

Oz Explorer – Visual Constraint Programming Support

Christian Schulte

**Version 1.2.3
December 1, 2001**



Abstract

The Oz Explorer is a graphical and interactive tool to visualize and analyze search trees. A research paper on the Oz Explorer is [1].

Credits

Mozart logo by Christian Lindig

License Agreement

This software and its documentation are copyrighted by the German Research Center for Artificial Intelligence (DFKI), the Swedish Institute of Computer Science (SICS), and other parties. The following terms apply to all files associated with the software unless explicitly disclaimed in individual files.

The authors hereby grant permission to use, copy, modify, distribute, and license this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. No written agreement, license, or royalty fee is required for any of the authorized uses. Modifications to this software may be copyrighted by their authors and need not follow the licensing terms described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE AND ITS DOCUMENTATION ARE PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Contents

1	Summary	1
2	Invoking the Explorer	3
3	The User Interface	5
4	The Search Tree	7
5	Status Information	9
6	The Menu Bar	11
6.1	The Explorer Menu	11
6.2	The Move Menu	12
6.3	The Search Menu	12
6.4	The Nodes Menu	13
6.5	The Hide Menu	13
6.6	The Options Menu	14
7	User-defined Actions	17
7.1	Information Actions	17
7.2	Compare Actions	18
7.3	Statistics Actions	18
8	The Explorer Object	21
8.1	Invoking	21
8.2	User-defined Actions	21
8.3	Options	21

Summary

This chapter briefly outlines the features of the Oz Explorer.

Invoking The Explorer is invoked by applying it to a *script*. It can be applied to both a script and an *order*, which switches the Explorer to branch and bound mode such that it supports best solution search.

Exploring The search tree of the current script can be explored in a user-guided manner. Starting from any node in the search tree further parts can be explored. Explored parts of the search tree are drawn as they are explored.

Search trees can be explored in a depth-first fashion up to the next solution. Entire subtrees as well as single nodes can be explored. Exploration and/or drawing can be stopped and resumed at any time.

Information Access Nodes in the tree representing choices and solutions carry as information their corresponding computation spaces. The Explorer gives first-class access to them by predefined or user-defined procedures.

Statistical information is available in a status bar during exploration. Detailed statistical information with respect to a subtree can be accessed by predefined or user-defined procedures.

Display Economics The display of large search trees can be kept economic by scaling the tree and by hiding subtrees. Support for automatically hiding complete subtrees not containing solutions is provided. Hidden subtrees can be unhidden on demand.

Recomputation The Explorer can run scripts with a large number of constraints and propagators and with large search trees efficiently with respect to both space and time. User-configurable recomputation can be employed to trade space for time for scripts which would require to much memory otherwise.

Postscript Output The search tree of a script can be dumped in postscript format.

Finding Nodes Nodes in the search tree can be selected directly or by convenient short cuts.

Invoking the Explorer

The Explorer is provided as an object `Explorer.object`. The Explorer can be applied to a script as follows.

script

```
{Explorer.object script(+ScriptP)}
{Explorer.object script(+ScriptP +OrderP)}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure). If the binary procedure *OrderP* is given, the Explorer runs this script in branch and bound mode.

one

```
{Explorer.object one(+ScriptP)}
{Explorer.object one(+ScriptP +OrderP)}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure) and starts exploration of the search tree up to the first solution. If the binary procedure *OrderP* is given, the Explorer runs this script in branch and bound mode.

all

```
{Explorer.object all(+ScriptP)}
{Explorer.object all(+ScriptP +OrderP)}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure) and starts exploration of the entire search tree. If the binary procedure *OrderP* is given, the Explorer runs this script in branch and bound mode.

The following procedures are provided for convenience.

one

```
{Explorer.one +ScriptP}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure) and starts exploration of the search tree up to the first solution.

all

```
{Explorer.all +ScriptP}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure) and starts exploration of the entire search tree.

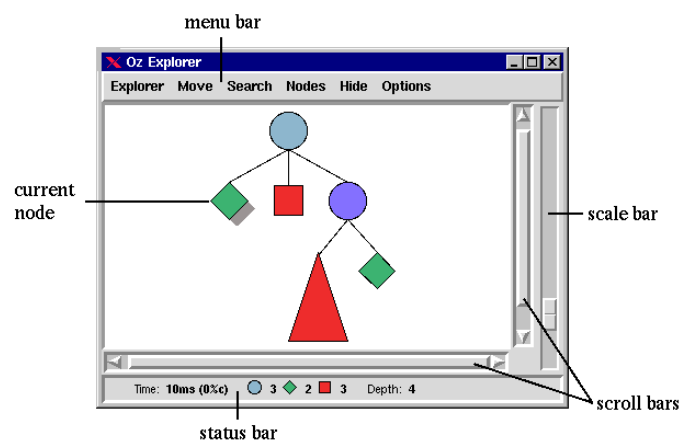
`best`

```
{Explorer.best +ScriptP +OrderP}
```

Initializes the Explorer with the script *ScriptP* (a unary procedure) and starts exploration of the entire search tree following a branch and bound strategy. Best solution is performed with respect to *OrderP* (a binary procedure).

The User Interface

Figure 3.1: The user interface of the Explorer.



The graphical user interface of the Explorer is shown in Figure 3.1. The main components of the graphical user interface are as follows.

Menu Bar The *menu bar* gives access to the Explorer's operations. Accelerators to operations are available with the keyboard and with the mouse. Chapter 6 describes the operations.

Scale Bar The *scale bar* allows to scale the search tree. Clicking with the right mouse button scales the search tree such that it fits the size of the window (if possible).

Scrollbars With the vertical and horizontal *scrollbars* the visible part of the search tree can be adjusted.

Status Bar The *status bar* contains status information on the Explorer and on the search tree. See Chapter 5 for more information.

Selected Node Within the search tree, a single node can be selected. The selected node is drawn shadowed. Most operations execute with respect to the currently selected node or with respect to the subtree starting at the currently selected node.

Current Node, Current Subtree To the selected node (subtree) we also refer to as the current node (current subtree).

Figure 3.1 does not show the mouse pointer. From the mouse pointer one can get additional status information. This is explained in Chapter 5.

The Search Tree

Figure 4.1: Nodes in the search tree.













Node	Kind	Color	BW
choice	open	 light blue	 thin border
choice	closed	 blue	 thick border
solved	suspended	 light green	 thin border
solved	entailed	 green	 thick border
failed		 red	 thin border
suspended		 orange	 thin border

Figure 4.1 sketches how the different nodes are drawn by the Explorer.





Choice nodes Choice nodes are drawn as circles. A choice node is *closed*, if all of its direct descendants are explored, otherwise it is *open*.

Solved nodes Solved nodes are drawn as diamonds. The Explorer distinguishes between nodes corresponding to *entailed* (i.e., spaces where no actors and propagators are left) or *suspended* (i.e., spaces where actors or propagators are left) computation spaces.

Failed nodes Failed nodes are drawn as rectangular boxes.

Suspended nodes Suspended nodes are drawn as stars. If the search tree contains a suspended node, it cannot be further explored. If the corresponding computation space becomes stable, the node will change to a node corresponding to the now stable space.

Figure 4.2: Hidden subtrees in the search tree.

Explored Solutions Color		
partially	yes	 light green
partially	no	 purple
fully	yes	 green
fully	no	 red

Hidden subtrees Besides of nodes, the search tree may also contain hidden subtrees. Hidden subtrees are drawn as triangles, see Figure 4.2. More on hiding subtrees can be found in Section 6.5.



Selecting nodes Choice nodes, solved nodes, and hidden subtrees can be selected by clicking on them with the left mouse button. The selected node is drawn shadowed (See also Chapter 3).

Numbered nodes Invoking an action on a node marks the node with a number. For invoking actions see Section 6.4.

Status Information

Status bar The status bar displays the most important information on the Explorer and the search tree.

BAB To the left of the status bar the letters *BAB* (for branch and bound) are displayed, if the Explorer is currently running in best solution search mode. Otherwise, no letters are displayed.

Time The time gives the runtime spent so far for exploration, where time for garbage collection is excluded. The percentage figure displayed in parenthesis shows the amount of time spent on copying.

Nodes The number of occurrences of each kind of node follows then to the right. The choice node drawn in the status bar is of special importance: if the search tree has been explored completely, the choice node is drawn as a closed node, otherwise as an open choice node (see Chapter 4 on open and closed choice nodes).

Depth The depth shows the depth of the search tree, that is the number of nodes on the longest path in the search tree.

Mouse pointer When the Explorer is idle, the mouse pointer is displayed as an arrow. If the Explorer is exploring the search tree the mouse pointer has the shape of a watch. During drawing, the mouse pointer shows a pencil.

The Menu Bar

All operations available with the Explorer can be chosen from menu entries. Some operations can be invoked by keyboard accelerators. The most important operations can be invoked with the mouse.

Keyboard accelerators are shown to the right of menu entries. Accelerators beginning with C- require the control-key to be pressed together with the key. Operations available via mouse buttons are tagged by a small figure to the left of the explanation.

6.1 The Explorer Menu

This menu contains operations to stop, reset, and quit the Explorer.

About . .

Displays a window containing short information on the Explorer.

Halt

C-g

Halts exploration of the search tree, but does not halt drawing of newly explored parts.

Break

C-c

Breaks both exploration and drawing of the search tree. Drawing is stopped as follows: not yet drawn subtrees are drawn as hidden.

Reset

C-r

Resets the Explorer such that only the top node of the search tree is explored and drawn.

Export Postscript . .

Opens a window to choose a file. After a file has been selected, the drawing of the search tree is dumped to this file in postscript format.

Close

C-x

The Explorer window is closed. When the Explorer is invoked again, a new Explorer window is created.

6.2 The Move Menu

Operations accessible from this menu manipulate the current node.

Center **c**

Scrolls the search tree such that the current node is centered (if possible).

Top Node **t**

Makes the top most node of the search tree the current node.

Leftmost **-**

Makes the leftmost node of the search tree the current node.

Rightmost **+**

Makes the rightmost node of the search tree the current node.

Backtrack **b**

Makes the nearest open choice node the current node. Nearest means the nearest node which is above and to the left of the current node.

Previous Solution **<**

Makes the previous solution (i.e., the nearest solution to the left of the current node) the current node.

Next Solution **>**

Makes the next solution (i.e., the nearest solution to the right of the current node) the current node.

6.3 The Search Menu

Operations available from this menu explore the search tree.

Next Solution **n**



Explores the search tree starting from the current node up to the next solution. Exploration can be stopped as described in Section 6.1.

All Solutions **a**

Explores the entire current subtree. Exploration can be stopped as described in Section 6.1.

One Step **o**



Performs a single distribution step starting from the current choice node.

6.4 The Nodes Menu

Operations available from this menu allow to select and invoke actions.

Information Action

Shows a sub menu from which an information action can be selected. Chapter 7 describes how new actions can be defined.

Information

i

Invokes the currently selected information action (see above) on the current node.

Compare Action

Shows a sub menu from which a compare action can be selected. Chapter 7 describes how new actions can be defined.

Select Compare

1

Selects the current node as *compare node*. Selecting a different node draws an arrow from the compare node to the new current node.

Deselect Compare

0

Deselects the compare node and deletes the arrow.

Compare

2

Applies the currently selected compare action to the compare node (i.e., the node from which the arrow issues) and the current node.

Statistics Action

Shows a sub menu from which a statistics action can be selected. Chapter 7 describes how new actions can be defined.

Statistics

s

Applies the currently selected statistics action to the current node.

6.5 The Hide Menu

This menu features operations for hiding and unhiding subtrees. The drawing of subtrees during unhiding can be stopped as described in Section 6.1.

Hide/Unhide

h

If the current subtree is hidden, it gets unhidden. If the current subtree is not hidden, it gets hidden. Unhiding is not recursive: If the hidden tree contains hidden subtrees itself, they remain hidden.

Hide Failed **f**

All completely explored subtrees in the current subtree that do not contain a solution are hidden.

Unhide But Failed **u**

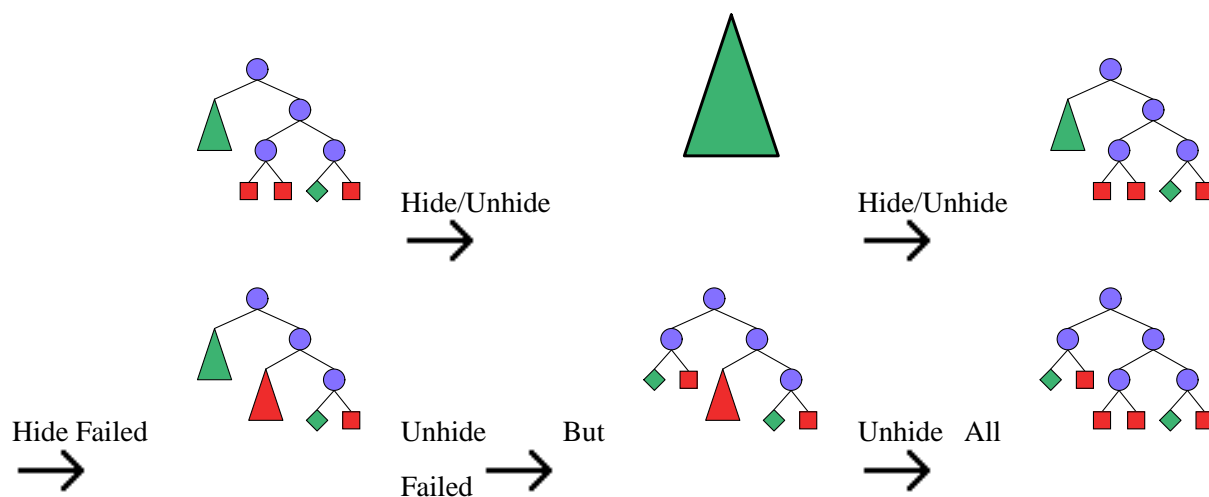


All subtrees of the current subtree with the exception of completely explored subtrees not containing a solution are unhidden.

Unhide All **C-u**

All subtrees of the current subtree are unhidden.

Figure 6.1: Example for hiding and unhiding subtrees.



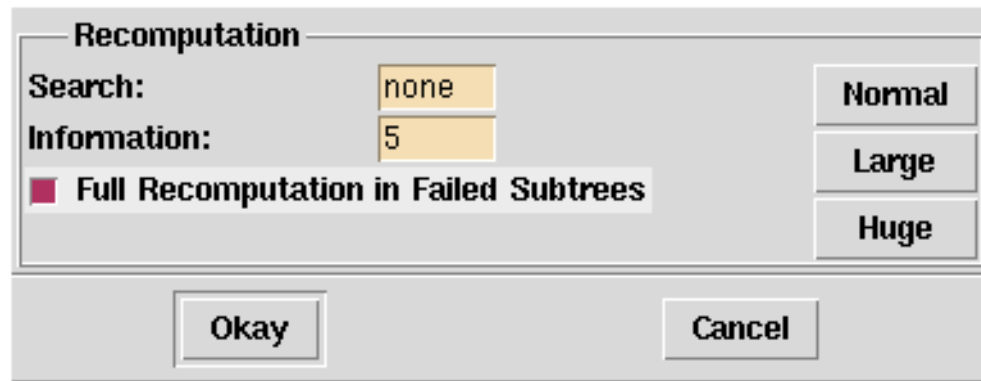
The different operations to hide and unhide trees are illustrated by an example in Figure 6.1. All operations are invoked with the entire tree as current subtree.

6.6 The Options Menu

This menu contains access to dialogs from which the Explorer can be configured.

Search ...

Creates a dialog to set options for the Explorer's search engine.



The **Recomputation** dialog box contains the following elements:

- Search:** A text entry field containing the word `none`.
- Information:** A text entry field containing the number `5`.
- Full Recomputation in Failed Subtrees:** A checkbox that is currently checked.
- Recomputation Level Buttons:** Three buttons labeled **Normal**, **Large**, and **Huge** are stacked vertically on the right side.
- Navigation Buttons:** **Okay** and **Cancel** buttons are at the bottom.

In the *Search* entry the kind of recomputation used during next and all solution search (see also Section 6.3) can be entered. Entering `none` means that in each distribution step during search a space is stored. Entering `full` means that no spaces at all are stored during search. Entering a number n means that only in each n -th distribution step a space gets stored. Roughly, with a recomputation distance of n , the time needed during search is increased by a factor of n and memory occupied is decreased by a factor of n .

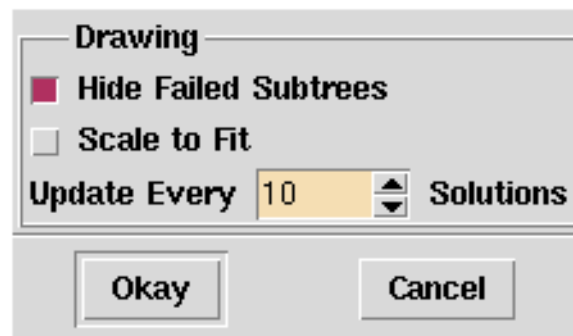
In the *Information* entry the kind of recomputation used for accessing the information attached to nodes. Entering `none` means that each choice and solved node stores a computation space. Entering `full` means that only the top node stores its corresponding computation space. The space of any other node is recomputed by redoing all distribution steps. The number of required distribution steps thus depends on the depth of the node. Entering a number n means that only nodes at a depth $1, n + 1, 2n + 1, \dots$ store a computation space. Thus, in the worst case the access to a node's space recomputes $n - 1$ distribution steps.

Selecting *Full Recomputation in Failed Subtrees* means that in subtrees that are explored completely and do not contain a solution, no spaces are stored.

Pressing the buttons *Normal*, *Large*, and *Huge* enter values to the recomputation entries. *Normal* is the default setting for scripts which do not contain very much propagators (i.e., about a few hundred). The other two buttons suggest values for scripts with more propagators or deep search trees.

Drawing ...

Creates a dialog to set options used for the drawing of the search tree.



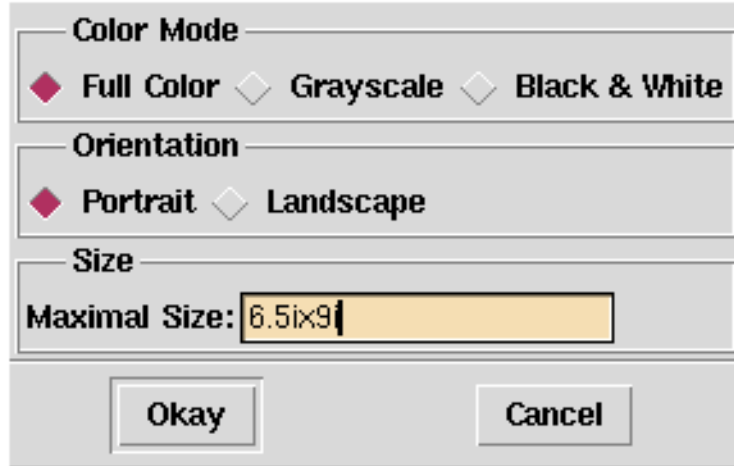
The **Drawing** dialog box contains the following elements:

- Hide Failed Subtrees:** A checked checkbox.
- Scale to Fit:** An unchecked checkbox.
- Update Every:** A text entry field containing the number `10`, followed by a spin button.
- Solutions:** A text entry field.
- Navigation Buttons:** **Okay** and **Cancel** buttons are at the bottom.

The entries in the dialog are self-explanatory.

Postscript ...

Creates a dialog to set options used for dumping the search tree in postscript format (see also Section 6.1).



The format of the string to be entered in the *Maximal size* field must be as follows:

num dim x num dim

where *dim* must be one of *i* (inch), *c* (centimeter), *m* (millimeter), or *p* (point, that is 1/72 inch).

The options can also be configured by sending a message to `Explorer.object`, for details see Chapter 8.

User-defined Actions

Actions for the Explorer can be user-defined. They must be provided as procedures to the `Explorer.object`.

7.1 Information Actions

`add`

```
{Explorer.object add(information +PA
                      label: +V <= _
                      type:  +A <= root
```

If *PA* is the atom *separator*, a separator entry is added as last entry of the *Information Action* sub menu.

Otherwise, an entry with label *V* (a virtual string) is added to the *Information Action* sub menu, from which *PA* can be selected. If the `label` feature is missing, the print-name of *PA* is taken as label instead. *PA* must be either a binary or ternary procedure.

When the information action is invoked, *PA* is applied with the integer tagging the current node as first actual argument. The second actual argument depends on *A*. If *A* is `root` (the default) the argument is the root variable of the space attached to the current node. If *A* is `space` the argument is the space itself. If *A* is `procedure` the argument is a unary procedure *P*. On application, *P* returns the root variable of a copy of the space.

If *PA* is a ternary procedure, it must return either a nullary procedure *P* or a pair *O#M* of an object *O* and a message *M*. When the Explorer is cleared, reset, or closed, a new thread is created that runs either *{P}* or *{O M}* is executed.

For example, the default information action can be defined as follows:

```
{Explorer.object add(information proc {$ I X}
                      {Inspector.inspect I#X}
                      end
                      label: 'Inspect')}}}
```

`delete`

```
{Explorer.object delete(information +PA)}
```

Deletes the information action *PA* from the *Information Action* submenu. If *PA* is the atom `all`, all but the default information actions are deleted from the submenu.

For example, after adding an information action *P* by

```
{Explorer add(information P)}
```

it can be deleted by

```
{Explorer delete(information P)}
```

7.2 Compare Actions

add

```
{Explorer.object add(compare +PA
                        label: +V <= _
                        type:  +A <= root
```

If *PA* is the atom `separator`, a separator entry is added to the bottom of the *Compare Action* submenu.

Otherwise, an entry with label *V* (a virtual string) is added to the *Compare Action* submenu, from which *PA* can be selected. If the `label` feature is missing, the printname of *PA* is taken as label instead. *PA* must be either a \$4\$-ary or \$5\$-ary procedure. When the compare action is invoked, *PA* is applied with the integer tagging the compare (current) node as first (third) actual argument. The second (fourth) actual argument is the computation space attached to the compare (current) node. These arguments depend on *A* as explained for adding information actions.

If *PA* is a \$5\$-ary procedure, it must return either a nullary procedure *P* or a pair *O#M* of an object *O* and a message *M*. When the Explorer is cleared, reset, or closed, a new thread is created that runs either *{P}* or *{O M}* is executed.

For example, the default compare action can be defined as follows:

```
{Explorer.object add(compare proc {$ I1 X1 I2 X2}
                             {Inspector.inspect I1#I2#X1#X2}
                             end
                        label: 'Inspect' )}
```

delete

```
{Explorer.object delete(compare +PA)}
```

Deletes the compare action *P* from the *Compare Action* submenu. If *PA* is the atom `all`, all but the default compare actions are deleted from the submenu.

7.3 Statistics Actions

add

```
{Explorer.object add(statistics +PA
                      label: +V <= _)
```

If *PA* is the atom `separator`, a separator entry is added to the bottom of the *Statistics Action* sub menu.

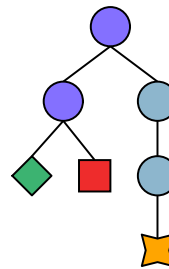
Otherwise, an entry with label *V* (a virtual string) is added to the *Statistics Action* sub menu, from which *PA* can be selected. If the `label` feature is missing, the printname of *PA* is taken as label instead. *PA* must be either a binary or ternary procedure. When

the information action is invoked, *PA* is applied with the integer tagging the current node as first actual argument. The second actual argument is a record as follows:

```
stat(c: ... % number of choice nodes
    s: ... % number of solved nodes
    f: ... % number of failed nodes
    b: ... % number of suspended nodes
    start: ... % depth of current node
    depth: ... % depth of current subtree
    shape: ... % describes current subtree
)
```

The value for the feature *shape* is either *s*, *f*, or *b* with the meaning from above or a unary tuple with label *c*. The argument of the tuple is a list where the elements recursively describe the shapes of the choice node's subtrees.

For example, invoking a statistics action on the following subtree;



the second actual argument is as follows:

```
stat(c:4 s:1 f:1 b:1
    start:1
    depth:4
    shape: c([c([s f])
              c([c([b])])])])
)
```

If *PA* is a ternary procedure, it must return either a nullary procedure *P* or a pair *O#M* of an object *O* and a message *M*. When the Explorer is cleared, reset, or closed, a new thread is created that runs either *{P}* or *{O M}* is executed.

For example, the default statistics action can be defined as follows:

```
{Explorer.object add(statistics
    proc {$ I R}
        {Inspector.inspect I#{Record.subtract R shape}}
    end
    label: 'Inspect' )}
```

delete

```
{Explorer.object delete(statistics +PA)}
```

Deletes the statistics action *P* from the *Statistics Action* submenu. If *PA* is the atom *all*, all but the default statistics actions are deleted from the submenu.

The Explorer Object

This section lists all methods of `Explorer.object`. New Explorers can be created by creating new objects from the class `Explorer.class`. Execution of

```
MyExplorer = {New Explorer.class init}
```

creates a new Explorer which can be accessed by the variable `MyExplorer`.

8.1 Invoking

`script, one, all`

```
{Explorer.object script(+ScriptP +OrderP <= _)}
{Explorer.object one(+ScriptP +OrderP <= _)}
{Explorer.object all(+ScriptP +OrderP <= _)}
```

Invokes the Explorer. For a detailed description see Chapter 2.

8.2 User-defined Actions

`add`

```
{Explorer.object add(...)}
```

Adds new actions to the Explorer. For a detailed description see Chapter 7.

`delete`

```
{Explorer.object delete(...)}
```

Deletes actions from the Explorer. For a detailed description see Chapter 7.

8.3 Options

`option`

```
{Explorer.object option(search search:      +AI1<=_
                           information: +AI2<=_
                           failed:      +B<=_)}
```

Sets the options for search as described in Section 6.6. *AI1* and *AI2* must be either `none`, `full`, or an integer.

option

```
{ {Explorer.object option(drawing hide: +B1<=_  
                           scale: +B2<=_  
                           update: +I<=_)}
```

Sets the options for search as described in Section 6.6. *A11* and *A12* must be either *none*, *full*, or an integer.

option

```
{ {Explorer.object option(postscript color: +A1<=_  
                           orientation: +A2<=_  
                           size: +V<=_)}
```

Sets the options for postscript output as described in Section 6.6. *A1* must be one of *full*, *grayscale*, or *bw*. *A2* must be one of *portrait* or *landscape*. *V* must be a virtual string built according to the same rules as described in Section 6.6.

Bibliography

- [1] Christian Schulte. Oz Explorer: A visual constraint programming tool. In Lee Naish, editor, *Proceedings of the Fourteenth International Conference on Logic Programming*, pages 286–300, Leuven, Belgium, July 1997. The MIT Press.

Explorer

Explorer, all, 3

Explorer, best, 4

object

Explorer, object, add, 17, 18

Explorer, object, all, 3

Explorer, object, delete, 17–
19

Explorer, object, one, 3

Explorer, object, script, 3

Explorer, one, 3