _d_constraint_queue

  Constraint queue: constraints to be propagated on device.
- size_t _scope_state_size

  Number of bytes needed to store the state of the variables in the scope of a constraint.
- int _loop_out

  Propagation loop out parameter.
- std::vector< size_t > _h_constraint_queue

  Constraint queue on device.
- CudaCPModel * _cp_model_ptr

5.97.1 Constructor & Destructor Documentation

5.97.1.1 CudaSimpleConstraintStore::CudaSimpleConstraintStore ( )

Default constructor. It initializes the internal data structures of this constraint store.

5.97.2 Member Function Documentation

5.97.2.1 bool CudaSimpleConstraintStore::consistency ( ) [override], [virtual]

Verify and propagate consistency of constraints in the constraint queue.

Note

consistency if performed in parallel on device.

Implements ConstraintStore.

5.97.2.2 void CudaSimpleConstraintStore::finalize (CudaCPModel * *ptr*) [virtual]

Allocate memory on device.

Parameters

num_cons	total number of constraints.

Note

if num_cons == 0, this function will use the parameter "_number_of_constraints" of SimpleConstraintStore.

5.97.2.3 bool CudaSimpleConstraintStore::move_states_from_device ( ) [protected]

Copy the current states (domains) from device to host

Returns

true if the copy has been completed successfully AND no domain is empty. False otherwise.

Note

copy is performed synchronously.

5.97.2.4 bool CudaSimpleConstraintStore::move_states_to_device ( ) [protected]

Copy the current states (domains) on device

Returns

true if the copy has been completed successfully. False otherwise.

Note

copy is performed synchronously.

5.97.2.5 void CudaSimpleConstraintStore::set_prop_loop_out (int *loop_out* = 1)

Sets the number of iterations to perform to propagate constrains on device. This represents the number of time the propagation kernel is invoked (it may not reach the fix point if the number of propagations is small).

Parameters

loop_out	integer value representing the number of times the propagation kernel is invoked.

Note

default is 1.

5.97.3 Member Data Documentation

5.97.3.1 CudaCPModel* CudaSimpleConstraintStore::_cp_model_ptr [protected]

Pointer to the current CP_model.

Note

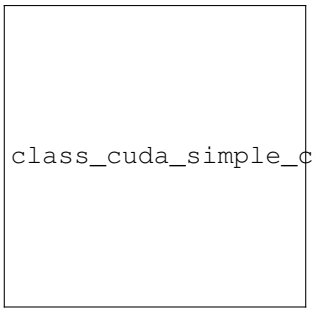
this is used to move data from-to device

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

- src/cuda_simple_constraint_store.h
- src/cuda_simple_constraint_store.cpp

5.98 CudaSimpleConstraintStore1b1c Class Reference

Inheritance diagram for CudaSimpleConstraintStore1b1c:



Protected Member Functions

- void dev_consistency ()

  Invoke the kernel which performs consistency on device.

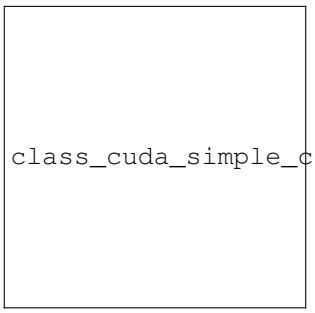
Additional Inherited Members

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

- src/cuda_simple_constraint_store_1b1c.h
- src/cuda_simple_constraint_store_1b1c.cpp

5.99 CudaSimpleConstraintStoreSeq Class Reference

Inheritance diagram for CudaSimpleConstraintStoreSeq:



Protected Member Functions

- void dev_consistency ()

  Invoke the kernel which performs consistency on device.

Additional Inherited Members

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

- src/cuda_simple_constraint_store_seq.h
- src/cuda_simple_constraint_store_seq.cpp

5.100 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

5.100.1 Constructor & Destructor Documentation

5.100.1.1 CudaVariable::CudaVariable ( )

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

5.100.1.2 CudaVariable::CudaVariable (int *idv*)

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

Parameters

idv	identifier to give to the variable

5.100.2 Member Function Documentation

5.100.2.1 void CudaVariable::restore_state ( ) [override], [virtual]

Restore a state from the current state hold by the BacktrackableObject.

Note

override backtrackable object methods.

Implements BacktrackableObject.

5.100.2.2 void CudaVariable::set_domain ( ) [virtual]

Set domain's bounds. If no bounds are provided, an unbounded domain (int) is istantiated. 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.

5.100.2.3 void CudaVariable::set_domain (int *lw*, int *ub*) [virtual]

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

Parameters

lw	lower bound
ub	upper bound

Implements IntVariable.

5.100.2.4 void CudaVariable::set_domain (std::vector< std::vector< int > > *elems*) [virtual]

Set domain's elements. A domain {d_1, ..., d_n} is generated.

Parameters

elems	vector of vectors (subsets) of domain's elements

Todo implement set of sets of elements.

Implements IntVariable.

5.100.2.5 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:

- src/cuda_variable.h
- src/cuda_variable.cpp

5.101 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.

5.101.1 Constructor & Destructor Documentation

5.101.1.1 DataStore::DataStore (std::string *in_file*) [protected]

Constructor.

Parameters

in_file	file path of the model to parse.

5.101.2 Member Function Documentation

5.101.2.1 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.

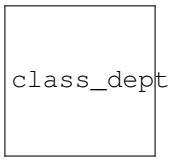
Implemented in CPStore.

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

- src/data_store.h
- src/data_store.cpp

5.102 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
- void set_time_watcher (bool watcher_on) 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 _peak_depth

  Peak depth reached so far.
- 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 _time_watcher

  Specifies if the time-watcher is 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.

5.102.1 Member Function Documentation

5.102.1.1 size_t DepthFirstSearch::get_backtracks ( ) const [override], [virtual]

Returns the number of backtracks performed by the search.

Returns

the number of backtracks.

Implements SearchEngine.

5.102.1.2 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.

5.102.1.3 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.

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

Return the n^th solution found if any.

Parameters

n_sol	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.

5.102.1.5 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.

5.102.1.6 void DepthFirstSearch::init_search ( ) [protected], [virtual]

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

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

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

Parameters

var	the index of the variable (not grounded) to assign.

Returns

true if the solution was found.

Implements SearchEngine.

5.102.1.8 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.

5.102.1.9 void DepthFirstSearch::print_solution (size_t *sol_idx*) const [override], [virtual]

Print on standard output a solutions represented by its index.

Parameters

sol_idx	the index of the solution to print.

Note

first solution has index 1.

Implements SearchEngine.

5.102.1.10 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.

5.102.1.11 void DepthFirstSearch::set_backtrack_manager (BacktrackManagerPtr *bkt_manager*) [virtual]

Sets a backtrackable manager to this class.

Parameters

bkt_manager	a reference to a backtrack manager.

Implements SearchEngine.

5.102.1.12 void DepthFirstSearch::set_backtrack_out (size_t *out_b*) [override], [virtual]

Set a maximum number of backtracks to perform during search.

Parameters

the	number of backtracks to consider as a limit during the search.

Implements SearchEngine.

5.102.1.13 void DepthFirstSearch::set_debug (bool *debug_on*) [virtual]

Set debug options.

Parameters

debug_on	boolean value indicating if debug should be enabled.

Note

default debug is off.

Implements SearchEngine.

5.102.1.14 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

a	reference to a heuristic.

Implements SearchEngine.

5.102.1.15 void DepthFirstSearch::set_nodes_out (size_t *out_n*) [override], [virtual]

Set a maximum number of nodes to visit during search.

Parameters

the	number of nodes to visit and to be considered as a limit during the search.

Implements SearchEngine.

5.102.1.16 void DepthFirstSearch::set_solution_limit (size_t *num_sol*) [override], [virtual]

Set maximum number of solutions to be found.

Parameters

num_sol	the maximum number of solutions.

Note

-1 states for "find all solutions".

Implements SearchEngine.

5.102.1.17 void DepthFirstSearch::set_solution_manager (SolutionManager * *sol_manager*) [virtual]

Set a solution manager for this search engine.

Parameters

a	reference to a solution manager.

Implements SearchEngine.

5.102.1.18 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

a	reference to a constraint store.

Implements SearchEngine.

5.102.1.19 void DepthFirstSearch::set_time_watcher (bool *watcher_on*) [override], [virtual]

Sets the time-watcher, i.e., it stores the computational times of consistency, backtrack, etc.

Parameters

watcher_on	the boolean value that turns on the of turns off the time watcher.

Implements SearchEngine.

5.102.1.20 void DepthFirstSearch::set_timeout_limit (double *timeout*) [override], [virtual]

Imposes a timeoutlimit.

Parameters

timeout	timeout limit.

Note

-1 for no timeout.

Implements SearchEngine.

5.102.1.21 void DepthFirstSearch::set_trail_debug (bool *debug_on*) [virtual]

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

Parameters

debug_on	boolean value indicating if debug should be enabled.

Implements SearchEngine.

5.102.1.22 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

the	number of wrong decisions to set as a limit during the search.

Implements SearchEngine.

5.102.2 Member Data Documentation

5.102.2.1 size_t DepthFirstSearch::_num_backtracks [protected]

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

5.102.2.2 size_t DepthFirstSearch::_num_nodes [protected]

Stores the number of search nodes explored during search.

5.102.2.3 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:

- src/depth_first_search.h
- src/depth_first_search.cpp

5.103 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.

5.103.1 Member Function Documentation

5.103.1.1 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.

5.103.1.2 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.

5.103.1.3 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.

5.103.1.4 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.

5.103.1.5 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.

5.103.1.6 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.

5.103.1.7 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.

5.103.1.8 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.

5.103.1.9 void Domain::set_type (DomainType *dt*)

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

Parameters

dt	domain type of type DomainType

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

- src/domain.h
- src/domain.cpp

5.104 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 void set_domain_status (void ∗concrete_domain)
- virtual std::pair< size_t, const void ∗ > get_domain_status () const
- virtual std::string get_string_representation () const

Protected Attributes

- IntDomainPtr **_domain**

5.104.1 Member Function Documentation

5.104.1.1 size_t DomainIterator::domain_size ( ) const [virtual]

Returns the current domain's size.

Returns

current domain's size.

5.104.1.2 std::pair< size_t, const void * > DomainIterator::get_domain_status ( ) const [virtual]

Returns a pointer to an area of memory storing the current internal domain.

Returns

a pair containing: (1) the size of the current internal domain's representation; (2) a void pointer to an area of memory where the current internal representation of the domain associated to this iterator is store.

5.104.1.3 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.

5.104.1.4 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).

5.104.1.5 int DomainIterator::max_val ( ) const [virtual]

Returns the current maximal value in domain.

Returns

the maximum value belonging to the domain.

5.104.1.6 int DomainIterator::min_val ( ) const [virtual]

Returns the current minimal value in domain.

Returns

the minimum value belonging to the domain.

5.104.1.7 int DomainIterator::random_val ( ) const [virtual]

Returns a random value from domain.

Returns

the a random value belonging to the domain.

5.104.1.8 void DomainIterator::set_domain_status (void * *concrete_domain*) [virtual]

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

Note

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

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

- src/domain_iterator.h
- src/domain_iterator.cpp

5.105 FactoryCPModel Class Reference

Static Public Member Functions

- static CPModel * get_cp_model (CPModelType cp_model)

  Get the right parser based on the input.

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

- src/factory_cp_model.h

5.106 FactoryCStore Class Reference

Static Public Member Functions

- static ConstraintStorePtr get_cstore (bool on_device=false, int type=0)

5.106.1 Member Function Documentation

5.106.1.1 static ConstraintStorePtr FactoryCStore::get_cstore (bool *on_device* = false, int *type* = 0) [inline], [static]

Get the right instance of constraint store based on input options.

Parameters

on_device,if	True it generates a constraint store for device propagation, otherwise it generates the constraint store for host propagation.

type,type	of constraint store to generate.

Todo propagation 1 block per variable

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

- src/factory_cstore.h

5.107 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:

- src/factory_generator.h

5.108 FactoryParser Class Reference

Static Public Member Functions

- static Parser * get_parser (ParserType pt)

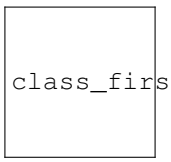
  Get the right parser based on the input.

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

- src/factory_parser.h

5.109 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

5.109.1 Member Function Documentation

5.109.1.1 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.

5.109.1.2 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.

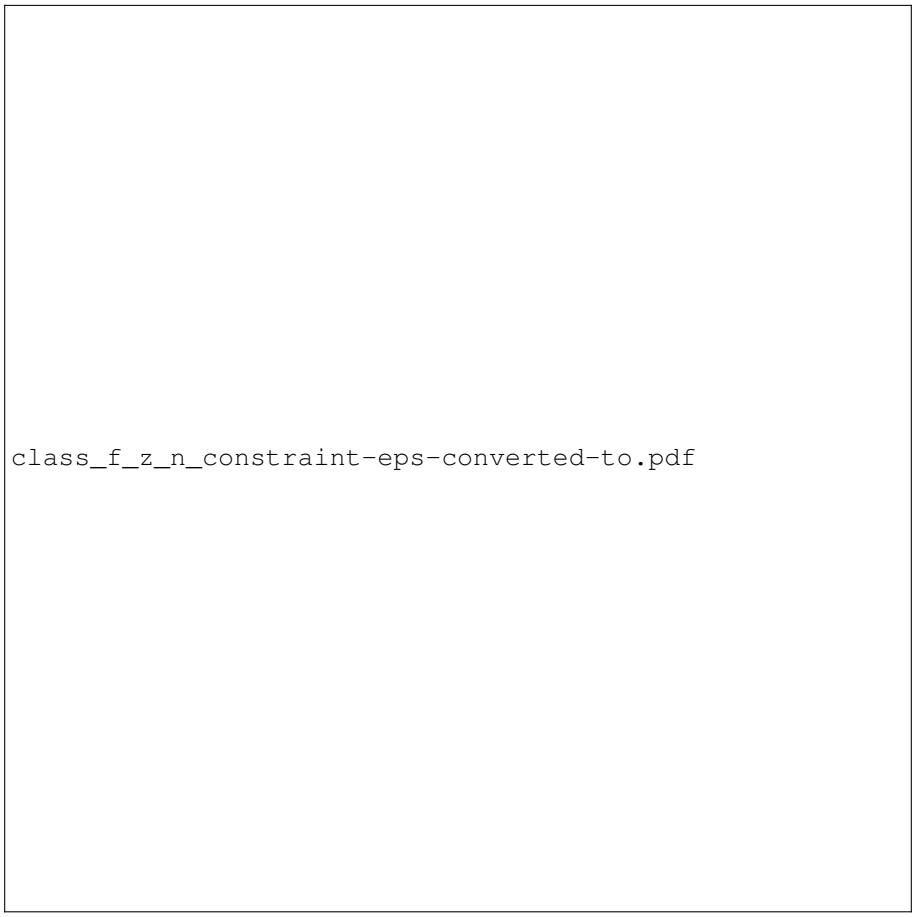
Implements VariableChoiceMetric.

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

- src/first_fail_metric.h
- src/first_fail_metric.cpp

5.110 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_LT** = "int_lt"
- static const std::string **INT_LT_REIF** = "int_lt_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.

5.110.1 Constructor & Destructor Documentation

5.110.1.1 FZNConstraint::FZNConstraint (std::string *name*) [protected]

Base constructor.

Parameters

name	the name of the FlatZinc constraint.
vars	the vector of (shared) pointers to the variables in the scope of this constraint.
args	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.

5.110.2 Member Function Documentation

5.110.2.1 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.

5.110.2.2 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 IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubset↩Reif, SetSymDiff, and SetUnion.

5.110.2.3 FZNConstraintType FZNConstraint::int_to_type (int *number_id*) [static]

It converts a number_id name to the correspondent FZNConstraintType type.

Parameters

number_id	the number id of the FlatZinc constraint.

Returns

the type of the FlatZinc constraint.

5.110.2.4 int FZNConstraint::name_to_id (std::string *c_name*) [static]

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

Parameters

c_name	name of a FlatZinc constraint.

Returns

the number_id correspondent to name.

5.110.2.5 void FZNConstraint::remove_constraint ( ) [virtual]

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

Implements Constraint.

5.110.2.6 bool FZNConstraint::satisfied ( ) [override], [virtual]

It checks if the constraint is satisfied.

Returns

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

Note

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

Implements Constraint.

Reimplemented in IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubset↩Reif, SetSymDiff, and SetUnion.

5.110.2.7 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

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

Implemented in IntEq, IntLe, IntNe, IntLt, IntLinNe, IntLinEq, ArrayBoolAnd, ArrayBoolElement, ArrayBoolOr, ArrayIntElement, ArraySetElement, ArrayVarBoolElement, ArrayVarIntElement, ArrayVarSetElement, Bool2Int, BoolAnd, BoolClause, BoolEq, BoolEqReif, BoolLe, BoolLeReif, BoolLt, BoolLtReif, BoolNot, BoolOr, BoolXor, IntAbs, IntDiv, IntEqReif, IntLeReif, IntLinEqReif, IntLinLe, IntLinLeReif, IntLinNeReif, IntLtReif, IntMaxC, IntMinC, IntMod, IntNeReif, IntPlus, IntTimes, SetCard, SetDiff, SetEq, SetEqReif, SetIn, SetInReif, SetIntersect, SetLe, SetLt, SetNe, SetNeReif, SetSubset, SetSubsetReif, SetSymDiff, and SetUnion.

5.110.2.8 int FZNConstraint::type_to_int (FZNConstraintType *c_type*) [static]

It converts a FZNConstraintType to the correspondent integer type.

Parameters

c_type	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:

- src/fzn_constraint.h
- src/fzn_constraint.cpp

5.111 FZNConstraintFactory Class Reference

Static Public Member Functions

- static ConstraintPtr get_fzn_constraint_shr_ptr (std::string c_name, std::vector< VariablePtr > vars, std::vector< std::string > args)

5.111.1 Member Function Documentation

5.111.1.1 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

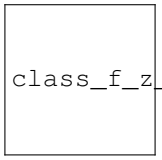
c_name	the FlatZinc name of the constraint to instantiate.
vars	the vector of (shared) pointer to the FD variables in the scope of the constraint to instantiate.
args	the vector of strings representing the auxiliary arguments needed by the constraint to instantiate in order to be propagated.

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

- src/fzn_constraint_generator.h

5.112 FZNParser Class Reference

Inheritance diagram for FZNParser:



Public Member Functions

- **FZNParser** (std::string ifile)
- bool parse ()

  Parses the file filling the internal state of the parser.
- bool more_variables () const

  Ask whether there are more variables to get.
- bool more_constraints () const

  Ask whether there are more constraits to get.
- bool more_search_engines () const

  Ask whether there are more search engines to get.
- UTokenPtr get_variable ()
- UTokenPtr get_constraint ()
- UTokenPtr get_search_engine ()
- void print () const

  Print info about the parser.

Additional Inherited Members

5.112.1 Member Function Documentation

5.112.1.1 UTokenPtr FZNParser::get_constraint ( ) [virtual]

Get a "constraint" token.

Returns

token pointer to a "constraint" token.

Note

the returned pointer is a unique_ptr.

Implements Parser.

5.112.1.2 UTokenPtr FZNParser::get_search_engine ( ) [virtual]

Get a "search_engine" token.

Returns

token pointer to a "search_engine" token.

Note

the returned pointer is a unique_ptr.

Implements Parser.

5.112.1.3 UTokenPtr FZNParser::get_variable ( ) [virtual]

Get a "variable" token.

Returns

token pointer to a "variable" token.

Note

the returned pointer is a unique_ptr.

Implements Parser.

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

- src/fzn_parser.h
- src/fzn_parser.cpp

5.113 FZNSearchFactory Class Reference

Static Public Member Functions

- static SearchEnginePtr get_fzn_search_shr_ptr (std::vector< Variable * > variables, TokenSol *search_tkn)

5.113.1 Member Function Documentation

5.113.1.1 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

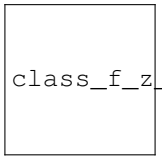
variables	a vector of pointers to all the variables in the model.
search_tkn	reference to a search token in order to define the right instance of search engine.

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

- src/fzn_search_generator.h

5.114 FZNTokenization Class Reference

Inheritance diagram for FZNTokenization:



Public Member Functions

- void set_internal_state (std::string str)
- UTokenPtr get_token ()

Additional Inherited Members

5.114.1 Member Function Documentation

5.114.1.1 UTokenPtr FZNTokenization::get_token ( ) [virtual]

Specialized method: It actually gets the right token according to the FlatZinc format. Analysis is perfomed on "_c_token".

Implements Tokenization.

5.114.1.2 void FZNTokenization::set_internal_state (std::string *str*)

This function is used for testing. Sets the current line to tokenize.

Parameters

	string of at most 250 chars to tokenize.

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

- src/fzn_tokenization.h
- src/fzn_tokenization.cpp

5.115 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.

5.115.1 Member Function Documentation

5.115.1.1 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 huristic).

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.

5.115.1.2 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

idx	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.

5.115.1.3 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:

- src/heuristic.h
- src/heuristic.cpp

5.116 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.

5.116.1 Constructor & Destructor Documentation

5.116.1.1 IdGenerator::IdGenerator ( ) [protected]

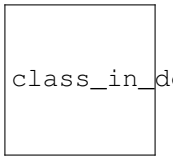
Protected constructor: a client cannot instantiate Singleton directly.

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

- src/id_generator.h
- src/id_generator.cpp

5.117 InDomain Class Reference

Inheritance diagram for InDomain:



Public Member Functions

- int metric_value (Variable *var)
- void print () const
  Print info about this value choice metric.

Additional Inherited Members

5.117.1 Member Function Documentation

5.117.1.1 int InDomain::metric_value (Variable * *var*) [virtual]

Gets value to assign to var using indomain choice.

Parameters

var	the (pointer to) variable for which a value if needed.

Returns

the value to assign to var.

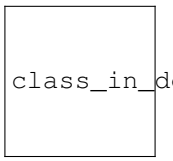
Implements ValueChoiceMetric.

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

- src/indomain_metric.h
- src/indomain_metric.cpp

5.118 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

5.118.1 Member Function Documentation

5.118.1.1 int InDomainMax::metric_value (Variable * *var*) [virtual]

Gets value to assign to var using indomain_max choice.

Parameters

var	the (pointer to) variable for which a value if needed.

Returns

the value to assign to var.

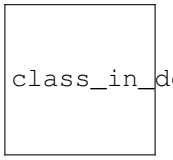
Implements ValueChoiceMetric.

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

- src/indomain_max_metric.h
- src/indomain_max_metric.cpp

5.119 InDomainMedian Class Reference

Inheritance diagram for InDomainMedian:



Public Member Functions

- int metric_value (Variable *var)
- void print () const

  Print info about this value choice metric.

Additional Inherited Members

5.119.1 Member Function Documentation

5.119.1.1 int InDomainMedian::metric_value (Variable * *var*) [virtual]

Gets value to assign to var using indomain_median choice.

Parameters

var	the (pointer to) variable for which a value if needed.

Returns

the value to assign to var.

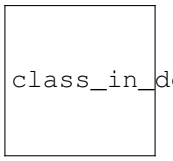
Implements ValueChoiceMetric.

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

- src/indomain_median_metric.h
- src/indomain_median_metric.cpp

5.120 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

5.120.1 Member Function Documentation

5.120.1.1 int InDomainMin::metric_value (Variable * *var*) [virtual]

Gets value to assign to var using indomain_min choice.

Parameters

var	the (pointer to) variable for which a value if needed.

Returns

the value to assign to var.

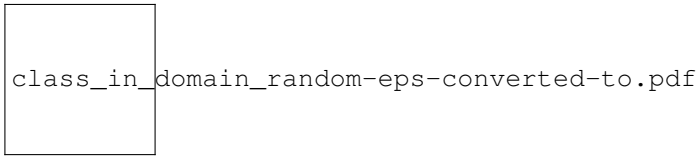
Implements ValueChoiceMetric.

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

- src/indomain_min_metric.h
- src/indomain_min_metric.cpp

5.121 InDomainRandom Class Reference

Inheritance diagram for InDomainRandom:



Public Member Functions

- int metric_value (Variable *var)
- void print () const

  Print info about this value choice metric.

Additional Inherited Members

5.121.1 Member Function Documentation

5.121.1.1 int InDomainRandom::metric_value (Variable * *var*) [virtual]

Gets value to assign to var using indomain_random choice.

Parameters

var	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:

- src/indomain_random_metric.h
- src/indomain_random_metric.cpp

5.122 InputData Class Reference

Public Member Functions

- **InputData** (const InputData &other)=delete
- InputData & **operator=** (const InputData &other)=delete
- 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[])

5.122.1 Constructor & Destructor Documentation

5.122.1.1 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!

5.122.2 Member Function Documentation

5.122.2.1 std::string InputData::get_in_file ( ) const

Get input file (path to).

Returns

the path where the input file is located.

5.122.2.2 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.

5.122.2.3 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.

5.122.2.4 double InputData::timeout ( ) const

Returns the timeout limit set by the user (default: inf).

Returns

the timeout limit.

5.122.2.5 bool InputData::timer ( ) const

Informs about the time option.

Returns

true if timer is on.

5.122.2.6 bool InputData::verbose ( ) const

Informs about the verbose option.

Returns

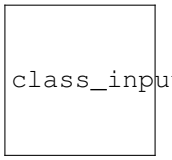
true if verbose is on.

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

- src/input_data.h
- src/input_data.cpp

5.123 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

5.123.1 Member Function Documentation

5.123.1.1 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.

5.123.1.2 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.

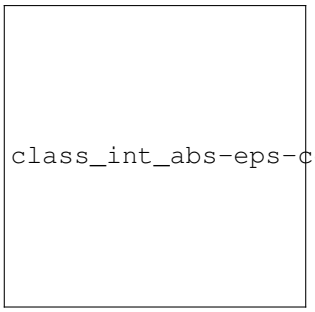
Implements VariableChoiceMetric.

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

- src/input_order_metric.h
- src/input_order_metric.cpp

5.124 IntAbs Class Reference

Inheritance diagram for IntAbs:



Public Member Functions

- IntAbs ()
- IntAbs (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.124.1 Constructor & Destructor Documentation

5.124.1.1 IntAbs::IntAbs ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.124.1.2 IntAbs::IntAbs (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.

5.124.2 Member Function Documentation

5.124.2.1 const std::vector< VariablePtr > IntAbs::scope ( ) const [override], [virtual]

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

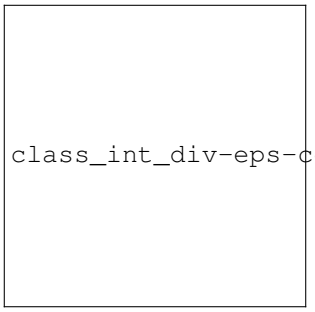
Implements Constraint.

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

- src/int_abs.h
- src/int_abs.cpp

5.125 IntDiv Class Reference

Inheritance diagram for IntDiv:



Public Member Functions

- IntDiv ()
- IntDiv (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.125.1 Constructor & Destructor Documentation

5.125.1.1 IntDiv::IntDiv ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.125.1.2 IntDiv::IntDiv (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.

5.125.2 Member Function Documentation

5.125.2.1 const std::vector< VariablePtr > IntDiv::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:

- src/int_div.h
- src/int_div.cpp

5.126 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 set_domain_status (void *domain)
- virtual size_t get_domain_size () const
- virtual const void * get_domain_status () const
- 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

5.126.1 Member Function Documentation

5.126.1.1 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

val	element to add to the current domain.

Implemented in CudaDomain.

5.126.1.2 virtual bool IntDomain::contains (int *value*) const [pure virtual]

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

Parameters

value	to check whether it is in the current domain.

Returns

true if value is in this domain, false othewise

Implemented in CudaDomain.

5.126.1.3 size_t IntDomain::get_domain_size ( ) const [virtual]

Get the size if the current domain (internal representation).

Returns

number of bytes of the internal domain representaion.

Note

default is 0.

Reimplemented in CudaDomain.

5.126.1.4 const void * IntDomain::get_domain_status ( ) const [virtual]

Get a pointer to the area of memory representing the current internal representation of this domain.

Returns

const void pointer to the current domain (internal representation)

Note

default is nullptr

Reimplemented in CudaDomain.

5.126.1.5 virtual void IntDomain::in_max (int *max*) [pure virtual]

It updates the domain according to the maximum value.

Parameters

max	domain value.

Implemented in CudaDomain.

5.126.1.6 virtual void IntDomain::in_min (int *min*) [pure virtual]

It updates the domain according to the minimum value.

Parameters

min	domain value.

Implemented in CudaDomain.

5.126.1.7 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.

5.126.1.8 void IntDomain::set_domain_status (void * *domain*) [virtual]

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

Note

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

Reimplemented in CudaDomain.

5.126.1.9 virtual bool IntDomain::set_singleton (int *val*) [pure virtual]

Set domain to the singleton element given in input.

Parameters

val	the value to set as singleton

Returns

true if the domain has been set to singleton, false otherwise.

Implemented in CudaDomain.

5.126.1.10 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

lower	lower bound value

upper	upper bound value

Implemented in CudaDomain.

5.126.1.11 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

val	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:

- src/int_domain.h
- src/int_domain.cpp

5.127 IntEq Class Reference

Inheritance diagram for IntEq:

Public Member Functions

- IntEq ()
- IntEq (std::vector< VariablePtr > vars, std::vector< std::string > args)
- IntEq (int x, int y)
- IntEq (IntVariablePtr x, int y)
- IntEq (int x, IntVariablePtr y)
- IntEq (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 = y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.127.1 Constructor & Destructor Documentation

5.127.1.1 IntEq::IntEq ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.127.1.2 IntEq::IntEq (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.

5.127.1.3 IntEq::IntEq (int *x*, int *y*)

Basic constructor: it checks if x = y.

Parameters

x	an integer value.
y	an integer value.

5.127.1.4 IntEq::IntEq (IntVariablePtr *x*, int *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	an integer value.

Note

It subtracts the value y from the domain of the variable x if x has a domain defined on integers.

5.127.1.5 IntEq::IntEq (int *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	an integer value.
y	(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.

5.127.1.6 IntEq::IntEq (IntVariablePtr *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	(pointer to) a FD variable.

5.127.2 Member Function Documentation

5.127.2.1 const std::vector< VariablePtr > IntEq::scope ( ) const [override], [virtual]

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

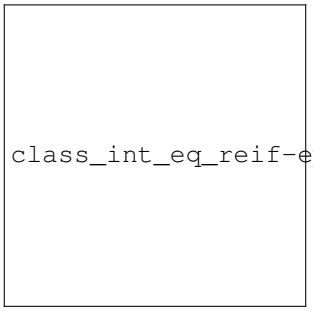
Implements Constraint.

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

- src/int_eq.h
- src/int_eq.cpp

5.128 IntEqReif Class Reference

Inheritance diagram for IntEqReif:



Public Member Functions

- IntEqReif ()
- IntEqReif (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.128.1 Constructor & Destructor Documentation

5.128.1.1 IntEqReif::IntEqReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.128.1.2 IntEqReif::IntEqReif (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.

5.128.2 Member Function Documentation

5.128.2.1 const std::vector< VariablePtr > IntEqReif::scope ( ) const [override], [virtual]

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

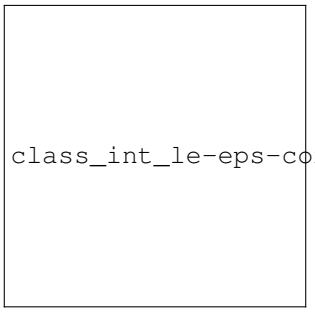
Implements Constraint.

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

- src/int_eq_reif.h
- src/int_eq_reif.cpp

5.129 IntLe Class Reference

Inheritance diagram for IntLe:



Public Member Functions

- IntLe ()
- IntLe (std::vector< VariablePtr > vars, std::vector< std::string > args)
- IntLe (int x, int y)
- IntLe (IntVariablePtr x, int y)
- IntLe (int x, IntVariablePtr y)
- IntLe (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 != y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.129.1 Constructor & Destructor Documentation

5.129.1.1 IntLe::IntLe ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.129.1.2 IntLe::IntLe (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.

5.129.1.3 IntLe::IntLe (int *x*, int *y*)

Basic constructor: it checks if x != y.

Parameters

x	an integer value.
y	an integer value.

5.129.1.4 IntLe::IntLe (IntVariablePtr *x*, int *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	an integer value.

Note

It subtracts the value y from the domain of the variable x if x has a domain defined on integers.

5.129.1.5 IntLe::IntLe (int *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	an integer value.
y	(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.

5.129.1.6 IntLe::IntLe (IntVariablePtr *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	(pointer to) a FD variable.

5.129.2 Member Function Documentation

5.129.2.1 bool IntLe::satisfied ( ) [override], [virtual]

It checks if x != y.

It checks if x <= y.

Reimplemented from FZNConstraint.

5.129.2.2 const std::vector< VariablePtr > IntLe::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:

- src/int_le.h
- src/int_le.cpp

5.130 IntLeReif Class Reference

Inheritance diagram for IntLeReif:

Public Member Functions

- IntLeReif ()
- IntLeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.130.1 Constructor & Destructor Documentation

5.130.1.1 IntLeReif::IntLeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.130.1.2 IntLeReif::IntLeReif (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.

5.130.2 Member Function Documentation

5.130.2.1 const std::vector< VariablePtr > IntLeReif::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:

- src/int_le_reif.h
- src/int_le_reif.cpp

5.131 IntLinEq Class Reference

Inheritance diagram for IntLinEq:

Public Member Functions

- IntLinEq ()
- IntLinEq (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 != y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.131.1 Constructor & Destructor Documentation

5.131.1.1 IntLinEq::IntLinEq ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.131.1.2 IntLinEq::IntLinEq (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.

5.131.2 Member Function Documentation

5.131.2.1 void IntLinEq::consistency ( ) [override], [virtual]

It performs domain consistency.

This function propagates on bounds.

See also

Apt K. Principles of constraint programming (CUP, 2003) pp 196.

Reimplemented from FZNConstraint.

5.131.2.2 const std::vector< VariablePtr > IntLinEq::scope ( ) const [override], [virtual]

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

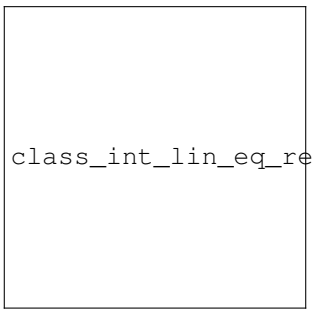
Implements Constraint.

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

- src/int_lin_eq.h
- src/int_lin_eq.cpp

5.132 IntLinEqReif Class Reference

Inheritance diagram for IntLinEqReif:



Public Member Functions

- IntLinEqReif ()
- IntLinEqReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.132.1 Constructor & Destructor Documentation

5.132.1.1 IntLinEqReif::IntLinEqReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.132.1.2 IntLinEqReif::IntLinEqReif (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.

5.132.2 Member Function Documentation

5.132.2.1 const std::vector< VariablePtr > IntLinEqReif::scope ( ) const [override], [virtual]

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

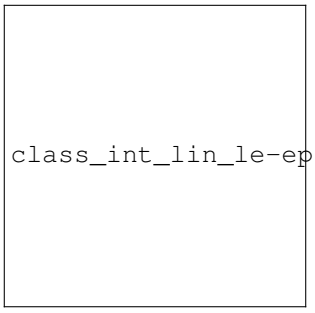
Implements Constraint.

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

- src/int_lin_eq_reif.h
- src/int_lin_eq_reif.cpp

5.133 IntLinLe Class Reference

Inheritance diagram for IntLinLe:



Public Member Functions

- IntLinLe ()
- IntLinLe (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.133.1 Constructor & Destructor Documentation

5.133.1.1 IntLinLe::IntLinLe ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.133.1.2 IntLinLe::IntLinLe (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.

5.133.2 Member Function Documentation

5.133.2.1 const std::vector< VariablePtr > IntLinLe::scope ( ) const [override], [virtual]

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

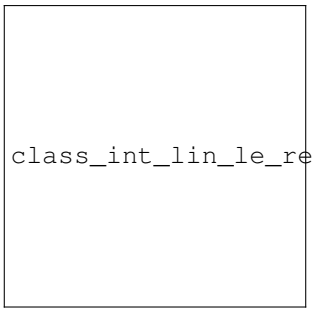
Implements Constraint.

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

- src/int_lin_le.h
- src/int_lin_le.cpp

5.134 IntLinLeReif Class Reference

Inheritance diagram for IntLinLeReif:



Public Member Functions

- IntLinLeReif ()
- IntLinLeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.134.1 Constructor & Destructor Documentation

5.134.1.1 IntLinLeReif::IntLinLeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.134.1.2 IntLinLeReif::IntLinLeReif (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.

5.134.2 Member Function Documentation

5.134.2.1 const std::vector< VariablePtr > IntLinLeReif::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:

- src/int_lin_le_reif.h
- src/int_lin_le_reif.cpp

5.135 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 != y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.135.1 Constructor & Destructor Documentation

5.135.1.1 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.

5.135.1.2 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.

5.135.2 Member Function Documentation

5.135.2.1 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.

5.135.2.2 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).

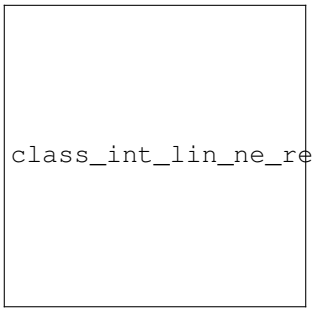
Implements Constraint.

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

- src/int_lin_ne.h
- src/int_lin_ne.cpp

5.136 IntLinNeReif Class Reference

Inheritance diagram for IntLinNeReif:



Public Member Functions

- IntLinNeReif ()
- IntLinNeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.136.1 Constructor & Destructor Documentation

5.136.1.1 IntLinNeReif::IntLinNeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.136.1.2 IntLinNeReif::IntLinNeReif (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.

5.136.2 Member Function Documentation

5.136.2.1 const std::vector< VariablePtr > IntLinNeReif::scope ( ) const [override], [virtual]

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

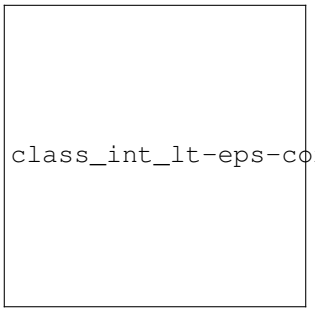
Implements Constraint.

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

- src/int_lin_ne_reif.h
- src/int_lin_ne_reif.cpp

5.137 IntLt Class Reference

Inheritance diagram for IntLt:



Public Member Functions

- IntLt ()
- IntLt (std::vector< VariablePtr > vars, std::vector< std::string > args)
- IntLt (int x, int y)

- IntLt (IntVariablePtr x, int y)
- IntLt (int x, IntVariablePtr y)
- IntLt (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 != y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.137.1 Constructor & Destructor Documentation

5.137.1.1 IntLt::IntLt ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.137.1.2 IntLt::IntLt (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.

5.137.1.3 IntLt::IntLt (int *x*, int *y*)

Basic constructor: it checks if x != y.

Parameters

x	an integer value.
y	an integer value.

5.137.1.4 IntLt::IntLt (IntVariablePtr *x*, int *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.

y	an integer value.

Note

It subtracts the value y from the domain of the variable x if x has a domain defined on integers.

5.137.1.5 IntLt::IntLt (int *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	an integer value.
y	(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.

5.137.1.6 IntLt::IntLt (IntVariablePtr *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	(pointer to) a FD variable.

5.137.2 Member Function Documentation

5.137.2.1 bool IntLt::satisfied ( ) [override], [virtual]

It checks if x != y.

It checks if x $<$ y.

Reimplemented from FZNConstraint.

5.137.2.2 const std::vector< VariablePtr > IntLt::scope ( ) const [override], [virtual]

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

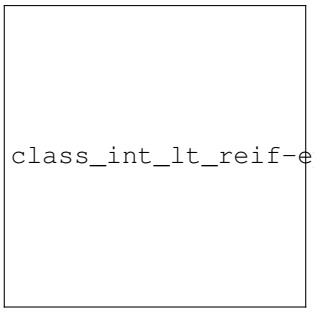
Implements Constraint.

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

- src/int_lt.h
- src/int_lt.cpp

5.138 IntLtReif Class Reference

Inheritance diagram for IntLtReif:



Public Member Functions

- IntLtReif ()
- IntLtReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.138.1 Constructor & Destructor Documentation

5.138.1.1 IntLtReif::IntLtReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.138.1.2 IntLtReif::IntLtReif (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.

5.138.2 Member Function Documentation

5.138.2.1 const std::vector< VariablePtr > IntLtReif::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:

- src/int_lt_reif.h
- src/int_lt_reif.cpp

5.139 IntMaxC Class Reference

Inheritance diagram for IntMaxC:

Public Member Functions

- IntMaxC ()
- IntMaxC (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.139.1 Constructor & Destructor Documentation

5.139.1.1 IntMaxC::IntMaxC ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.139.1.2 IntMaxC::IntMaxC (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.

5.139.2 Member Function Documentation

5.139.2.1 const std::vector< VariablePtr > IntMaxC::scope ( ) const [override], [virtual]

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

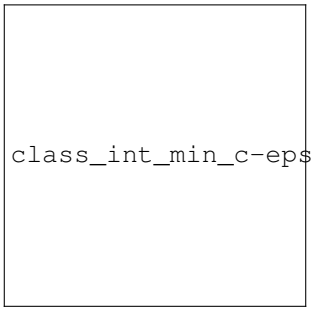
Implements Constraint.

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

- src/int_max_c.h
- src/int_max_c.cpp

5.140 IntMinC Class Reference

Inheritance diagram for IntMinC:



Public Member Functions

- IntMinC ()
- IntMinC (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.140.1 Constructor & Destructor Documentation

5.140.1.1 IntMinC::IntMinC ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.140.1.2 IntMinC::IntMinC (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.

5.140.2 Member Function Documentation

5.140.2.1 const std::vector< VariablePtr > IntMinC::scope ( ) const [override], [virtual]

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

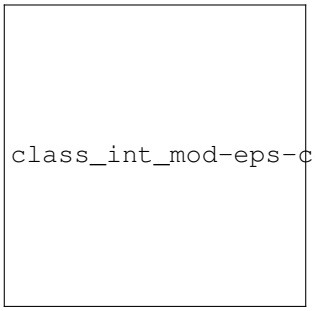
Implements Constraint.

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

- src/int_min_c.h
- src/int_min_c.cpp

5.141 IntMod Class Reference

Inheritance diagram for IntMod:



Public Member Functions

- IntMod ()
- IntMod (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.141.1 Constructor & Destructor Documentation

5.141.1.1 IntMod::IntMod ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.141.1.2 IntMod::IntMod (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.

5.141.2 Member Function Documentation

5.141.2.1 const std::vector< VariablePtr > IntMod::scope ( ) const [override], [virtual]

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

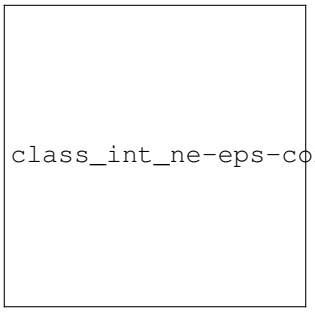
Implements Constraint.

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

- src/int_mod.h
- src/int_mod.cpp

5.142 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 != y.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.142.1 Constructor & Destructor Documentation

5.142.1.1 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.

5.142.1.2 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.

5.142.1.3 IntNe::IntNe (int *x*, int *y*)

Basic constructor: it checks if x != y.

Parameters

x	an integer value.
y	an integer value.

5.142.1.4 IntNe::IntNe (IntVariablePtr *x*, int *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.

y	an integer value.

Note

It subtracts the value y from the domain of the variable x if x has a domain defined on integers.

5.142.1.5 IntNe::IntNe (int *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	an integer value.
y	(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.

5.142.1.6 IntNe::IntNe (IntVariablePtr *x*, IntVariablePtr *y*)

Constructor.

Parameters

x	(pointer to) a FD variable.
y	(pointer to) a FD variable.

5.142.2 Member Function Documentation

5.142.2.1 const std::vector< VariablePtr > 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:

- src/int_ne.h
- src/int_ne.cpp

5.143 IntNeReif Class Reference

Inheritance diagram for IntNeReif:

Public Member Functions

- IntNeReif ()
- IntNeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.143.1 Constructor & Destructor Documentation

5.143.1.1 IntNeReif::IntNeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.143.1.2 IntNeReif::IntNeReif (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.

5.143.2 Member Function Documentation

5.143.2.1 const std::vector< VariablePtr > IntNeReif::scope ( ) const [override], [virtual]

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

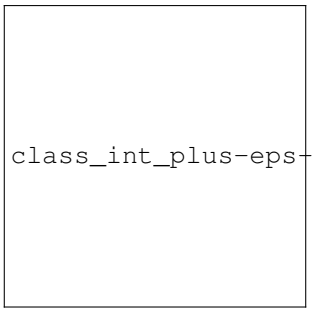
Implements Constraint.

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

- src/int_ne_reif.h
- src/int_ne_reif.cpp

5.144 IntPlus Class Reference

Inheritance diagram for IntPlus:



Public Member Functions

- IntPlus ()
- IntPlus (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.144.1 Constructor & Destructor Documentation

5.144.1.1 IntPlus::IntPlus ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.144.1.2 IntPlus::IntPlus (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.

5.144.2 Member Function Documentation

5.144.2.1 const std::vector< VariablePtr > IntPlus::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:

- src/int_plus.h
- src/int_plus.cpp

5.145 IntTimes Class Reference

Inheritance diagram for IntTimes:

Public Member Functions

- IntTimes ()
- IntTimes (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.145.1 Constructor & Destructor Documentation

5.145.1.1 IntTimes::IntTimes ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.145.1.2 IntTimes::IntTimes (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.

5.145.2 Member Function Documentation

5.145.2.1 const std::vector< VariablePtr > IntTimes::scope ( ) const [override], [virtual]

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

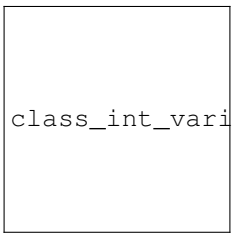
Implements Constraint.

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

- src/int_times.h
- src/int_times.cpp

5.146 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)
- virtual void notify_observers ()
- 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 ()

Protected Attributes

- IntDomainPtr _domain_ptr
- BacktrackManagerPtr _backtack_manager

Additional Inherited Members

5.146.1 Member Function Documentation

5.146.1.1 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.

5.146.1.2 void IntVariable::in_max (int *max*) [virtual]

It updates the domain according to the maximum value.

Parameters

max	domain value.

5.146.1.3 void IntVariable::in_min (int *min*) [virtual]

It updates the domain according to the minimum value.

Parameters

min	domain value.

5.146.1.4 bool IntVariable::is_empty ( ) const [virtual]

It checks if the domain is empty.

Returns

true if variable domain is empty. false otherwise.

Implements Variable.

5.146.1.5 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.

5.146.1.6 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.

5.146.1.7 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.

5.146.1.8 void IntVariable::notify_backtrack_manager ( ) [protected], [virtual]

Notifies the backtrack manager that a change happened on this variable, so the manager can manage this backtrackable object.

5.146.1.9 void IntVariable::notify_observers ( ) [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.

Reimplemented from Variable.

5.146.1.10 void IntVariable::set_backtrack_manager (BacktrackManagerPtr *bkt_manager*) [virtual]

Set a backtrack manager for this backtrackable object.

Parameters

bkt_manager	a reference to the backtrack manager that will manage this backtrackable object.

5.146.1.11 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.

5.146.1.12 virtual void IntVariable::set_domain ( ) [pure virtual]

Set domain's bounds. If no bounds are provided, an unbounded domain (int) is istantiated. 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.

5.146.1.13 virtual void IntVariable::set_domain (int *lw,* int *ub*) [pure virtual]

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

Parameters

lw	lower bound
ub	upper bound

Implemented in CudaVariable.

5.146.1.14 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

elems	vector of vectors (subsets) of domain's elements

Todo implement set of sets of elements.

Implemented in CudaVariable.

5.146.1.15 void IntVariable::set_domain_type (DomainType *dt*) [virtual]

Set domain according to the specific variable implementation.

Note

: different types of variable

Parameters

dt	domain type of type DomainType to set to the current variable

Implements Variable.

5.146.1.16 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

lower	lower bound value
upper	upper bound value

5.146.1.17 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

val	the value to subtract from the current domain

Returns

true if succeed, false otherwise.

5.146.2 Member Data Documentation

5.146.2.1 BacktrackManagerPtr IntVariable::_backtack_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.

5.146.2.2 IntDomainPtr IntVariable::_domain_ptr [protected]

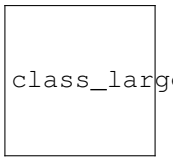
Reference to the domain of the variable. IntDomain for IntVariable

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

- src/int_variable.h
- src/int_variable.cpp

5.147 Largest Class Reference

Inheritance diagram for Largest:



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

5.147.1 Member Function Documentation

5.147.1.1 int Largest::compare (double *metric,* Variable * *var*) [virtual]

Compare a metric value and a variable. Metric is given by their largest value in their domain.

Implements VariableChoiceMetric.

5.147.1.2 int Largest::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

Compare variables w.r.t. their metrics. Metric is given by their largest value in their domain.

Implements VariableChoiceMetric.

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

- src/largest_metric.h
- src/largest_metric.cpp

5.148 Logger Class Reference

Public Member Functions

- **Logger** (const Logger &other)=delete
- Logger & **operator=** (const Logger &other)=delete
- template<typename T >
  Logger & **operator**<< (const T &v)
- Logger const & **operator**<< (std::ostream &(*F)(std::ostream &))
- 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::ostream &out, std::string log_file="")

  Constructor get (static) instance.

Protected Member Functions

- **Logger** (std::ostream &out, std::string="")
- virtual std::string get_time_stamp ()

  Time stamp.
- template<typename T >
  void log (const T &v, bool flush=false)

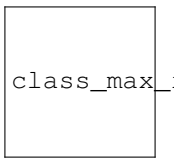
  Print log on stdout or file.
- template<typename T >
  void **oflog** (const T &v)

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

- src/logger.h
- src/logger.cpp

5.149 MaxRegret Class Reference

Inheritance diagram for MaxRegret:



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

5.149.1 Member Function Documentation

5.149.1.1 int MaxRegret::compare (double *metric,* Variable * *var*) [virtual]

Compare a metric value and a variable. Metric is given by their largest difference between the max and min values in its domain.

Implements VariableChoiceMetric.

5.149.1.2 int MaxRegret::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

Compare variables w.r.t. their metrics. Metric is given by their largest difference between the max and min values in its domain.

Implements VariableChoiceMetric.

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

- src/max_regret_metric.h
- src/max_regret_metric.cpp

5.150 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**

5.150.1 Member Function Documentation

5.150.1.1 virtual MementoState* Memento::get_state ( ) [inline], [protected], [virtual]

Get the current state saved as memento.

Returns

the current state/memento.

5.150.1.2 virtual void Memento::set_state (MementoState * *state*) [inline], [protected], [virtual]

Set a state as a memento object.

Parameters

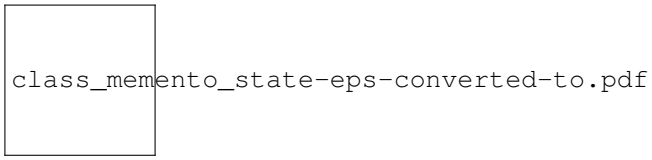
state	the current state representing a mememnto object.

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

- src/memento.h

5.151 MementoState Class Reference

Inheritance diagram for MementoState:



Public Member Functions

- virtual void print () const =0

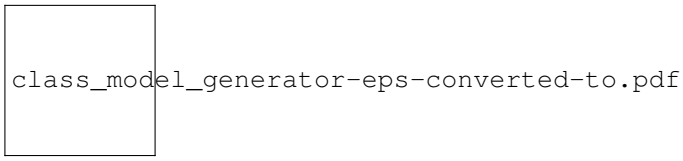
  Print information about this memento state.

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

- src/memento_state.h

5.152 ModelGenerator Class Reference

Inheritance diagram for ModelGenerator:



Public Member Functions

- virtual VariablePtr get_variable (UTokenPtr)=0
- virtual ConstraintPtr get_constraint (UTokenPtr)=0
- virtual SearchEnginePtr get_search_engine (UTokenPtr)=0
- virtual ConstraintStorePtr get_store ()=0

5.152.1 Member Function Documentation

5.152.1.1 virtual ConstraintPtr ModelGenerator::get_constraint (UTokenPtr) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

Parameters

TokenPtr	pointer to the token describing a constraint. If the token does not correspond to the object to instantiate, it returns nullptr.

Implemented in CudaGenerator.

5.152.1.2 virtual SearchEnginePtr ModelGenerator::get_search_engine (UTokenPtr) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

Parameters

TokenPtr	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.

5.152.1.3 virtual ConstraintStorePtr ModelGenerator::get_store ( ) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

Parameters

TokenPtr	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.

5.152.1.4 virtual VariablePtr ModelGenerator::get_variable (UTokenPtr) [pure virtual]

These methods create the instances of the objects and return the correspondent (shared) pointers to them.

Parameters

TokenPtr	pointer to the token describing a variable. If the token does not correspond to the object to instantiate, it returns nullptr.

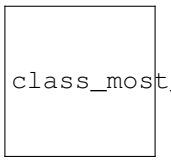
Implemented in CudaGenerator.

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

- src/model_generator.h

5.153 MostConstrained Class Reference

Inheritance diagram for MostConstrained:



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

5.153.1 Member Function Documentation

5.153.1.1 int MostConstrained::compare (double *metric,* Variable * *var*) [virtual]

Compare a metric value and a variable. Metric is given by their smallest domains, breaking ties using the number of constraints.

Implements VariableChoiceMetric.

5.153.1.2 int MostConstrained::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

Compare variables w.r.t. their metrics. Metric is given by their smallest domains, breaking ties using the number of constraints.

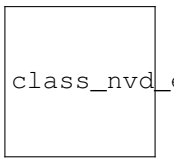
Implements VariableChoiceMetric.

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

- src/most_constrained_metric.h
- src/most_constrained_metric.cpp

5.154 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.

5.154.1 Constructor & Destructor Documentation

5.154.1.1 NvdException::NvdException (const char * *msg* = " ")

Constructor.

Parameters

msg	the message related to the exception.

5.154.1.2 NvdException::NvdException (const char * *msg*, const char * *file*)

Constructor.

Parameters

msg	the message related to the exception.
file	where the excpetion has been raised.

5.154.1.3 NvdException::NvdException (const char * *msg*, const char * *file*, int *line*)

Constructor.

Parameters

msg	the message related to the exception.
file	where the excpetion has been raised.
line	of code where the excpetion has been raised.

5.154.2 Member Function Documentation

5.154.2.1 const char * NvdException::what ( ) const [virtual], [noexcept]

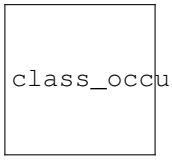
Overwrite the what method to print other information about the exception.

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

- src/nvd_exception.h
- src/nvd_exception.cpp

5.155 Occurence Class Reference

Inheritance diagram for Occurence:



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

5.155.1 Member Function Documentation

5.155.1.1 int Occurence::compare (double *metric,* Variable * *var*) [virtual]

Compare a metric value and a variable. Metric is given by their number of attached constraints.

Implements VariableChoiceMetric.

5.155.1.2 int Occurence::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

Compare variables w.r.t. their metrics. Metric is given by their number of attached constraints.

Implements VariableChoiceMetric.

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

- src/occurence_metric.h
- src/occurence_metric.cpp

5.156 ParamData Class Reference

Public Member Functions

- **ParamData** (std::string in_file)
- void set_param_path (std::string path)

  Set input (parameters) path.
- virtual void set_parameters ()

  Read parameters from file.
- std::string get_param_path () const
- bool **search_get_debug** () const
- bool **search_get_trail_debug** () const
- bool **search_get_time_watcher** () const
- int **search_get_solution_limit** () const
- int **search_get_backtrack_limit** () const
- int **search_get_nodes_limit** () const
- int **search_get_wrong_decision_limit** () const
- double **search_get_timeout** () const
- bool **cstore_get_consistency** () const
- bool **cstore_get_satisfiability** () const
- int **cstore_get_dev_loop_out** () const
- int **cstore_type_to_int** (CudaPropParam ctype) const
- CudaPropParam **cstore_int_to_type** (int ctype) const
- CudaPropParam **cstore_get_dev_propagation** () const
- virtual void **print** () const

Protected Member Functions

- void open ()
  Open parameters file.
- void close ()
  Close parameters file.
- std::string get_param_value (std::string line)
  Get parameter value.
- virtual void set_default_parameters ()
  Set default parameters.
- virtual void read_params ()
  Read parameters from file.
- virtual void set_search_parameters (std::string &line)
  Search engine parameters.
- virtual void set_constraint_engine_parameters (std::string &line)
  Constraint store parameters.

5.156.1 Member Function Documentation

5.156.1.1 string ParamData::get_param_path ( ) const

Get path where parameters file is located.

Returns

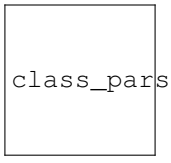
the path where the parameters file is located.

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

- src/param_data.h
- src/param_data.cpp

5.157 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 failed to parse the file.

- virtual bool more_tokens ()
- virtual void open ()
- virtual void close ()
- virtual std::string get_next_token ()
- virtual UTokenPtr get_next_content ()
- virtual bool parse ()=0
- virtual bool more_variables () const =0
- virtual bool **more_constraints** () const =0
- virtual bool **more_search_engines** () const =0
- virtual UTokenPtr get_variable ()=0
- virtual UTokenPtr **get_constraint** ()=0
- virtual UTokenPtr **get_search_engine** ()=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

  Positions in stream (file)

5.157.1 Member Function Documentation

5.157.1.1 void Parser::close ( ) [virtual]

Close the file.

Note

: alternating open() and close() the client can decided how much text has to be parsed. For example, parse only the first n lines of the text file.

5.157.1.2 UTokenPtr Parser::get_next_content ( ) [virtual]

Returns a token at a time from the set of tokens currently stored in the parser. This is equivalent to call get_variable (); get_constraint (); get_search_engine (); until no tokens are available.

Returns

a (unique_ptr) pointer to the current token read from input

Note

if no token can be read, it returns a null, empty object.

5.157.1.3 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).

5.157.1.4 virtual UTokenPtr 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.

Returns

return a unique_ptr

Implemented in FZNParser.

5.157.1.5 bool Parser::more_tokens ( ) [virtual]

Check if the internal status has more tokens to give back to the client.

5.157.1.6 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.

5.157.1.7 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.

5.157.1.8 virtual bool Parser::parse ( ) [pure virtual]

Parses the file. It fills the internal state with tokens created by reading the model.

Returns

True if parsed succeeded, False otherwise.

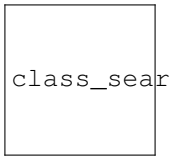
Implemented in FZNParser.

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

- src/parser.h
- src/parser.cpp

5.158 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 set_time_watcher (bool watcher_on)=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.

5.158.1 Member Function Documentation

5.158.1.1 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.

5.158.1.2 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.

5.158.1.3 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.

5.158.1.4 virtual std::vector<DomainPtr> SearchEngine::get_solution (int *n_sol*) const [pure virtual]

Return the n^th solution found if any.

Parameters

n_sol	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.

5.158.1.5 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.

5.158.1.6 virtual bool SearchEngine::label (int *var*) [pure virtual]

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

Parameters

var	the index of the variable (not grounded) to assign.

Returns

true if the solution was found.

Implemented in DepthFirstSearch.

5.158.1.7 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.

5.158.1.8 virtual void SearchEngine::print_solution (size_t *sol_idx*) const [pure virtual]

Print on standard output a solutions represented by its index.

Parameters

sol_idx	the index of the solution to print.

Note

first solution has index 1.

Implemented in DepthFirstSearch.

5.158.1.9 virtual void SearchEngine::set_backtrack_manager (BacktrackManagerPtr *bkt_manager*) [pure virtual]

Sets a backtrackable manager to this class.

Parameters

bkt_manager	a reference to a backtrack manager.

Implemented in DepthFirstSearch.

5.158.1.10 virtual void SearchEngine::set_backtrack_out (size_t *out_b*) [pure virtual]

Set a maximum number of backtracks to perform during search.

Parameters

the	number of backtracks to consider as a limit during the search.

Implemented in DepthFirstSearch.

5.158.1.11 virtual void SearchEngine::set_debug (bool *debug_on*) [pure virtual]

Set debug option.

Parameters

debug_on	boolean value indicating if debug should be enabled.

Implemented in DepthFirstSearch.

5.158.1.12 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

a	reference to a heuristic.

Implemented in DepthFirstSearch.

5.158.1.13 virtual void SearchEngine::set_nodes_out (size_t *out_n*) [pure virtual]

Set a maximum number of nodes to visit during search.

Parameters

the	number of nodes to visit and to be considered as a limit during the search.

Implemented in DepthFirstSearch.

5.158.1.14 virtual void SearchEngine::set_solution_limit (size_t *num_sol*) [pure virtual]

Set maximum number of solutions to be found.

Parameters

num_sol	the maximum number of solutions.

Note

-1 for finding all solutions.

Implemented in DepthFirstSearch.

5.158.1.15 virtual void SearchEngine::set_solution_manager (SolutionManager * *sol_manager*) [pure virtual]

Set a solution manager for this search engine.

Parameters

a	reference to a solution manager.

Implemented in DepthFirstSearch.

5.158.1.16 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

a	reference to a constraint store.

Implemented in DepthFirstSearch.

5.158.1.17 virtual void SearchEngine::set_time_watcher (bool *watcher_on*) [pure virtual]

Sets the time-watcher, i.e., it stores the computational times of consistency, backtrack, etc.

Parameters

watcher_on	the boolean value that turns on the of turns off the time watcher.

Implemented in DepthFirstSearch.

5.158.1.18 virtual void SearchEngine::set_timeout_limit (double *timeout*) [pure virtual]

Imposes a timeoutlimit.

Parameters

timeout	timeout limit.

Note

-1 for no timeout.

Implemented in DepthFirstSearch.

5.158.1.19 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

debug_on	boolean value indicating if debug should be enabled.

Implemented in DepthFirstSearch.

5.158.1.20 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

the	number of wrong decisions to set as a limit during the search.

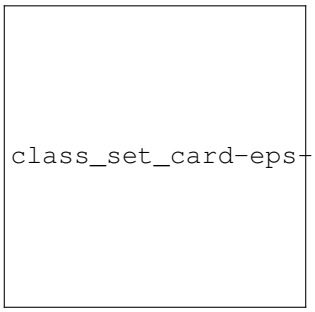
Implemented in DepthFirstSearch.

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

- src/search_engine.h

5.159 SetCard Class Reference

Inheritance diagram for SetCard:



Public Member Functions

- SetCard ()
- SetCard (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.159.1 Constructor & Destructor Documentation

5.159.1.1 SetCard::SetCard ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.159.1.2 SetCard::SetCard (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.

5.159.2 Member Function Documentation

5.159.2.1 const std::vector< VariablePtr > SetCard::scope ( ) const [override], [virtual]

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

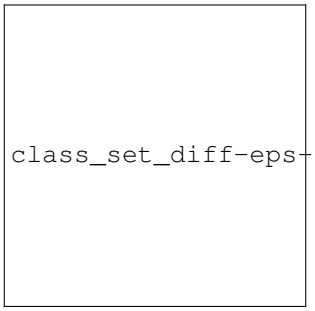
Implements Constraint.

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

- src/set_card.h
- src/set_card.cpp

5.160 SetDiff Class Reference

Inheritance diagram for SetDiff:



Public Member Functions

- SetDiff ()
- SetDiff (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.160.1 Constructor & Destructor Documentation

5.160.1.1 SetDiff::SetDiff ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.160.1.2 SetDiff::SetDiff (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.

5.160.2 Member Function Documentation

5.160.2.1 const std::vector< VariablePtr > SetDiff::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:

- src/set_diff.h
- src/set_diff.cpp

5.161 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

5.161.1 Member Function Documentation

5.161.1.1 EventType SetDomain::get_event ( ) const [virtual]

Get event on this domain

Todo implement this function

Implements Domain.

5.161.1.2 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

5.161.1.3 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.

5.161.1.4 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

elems	vector of domain's elements

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

- src/set_domain.h
- src/set_domain.cpp

5.162 SetEq Class Reference

Inheritance diagram for SetEq:

Public Member Functions

- SetEq ()
- SetEq (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.162.1 Constructor & Destructor Documentation

5.162.1.1 SetEq::SetEq ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.162.1.2 SetEq::SetEq (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.

5.162.2 Member Function Documentation

5.162.2.1 const std::vector< VariablePtr > SetEq::scope ( ) const [override], [virtual]

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

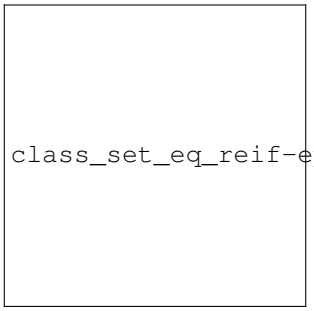
Implements Constraint.

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

- src/set_eq.h
- src/set_eq.cpp

5.163 SetEqReif Class Reference

Inheritance diagram for SetEqReif:



Public Member Functions

- SetEqReif ()
- SetEqReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.163.1 Constructor & Destructor Documentation

5.163.1.1 SetEqReif::SetEqReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.163.1.2 SetEqReif::SetEqReif (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.

5.163.2 Member Function Documentation

5.163.2.1 const std::vector< VariablePtr > SetEqReif::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:

- src/set_eq_reif.h
- src/set_eq_reif.cpp

5.164 SetIn Class Reference

Inheritance diagram for SetIn:

Public Member Functions

- SetIn ()
- SetIn (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.164.1 Constructor & Destructor Documentation

5.164.1.1 SetIn::SetIn ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.164.1.2 SetIn::SetIn (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.

5.164.2 Member Function Documentation

5.164.2.1 const std::vector< VariablePtr > SetIn::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:

- src/set_in.h
- src/set_in.cpp

5.165 SetInReif Class Reference

Inheritance diagram for SetInReif:

Public Member Functions

- SetInReif ()
- SetInReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.165.1 Constructor & Destructor Documentation

5.165.1.1 SetInReif::SetInReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.165.1.2 SetInReif::SetInReif (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.

5.165.2 Member Function Documentation

5.165.2.1 const std::vector< VariablePtr > SetInReif::scope ( ) const [override], [virtual]

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

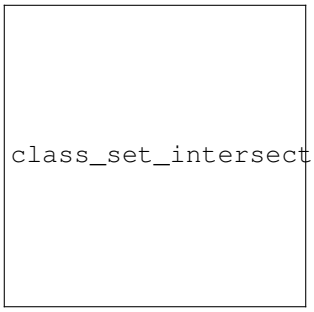
Implements Constraint.

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

- src/set_in_reif.h
- src/set_in_reif.cpp

5.166 SetIntersect Class Reference

Inheritance diagram for SetIntersect:



Public Member Functions

- SetIntersect ()
- SetIntersect (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.166.1 Constructor & Destructor Documentation

5.166.1.1 SetIntersect::SetIntersect ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.166.1.2 SetIntersect::SetIntersect (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.

5.166.2 Member Function Documentation

5.166.2.1 const std::vector< VariablePtr > SetIntersect::scope ( ) const [override], [virtual]

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

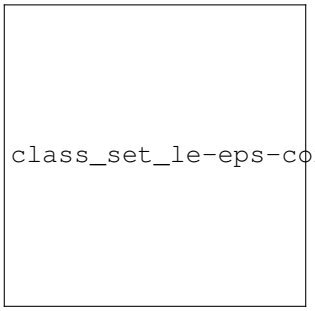
Implements Constraint.

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

- src/set_intersect.h
- src/set_intersect.cpp

5.167 SetLe Class Reference

Inheritance diagram for SetLe:



Public Member Functions

- SetLe ()
- SetLe (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.167.1 Constructor & Destructor Documentation

5.167.1.1 SetLe::SetLe ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.167.1.2 SetLe::SetLe (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.

5.167.2 Member Function Documentation

5.167.2.1 const std::vector< VariablePtr > SetLe::scope ( ) const [override], [virtual]

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

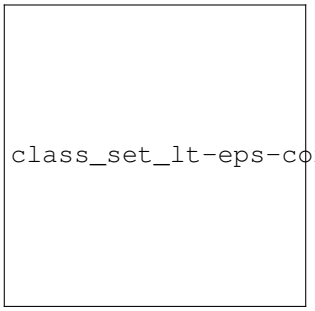
Implements Constraint.

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

- src/set_le.h
- src/set_le.cpp

5.168 SetLt Class Reference

Inheritance diagram for SetLt:



Public Member Functions

- SetLt ()
- SetLt (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.168.1 Constructor & Destructor Documentation

5.168.1.1 SetLt::SetLt ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.168.1.2 SetLt::SetLt (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.

5.168.2 Member Function Documentation

5.168.2.1 const std::vector< VariablePtr > SetLt::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:

- src/set_lt.h
- src/set_lt.cpp

5.169 SetNe Class Reference

Inheritance diagram for SetNe:

Public Member Functions

- SetNe ()
- SetNe (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.

- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.169.1 Constructor & Destructor Documentation

5.169.1.1 SetNe::SetNe ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.169.1.2 SetNe::SetNe (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.

5.169.2 Member Function Documentation

5.169.2.1 const std::vector< VariablePtr > SetNe::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:

- src/set_ne.h
- src/set_ne.cpp

5.170 SetNeReif Class Reference

Inheritance diagram for SetNeReif:

Public Member Functions

- SetNeReif ()
- SetNeReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.170.1 Constructor & Destructor Documentation

5.170.1.1 SetNeReif::SetNeReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.170.1.2 SetNeReif::SetNeReif (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.

5.170.2 Member Function Documentation

5.170.2.1 const std::vector< VariablePtr > SetNeReif::scope ( ) const [override], [virtual]

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

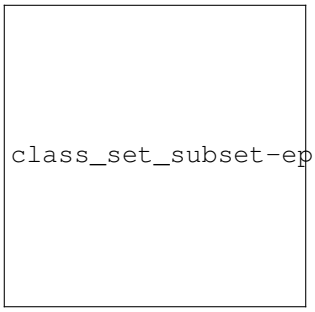
Implements Constraint.

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

- src/set_ne_reif.h
- src/set_ne_reif.cpp

5.171 SetSubset Class Reference

Inheritance diagram for SetSubset:



Public Member Functions

- SetSubset ()
- SetSubset (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.171.1 Constructor & Destructor Documentation

5.171.1.1 SetSubset::SetSubset ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.171.1.2 SetSubset::SetSubset (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.

5.171.2 Member Function Documentation

5.171.2.1 const std::vector< VariablePtr > SetSubset::scope ( ) const [override], [virtual]

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

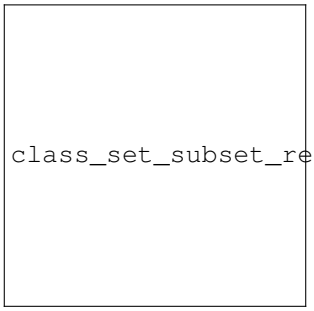
Implements Constraint.

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

- src/set_subset.h
- src/set_subset.cpp

5.172 SetSubsetReif Class Reference

Inheritance diagram for SetSubsetReif:



Public Member Functions

- SetSubsetReif ()
- SetSubsetReif (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.172.1 Constructor & Destructor Documentation

5.172.1.1 SetSubsetReif::SetSubsetReif ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.172.1.2 SetSubsetReif::SetSubsetReif (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.

5.172.2 Member Function Documentation

5.172.2.1 const std::vector< VariablePtr > SetSubsetReif::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:

- src/set_subset_reif.h
- src/set_subset_reif.cpp

5.173 SetSymDiff Class Reference

Inheritance diagram for SetSymDiff:

Public Member Functions

- SetSymDiff ()
- SetSymDiff (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.173.1 Constructor & Destructor Documentation

5.173.1.1 SetSymDiff::SetSymDiff ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.173.1.2 SetSymDiff::SetSymDiff (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.

5.173.2 Member Function Documentation

5.173.2.1 const std::vector< VariablePtr > SetSymDiff::scope ( ) const [override], [virtual]

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

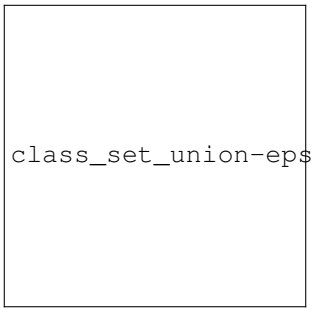
Implements Constraint.

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

- src/set_sym_diff.h
- src/set_sym_diff.cpp

5.174 SetUnion Class Reference

Inheritance diagram for SetUnion:



Public Member Functions

- SetUnion ()
- SetUnion (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.
- void print_semantic () const override

  Prints the semantic of this constraint.

Additional Inherited Members

5.174.1 Constructor & Destructor Documentation

5.174.1.1 SetUnion::SetUnion ( )

Basic constructor.

Note

after this constructor the client should call the setup method to setup the variables and parameters needed by this constraint.

5.174.1.2 SetUnion::SetUnion (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.

5.174.2 Member Function Documentation

5.174.2.1 const std::vector< VariablePtr > SetUnion::scope ( ) const [override], [virtual]

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

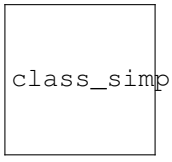
Implements Constraint.

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

- src/set_union.h
- src/set_union.cpp

5.175 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::unordered_map< size_t, BacktrackableObject ∗ > _backtrackable_objects
- std::set< size_t > _changed_backtrackables
- std::stack< std::pair< size_t, std::vector< std::pair< size_t, Memento ∗ > > > > _trail_stack
- std::stack< std::vector< size_t > > _trail_stack_info

5.175.1 Member Function Documentation

5.175.1.1 void SimpleBacktrackManager::add_changed (size_t *idx*) [virtual]

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

Parameters

idx	the (unique) id of the backtrackable object which is changed.

Note

only object already registered with this manager can be restored later.

Implements BacktrackManager.

5.175.1.2 void SimpleBacktrackManager::attach_backtracable (BacktrackableObject ∗ *bkt_obj*) [virtual]

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

Parameters

bkt_obj	a reference to a backtrackable object.

Implements BacktrackManager.

5.175.1.3 void SimpleBacktrackManager::detach_backtracable (size_t *bkt_id*) [virtual]

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

Parameters

bkt_id	the id of the backtrackable object to detach.

Implements BacktrackManager.

5.175.1.4 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.

5.175.1.5 size_t SimpleBacktrackManager::get_level ( ) const [virtual]

Get the current active level.

Returns

current active level in the manager.

Implements BacktrackManager.

5.175.1.6 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.

5.175.1.7 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.

5.175.1.8 void SimpleBacktrackManager::remove_level (size_t *lvl*) [override], [virtual]

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

Parameters

lvl	the level which is being removed.

Implements BacktrackManager.

5.175.1.9 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

lvl	the level to backtrack to.

Implements BacktrackManager.

5.175.1.10 void SimpleBacktrackManager::set_level (size_t *lvl*) [override], [virtual]

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

Parameters

lvl	the active level at which the changes will be recorded.

Implements BacktrackManager.

5.175.2 Member Data Documentation

5.175.2.1 std::unordered_map< size_t, BacktrackableObject * > SimpleBacktrackManager::_backtrackable_objects [protected]

Ordered list of backtrackable objects that are subjects of this BacktrackManager observer.

5.175.2.2 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).

5.175.2.3 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), ...] >

5.175.2.4 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.

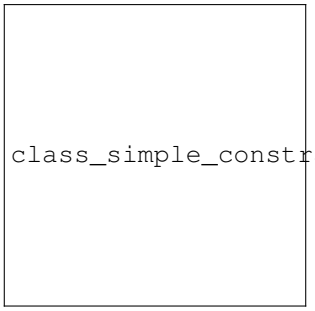
Todo implement this functionality.

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

- src/simple_backtrack_manager.h
- src/simple_backtrack_manager.cpp

5.176 SimpleConstraintStore Class Reference

Inheritance diagram for SimpleConstraintStore:



Public Member Functions

- SimpleConstraintStore ()
- void fail () override
- void sat_check (bool sat_check=true) override
- void con_check (bool con_check=true) override
- void add_changed (std::vector< size_t > &c_id, EventType event) override
- void impose (ConstraintPtr c) override
- bool consistency () override
- Constraint * getConstraint () override
- void clear_queue () override
- size_t num_constraints () const override
- size_t num_constraints_to_reevaluate () const override
- size_t num_propagations () const override
- void print () const override

  Print infoformation 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::unordered_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.

- size_t _number_of_propagations

  Number of propagations performed so far.
- bool _satisfiability_check
- bool _consistecy_propagation
- bool _failure

5.176.1 Constructor & Destructor Documentation

5.176.1.1 SimpleConstraintStore::SimpleConstraintStore ( )

Default constructor. It initializes the internal data structures of this constraint store.

5.176.2 Member Function Documentation

5.176.2.1 void SimpleConstraintStore::add_changed (std::vector< size_t > & *c_id*, EventType *event*) [override], [virtual]

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

Parameters

c_id	the vector of constraints ids to re-evaluate.
event	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.

5.176.2.2 void SimpleConstraintStore::clear_queue ( ) [override], [virtual]

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

Implements ConstraintStore.

5.176.2.3 void SimpleConstraintStore::con_check (bool *con_check* = true) [override], [virtual]

Sets consistecy propagation to con_check.

Parameters

con_check	boolean value. True if constraint propagation has to be performed, False otherwise.

Implements ConstraintStore.

5.176.2.4 bool SimpleConstraintStore::consistency ( ) [override], [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.

5.176.2.5 void SimpleConstraintStore::fail ( ) [override], [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.

5.176.2.6 Constraint * SimpleConstraintStore::getConstraint ( ) [override], [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.

5.176.2.7 void SimpleConstraintStore::impose (ConstraintPtr *c*) [override], [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

c	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.

5.176.2.8 size_t SimpleConstraintStore::num_constraints ( ) const [override], [virtual]

Returns the total number of constraints in this constraint store.

Implements ConstraintStore.

5.176.2.9 size_t SimpleConstraintStore::num_constraints_to_reevaluate ( ) const [override], [virtual]

Returns the number of constraints to re-evaluate.

Returns

number of constraints to re-evaluate.

Implements ConstraintStore.

5.176.2.10 size_t SimpleConstraintStore::num_propagations ( ) const [override], [virtual]

Returns the total number of propagations performed by this constraint store so far.

Implements ConstraintStore.

5.176.2.11 void SimpleConstraintStore::sat_check (bool *sat_check* = `true`) [override], [virtual]

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

Parameters

sat_check	boolean value representing whether or not the satisfiability check should be performed (default: true).

Implements ConstraintStore.

5.176.3 Member Data Documentation

5.176.3.1 bool SimpleConstraintStore::_consistecy_propagation [protected]

Defines whether the consistency propagation should be performed or not (default: true).

5.176.3.2 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).

5.176.3.3 bool SimpleConstraintStore::_failure [protected]

Keeps track whether some failure happened during some operations on this constraint store.

5.176.3.4 std::unordered_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.

5.176.3.5 bool SimpleConstraintStore::_satisfiability_check [protected]

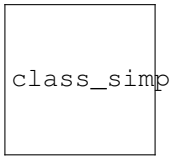
Defines whether the satisfiability check should be performed or not (default: true).

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

- src/simple_constraint_store.h
- src/simple_constraint_store.cpp

5.177 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

5.177.1 Constructor & Destructor Documentation

5.177.1.1 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

vars	a vector of pointer to variables to label.
var_cm	the variable metric used to select the next variable to label.
val_cm	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.

5.177.2 Member Function Documentation

5.177.2.1 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.

5.177.2.2 Variable * SimpleHeuristic::get_choice_variable (int *idx*) [virtual]

Gets next variable to label according to the VariableChoiceMetric.

Parameters

idx	the index of the last variable returned by this heuristic.

Returns

a pointer to the next variable to label.

Implements Heuristic.

5.177.3 Member Data Documentation

5.177.3.1 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.

5.177.3.2 ValueChoiceMetric* SimpleHeuristic::_value_metric [protected]

The metric used to select the next value to assign to the selected variable.

5.177.3.3 VariableChoiceMetric* SimpleHeuristic::_variable_metric [protected]

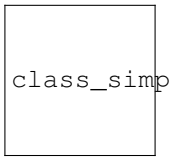
The metric used to select the next variable to label.

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

- src/simple_heuristic.h
- src/simple_heuristic.cpp

5.178 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

5.178.1 Constructor & Destructor Documentation

5.178.1.1 SimpleSolutionManager::SimpleSolutionManager (std::vector< Variable * > & *vars*)

Constructor. It creates a new simple solution manager attached to the given list of variables.

Parameters

vars	a vector of references to variables.

5.178.2 Member Function Documentation

5.178.2.1 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.

5.178.2.2 std::string SimpleSolutionManager::get_solution ( ) const [override], [virtual]

Get the last solution found.

Returns

a string representing the last solution found.

Implements SolutionManager.

5.178.2.3 std::string SimpleSolutionManager::get_solution (size_t *sol_idx*) const [override], [virtual]

Get the solution identified by its index.

Parameters

sol_idx	the index of the required solution.

Returns

a string representing the required solution.

Note

first solution has index 1.

Implements SolutionManager.

5.178.2.4 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.

5.178.2.5 size_t SimpleSolutionManager::number_of_solutions ( ) [override], [virtual]

Returns the number of solutions found so far.

Returns

the number of solutions.

Implements SolutionManager.

5.178.2.6 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.

5.178.2.7 void SimpleSolutionManager::set_solution_limit (int *n_sol*) [override], [virtual]

Sets a maximum number of solutions.

Parameters

n_sol	the number of solutions to compute.

Note

-1 stands for "find all solutions".

Implements SolutionManager.

5.178.2.8 void SimpleSolutionManager::set_variables (std::vector< Variable * > & *vars*) [override], [virtual]

Set the list of variables for which a solution is required.

Parameters

vars	a vector of references to variables.

Implements SolutionManager.

5.178.3 Member Data Documentation

5.178.3.1 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.

5.178.3.2 std::vector< std::string > SimpleSolutionManager::_solution_strings [protected]

Store the string representations of the solutions found so far.

5.178.3.3 std::map< int, Variable * > SimpleSolutionManager::_variables [protected]

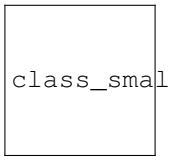
Stores the ordered list of variables that represent a solution. The order is given by variables' ids.

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

- src/simple_solution_manager.h
- src/simple_solution_manager.cpp

5.179 Smallest Class Reference

Inheritance diagram for Smallest:



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

5.179.1 Member Function Documentation

5.179.1.1 int Smallest::compare (double *metric,* Variable * *var*) [virtual]

Compare a metric value and a variable. Metric is given by their smallest value in their domin.

Implements VariableChoiceMetric.

5.179.1.2 int Smallest::compare (Variable * *var_a,* Variable * *var_b*) [virtual]

Compare variables w.r.t. their metrics. Metric is given by their smallest value in their domain.

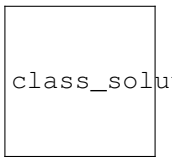
Implements VariableChoiceMetric.

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

- src/smallest_metric.h
- src/smallest_metric.cpp

5.180 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.

5.180.1 Member Function Documentation

5.180.1.1 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.

5.180.1.2 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.

5.180.1.3 virtual std::string SolutionManager::get_solution (size_t *sol_idx*) const [pure virtual]

Get the solution identified by its index.

Parameters

sol_idx	the index of the required solution.

Returns

a string representing the required solution.

Note

first solution has index 1.

Implemented in SimpleSolutionManager.

5.180.1.4 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.

5.180.1.5 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.

5.180.1.6 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.

5.180.1.7 virtual void SolutionManager::set_solution_limit (int *n_sol*) [pure virtual]

Sets a maximum number of solutions.

Parameters

n_sol	the number of solutions to compute.

Note

-1 stands for "find all solutions".

Implemented in SimpleSolutionManager.

5.180.1.8 virtual void SolutionManager::set_variables (std::vector< Variable * > & *vars*) [pure virtual]

Set the list of variables for which a solution is required.

Parameters

vars	a vector of references to variables.

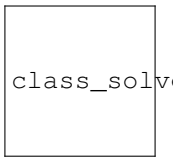
Implemented in SimpleSolutionManager.

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

- src/solution_manager.h

5.181 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.

5.181.1 Member Function Documentation

5.181.1.1 virtual void Solver::add_model (CPModel * *model*) [pure virtual]

Add a model to the solver.

Parameters

model	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.

5.181.1.2 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_data	a reference to a input_data class where options are retrieved.
model_idx	the index of the model to customize (default: 0, i.e., first model).

Implemented in CPSolver.

5.181.1.3 virtual CPModel* Solver::get_model (int *model_idx*) const [pure virtual]

Returns a reference to model.

Parameters

model_idx	the index of the model to return (model_idx = 0 means first model).

Implemented in CPSolver.

5.181.1.4 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.

5.181.1.5 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.

5.181.1.6 virtual void Solver::remove_model (int *model_idx*) [pure virtual]

Removes a model actually destroying it.

Parameters

model_idx	the index of the model to destroy, (model_idx = 0 means first model).

Implemented in CPSolver.

5.181.1.7 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.

5.181.1.8 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

model_idx	the index of the model to solve (model_idx = 0 means first model).

Implemented in CPSolver.

5.181.1.9 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.

5.181.1.10 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:

- src/solver.h

5.182 Statistics Class Reference

Public Types

- enum **TIMING** {
  GENERAL, **SEARCH**, **FIRST_SOL**, **PREPROCESS**,
  FILTERING, **BACKTRACK**, **ALL**, **Count** }

Public Member Functions

- **Statistics** (const Statistics &other)=delete
- Statistics & **operator=** (const Statistics &other)=delete
- void set_timer ()

  Set timer (starts "watching" the running time)
- void set_timer (TIMING t)
- void stopwatch (TIMING t=TIMING::GENERAL)
- void stopwatch_and_add (TIMING t=TIMING::GENERAL)
- double get_timer (TIMING t=TIMING::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.

Protected Member Functions

- virtual int timing_to_int (Statistics::TIMING t) const
- virtual Statistics::TIMING int_to_timing (int i) const

Protected Attributes

- std::string _dbg

  Debug string info.
- std::chrono::time_point
  < std::chrono::system_clock > **_time_start**
- double _time [MAX_T_TYPE]

  Computational times are recorded here.
- std::chrono::time_point
  < std::chrono::system_clock > _partial_time [MAX_T_TYPE]

  Partial times (i.e., from set timer to stop watch) 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 int MAX_T_TYPE = 100

  Max size of the array of times.

5.182.1 Member Function Documentation

5.182.1.1 double Statistics::get_timer (TIMING *t* = TIMING::GENERAL)

Get the value of the running time in seconds.

Parameters

tt	describes which kind of computation time must be returned,

Returns

the computational time related to tt in seconds.

5.182.1.2 Statistics::TIMING Statistics::int_to_timing (int *i*) const [protected], [virtual]

Converter from integer to TIMING values. This is mostly used for ease of implementation and decoupling from TIMING class and integer indeces for arrays.

Parameters

i	integer value

Returns

TIMING value corresponding to i or count if if no mapping exist for the given i.

5.182.1.3 void Statistics::set_timer (TIMING *t*)

Set timer for a given computation which will be observed.

Parameters

tt	describes which kind of computation will be observed.

5.182.1.4 void Statistics::stopwatch (TIMING *t* = TIMING::GENERAL)

Stop watching the running time.

Parameters

tt	describes which kind of computation has been observed.

5.182.1.5 void Statistics::stopwatch_and_add (TIMING *t* = TIMING::GENERAL)

Stop watching the running time and add the time to the previous times watched for tt.

Parameters

tt	describes which kind of computation has been observed.

5.182.1.6 int Statistics::timing_to_int (Statistics::TIMING *t*) const [protected], [virtual]

Converter from TIMING enum values to int values.

Parameters

t	TIMING value

Returns

integer value mapping t, or -1 if no mapping exists for the given t.

Note

This is done to decouple TIMING with integer indeces.

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

- src/statistics.h
- src/statistics.cpp

5.183 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.
- virtual bool set_token (std::string &token_string)=0

Protected Attributes

- std::string **_dbg**
- TokenType _tkn_type

  Specifies the type of token (e.g., FD_VARIABLE)

5.183.1 Member Function Documentation

5.183.1.1 virtual bool Token::set_token (std::string & *token_string*) [pure virtual]

Set the token (initialization) given the string representing the token.

Parameters

token_string	the string corresponding to the token

Returns

True if token has been created, False otherwise

Implemented in TokenVar, TokenArr, TokenSol, and TokenCon.

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

- src/token.h
- src/token.cpp

5.184 TokenArr Class Reference

Inheritance diagram for TokenArr:

Public Member Functions

- bool set_token (std::string &token_string) override
- void **set_size_arr** (int)
- int **get_size_arr** () const
- bool **is_valid_array** () 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

5.184.1 Member Function Documentation

5.184.1.1 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

5.184.1.2 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

5.184.1.3 bool TokenArr::is_var_in (int *var*) const

Check whether a given variable (idx) is indexed by the array (i.e., is whithin the array.

Note

: check is performed w.r.t. both the variable string identifier (e.g., a[i]) and its global id.

Parameters

var	the variable to check membership

Returns

true if var is in the current array, false otherwise

5.184.1.4 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

lw	lower bound
up	upper bound

5.184.1.5 bool TokenArr::set_token (std::string & *token_string*) [override], [virtual]

Set internal parameters according to the string representing the array of variables

Parameters

string	representing the array statement

Returns

True if params are set correctly, False otherwise

Note

var_decl ::= var_type: identifier annotations

Implements Token.

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

- src/token_arr.h
- src/token_arr.cpp

5.185 TokenCon Class Reference

Inheritance diagram for TokenCon:

Public Member Functions

- bool set_token (std::string &token_string) override
- 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.

5.185.1 Member Function Documentation

5.185.1.1 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

str	string representing the expression.

5.185.1.2 std::string TokenCon::get_expr (int *idx*) const

Get the string represeting the ith expression that defines the constraint.

Parameters

idx	index of the expression to return

Returns

return the idx^th expression

5.185.1.3 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.

5.185.1.4 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.

5.185.1.5 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.

5.185.1.6 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.

5.185.1.7 bool TokenCon::set_token (std::string & *token_string*) [override], [virtual]

Set the token (initialization) given the string representing the token.

Parameters

token_string	the string corresponding to the token

Returns

True if token has been created, False otherwise

Implements Token.

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

- src/token_con.h
- src/token_con.cpp

5.186 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 UTokenPtr 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 a line must be skipped or not.
- virtual bool **skip_line** (std::string)
- virtual bool set_new_line ()
- virtual void clear_line ()
- virtual UTokenPtr analyze_token ()=0

Protected Attributes

- std::string **_dbg**
- std::string **DELIMITERS**
- std::string **WHITESPACE**
- std::string **_comment_lines**
- bool **_new_line**
- bool **_need_line**
- bool **_failed**
- char * **_c_token**
- char * _parsed_line

  Parsed line.

5.186.1 Member Function Documentation

5.186.1.1 virtual UTokenPtr 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.

5.186.1.2 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.

5.186.1.3 bool Tokenization::set_new_line ( ) [protected], [virtual]

It states whether a new line has been found. Different heuristics may be used here.

5.186.1.4 void Tokenization::set_new_tokenizer (std::string *line*)

Prepare a new tokenizer (i.e., string for strtok).

Parameters

line	the string to tokenize.

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

- src/tokenization.h
- src/tokenization.cpp

5.187 TokenSol Class Reference

Inheritance diagram for TokenSol:

Public Member Functions

- bool set_token (std::string &token_string) override
- bool set_solve_params (std::string &annotation)
- void **set_var_goal** (std::string)
- void **set_solve_goal** (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
- 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

5.187.1 Member Function Documentation

5.187.1.1 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.

5.187.1.2 string TokenSol::get_var_to_label (int *idx*) const

Get the string corresponding to the ith variable to label.

Parameters

idx	the index of the variable to label.

Returns

the string identifier of the idx^th variable to label.

5.187.1.3 int TokenSol::num_var_to_label ( ) const

Number of variables to label if specified by the model.

Returns

the number of variables to label.

5.187.1.4 bool TokenSol::set_solve_params (std::string & *annotation*)

Given a FlatZinc solve statement, parses the statement and set strategy methods.

Returns

True if parsing succeed , False otherwise.

Note

annotation (annotationarg, ...)

5.187.1.5 bool TokenSol::set_token (std::string & *token_string*) [override], [virtual]

Set the token (initialization) given the string representing the token.

Parameters

token_string	the string corresponding to the token

Returns

True if token has been created, False otherwise

Implements Token.

5.187.2 Member Data Documentation

5.187.2.1 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:

- src/token_sol.h
- src/token_sol.cpp

5.188 TokenVar Class Reference

Inheritance diagram for TokenVar:

Public Member Functions

- bool set_token (std::string &token_string) override
- 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)
- std::vector< std::vector< int > > get_subset_domain ()
- void print () const

  Print info methods.

Protected Member Functions

- virtual std::pair< int, int > get_range (std::string str) const
- virtual std::vector< int > get_subset (std::string str) const
- virtual bool set_type_var (std::string &type)

  Set var type from token string.
- virtual void set_id (std::string &id)

  Set var id from token string.

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**

5.188.1 Member Function Documentation

5.188.1.1 pair< int, int > TokenVar::get_range (std::string *str*) const [protected], [virtual]

Get a pair of integers <x1, x2> from a string of type containing a range value *x1..x2*, where x1 is the lower bound and x2 the upper bound.

Parameters

str	string to parse

Returns

a pair representing the range

Note

if more than one range is present, return the first one

5.188.1.2 vector< int > TokenVar::get_subset (std::string *str*) const [protected], [virtual]

Get a vector of elements from a string of type "∗{x1, x2, ...xk}∗".

Parameters

str	string to parse

Returns

a pair representing the range expressed with str

5.188.1.3 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.

5.188.1.4 void TokenVar::set_range_domain (std::string *str*)

Specifies a range domain for the variable with a given a string of type "∗x1..x2∗".

5.188.1.5 void TokenVar::set_range_domain (int *lw,* int *ub*)

Specifies a range domain for the variable with a given lower and upper bound.

Parameters

lw	lower bound
ub	upper bound

5.188.1.6 void TokenVar::set_subset_domain (std::string *str*)

Call the right subset function, parsing the string given in input.

5.188.1.7 void TokenVar::set_subset_domain ( )

Specifies a set of int domain.

Note

set of int;

5.188.1.8 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

elems	vector of elements

Note

set of {x1, x2, ...xk}

5.188.1.9 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

elems	vector of vectors of elements

Note

set as {{x1, x2, ...xk}, ...}

5.188.1.10 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

range	pair of int elements for range

Note

set of x1..x2

5.188.1.11 bool TokenVar::set_token (std::string & *token_string*) [override], [virtual]

Set the token (initialization) given the string representing the token.

Parameters

token_string	the string corresponding to the token

Returns

True if token has been created, False otherwise

Implements Token.

5.188.1.12 void TokenVar::set_var_dom_type (VarDomainType *vdt*)

Set the type of the current (token) variable.

Parameters

vdt	the variable domain type of type VarDomainType.

5.188.1.13 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

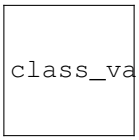
str	the string identifier of the variable.

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

- src/token_var.h
- src/token_var.cpp

5.189 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.

5.189.1 Member Function Documentation

5.189.1.1 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.

5.189.1.2 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

var	(pointer to) the variable for which value for assignment is given.

Returns

the value to assign to the given variable.

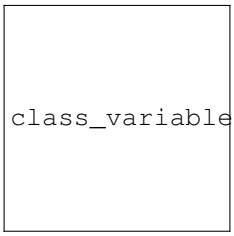
Implemented in InDomainMax, InDomainMedian, InDomain, InDomainMin, and InDomainRandom.

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

- src/value_choice_metric.h
- src/value_choice_metric.cpp

5.190 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_observers ()
- 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)
- virtual void notify_constraint ()
- virtual void notify_store ()
- 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

5.190.1 Constructor & Destructor Documentation

5.190.1.1 Variable::Variable ( ) [protected]

Base constructor.

Note

a global unique id is assigned to this variable.

5.190.1.2 Variable::Variable (int *v_id*) [protected]

Base constructor.

Parameters

v_id	the id to assign to this variable.

5.190.2 Member Function Documentation

5.190.2.1 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

c	the (pointer to) the constraint which is added to this variable.

5.190.2.2 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

store	the constraint store to attach to this variable.

5.190.2.3 void Variable::detach_constraint (ConstraintPtr *c*) [virtual]

It detaches constraint from this variable, so change in variable will not cause constraint reevaluation.

Parameters

c	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 destroyes the constraint c. The client must be care of storing c somewhere else in order to restore the state (e.g. for backtrack actions).

5.190.2.4 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

c	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 destroyes the constraint c. The client must be care of storing c somewhere else in order to restore the state (e.g. for backtrack actions).

5.190.2.5 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.

5.190.2.6 string Variable::get_str_id ( ) const

Get the string id of this variable.

Returns

a string representing the id of this variable.

5.190.2.7 bool Variable::is_attached (size_t *c_id*) [protected], [virtual]

It checks whether a given id belongs to the list of detached constraints.

Parameters

c_id	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.

5.190.2.8 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.

5.190.2.9 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.

5.190.2.10 void Variable::notify_constraint ( ) [protected], [virtual]

It notifies all the constraints attached to this variables that a change has been done on this very variable.

5.190.2.11 void Variable::notify_observers ( ) [virtual]

It notifies the current observers attached to this variable that a change has been done on this very variable.

Reimplemented in IntVariable.

5.190.2.12 void Variable::notify_store ( ) [protected], [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.

5.190.2.13 virtual void Variable::set_domain_type (DomainType *dt*) [pure virtual]

Set domain according to the specific variable implementation.

Note

: different types of variable

Parameters

dt	domain type of type DomainType to set to the current variable

Implemented in IntVariable.

5.190.2.14 void Variable::set_str_id (std::string *str*)

Set the (string) id of the variable.

Parameters

str	the string to set as variable's identifier.

5.190.2.15 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.

5.190.2.16 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.

5.190.3 Member Data Documentation

5.190.3.1 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.

5.190.3.2 ConstraintStorePtr Variable::_constraint_store [protected]

The constraint store on which this variable operates (i.e., constraint store to notify).

5.190.3.3 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.

5.190.3.4 DomainIterator* Variable::domain_iterator

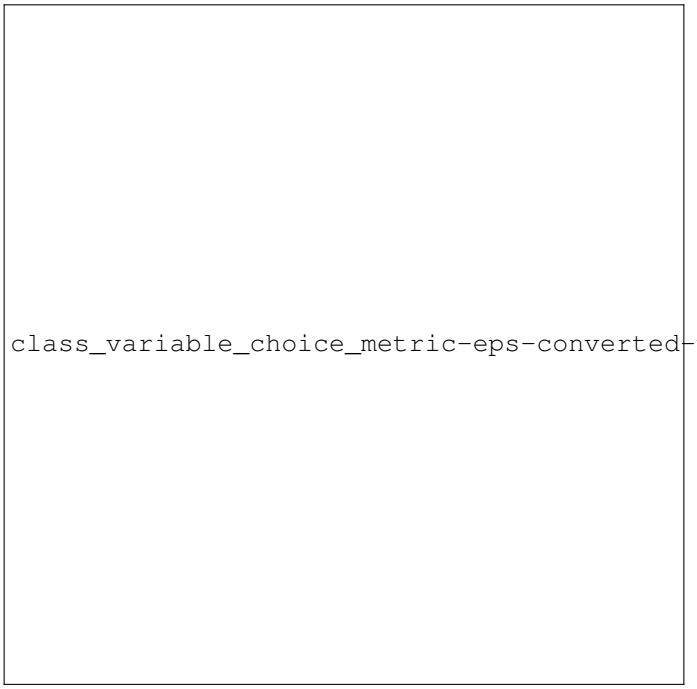
Iterator to use to get domain's elements from the current variable's domain. Domains should be accessed only through this iterator.

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

- src/variable.h
- src/variable.cpp

5.191 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**

5.191.1 Member Function Documentation

5.191.1.1 virtual int VariableChoiceMetric::compare (double *metric,* Variable ∗ *var*) [pure virtual]

Compares the metric value with a given variable.

Parameters

metric	the (metric) value to compare with.

var	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, MaxRegret, MostConstrained, AntiFirstFail, FirstFail, Largest, Occurence, and Smallest.

5.191.1.2 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

var_a	the (pointer to) variable to compare with the metric value of var_b.
var_b	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, MaxRegret, MostConstrained, AntiFirstFail, FirstFail, Largest, Occurence, and Smallest.

5.191.1.3 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.

5.191.1.4 virtual double VariableChoiceMetric::metric_value (Variable * *var*) [pure virtual]

Returns the value of the metric of a given variable.

Parameters

var	the variable for which the metric is required.

Returns

the value of the metric.

Implemented in InputOrder, MaxRegret, MostConstrained, AntiFirstFail, FirstFail, Largest, Occurence, and Smallest.

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

- src/variable_choice_metric.h
- src/variable_choice_metric.cpp