vifid: Variable vIsualization for FInite Domains

An Automatic Documentation Generator for (C)LP Systems
The CIAO System Documentation Series
Draft printed on: 15 October 1998

José Manuel Ramos and the CLIP Group
`jramos@clip.dia.fi.upm.es`
`http://www.clip.dia.fi.upm.es/~jramos/pl2texi_docs/`
Facultad de Informática
Universidad Politécnica de Madrid

Summary

`VIFID` is a graphical interface aimed at visualizing the execution of constraint logic programs using finite domains. This library provides a way for debugging constraint logic programs which use finite domains by insert breakpoints in the user code. It is also useful for educational purposes in order for novel users to learn how finite domain solvers work. It has been developed with Sicstus Prolog and Tcl/Tk under UNIX. This internals manual has been generated with the `pl2texi` package.

1 Introduction

Traditional visualization paradigms are often inadequate for CLP programming paradigms, because declarative and operational semantics differ in crucial ways from those of other paradigms. In this document we will describe the implementation interface of VIFID, which visualizes data evolution and the runtime constraints in constraint logic programs. The first chapters will present the engine of the visualizer with a description of all the internal predicates. The following chapters will present the four visualizations developed in VIFID (variable history, 2-D plot, all variables and enumerate all variables) as a description of their internal predicates.

1.1 Applications

The proposed graphical representations have been chosen based on the perceived needs of a programmer trying to analyze the behaviour of a program execution. These representations give an intuitive depiction which it is also useful for educational purposes. This is done by adding breakpoints in the source code. It is also possible to add user input constraints at runtime in order to try different strategies or to check the state of the constraint store.

1.2 More info

The facilities provided by the library are documented in the description of its modules. This documentation is intended to provide information at a "reference manual" level. The purpose of this manual is to document precisely what the implementation of the library supports at any point in time.

2 Interface to VIFID (traceclpfd)

Version: 0.5#3 (1998/9/6, 11:30:45 MET DST)

This library provides the *user interface* of the *finite domain visualization* .

2.1 Usage and interface (traceclpfd)

- **Library usage:**
  `:- use_module(library(traceclpfd)).`
- **Exports:**
  - *Predicates:*
    `init_state/3`, `show_state/2`, `show_state/1`, `final_state/1`, `list_of_vars_names/6`, `list_of_vars_names/5`.
- **Other modules used:**
  - *User modules:*
    `allv`, `allven`, `event`, `fd_general`, `internal_types`, `misc`, `plot_2d`, `tcltk_general`, `var_hist`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

2.2 Documentation on exports (traceclpfd)

init_state/3: PREDICATE

Usage: `init_state(+Vars,+Names,-(State))`

- *Description:* That is the first predicate you must call to create the visualization's windows and its buttons and to initialize the `State` properly. It also takes care of cleaning possible garbage from previous calls. `Names` are the *names of the variables* in `Vars`. State is a returned struct you need for every call to `show_state/2` or `show_state/1` and `final_state/1`
- *Call and exit should be compatible with:*

  `+Vars` is a list of `fdvars`. (`list/2`)

  `+Names` is a list of `atms`. (`list/2`)

  `-(State)` is the structure which keeps the information about the visualization interface. (`state/1`)

show_state/2: PREDICATE

Usage: `show_state(+State,+Name)`

- *Description:* Sets a breakpoint which checks the execution mode ('Stopped', 'Draw', 'Ignore') and, if needed, shows the state of the variables in the active windows. It also disables buttons in order to avoid inconsistent states. `State` is the variable returned by `init_state/3` . `Name` is an identifier for the breakpoint, displayed in the status line of the main window, in order to know where you are during the execution.

– *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

  +Name is an atom (a non-numeric constant). (atm/1)

show_state/1: PREDICATE

Usage: show_state(State)

– *Description:* See show_state/2 .

final_state/1: PREDICATE

Usage: final_state(+State)

– *Description:* Shows the last state of the variables (usually the solution of the problem). Upon exit all windows are closed and all internal values deleted. State is the variable returned by init_state/3 .

– *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

list_of_vars_names/6: PREDICATE

Usage: list_of_vars_names(+N,-(Vars),-(Names),+Min,+Max,+Base)

– *Description:* Creates a list of N variables and a list of N names for them, where each name is created concatenating Base with a positive integer starting at 0. Min and Max are the initial *domain bounds* for each variable. Base is the first part of the name.

– *Call and exit should be compatible with:*

  +N is an integer. (int/1)

  -(Vars) is a list of fdvars. (list/2)

  -(Names) is a list of atms. (list/2)

  +Min is an integer. (int/1)

  +Max is an integer. (int/1)

  +Base is an atom (a non-numeric constant). (atm/1)

list_of_vars_names/5: PREDICATE

Usage: list_of_vars_names(N,Vars,Names,Min,Max)

– *Description:* It is the same as list_of_vars/6 except that the first part of the name for each variable is 'Var'.

3 "Variable History" Visualization (var_hist)

This module provides predicates for the visualization of 'variable history', which initialize the window and show domain variables each time it is required.

3.1 Usage and interface (var_hist)

- **Library usage:**
  `:- use_module(library(var_hist)).`
- **Exports:**
  - *Predicates:*
    `visualize_vh/1`.
- **Other modules used:**
  - *User modules:*
    `fd_general`, `internal_types`, `misc`, `tcltk_general`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

3.2 Documentation on exports (var_hist)

visualize_vh/1: PREDICATE

Usage: `visualize_vh(+State)`

- *Description:* It is called each time the 'variable history' visualization is to be shown. It checks whether the 'variable history' visualization window is active or not, creating it if necessary. `State` is the structure returned by `init_state/3` .
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

3.3 Documentation on internals (var_hist)

init_hist/1: PREDICATE

Usage: `init_hist(+State)`

- *Description:* Creates the 'variable history' visualization with all buttons and frames, and initializes all internal values. The arguments are the same as in `visualize_vh/1` .
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

show_hist/1: PREDICATE

Usage: `show_hist(+State)`

- *Description:* Shows the actual domain of each variable in the 'variable history' visualization window when that window is active. The arguments are the same as in `visualize_vh/1` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

show_variables_tr/1: PREDICATE

Usage: `show_variables_tr(+State)`

- *Description:* Draws the domain values of each variable at the next coordinate. If the 'refresh' button has been pressed a brown circle appears on the left of the domain values. If backtracking from constraint evaluation occurs, a hook-shaped arrow also appears at the left side.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

plot_them_tr/4: PREDICATE

Usage: `plot_them_tr(+Domains,+Values,+Interp,+Coord)`

- *Description:* Draws the domain values of each variable we are inspecting in the 'variable history' window. It calculates the right coordinate to draw the representation of each variable. `Values` are the current domain values of each variable. `Domains` is the domain limits of each variable. `Coord` is the starting point to draw at.
- *Call and exit should be compatible with:*

  `+Domains` is a list of `limits`. (`list/2`)

  `+Values` is a list of `list(int)`s. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord` is a point. (`point/1`)

plot_1_tr/6: PREDICATE

Usage: `plot_1_tr(+Current,+Max,+Values,+Interp,+Coord1,-(Coord2))`

- *Description:* Draws the representation of the domain values of a variable beginning at the starting point `Coord1` with the first domain value. `Coord2` is the next coordinate to be used to draw at. If `Current` is in `Values`, a green rectangle is drawn, otherwise a red rectangle is drawn .
- *Call and exit should be compatible with:*

  `+Current` is an integer. (`int/1`)

  `+Max` is an integer. (`int/1`)

  `+Values` is a list of `ints`. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord1` is a point. (`point/1`)

  `-(Coord2)` is a point. (`point/1`)

4 "2-D Plot" Visualization (plot_2d)

This module provides predicates for '2-D plot' visualization, which initialize the window and show domain variables each time it is required

4.1 Usage and interface (plot_2d)

- **Library usage:**
  `:- use_module(library(plot_2d)).`
- **Exports:**
  - *Predicates:*
    `visualize_2dp/1`.
- **Other modules used:**
  - *User modules:*
    `fd_general`, `internal_types`, `misc`, `tcltk_general`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

4.2 Documentation on exports (plot_2d)

visualize_2dp/1: PREDICATE

Usage: `visualize_2dp(+State)`

- *Description:* Called each time the '2-D plot' visualization is to be shown. It checks whether the 'all variables' window is active or not, creating it if neccesary. `State` is the structure returned by `init_state/3` .
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

4.3 Documentation on internals (plot_2d)

init_2dp/1: PREDICATE

Usage: `init_2dp(+State)`

- *Description:* Creates the '2-D plot' visualization with all buttons and frames, and initializes all internal values. The arguments are the same as in `visualize_2dp/1` . By default the first and second variables are selected for displaying.
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

show_vars_data/1: PREDICATE

Usage: `show_vars_data(+State)`

- *Description:* A rectangle shows the 2-D points with compatible values for the two displayed variables. For each displayed variable, the outermost rectangle depicts the values in the domain according to the *solver domains* , and the innermost rectangle shows the *active values* obtained from enumeration.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

plot_var_st/4: PREDICATE

Usage: `plot_var_st(+State,+Name,+Orient,+Row)`

- *Description:* Draws the internal values that the finite domain solver is not able to discard. `Name` is the name of the selected variable. The orientation (vertical or horizontal) is given by `Orient`. `Row` is the row-coordinate where to draw it.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Orient` is an atom (a non-numeric constant). (`atm/1`)

  `+Row` is an integer. (`int/1`)

plot_var_st2/4: PREDICATE

Usage: `plot_var_st2(+State,+Name,+Orient,+Row)`

- *Description:* Draws an enumeration of the variable values. The arguments have the same meaning as in `plot_var_st/4` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Orient` is an atom (a non-numeric constant). (`atm/1`)

  `+Row` is an integer. (`int/1`)

plot_them_st/7: PREDICATE

Usage: `plot_them_st(+Name,+Current,+Max,+Values,+Interp,+Coord,+Orient)`

- *Description:* Draws the representation of the domain values of a variable beginning at the starting point with the first domain value. If `Current` belongs to `Values`, a green rectangle is drawn, otherwise a red one is drawn. Each rectangle has an identifier of the form `Name@Value`. `Orient` should be vertical or horizontal. `Coord` is the current coordinate where to draw.
- *Call and exit should be compatible with:*

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Current` is an integer. (`int/1`)

  `+Max` is an integer. (`int/1`)

+Values is a list of ints. (list/2)
+Interp represents a TCL/TK interpreter identifier. (interp/1)
+Coord is a point. (point/1)
+Orient is an atom (a non-numeric constant). (atm/1)

show_data/2: PREDICATE

Usage: show_data(+State,+Row)

- *Description:* Draws the relation between selected variables by enumeration. This enumeration represents the domain values of the vertical selected variable with each of the domain values of the horizontal selected variable. It starts drawing at the Row coordinate.
- *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

  +Row is an integer. (int/1)

show_box/5: PREDICATE

Usage: show_box(+State,+Max,+Enumeration,+Current,+Coord)

- *Description:* Draws the relation between the selected variables. It draws a line of coloured rectangles for each variable value. Max is the maximum domain value of the horizontal selected variable. Enumeration is given by show_data/2 . Current is the horizontal selected variable value.
- *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

  +Max is an integer. (int/1)

  +Enumeration is a list of int*list(int)s. (list/2)

  +Current is an integer. (int/1)

  +Coord is a point. (point/1)

draw_line/3: PREDICATE

Usage: draw_line(+State,+Coord,+Values)

- *Description:* Draws a vertical line with the domain values of the vertical selected variable at the given coordinate by Coord.
- *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

  +Coord is a point. (point/1)

  +Values is a list of ints. (list/2)

5 "All Variables" Visualization (allv)

This module provides predicates for the 'all variables' visualization, which initialize the window and show domain variables each time it is required.

5.1 Usage and interface (allv)

- **Library usage:**
  `:- use_module(library(allv)).`
- **Exports:**
  - *Predicates:*
    `visualize_allv/2`.
- **Other modules used:**
  - *User modules:*
    `fd_general`, `internal_types`, `misc`, `tcltk_general`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

5.2 Documentation on exports (allv)

visualize_allv/2: PREDICATE

Usage: `visualize_allv(+State,+Values)`

- *Description:* Called each time the 'all variables' visualization is to be shown. It checks whether the visualization window is active or not, creating it if necessary. `State` is the structure returned by `init_state/3` . `Values` is a list of the values of each variable which is used to show in a different colour the domain of a variable when the user posts a constraint.
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)
  `+Values` is a list of `list(int)`s. (`list/2`)

5.3 Documentation on internals (allv)

init_allv/2: PREDICATE

Usage: `init_allv(+State,+Values)`

- *Description:* Creates the 'all variables' window with all buttons and frames, and initializes all internal values. The arguments are the same as for `visualize_allv/2` .

- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

show_allv/2: PREDICATE

Usage: `show_allv(+State,+Values)`

- *Description:* Shows the current domain of each variable in the 'all variables' window when it is active. The arguments are the same as in `visualize_allv/2` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

show_allvar_data/2: PREDICATE

Usage: `show_allvar_data(+State,+Values)`

- *Description:* Retrieves the current domain of each variable and shows it in the 'all variables' window. All canvas elements are deleted beforehand. The arguments are the same as in `visualize_allv/2` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

plot_them_allv/6: PREDICATE

Usage: `plot_them_allv(+Domains,+Names,+Values,+OldValues,+Interp,+Coord)`

- *Description:* Draws all domain values of each variable we are inspecting in the 'all variables' window. It calculates the correct coordinate to draw the representation of each variable. `Values` are the current domain values of each variable. `OldValues` are the domain values before posting a constraint. If `OldValues` is an empty list, each domain value is drawn; otherwise it only changes the colour of some domain values. `Domains` is the domain limits of each variable. `Names` is the names of all the variables. `Coord` is the starting point to draw at.
- *Call and exit should be compatible with:*

  `+Domains` is a list of `limits`. (`list/2`)

  `+Names` is a list of `atms`. (`list/2`)

  `+Values` is a list of `list(int)`s. (`list/2`)

  `+OldValues` is a list of `list(int)`s. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord` is a point. (`point/1`)

plot_1_allv/6: PREDICATE

Usage: `plot_1_allv(+Current,+Max,+Name,+Values,+Interp,+Coord)`

- *Description:* Draws the representation of the domain values of a variable beginning at the starting point with the first domain value. If `Current` belongs to the domain `Values` a green rectangle is drawn, otherwise the rectangle is red. Each rectangle has an identifier of the form `Name@Value`.
- *Call and exit should be compatible with:*

  `+Current` is an integer. (`int/1`)

  `+Max` is an integer. (`int/1`)

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Values` is a list of `ints`. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord` is a point. (`point/1`)

change_1_allv/4: PREDICATE

Usage: `change_1_allv(+Name,+Values,+OldValues,+Interp)`

- *Description:* Changes the colour of the domain variables represented. If an element in `OldValues` does not appear in `Values`, then the rectangle representing it changes its colour to yellow.
- *Call and exit should be compatible with:*

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Values` is a list of `ints`. (`list/2`)

  `+OldValues` is a list of `ints`. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

6 "Enumerate All Variables" Visualization (allven)

This module provides predicates for the 'enumerate all variables' visualization, which initialize the window and show the domain variables each time it is required

6.1 Usage and interface (allven)

- **Library usage:**

  `:- use_module(library(allven)).`
- **Exports:**
  - *Predicates:*

    `visualize_allven/2`.
- **Other modules used:**
  - *User modules:*

    `fd_general`, `internal_types`, `misc`, `tcltk_general`.
  - *System library modules:*

    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*

    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

6.2 Documentation on exports (allven)

visualize_allven/2: PREDICATE

Usage: `visualize_allven(+State,+Values)`

- *Description:* Called each time the 'enumerate all variables' visualization is to be shown. It checks whether the 'enumerate all variables' window is active or not, creating it if necessary. `State` is the structure returned by `init_state/3` . `Values` is a list of the values of each variable which is used to show in a different colour the domain of a variable when the user posts a constraint.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

6.3 Documentation on internals (allven)

init_allven/2: PREDICATE

Usage: `init_allven(+State,+Values)`

- *Description:* Creates the 'enumerate all variables' window with all buttons and frames, and initializes all internal values. The arguments are the same as for `visualize_allven/2` .

- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

show_allven/2: PREDICATE

Usage: `show_allven(+State,+Values)`

- *Description:* Shows the current domain of each variable in the 'enumerate all variables' window when that window is active. The arguments are the same as in `visualize_allven/2` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

show_allvar2_data/2: PREDICATE

Usage: `show_allvar2_data(+State,+Values)`

- *Description:* Retrieves the current domain of each variable and shows it in the 'enumerate all variables' window. All canvas elements are deleted beforehand. The arguments are the same as in `visualize_allven/2` .
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Values` is a list of `list(int)`s. (`list/2`)

plot_them_allven/6: PREDICATE

Usage: `plot_them_allven(+Domains,+Names,+Values,+OldValues,+Interp,+Coord)`

- *Description:* Draws the domain values of each variable we are inspecting in the 'enumerate all variables' window. It calculates the correct coordinate to draw the representation of each variable. `Values` are the current domain values of each variable. `OldValues` are the domain values before posting a constraint. If `OldValues` is an empty list, each domain value is drawn, otherwise it only changes the colour of some domain values. `Domains` is the domain limits of each variable. `Names` is the names of all the variables names. `Coord` is the starting point to draw at.
- *Call and exit should be compatible with:*

  `+Domains` is a list of `limits`. (`list/2`)

  `+Names` is a list of `atms`. (`list/2`)

  `+Values` is a list of `list(int)`s. (`list/2`)

  `+OldValues` is a list of `list(int)`s. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord` is a point. (`point/1`)

plot_1_allven/6: PREDICATE

Usage: `plot_1_allven(+Current,+Max,+Name,+Values,+Interp,+Coord)`

- *Description:* Draws the representation of the domain values of a variable beginning at the starting point with the first domain value. If `Current` belongs to the domain `Values` a green rectangle is drawn, otherwise a red one is drawn. Each rectangle has an identifier of the form `Name@Value`.
- *Call and exit should be compatible with:*

  `+Current` is an integer. (`int/1`)

  `+Max` is an integer. (`int/1`)

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Values` is a list of `int`s. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Coord` is a point. (`point/1`)

change_1_allven/4: PREDICATE

Usage: `change_1_allven(+Name,+Values,+OldValues,+Interp)`

- *Description:* Changes the colour of the domain variables represented. If an element in `OldValues` does not appear in `Values`, then the rectangle representing it changes its colour to yellow.
- *Call and exit should be compatible with:*

  `+Name` is an atom (a non-numeric constant). (`atm/1`)

  `+Values` is a list of `int`s. (`list/2`)

  `+OldValues` is a list of `int`s. (`list/2`)

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

7 Event management (event)

This module provides some predicates to manage GUI (Graphical User Interface) events, such as button pressed and mouse button pressed. It also provides a loop to deal with these events when the program execution stops at a breakpoint.

7.1 Usage and interface (event)

- **Library usage:**
  `:- use_module(library(event)).`
- **Exports:**
  - *Predicates:*
    `top_level/2`, `$no_back/1`.
- **Other modules used:**
  - *User modules:*
    `allv`, `allven`, `fd_general`, `internal_types`, `misc`, `plot_2d`, `tcltk_general`, `var_hist`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

7.2 Documentation on exports (event)

top_level/2: PREDICATE

Usage: `top_level(+State,+Num)`

- *Description:* Waits until the next *event* is produced and executes the action associated to this event when in stop mode. `Num` is a counter initialized each time the user starts 'stop on next breakpoint' mode in order to properly treat the reset event.
- *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)
  `+Num` is an integer. (`int/1`)

7.3 Documentation on internals (event)

do_event/3: PREDICATE

This predicate executes the action associated to the events produced while 'stop on next breakpoint' mode is active. The events this predicate reacts to are:

- reset button pressed,
- commit button pressed,

- exit button pressed,
- variable selected,
- constraint posted,
- go on executing by pressing stop, draw or ignore buttons.

Usage 1: `do_event(State,Event,Num)`

- *Description:* Draws the domain values of the two selected variables on the 2-D plot visualization windows and their relation each time a variable is selected.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `plot2dp`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 2: `do_event(State,Event,Num)`

- *Description:* Redraws the domain values of the variables in all active windows. Called each time the 'refresh' button is pressed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `refresh`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 3: `do_event(State,Event,Num)`

- *Description:* Removes (by backtracking) from the constraint store the user constraints executed but not yet commited to.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `reset`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 4: `do_event(State,Event,Num)`

- *Description:* User constraints are accepted and cannot be discarded.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `accept`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 5: `do_event(State,Event,Num)`

- *Description:* Posts a user constraint created by pressing the mouse left button on the '2-D plot' visualization window. That constraint removes a value from a FD variable. A message appears at the bottom of the window if the constraint did not succeed. A message also appears if it succeeds, showing the variable and the value removed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `constraint_2dp`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 6: `do_event(State,Event,Num)`

- *Description:* Posts a user constraint created pressing the mouse left button on the 'all variables' visualization window. That constraint removes a value from a FD variable. A message appears at the bottom of the window if the constraint did not succeed. A message also appears if it succeeds, showing the variable and the value removed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `constraint_allv`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 7: `do_event(State,Event,Num)`

- *Description:* Posts a user constraint created pressing the mouse left button on the 'enumerate all variables' visualization window. That constraint removes a value from a FD variable. A message appears at the bottom of the window if the constraint did not succeed. A message also appears if it succeeds, showing the variable and the value removed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `constraint_allven`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 8: `do_event(State,Event,Num)`

- *Description:* Creates a window to input a user *constraint* , and redraws the domain values of the variables in all *active windows* when the 'user input contraint' button is pressed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `add`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 9: `do_event(State,Event,Num)`

- *Description:* Disables buttons and goes on executing each time 'stop on next break point', 'draw on next breakpoint' or 'ignore breakpoints' is pressed
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `go`. (`= /2`)

  `Num` is an integer. (`int/1`)

Usage 10: `do_event(State,Event,Num)`

- *Description:* Stops executing and closes all windows when exit button is pressed.
- *Should hold at call time:*

  `State` is the structure which keeps the information about the visualization interface. (`state/1`)

  Unify `Event` and `quit`. (`= /2`)

  `Num` is an integer. (`int/1`)

8 TCL/TK miscellaneous predicates (tcltk_general)

Version: 0.5#4 (1998/4/1)

This module provides different utility predicates for TCL/TK objects.

8.1 Usage and interface (tcltk_general)

- **Library usage:**
  `:- use_module(library(tcltk_general)).`
- **Exports:**
  - *Predicates:*
    `init_env_vh/1`, `tk_rectangle_tr/3`, `put_names/3`, `init_env_2dp/1`, `draw_vars_st/2`, `tk_rectangle_st/4`, `init_env_allv/1`, `tk_rectangle_allv/4`, `ch_color_rect/4`, `init_env_allven/1`, `tk_rectangle_allven/4`, `create_main_env/1`, `update/1`, `update_2/0`, `delete_dom/2`, `create_canvas_scrolled/2`, `resize_scroll/2`, `resize_width_tr/1`, `set_tcl_var/3`, `get_tcl_var/3`, `resize_cv/2`, `enable_b/2`, `disable_b/2`, `exist_window/3`, `ask_constraint/2`.
- **Other modules used:**
  - *User modules:*
    `fd_general`, `internal_types`, `misc`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

8.2 Documentation on exports (tcltk_general)

init_env_vh/1: PREDICATE

Usage: `init_env_vh(+Interp)`

- *Description:* Creates the variable history visualization window.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

tk_rectangle_tr/3: PREDICATE

Usage: `tk_rectangle_tr(+Interp,+Coord,+Colour)`

- *Description:* Creates a rectangle in the 'variable history' visualization window whose upper left corner is placed at `Coord` and whose colour is `Colour`.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Coord` is a point. (`point/1`)
  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

put_names/3: PREDICATE

Usage: `put_names(+Interp,+Names,+Domain)`

- *Description:* Calculates the coordinate where the variable names `Names` are to be drawn and calls `show_names/4` .
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Names` is a list of `atms`. (`list/2`)

  `+Domain` is a list of `limits`. (`list/2`)

init_env_2dp/1: PREDICATE

Usage: `init_env_2dp(+Interp)`

- *Description:* Creates the 2-D plot visualization window.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

draw_vars_st/2: PREDICATE

Usage: `draw_vars_st(+Interp,+Names)`

- *Description:* Prints text and variable names `Names` associated to the radiobuttons in the '2-D plot' window.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Names` is a list of `atms`. (`list/2`)

tk_rectangle_st/4: PREDICATE

Usage: `tk_rectangle_st(+Interp,+Id,+Coordinate,+Colour)`

- *Description:* Creates a rectangle in the '2-D plot' visualization window whose upper-left corner is placed at `Coord`, whose colour is `Colour`, and associates the identifier `Id` with it.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Coordinate` is a point. (`point/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

init_env_allv/1: PREDICATE

Usage: `init_env_allv(+Interp)`

- *Description:* Creates the 'all variables' visualization window.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

tk_rectangle_allv/4: PREDICATE

Usage: `tk_rectangle_allv(+Interp,+Id,+Coordinate,+Colour)`

- *Description:* Creates a rectangle in the 'all variables' visualization window whose upper left corner is placed at `Coord`, whose colour is `Colour`, and associates the identifier `Id` with it.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Coordinate` is a point. (`point/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

ch_color_rect/4: PREDICATE

Usage: `ch_color_rect(+Interp,+Canvas,+Id,+Colour)`
- *Description:* Changes the color of the `Id` object on the canvas `Canvas` to `Colour`.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Canvas` is an atom (a non-numeric constant). (`atm/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

init_env_allven/1: PREDICATE

Usage: `init_env_allven(+Interp)`
- *Description:* Creates the 'enumerate all variables' visualization window.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

tk_rectangle_allven/4: PREDICATE

Usage: `tk_rectangle_allven(+Interp,+Id,+Coordinate,+Colour)`
- *Description:* Creates a rectangle in the 'enumerate all variables' visualization window whose upper left corner is placed at `Coord`, whose colour is `Colour`, and associates the identifier `Id` with it.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Coordinate` is a point. (`point/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

create_main_env/1: PREDICATE

Usage: `create_main_env(+Interp)`
- *Description:* Creates the main window and initializes all Tcl/Tk global variables.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

update/1: PREDICATE

Usage: `update(+Interp)`

- *Description:* Forces TCL/TK to process all window displaying events.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

update_2/0: PREDICATE

Usage:

- *Description:* Forces TCL/TK to process all internal events.

delete_dom/2: PREDICATE

Usage: `delete_dom(+Interp,+Canvas)`

- *Description:* Deletes all the elements on the canvas `Canvas`.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Canvas` is an atom (a non-numeric constant). (`atm/1`)

create_canvas_scrolled/2: PREDICATE

Usage: `create_canvas_scrolled(+Interp,+Window)`

- *Description:* Creates a canvas with two scrollbars in the window `Window`.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Window` is an atom (a non-numeric constant). (`atm/1`)

resize_scroll/2: PREDICATE

Usage: `resize_scroll(+Interp,+Canvas)`

- *Description:* Resizes the scrollable canvas region of `Canvas` to contain all the elements drawn on it.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Canvas` is an atom (a non-numeric constant). (`atm/1`)

resize_width_tr/1: PREDICATE

Usage: `resize_width_tr(+Interp)`

- *Description:* Resizes the canvas width of variable history visualization window. The maximum size of the window is fixed with the predicates `max_width` and `max_height` to 800x600 pixels.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

set_tcl_var/3: PREDICATE

Usage: `set_tcl_var(+Interp,+Var,+Value)`

- *Description:* Assigns `Value` to `Var`.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Var` defines a TCL/TK variable. (`tclvar/1`)
  `+Value` is any term. (`term/1`)

get_tcl_var/3: PREDICATE

Usage: `get_tcl_var(+Interp,+Var,-(Value))`

- *Description:* Provides the current `Value` of `Var`.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Var` defines a TCL/TK variable. (`tclvar/1`)
  `-(Value)` is any term. (`term/1`)

resize_cv/2: PREDICATE

Usage: `resize_cv(+Interp,+Canvas)`

- *Description:* Resizes canvas of `Canvas` in order to show all objects appearing in it. The maximum size of the canvas is 800x600 pixels
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Canvas` is an atom (a non-numeric constant). (`atm/1`)

enable_b/2: PREDICATE

Usage: `enable_b(+Interp,+Button)`

- *Description:* Enables `Button`, so it can be clicked on.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Button` is an atom (a non-numeric constant). (`atm/1`)

disable_b/2: PREDICATE

Usage: `disable_b(+Interp,+Button)`

- *Description:* Disables `Button`, so it will not react to clicking.
- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Button` is an atom (a non-numeric constant). (`atm/1`)

exist_window/3: PREDICATE

Usage: `exist_window(+Interp,+Window,-(Exists))`

- *Description:* Returns in `Exists` an atom which reflects the existence of `Window`. `Exists` can be `true` or `false`

– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Window` is an atom (a non-numeric constant). (`atm/1`)
  `-(Exists)` is an atom (a non-numeric constant). (`atm/1`)

ask_constraint/2: PREDICATE

Usage: `ask_constraint(+Interp,+Constraint)`
– *Description:* Creates a window where the user can type in a `Constraint`.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Constraint` is a character string. (`string/1`)

8.3 Documentation on internals (`tcltk_general`)

max_width/1: PREDICATE

Usage: `max_width(+Interp)`
– *Description:* Sets the maximum width of all the windows in `Interp`.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

max_height/1: PREDICATE

Usage: `max_height(+Interp)`
– *Description:* Sets the maximum height of all the windows in `Interp`.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

show_names/4: PREDICATE

Usage: `show_names(+Interp,+Names,+Domain,+Coord)`
– *Description:* Prints the variable `Names` in the 'variable history' visualization window starting at point `Coord`.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Names` is a list of `atm`s. (`list/2`)
  `+Domain` is a list of `limit`s. (`list/2`)
  `+Coord` is a point. (`point/1`)

plot_nmvars_st/1: PREDICATE

Usage: `plot_nmvars_st(+Interp)`
– *Description:* Draws radiobuttons which allow selecting a variable for vertical and horizontal plotting in the '2-D plot' visualization window.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

draw_it/3: PREDICATE

Usage: `draw_it(+Interp,+Num,+Names)`

- *Description:* Calls a TCL/TK procedure which displays the name of each variable.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Num` is an integer. (`int/1`)

  `+Names` is a list of `atm`s. (`list/2`)

assoc_bind/3: PREDICATE

Usage: `assoc_bind(+Interp,+Id,+Colour)`

- *Description:* Associates an action to the clicking of the left mouse button on a rectangle representing a value in the domain of the variable in the '2-D plot' visualization window. `Id` is the identifier given to the rectangle when it was created. `Colour` is the color of the rectangle to which this action is associated. It also generates an event.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

resize_cv_st/1: PREDICATE

Usage: `resize_cv_st(+Interp)`

- *Description:* Resizes the canvas of the '2-D plot' visualization window in order to redraw all objects appearing in it.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

assoc_bind_allv/3: PREDICATE

Usage: `assoc_bind_allv(+Interp,+Id,+Colour)`

- *Description:* Associates an action to the clicking of the left mouse button on a rectangle representing a value in the domain of the variable in the 'all variables' visualization window. `Id` is the identifier given to the rectangle when it was created. `Colour` is the color of the rectangle to which this action is associated.It also generates an event.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

assoc_bind_allven/3: PREDICATE

Usage: `assoc_bind_allven(+Interp,+Id,+Colour)`

- *Description:* Associates an action to the clicking of the left mouse button on a rectangle representing a value in the domain of the variable in 'enumerate all variables' visualization window. `Id` is the identifier given to the rectangle when it was created. `Colour` is the color of the rectangle to which this action is associated. It also generates an event.

- *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Id` is an atom (a non-numeric constant). (`atm/1`)
  `+Colour` is an atom (a non-numeric constant). (`atm/1`)

9 Finite Domains predicates (fd_general)

This module provides different utility predicates for finite domain values management and searching.

9.1 Usage and interface (fd_general)

- **Library usage:**
  `:- use_module(library(fd_general)).`
- **Exports:**
  - *Predicates:*
    `active_values_var/2`, `active_values/2`, `real_values/2`, `get_minmax/1`.
- **Other modules used:**
  - *User modules:*
    `internal_types`, `misc`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

9.2 Documentation on exports (fd_general)

active_values_var/2: PREDICATE

Usage: `active_values_var(+Var,-(Values))`
- *Description:* Returns the values in the domain of `Var`.
- *Call and exit should be compatible with:*
  `+Var` defines a finite domain variable. (`fdvar/1`)
  `-(Values)` is a list of `int`s. (`list/2`)

active_values/2: PREDICATE

Usage: `active_values(Vars,Values)`
- *Description:* Returns the values of a list of finite domain variables, as internally stored by the solver.
- *Call and exit should be compatible with:*
  `Vars` is a list of `fdvar`s. (`list/2`)
  `Values` is a list of `list(int)`s. (`list/2`)

real_values/2: PREDICATE

Usage: `real_values(+Vars,+Values)`
- *Description:* Returns the values of a list of finite domain variables, as obtained by enumeration.

– *Call and exit should be compatible with:*
  `+Vars` is a list of `fdvars`. (`list/2`)
  `+Values` is a list of `list(int)`s. (`list/2`)

get_minmax/1: PREDICATE

Usage: `get_minmax(+State)`

– *Description:* Gets the minimun and maximum value of each finite domain variable we are inspecting and adds it to `State` as a list.
– *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

9.3 Documentation on internals (`fd_general`)

lapp/2: PREDICATE

Usage: `lapp(+Vars,-(Values))`

– *Description:* See `lapp_aux/3` .
– *Call and exit should be compatible with:*
  `+Vars` is a list of `fdvars`. (`list/2`)
  `-(Values)` is a list of `limits`. (`list/2`)

lapp_aux/3: PREDICATE

Usage: `lapp_aux(+Vars,+Aux,-(Values))`

– *Description:* Provides the bounds of the variables in a list of finite domain variables. `Aux` is an intermediate list of bounds, initially empty.
– *Call and exit should be compatible with:*
  `+Vars` is a list of `fdvars`. (`list/2`)
  `+Aux` is a list of `limits`. (`list/2`)
  `-(Values)` is a list of `limits`. (`list/2`)

10 Mutable predicates (updatable)

This module provides three predicates to create, access, and change updatable terms.

10.1 Usage and interface (updatable)

- **Library usage:**
  `:- use_module(library(updatable)).`
- **Exports:**
  - *Predicates:*
    `create_updatable/2`, `get_updatable/2`, `update_updatable/2`.
- **Other modules used:**
  - *User modules:*
    `internal_types`.
  - *System library modules:*
    `aggregates`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

10.2 Documentation on exports (updatable)

create_updatable/2: PREDICATE

Usage: `create_updatable(-(Var),+Value)`
- *Description:* Creates a new updatable term `Var` whose initial value is `Value`.
- *Call and exit should be compatible with:*
  `-(Var)` defines a mutable variable. (`updatable/1`)
  `+Value` is any term. (`any/1`)

get_updatable/2: PREDICATE

Usage: `get_updatable(-(Var),+Value)`
- *Description:* `Value` is the current value of the updatable term `Var`.
- *Call and exit should be compatible with:*
  `-(Var)` defines a mutable variable. (`updatable/1`)
  `+Value` is any term. (`any/1`)

update_updatable/2: PREDICATE

Usage: `update_updatable(-(Var),+Value)`
- *Description:* Set the current value of the updatable term `Var`.
- *Call and exit should be compatible with:*
  `-(Var)` defines a mutable variable. (`updatable/1`)
  `+Value` is any term. (`any/1`)

11 Miscellaneous predicates (misc)

This module provides different utility predicates for managing object coordinates and internal structures, searching, and miscellaneous constraint related predicates.

11.1 Usage and interface (misc)

- **Library usage:**
  `:- use_module(library(misc)).`
- **Exports:**
  - *Predicates:*
    get_coord/3, delta_coord/4, scale_coord/3, save_coord/1, retrieve_coord/1, atom_concat/3, index_list/3, identif_tr/2, identif_st/3, state_interp/2, state_vars/2, state_names/2, state_dom/2, domain_range/3, create_crd/2, get_crd/2, set_crd/2, protect/1, protect_2/1, try_const/2, do_constraint/2, get_constraint/7, definite_variables/1, at_least_two/2, list_of_vars_names/6, list_of_vars_names/5, current_coord/1.
- **Other modules used:**
  - *User modules:*
    allv, allven, event, internal_types, plot_2d, tcltk_general, updatable, var_hist.
  - *System library modules:*
    aggregates, basicprops, types/basictypes, dynamic, iso_byte_char, iso_incomplete, iso_misc, lists, operators, read, streams, system, write.
  - *Internal (engine) modules:*
    arithmetic, atomic_basic, attributes, basiccontrol, concurrency, data_facts, exceptions, io_aux, io_basic, prolog_flags, streams_basic, system_info, term_basic, term_compare, term_typing.

11.2 Documentation on exports (misc)

get_coord/3: PREDICATE

Usage 1: `get_coord(+Coord,-(X),-(Y))`
- *Description:* Provides the X-value and Y-value of the coordinate
- *Call and exit should be compatible with:*
  `+Coord` is a point. (point/1)
  `-(X)` is currently instantiated to an integer. (integer/1)
  `-(Y)` is currently instantiated to an integer. (integer/1)

Usage 2: `get_coord(-(Coord),+X,+Y)`
- *Description:* Provides a coordinate with X-value and a Y-value
- *Call and exit should be compatible with:*
  `-(Coord)` is a point. (point/1)
  `+X` is currently instantiated to an integer. (integer/1)
  `+Y` is currently instantiated to an integer. (integer/1)

delta_coord/4: PREDICATE

Usage: `delta_coord(+Coord1,-(Coord2),+DeltaX,+DeltaY)`

- *Description:* Increases a coordinate `Coord1` adding `DeltaX` to X-value and `DeltaY` to Y-value.
- *Call and exit should be compatible with:*
  `+Coord1` is a point. (`point/1`)
  `-(Coord2)` is a point. (`point/1`)
  `+DeltaX` is currently instantiated to an integer. (`integer/1`)
  `+DeltaY` is currently instantiated to an integer. (`integer/1`)

scale_coord/3: PREDICATE

Usage: `scale_coord(+Coord1,-(Coord2),+Scale)`

- *Description:* Scales `Coord1` with a factor `Scale` given the result in `Coord2`.
- *Call and exit should be compatible with:*
  `+Coord1` is a point. (`point/1`)
  `-(Coord2)` is a point. (`point/1`)
  `+Scale` is currently instantiated to an integer. (`integer/1`)

save_coord/1: PREDICATE

Usage: `save_coord(+Coord)`

- *Description:* Asserts `Coord` in the data base as the fact `current_coord/1` .
- *Call and exit should be compatible with:*
  `+Coord` is a point. (`point/1`)

retrieve_coord/1: PREDICATE

Usage: `retrieve_coord(-(Coord))`

- *Description:* Retracts an atom asserted as `current_coord/1` .
- *Call and exit should be compatible with:*
  `-(Coord)` is a point. (`point/1`)

atom_concat/3: PREDICATE

Usage: `atom_concat(+Atom1,+Atom2,-(AtomRes))`

- *Description:* Concat `Atom1` with `Atom2` giving the result in `AtomRes`.
- *Call and exit should be compatible with:*
  `+Atom1` is an atom (a non-numeric constant). (`atm/1`)
  `+Atom2` is an atom (a non-numeric constant). (`atm/1`)
  `-(AtomRes)` is an atom (a non-numeric constant). (`atm/1`)

index_list/3: PREDICATE

Usage 1: `index_list(+Element,+List,-(Pos))`

- *Description:* Searches `Element` in `List`. If `Element` is found, `Pos` represent the position of `Element` in `List`.

- *Call and exit should be compatible with:*

  +Element is any term. (any/1)

  +List is a list of anys. (list/2)

  -(Pos) is currently instantiated to an integer. (integer/1)

Usage 2: `index_list(-(Element),+List,+Pos)`

- *Description:* Returns the element in List which is at Pos position.
- *Call and exit should be compatible with:*

  -(Element) is any term. (any/1)

  +List is any term. (any/1)

  +Pos is currently instantiated to an integer. (integer/1)

identif_tr/2: PREDICATE

Usage: `identif_tr(+Coord,-(Id))`

- *Description:* Creates an identifier Id for a TCL/TK widget using its coordinate Coord.
- *Call and exit should be compatible with:*

  +Coord is a point. (point/1)

  -(Id) is an atom (a non-numeric constant). (atm/1)

identif_st/3: PREDICATE

Usage: `identif_st(+Var,+Value,-(Id))`

- *Description:* Creates an identifier Id for a TCL/TK rectangle using the name of var Var and the value Value.
- *Call and exit should be compatible with:*

  +Var is an atom (a non-numeric constant). (atm/1)

  +Value is currently instantiated to an integer. (integer/1)

  -(Id) is an atom (a non-numeric constant). (atm/1)

state_interp/2: PREDICATE

Usage 1: `state_interp(+State,-(Interp))`

- *Description:* Returns the *TCL/TK interpreter* PROLOG is working with.
- *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

  -(Interp) represents a TCL/TK interpreter identifier. (interp/1)

Usage 2: `state_interp(-(State),+Interp)`

- *Description:* Assigns the TCL/TK interpreter to the structure State.
- *Call and exit should be compatible with:*

  -(State) is the structure which keeps the information about the visualization interface. (state/1)

  +Interp represents a TCL/TK interpreter identifier. (interp/1)

state_vars/2: PREDICATE

Usage 1: `state_vars(+State,-(Vars))`

- *Description:* Returns the *Finite Domain variables* we are inspecting.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `-(Vars)` is a list of `fdvars`. (`list/2`)

Usage 2: `state_vars(-(State),+Vars)`

- *Description:* Stores in the `State` the Finite Domain variables we will inspect.
- *Call and exit should be compatible with:*

  `-(State)` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Vars` is a list of `fdvars`. (`list/2`)

state_names/2: PREDICATE

Usage 1: `state_names(+State,-(Names))`

- *Description:* Returns the names of the variables we are inspecting.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `-(Names)` is a list of `atms`. (`list/2`)

Usage 2: `state_names(-(State),+Names)`

- *Description:* Stores in the `State` the names of the variables we will inspect.
- *Call and exit should be compatible with:*

  `-(State)` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Names` is a list of `atms`. (`list/2`)

state_dom/2: PREDICATE

Usage 1: `state_dom(+State,-(Domain))`

- *Description:* Returns the bounds for each of the Finite Domain variables we are inspecting.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `-(Domain)` is a list of `limits`. (`list/2`)

Usage 2: `state_dom(-(State),+Vars)`

- *Description:* Stores in the `State` the limits for each of the Finite Domain variables we will inspect.
- *Call and exit should be compatible with:*

  `-(State)` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Vars` is a list of `limits`. (`list/2`)

domain_range/3: PREDICATE

Usage 1: `domain_range(+Limit,-(Min),-(Max))`

- *Description:* Returns the minimum domain value `Min` and the maximum domain value `Max` from the initial domain of a variable.
- *Call and exit should be compatible with:*

  `+Limit` defines the bounds of a FD variable. (`limit/1`)

  `-(Min)` is currently instantiated to an integer. (`integer/1`)

  `-(Max)` is currently instantiated to an integer. (`integer/1`)

Usage 2: `domain_range(-(Limit),+Min,+Max)`

- *Description:* Assigns the lower and the upper bounds of a domain limits.
- *Call and exit should be compatible with:*

  `-(Limit)` defines the bounds of a FD variable. (`limit/1`)

  `+Min` is currently instantiated to an integer. (`integer/1`)

  `+Max` is currently instantiated to an integer. (`integer/1`)

create_crd/2: PREDICATE

Usage: `create_crd(+State,+Coord)`

- *Description:* Associates a coordinate given by `Coord` to `State` using backtrackable destructive assignement.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Coord` is a point. (`point/1`)

get_crd/2: PREDICATE

Usage: `get_crd(+State,-(Coord))`

- *Description:* Returns the last coordinate saved in `State`. Fail if you did not call `create_crd/2` before.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `-(Coord)` is a point. (`point/1`)

set_crd/2: PREDICATE

Usage: `set_crd(+State,+Coord)`

- *Description:* Changes coordinate value in `State`. Fail if `create_crd/2` was not called before.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Coord` is a point. (`point/1`)

protect/1: PREDICATE

Usage: `protect(+Action)`

- *Description:* Protects code from being backtracked over by creating a choicepoint. It is used to control backtracking when you press *reset* button.
- *Call and exit should be compatible with:*

  `+Action` defines a choicepoint. (`action/1`)

protect_2/1: PREDICATE

Usage: `protect_2(+Action)`

- *Description:* It is the same as protect, but it is used when you press commit.
- *Call and exit should be compatible with:*

  `+Action` defines other type of choicepoint. (`action2/1`)

try_const/2: PREDICATE

Usage: `try_const(+Constraint,-(Result))`

- *Description:* Executes `Constraint` and returns 'yes' in `Result` denoting it success. In case of failure it returns 'no'
- *Call and exit should be compatible with:*

  `+Constraint` defines a valid Finite Domains constraint using the variable names in the problem. (`constraint/1`)

  `-(Result)` is an atom (a non-numeric constant). (`atm/1`)

do_constraint/2: PREDICATE

Usage: `do_constraint(+State,+Constraint)`

- *Description:* Executes a `Constraint` given by the user as input and shows the effects of its execution in all active windows if success. If failure, it displays the message 'Failed contraint'.
- *Call and exit should be compatible with:*

  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)

  `+Constraint` defines a valid Finite Domains constraint using the variable names in the problem. (`constraint/1`)

get_constraint/7: PREDICATE

Usage: `get_constraint(+Interp,+Id,+Vars,+Names,-(Var),-(Value),+Status)`

- *Description:* Returns the Finite Domain variable (`Var`) and the value (`Value`) associated to the TCL/TK object identifier (`Id`), in order to make an user input constraint like `"Var` is not equal to `Value"`. `Id` has the form `Var@Value`.
- *Call and exit should be compatible with:*

  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)

  `+Id` is an atom (a non-numeric constant). (`atm/1`)

  `+Vars` is a list of `fdvars`. (`list/2`)

  `+Names` is a list of `atms`. (`list/2`)

-(Var) defines a finite domain variable. (fdvar/1)
-(Value) is currently instantiated to an integer. (integer/1)
+Status is a character string. (string/1)

definite_variables/1: PREDICATE

Usage: `definite_variables(+State)`

- *Description:* Checks the definiteness of the Finite Domain variables we are inspecting.
- *Call and exit should be compatible with:*

  +State is the structure which keeps the information about the visualization interface. (state/1)

at_least_two/2: PREDICATE

Usage: `at_least_two(+Interp,-(Num))`

- *Description:* Provides the number of visualizations currently active in Num.
- *Call and exit should be compatible with:*

  +Interp represents a TCL/TK interpreter identifier. (interp/1)
  -(Num) is currently instantiated to an integer. (integer/1)

list_of_vars_names/6: PREDICATE

Usage: `list_of_vars_names(+N,-(Vars),-(Names),+Min,+Max,+Base)`

- *Description:* Creates a list of N variables and a list of N names for that vars, where each name is Base concat with a value starting in 0. Min and Max are the initial domain limits of each variable. Base is the first part of the name. N is the number of variables given.
- *Call and exit should be compatible with:*

  +N is an integer. (int/1)
  -(Vars) is a list of fdvars. (list/2)
  -(Names) is a list of atms. (list/2)
  +Min is an integer. (int/1)
  +Max is an integer. (int/1)
  +Base is an atom (a non-numeric constant). (atm/1)

list_of_vars_names/5: PREDICATE

Usage: `list_of_vars_names(N,Vars,Names,Min,Max)`

- *Description:* It is the same as list_of_vars/6 except on the first part of the name for each variable is 'var'.

11.3 Documentation on internals (misc)

put_back/2: PREDICATE

Usage: `put_back(+L1,-(L2))`

– *Description:* Adds neccessary backslashes in order to show the input user constraint correctly. L1 is the string given by TCL/TK and L2 is the string that Prolog shows in the status label of the main window. That is needed to be done due to the escape sequences used in both languages.
– *Call and exit should be compatible with:*
  `+L1` is a character string. (`string/1`)
  `-(L2)` is a character string. (`string/1`)

unify_vars/3: PREDICATE

Usage: `unify_vars(+State,+Term,?(Names))`
– *Description:* Unifies each variable in `Term` with a Finite Domain variable of the problem we are inspecting when an input user constraint is given.
– *Call and exit should be compatible with:*
  `+State` is the structure which keeps the information about the visualization interface. (`state/1`)
  `+Term` is any term. (`term/1`)
  `?(Names)` is a list of `atm=vars`. (`list/2`)

valid_constraint/1: PREDICATE

Usage: `valid_constraint(+Term)`
– *Description:* Checks if `Term` is a valid constraint expansion, preventing errors in user input constraints.
– *Call and exit should be compatible with:*
  `+Term` defines a valid Finite Domains constraint using the variable names in the problem. (`constraint/1`)

split_id/3: PREDICATE

Usage: `split_id(+String,-(Name),-(Value))`
– *Description:* Generates the name of a Finite Domain variable and a value from a string that represents a identifier.
– *Call and exit should be compatible with:*
  `+String` is a character string. (`string/1`)
  `-(Name)` is an atom (a non-numeric constant). (`atm/1`)
  `-(Value)` is currently instantiated to an integer. (`integer/1`)

is_definite_variables/2: PREDICATE

Usage: `is_definite_variables(+Interp,+Vars)`
– *Description:* See `definite_variables/1` . It is called from it.
– *Call and exit should be compatible with:*
  `+Interp` represents a TCL/TK interpreter identifier. (`interp/1`)
  `+Vars` is a list of `fdvars`. (`list/2`)

create_vars/4: PREDICATE

Usage: `create_vars(+N,+Min,+Max,-(Vars))`

- *Description:* Creates a list of `N` variables. Each one has as domain limits `Min` and `Max`.
- *Call and exit should be compatible with:*

  `+N` is currently instantiated to an integer. (`integer/1`)

  `+Min` is currently instantiated to an integer. (`integer/1`)

  `+Max` is currently instantiated to an integer. (`integer/1`)

  `-(Vars)` is a list of `fdvars`. (`list/2`)

give_var_names/5: PREDICATE

Usage: `give_var_names(+Vars,-(Names),+Initial,-(Total),+Base)`

- *Description:* Creates a list of names `Names` for the variables in `Vars`, starting each name with `Base`. `Initial` is a number given to each variable, and `Total` is the number of names created.
- *Call and exit should be compatible with:*

  `+Vars` is a list of `fdvars`. (`list/2`)

  `-(Names)` is a list of `atms`. (`list/2`)

  `+Initial` is currently instantiated to an integer. (`integer/1`)

  `-(Total)` is currently instantiated to an integer. (`integer/1`)

  `+Base` is an atom (a non-numeric constant). (`atm/1`)

give_var_names/4: PREDICATE

Usage: `give_var_names(Vars,Names,Initial,Total)`

- *Description:* See `give_var_names/5` . In this case each variable starts with 'var'.

12 Defined types

These are the internal data types used in the predicates

12.1 Usage and interface (internal_types)

- **Library usage:**
  `:- use_module(library(internal_types)).`
- **Exports:**
  - *Predicates:*
    `mutable/1`.
  - *Types:*
    `state/1`, `interp/1`, `fdvar/1`, `tclvar/1`, `limit/1`, `point/1`, `constraint/1`, `action/1`, `action2/1`.
- **Other modules used:**
  - *System library modules:*
    `aggregates`, `basicprops`, `types/basictypes`, `dynamic`, `iso_byte_char`, `iso_incomplete`, `iso_misc`, `lists`, `operators`, `read`, `streams`, `system`, `write`.
  - *Internal (engine) modules:*
    `arithmetic`, `atomic_basic`, `attributes`, `basiccontrol`, `concurrency`, `data_facts`, `exceptions`, `io_aux`, `io_basic`, `prolog_flags`, `streams_basic`, `system_info`, `term_basic`, `term_compare`, `term_typing`.

12.2 Documentation on exports (internal_types)

state/1: TYPE

Usage: `state(X)`
- *Description:* `X` is the structure which keeps the information about the visualization interface.
- *Should hold globally:*
  Defines the regular type: `^info(interp,list(fdvar),list(atm),list(limit),point)`. (`defines_type/2`)

interp/1: TYPE

Usage: `interp(X)`
- *Description:* `X` represents a TCL/TK interpreter identifier.
- *Should hold globally:*
  Defines the regular type: `atm`. (`defines_type/2`)

fdvar/1: TYPE

Usage: `fdvar(X)`
- *Description:* `X` defines a finite domain variable.
- *Should hold globally:*
  Defines the regular type: `term`. (`defines_type/2`)

tclvar/1: TYPE

Usage: `tclvar(X)`

- *Description:* `X` defines a TCL/TK variable.
- *Should hold globally:*
  Defines the regular type: `atm`. (`defines_type/2`)

limit/1: TYPE

Usage: `limit(X)`

- *Description:* `X` defines the bounds of a FD variable.
- *Should hold globally:*
  Defines the regular type: `^min_max(int,int)`. (`defines_type/2`)

point/1: TYPE

Usage: `point(X)`

- *Description:* `X` is a point.
- *Should hold globally:*
  Defines the regular type: `^coord(int,int)`. (`defines_type/2`)

constraint/1: TYPE

Usage: `constraint(X)`

- *Description:* `X` defines a valid Finite Domains constraint using the variable names in the problem.
- *Should hold globally:*
  Defines the regular type: `term`. (`defines_type/2`)

action/1: TYPE

Usage: `action(X)`

- *Description:* `X` defines a choicepoint.
- *Should hold globally:*
  Defines the regular type: `^$no_back(atm)`. (`defines_type/2`)

action2/1: TYPE

Usage: `action2(X)`

- *Description:* `X` defines other type of choicepoint.
- *Should hold globally:*
  Defines the regular type: `^$no_back_ac(atm)`. (`defines_type/2`)

Concept Index

Predicate/Method Definition Index

R

S

T

U

V

Type Definition Index

A

C

F

I

L

P

S

T

Usage Index

This index contains pointers to places where predicates, functions, properties, types, applications, etc., are referred to in the text of the manual. Note that due to limitations of the info format only the first reference will appear in online versions of the manual.

L

M

O

P

R

S

T

U

V

W

Table of Contents