User Manual of the RODIN Platform

July 2007

Version 1.4

User Manual of the RODIN Platform

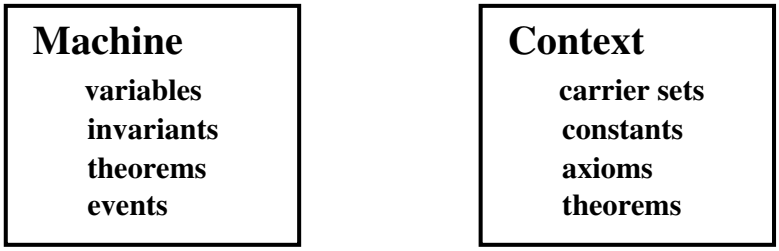
Contents

The reader of this Manual is supposed to have a basic acquaintance with Eclipse.

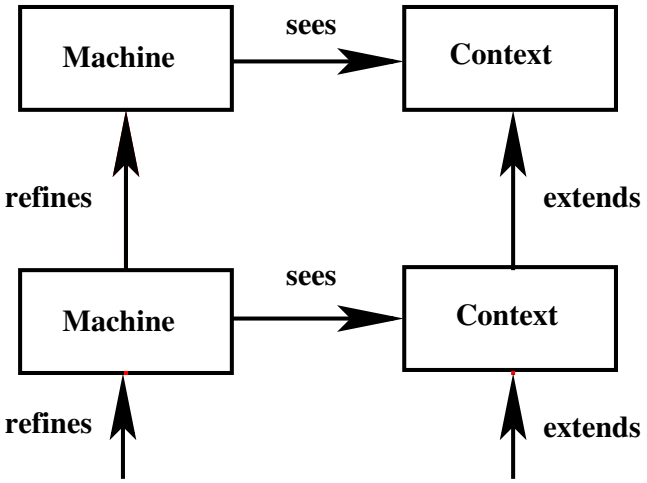
1 Project

1.1 Project Constituents and Relationships

The primary concept in doing formal developments with the Rodin Platform is that of a *project*. A project contains the complete mathematical development of a *Discrete Transition System*. It is made of several components of two kinds: *machines* and *contexts*. Machines contain the variables, invariants, theorems, and events of a project, whereas contexts contain the carrier sets, constants, axioms, and theorems of a project:

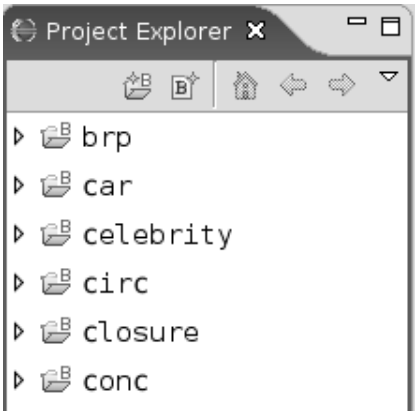


We remind the reader of the various relationships existing between machines and contexts. This is illustrated in the following figure. A machine can ”refined” by another one, and a context can be ”extended” by another one (no cycles are allowed in both these relationships). Moreover, a machine can ”see” one or several contexts. A typical example of machine and context relationship is shown below:

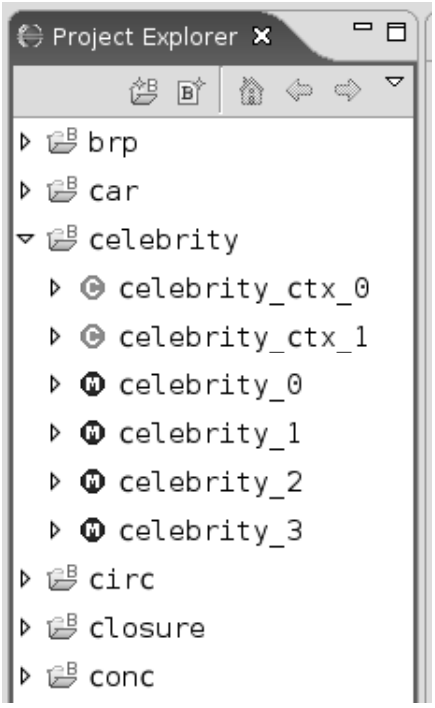


1.2 The Project Explorer

Projects are reachable in the RODIN platform by means of a window called the ”Project Explorer”. This window is usually situated on the left hand side of the screen (but, in Eclipse, the place of such windows can be changed very easily). Next is a screen shot showing a ”Project Explorer” window:



As can be seen on this screen shot, the Project Explorer window contains the list of current project names. Next to each project name is a little triangle. By pressing it, one can expand a project and see its components as shown below.

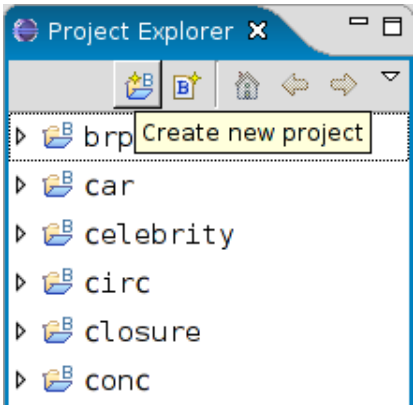


We expanded the project named ”celebrity”. This project contains 2 contexts named ”celebrity_ctx_0” and ”celebrity_ctx_1”. It also contains 4 machines named ”celebrity_0” to ”celebrity_3”. The icons ((c) or (m)) situated next to the components help recognizing their kind (context or machine respectively)

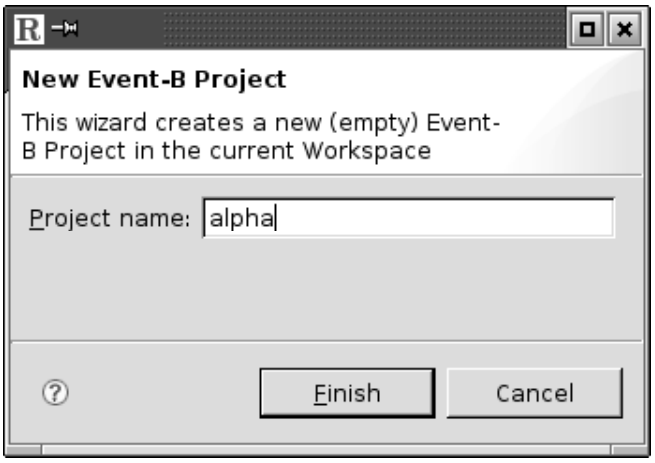
In the remaining parts of this section we are going to see how to create (section 1.3), remove (section 1.4), export (section 1.5), import (section 1.6), change the name (section 1.7) of a project, create a component (section 1.8), and remove a component (section 1.9). In the next two sections (2 and 3) we shall see how to manage contexts and machines.

1.3 Creating a Project

In order to create a new project, simply press the button as indicated below in the ”Project Explorer” window:



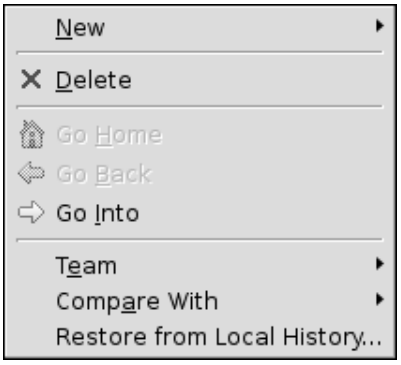
The following wizard will then appear, within which you type the name of the new project (here we type ”alpha”):



After pressing the ”Finish” button, the new project is created and appears in the ”Project Explorer” window.

1.4 Removing a Project

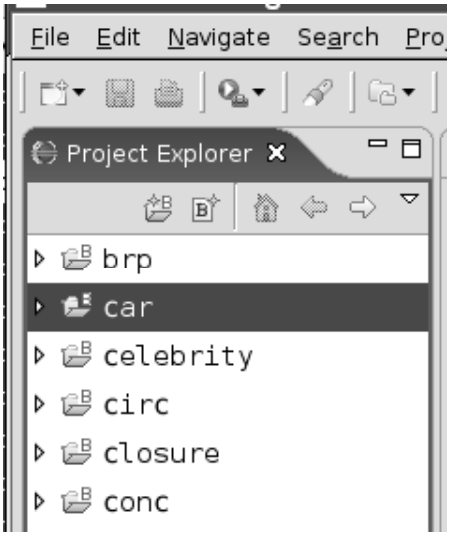
In order to remove a project, you first select it on the ”Project Explorer” window and then right click with the mouse. The following contextual menu will appear on the screen:



You simply click on ”Delete” and your project will be deleted (confirmation is asked). It is then removed from the Project Explorer window.

1.5 Exporting a Project

Exporting a project is the operation by which you can construct automatically a ”zip” file containing the entire project. Such a file is ready to be sent by mail. Once received, an exported project can be *imported* (next section), it then becomes a project like the other ones which were created locally. In order to export a project, first select it, and then press the ”File” button in the menubar as indicated below:



A menu will appear. You then press ”Export” on this menu. The Export wizard will pop up. In this window, select ”Archive File” and click ”Next ¿”. Specify the path and name of the archive file into which you want to export your project and finally click ”Finish”. This menu sequence belongs (as well as the various options) to Eclipse. For more information, have a look at the Eclipse documentation.

1.6 Importing a Project

A ”.zip” file corresponding to a project which has been exported elsewhere can be imported locally. In order to do that, click the ”File” button in the menubar (as done in the previous section). You then click

”Import” in the coming menu. In the import wizard, select ”Existing Projects into Workspace” and click ”Next ¿”. Then, enter the file name of the imported project and finally click ”Finish”. Like for exporting, the menu sequence and layout are part of Eclipse.

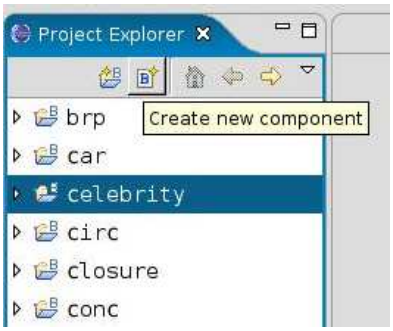
The importation is refused if the name of the imported project (not the name of the file), is the same as the name of an existing local project. The moral of the story is that when exporting a project to a partner you better modify its name in case your partner has already got a project with that same name (maybe a previous version of the exported project). Changing the name of a project is explained in the next section.

1.7 Changing the Name of a Project

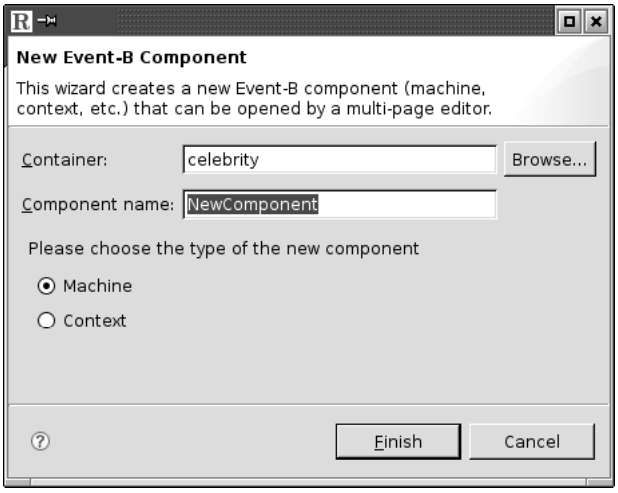
The procedure for changing the name of a project is a bit heavy at the present time. Here is what you have to do. Select the project whose name you want to modify. Enter the Eclipse ”Resource” perspective. A ”Navigator” window will appear in which the project names are listed (and your project still selected). Right click with the mouse and, in the coming menu, click ”Rename”. Modify the name and press enter. Return then to the original perspective. The name of your project has been modified accordingly.

1.8 Creating a Component

In this section, we learn how to create a component (context or machine). In order to create a component in a project, you have to first select the project and then click the corresponding button as shown below:



You may now choose the type of the component (machine or context) and give it a name as indicated:



Click ”Finish” to eventually create the component. The new component will appear in the Project Explorer window.

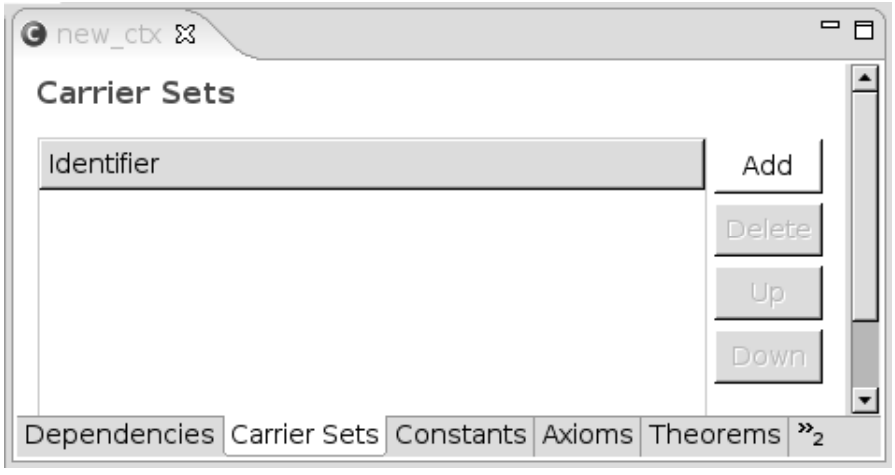
1.9 Removing a Component

In order to remove a component, press the right mouse button. In the coming menu, click ”Delete”. This component is removed from the Project Explorer window.

In the next two sections we describe in details first the contexts and then the machines.

2 Anatomy of a Context

Once a context is created, a window such as the following appears in the editing area (usually next to the center of the screen):



It shows that a context is made of various components, namely dependencies, carrier sets, constants, axioms, or theorems.

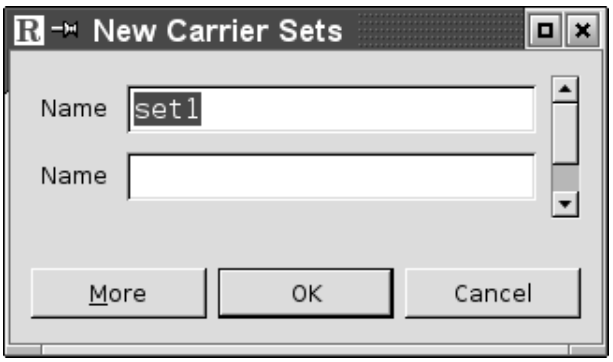
In the next sections we are going to see how to create, modify or remove such components. The creation of these components. except dependencies (studied in section 2.7), can be made by two distinct methods, either by using wizards or by editing them directly. In each section, we shall review both methods.

2.1 Carrier Sets

2.1.1 Carrier Sets Creation Wizard.

In order to activate the carrier set creation wizard, you have to press the corresponding button in the toolbar as indicated below:

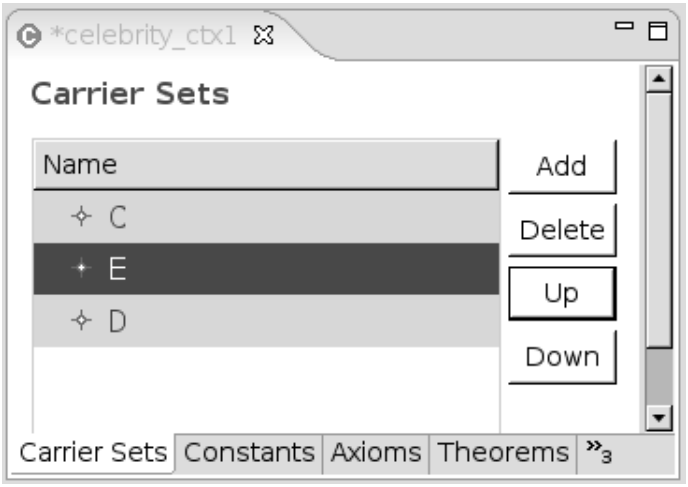
After pressing that button, the following wizard pops up:



You can enter as many carrier sets as you want by pressing the ”More” button. When you’re finished, press the ”OK” button.

2.1.2 Direct Editing of Carrier Sets.

It is also possible to create (button ”Add”) or remove (button ”Delete”) carrier sets by using the central editing window (see window below). For this, you have first to select the ”Carrier Sets” tab of the editor. Notice that you can change the order of carrier sets: first select the carrier set and then press button ”Up” or ”Down”.

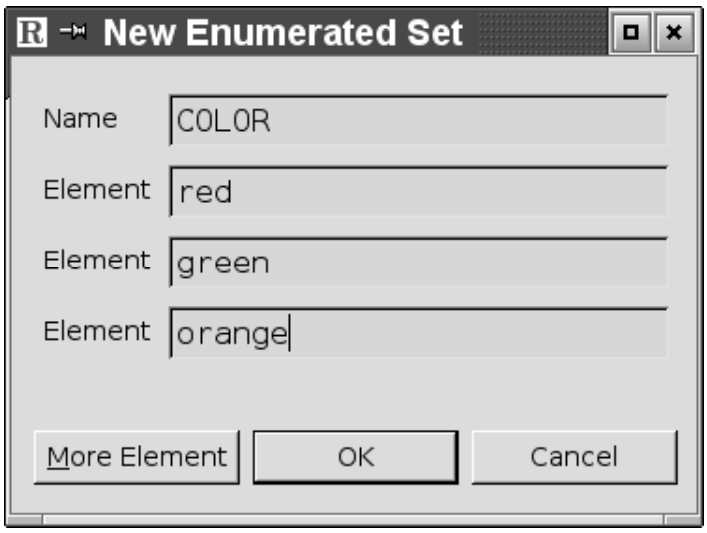


As can be seen, we have created three carrier sets C, E, and D.

2.2 Enumerated Sets

In order to activate the enumerated set creation wizard, you have to press the corresponding button in the toolbar as indicated below:

After pressing that button, the following wizard pops up:



You can enter the name of the new enumerated set as well as the name of its elements. By pressing the button "More Elements", you can enter additional elements. When you're finished, press the "OK" button. The effect of using this wizard as indicated is to add the new carrier set $COLOR$ (section 2.1) and the three constants (section 2.3) red, $green$, and $orange$. Finally, it adds the following axioms (section 2.4):

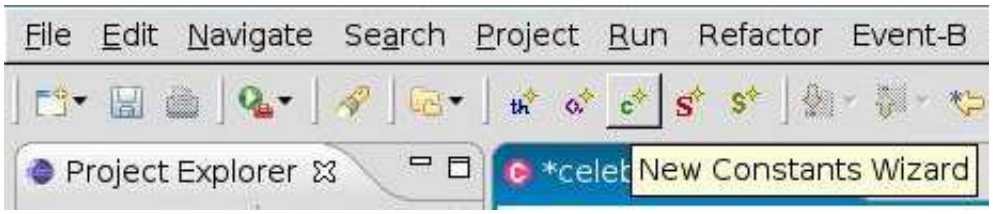
$$
\begin{array}{l}
COLOR = \{red, green, orange\} \\
red \neq green \\
red \neq orange \\
green \neq orange
\end{array}
$$

If you enter several time the same enumerated set element, you get an error message.

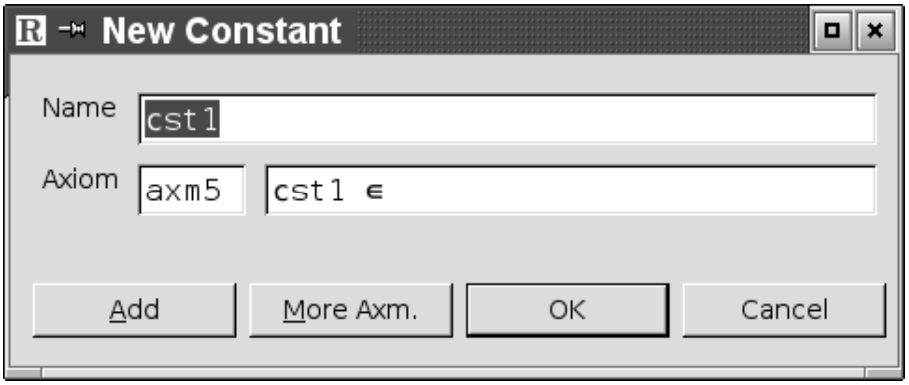
2.3 Constants

2.3.1 Constants Creation Wizard.

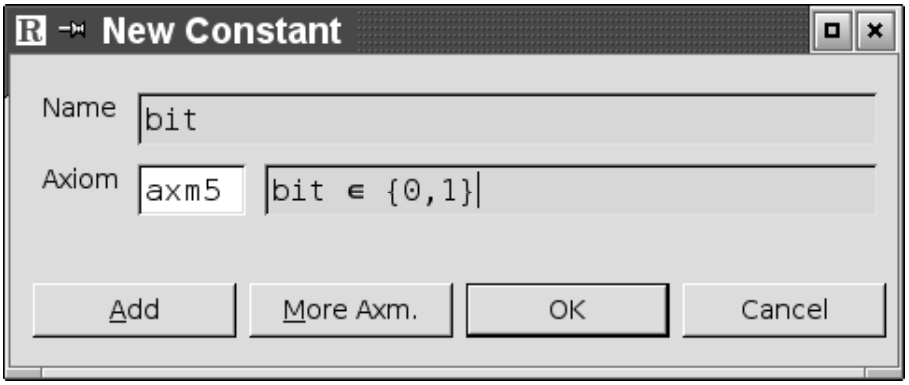
In order to activate the constants creation wizard, you have to press the corresponding button in the toolbar as indicated below:



After pressing that button, the following wizard pops up:



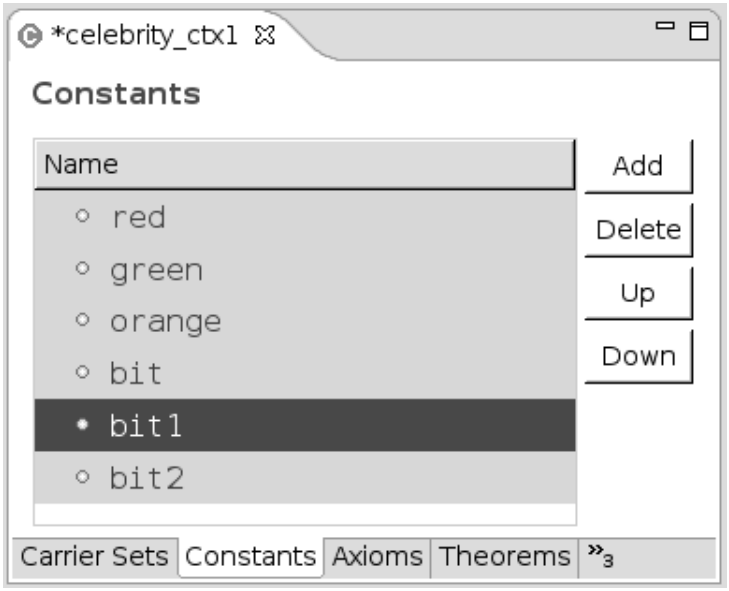
You can then enter the names of the constants, and an axiom which can be used to define its type. Here is an example:



By pressing the ”More Axm.” button you can enter additional axioms. For adding more constants, press ”Add” button. When you’re finished, press button ”OK”.

2.3.2 Direct Editing of Constants.

It is also possible to create (button ”Add”) or remove (button ”Delete”) constants by using the central editing window. For this, you have first to select the ”Constants” tab of the editor. You can also change the relative place of a constant: first select it and then press button ”Up” or ”Down”.

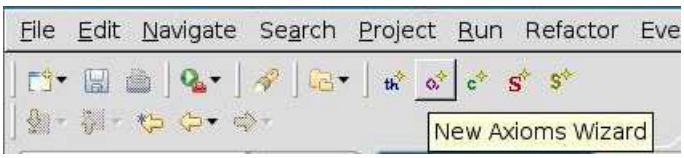


As can be seen, two more constants, $bit1$ and $bit2$ have been added. Note that this time the axioms concerning these constants have to be added directly (see next section 2.4).

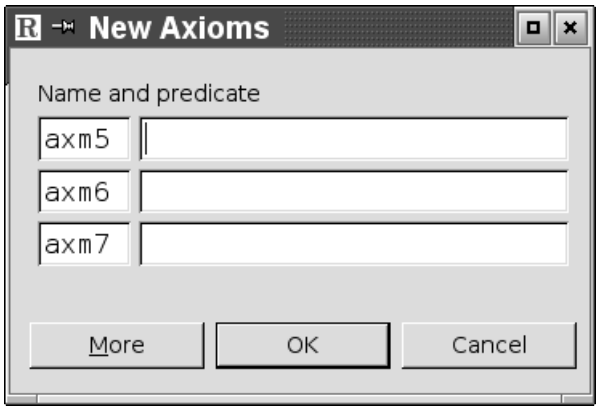
2.4 Axioms

2.4.1 Axioms Creation Wizard.

In order to activate the axioms creation wizard, you have to press the corresponding button in the toolbar as indicated below:



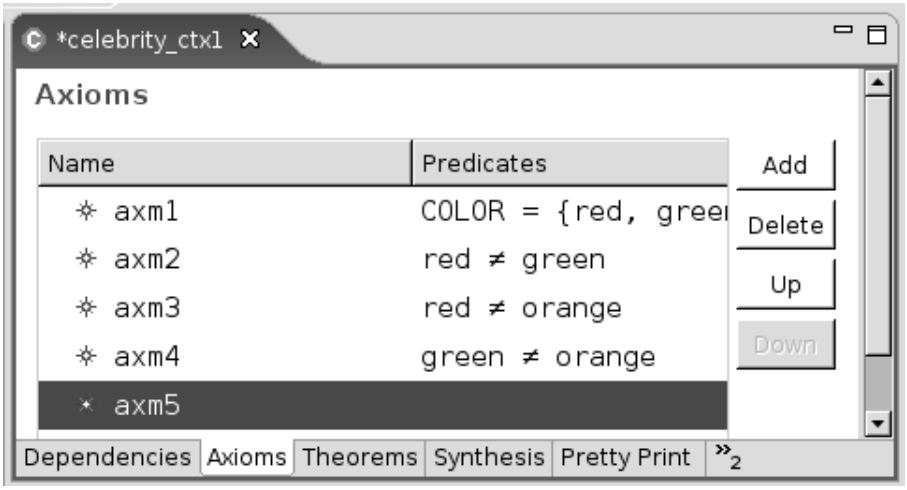
After pressing that button, the following wizard appears:



You can then enter the axioms you want. If more axioms are needed then press ”More”. When you are finished, press ”OK”.

2.4.2 Direct Editing of Axioms.

It is also possible to create (button ”Add”) or remove (button ”Delete”) axioms by using the central editing window. For this, you have first to select the ”Axioms” tab of the editor. You can also change the relative place of an axiom: first select it and then press button ”Up” or ”Down”.



Note that the ”Up” and ”Down” buttons for changing the order of axioms are important for well-definedness. For example the following axioms in that order

$$\text{axm1} : y/x = 3$$
$$\text{axm2} : x \neq 0$$

does not allow to prove the well-definedness of $y/x = 3$. The order must be changed to the following:

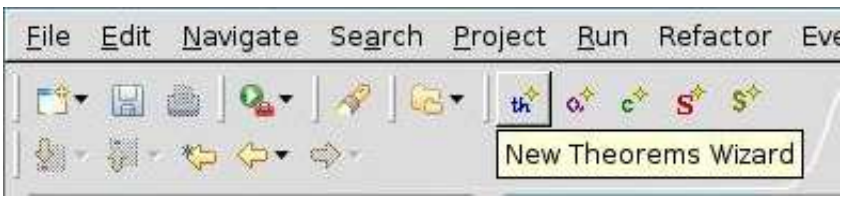
$$\text{axm2} : x \neq 0$$
$$\text{axm1} : y/x = 3$$

The same remark applies to theorems (section 2.5 and 3.4), invariants (section 3.3) and event guards (section3.5.2)

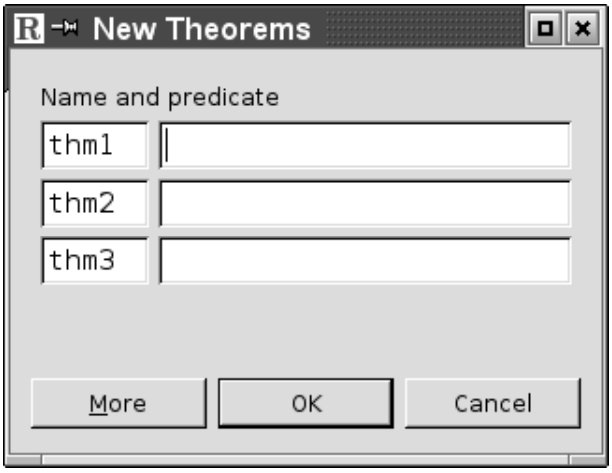
2.5 Theorems

2.5.1 Theorems Creation Wizard.

In order to activate the theorems creation wizard, you have to press the corresponding button in the toolbar as indicated below:



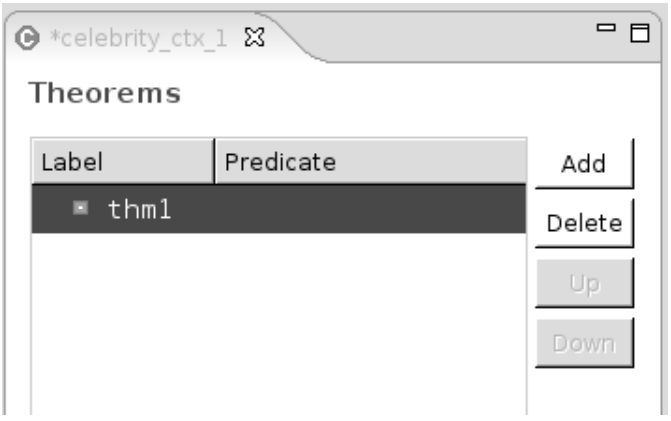
After pressing that button, the following wizard pops up:



You can then enter the theorems you want. If more theorems are needed then press ”More”. When you are finished, press ”OK”.

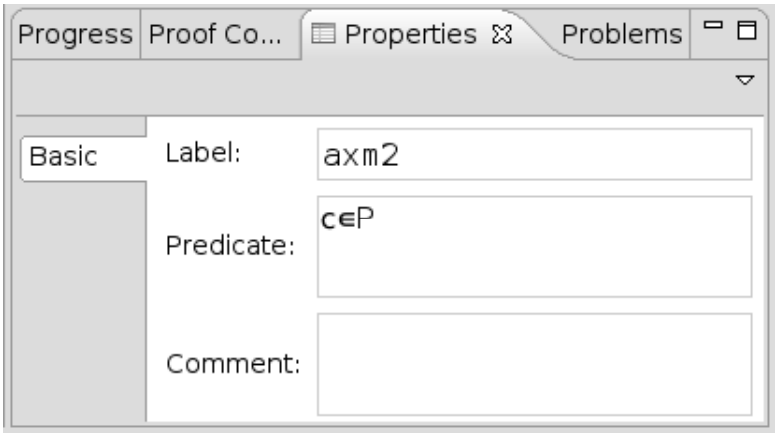
2.5.2 Direct Editing of Theorems.

It is also possible to create (button ”Add”) or remove (button ”Delete”) theorems by using the central editing window. For this, you have first to select the ”Theorems” tab of the editor. You can also change the relative place of a theorem: first select it and then press button ”Up” or ”Down”.



2.6 Adding Comments

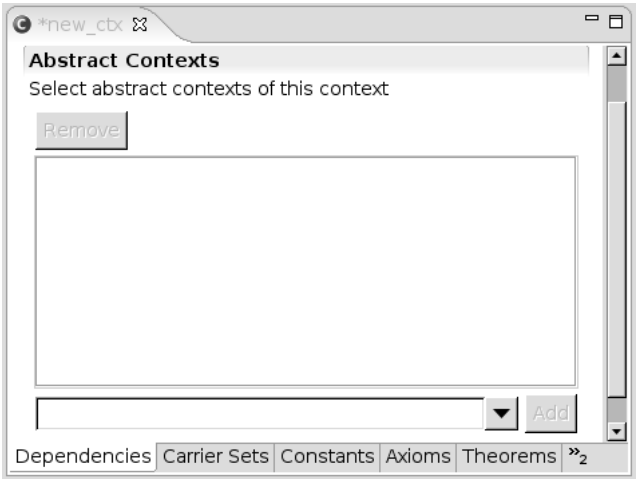
It is possible to add comments to carrier sets, constants, axioms and theorems. For doing so, select the corresponding modeling element and enter the ”Properties” window as indicated below where it is shown how to add comments on a certain axiom:



Multiline comments can be added in the editing area labeled ”Comments”.

2.7 Dependencies

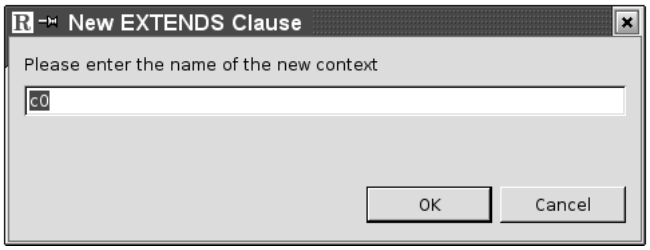
By selecting the ”Dependencies” tab of the editor, you obtain the following window:



This allows you to express that the current context is *extending* other contexts of the current project. In order to add the name of the context you want to extend, use the combobox which appears at the bottom of the window and then select the corresponding context name.

There exists another way to directly create a new context extending an existing context C. Select the context C in the project window, then press the right mouse key, you’ll get the following menu:

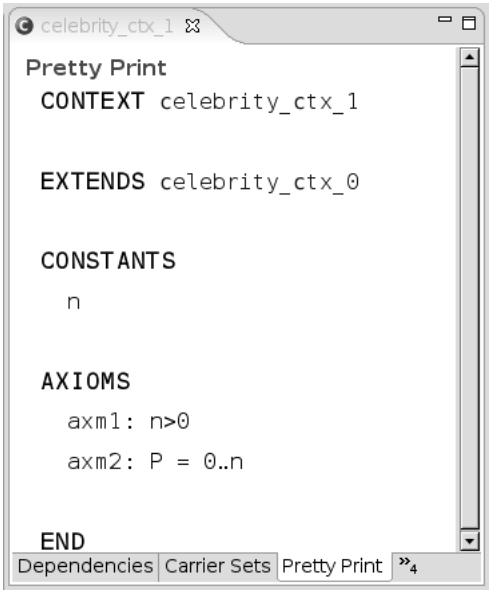
After pressing the ”Extend” button, the following wizard pops up:



You can then enter the name of the new context which will be made automatically an extension of C.

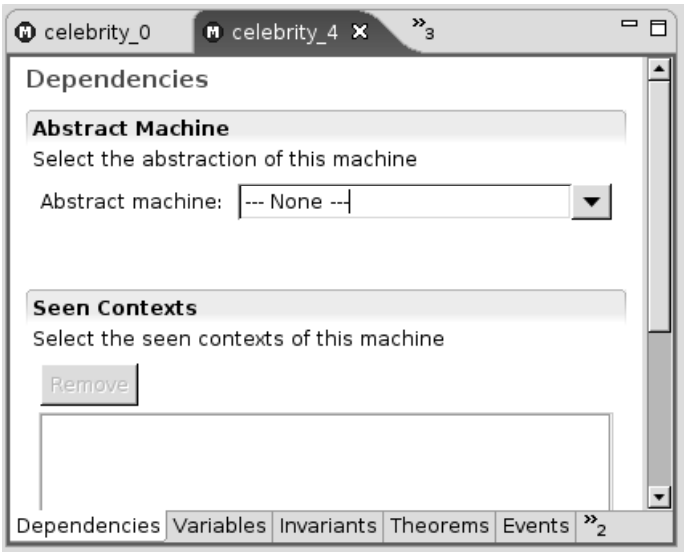
2.8 Pretty Print

By selecting the ”Pretty Print” tab of the editor, you may have a global view of your context as if it would have been entered through an input text file.



3 Anatomy of a Machine

Once a machine is created, a window such as the following appears in the editing area (usually next to the center of the screen):



It shows that a machine is made of various components, namely dependencies, variables, invariants, theorems, variant, and events. In the next sections, we are going to see how to create, modify or remove such components. Except for the dependencies, the creation of these components can be made by two distinct methods, either by using wizards or by editing them directly. In each section, we shall review both methods.

3.1 Dependencies

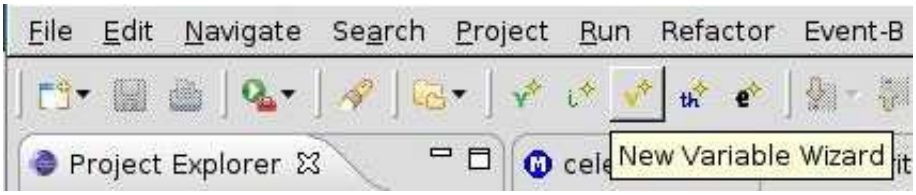
The ”Dependencies” window is shown automatically after creating a machine (you can also get it by selecting the ”Dependencies” tab). This was shown in the previous section so that we do not copy the screen shot again. As can be seen on this window, two kinds of dependencies can be established: machine dependency in the upper part and context dependencies in the lower part.

In this section, we only speak of context dependencies (machine dependency will be covered in section 3.8). It corresponds to the ”sees” relationship alluded in section 1. In the lower editing area, you can select some contexts ”seen” by the current machine.

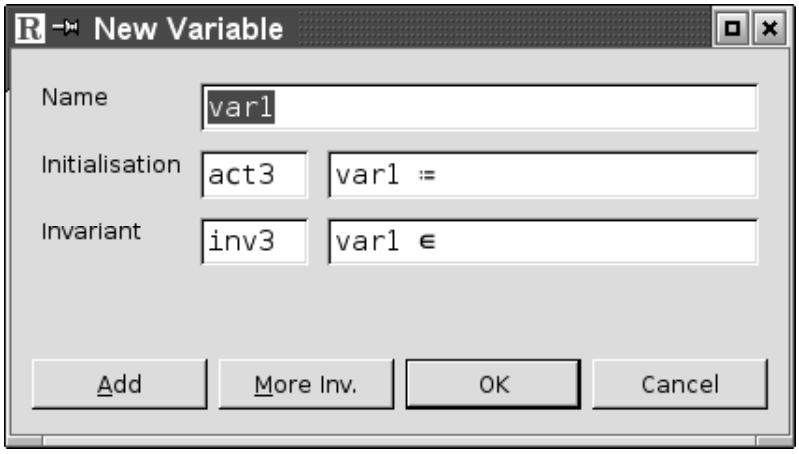
3.2 Variables

3.2.1 Variables Creation Wizard.

In order to activate the variables creation wizard, you have to press the corresponding button in the toolbar as indicated below:



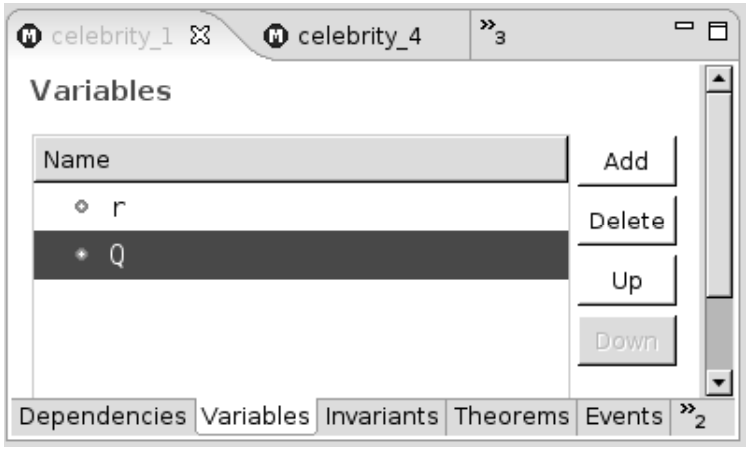
After pressing that button, the following wizard pops up:



You can then enter the names of the variables, its initialization, and an invariant which can be used to define its type. By pressing button ”More Inv.” you can enter additional invariants. For adding more variables, press the ”Add” button. When you’re finished, press the ”OK” button.

3.2.2 Direct Editing of Variables.

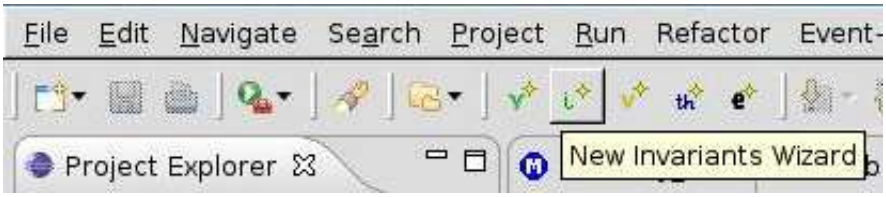
It is also possible to create (button ”Add”) or remove (button ”Delete”) variables by using the central editing window. For this, you have first to select the ”Variables” tab of the editor. You can also change the relative place of a variable: first select it and then press button ”Up” or ”Down”.



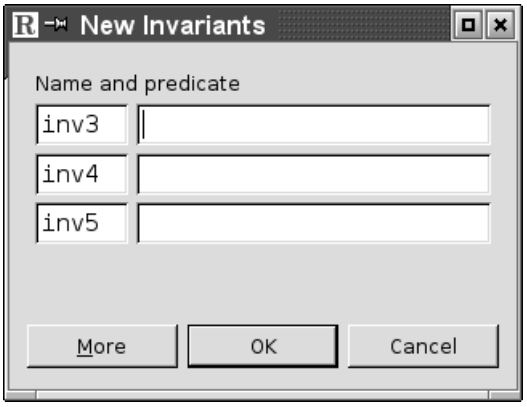
3.3 Invariants

3.3.1 Invariants Creation Wizard.

In order to activate the invariants creation wizard, you have to press the corresponding button:



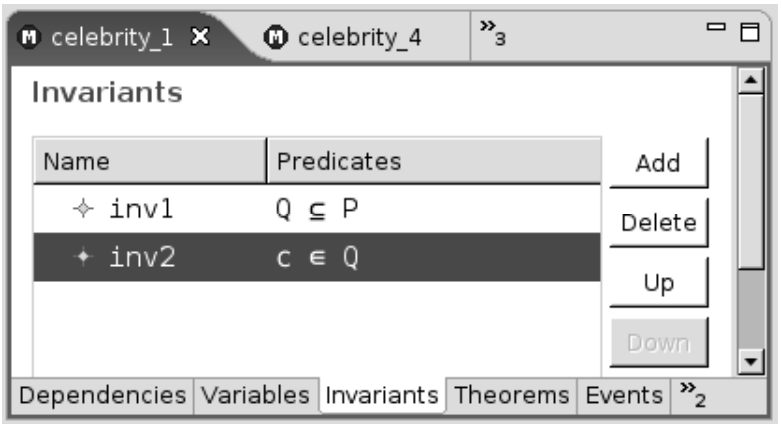
After pressing that button, the following wizard pops up:



You can then enter the invariants you want. If more invariants are needed then press ”More”.

3.3.2 Direct Editing of Invariants.

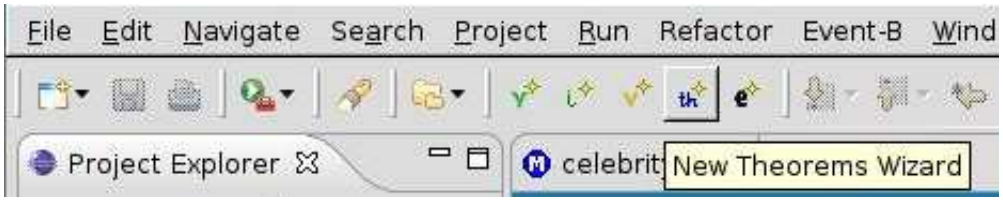
It is also possible to create (button ”Add”) or remove (button ”Delete”) invariants by using the central editing window. For this, you have first to select the ”Invariants” tab of the editor. You can also change the relative place of an invariant: first select it and then press button ”Up” or ”Down”.



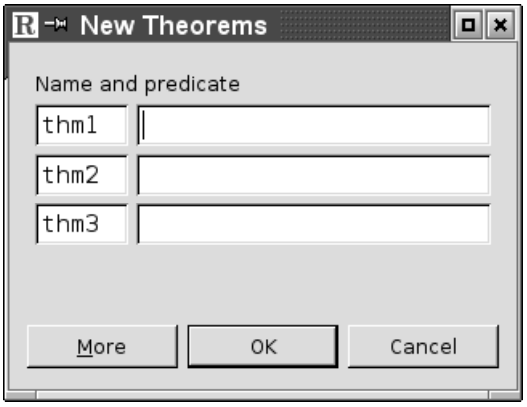
3.4 Theorems

3.4.1 Theorems Creation Wizard.

In order to activate the theorems creation wizard, you have to press the corresponding button:



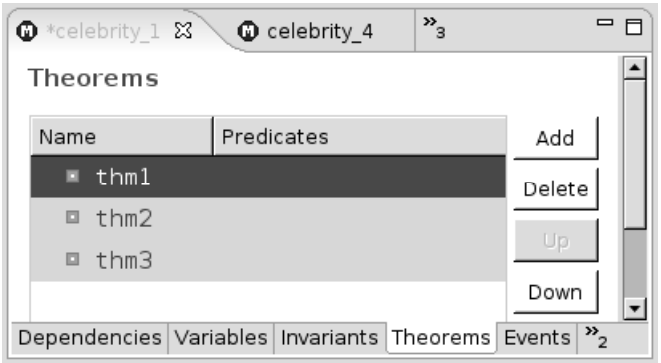
After pressing that button, the following wizard pops up:



You can then enter the theorems you want. If more theorems are needed then press ”More”. When you are finished, press the ”OK” button.

3.4.2 Direct Editing of Theorems.

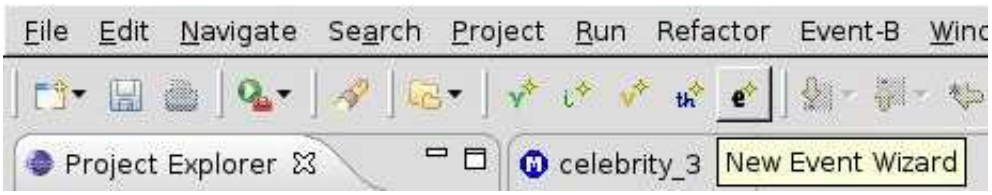
It is also possible to create (button ”Add”) or remove (button ”Delete”) theorems by using the central editing window. For this, you have first to select the ”Theorems” tab of the editor. You can also change the relative place of a theorem: first select it and then press button ”Up” or ”Down”.



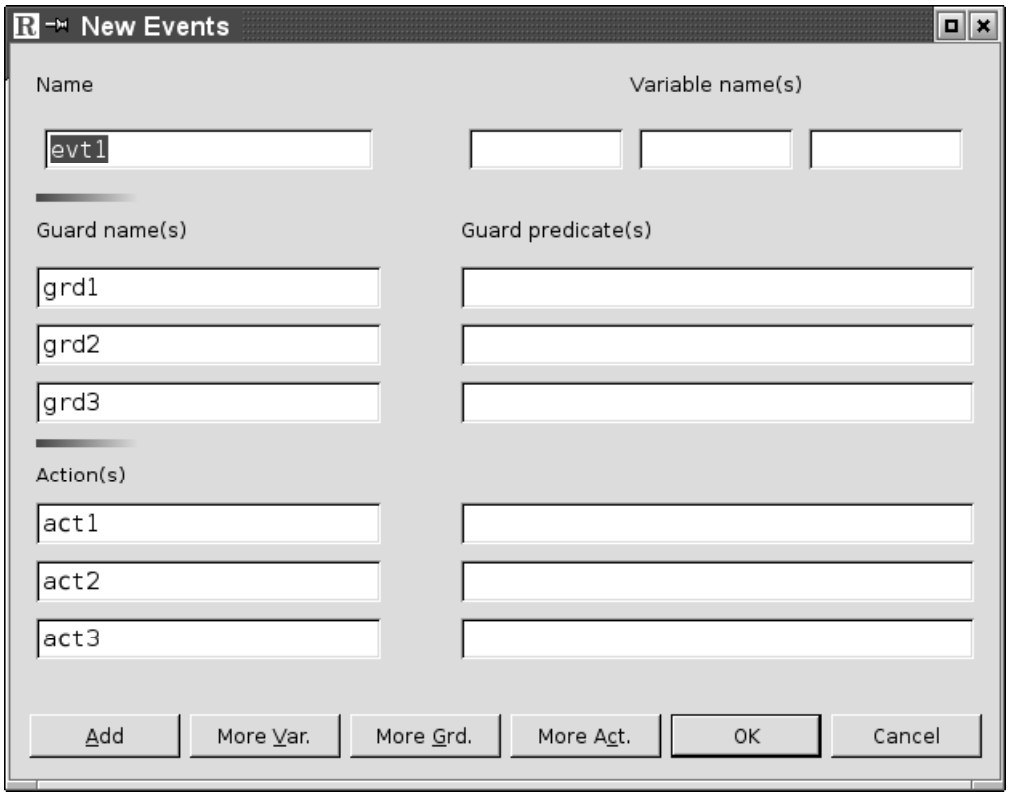
3.5 Events

3.5.1 Events Creation Wizard.

In order to activate the events creation wizard, you have to press the corresponding button in the toolbar as indicated below:



After pressing that button, the following wizard pops up:

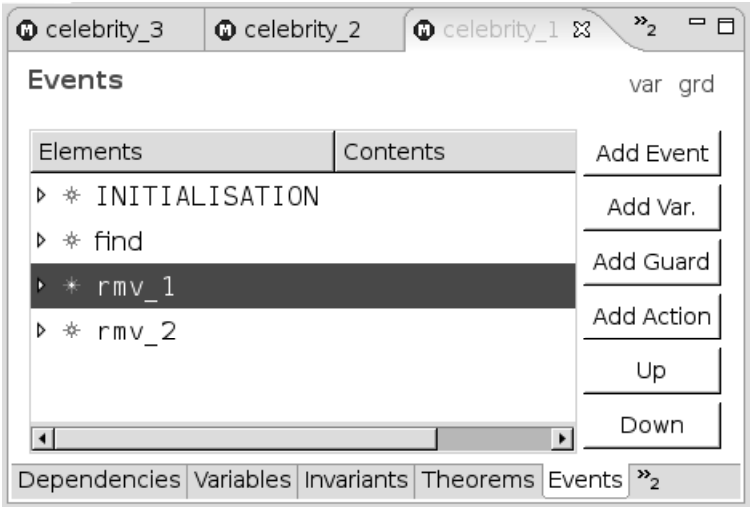


You can then enter the events you want. As indicated, the following elements can be entered: name, parameters, guards, and actions. More parameters, guards and actions can be entered by pressing the corresponding buttons. If more events are needed then press ”Add”. Press ”OK” when you’re finished.

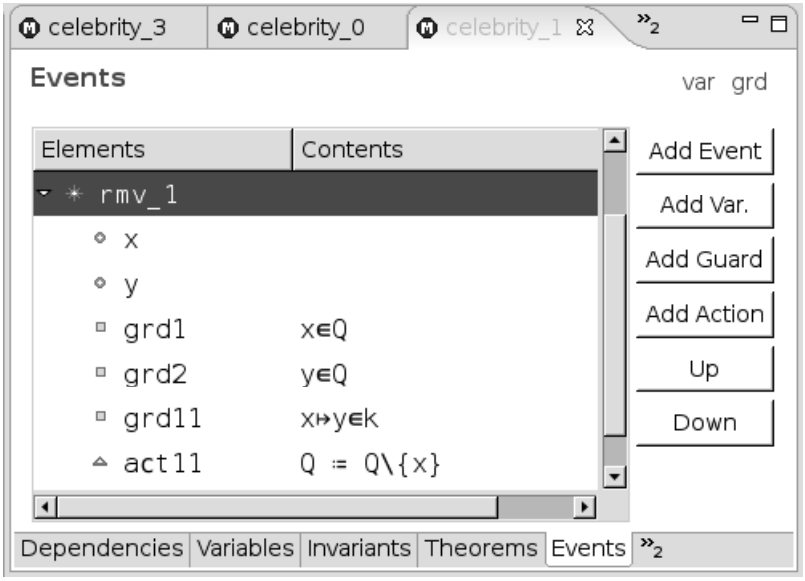
Note that an event with no guard is considered to have a true guard. Moreover, an event with no action is considered to have the ”skip” action.

3.5.2 Direct Editing of Events.

It is also possible to perform a direct creation (button ”Add Event”) of variables by using the central editing window. For this, you have first to select the ”Events” tab of the editor. You can also change the relative place of a variable: first select it and then press button ”Up” or ”Down”.



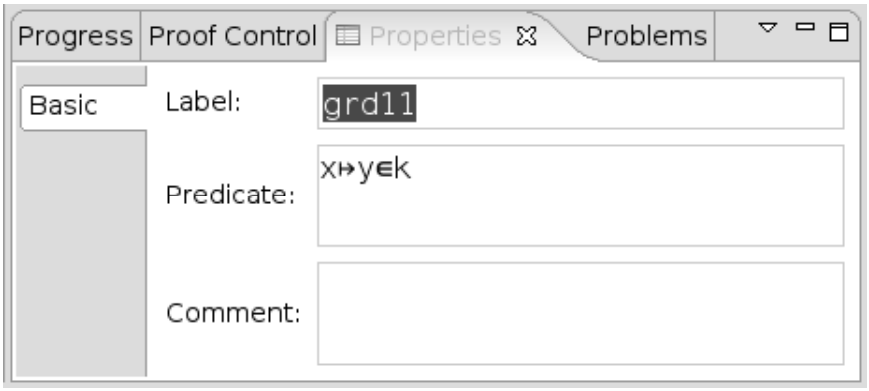
Once an event is selected you can add parameters, guards, and actions. The components of an events can be seen by pressing the little triangle situated on the left of the event name:



As can be seen, event rmv_1 is made of two parameters, x and y, three guards, $x \in Q$, $y \in Q$, and $x \mapsto y \in k$, and one action $Q := Q \setminus \{x\}$. These elements can be modified (select and edit) or removed (select, right click on the mouse, and press ”Delete”). Similar elements can be added by pressing the relevant buttons on the right of the window.

3.6 Adding Comments

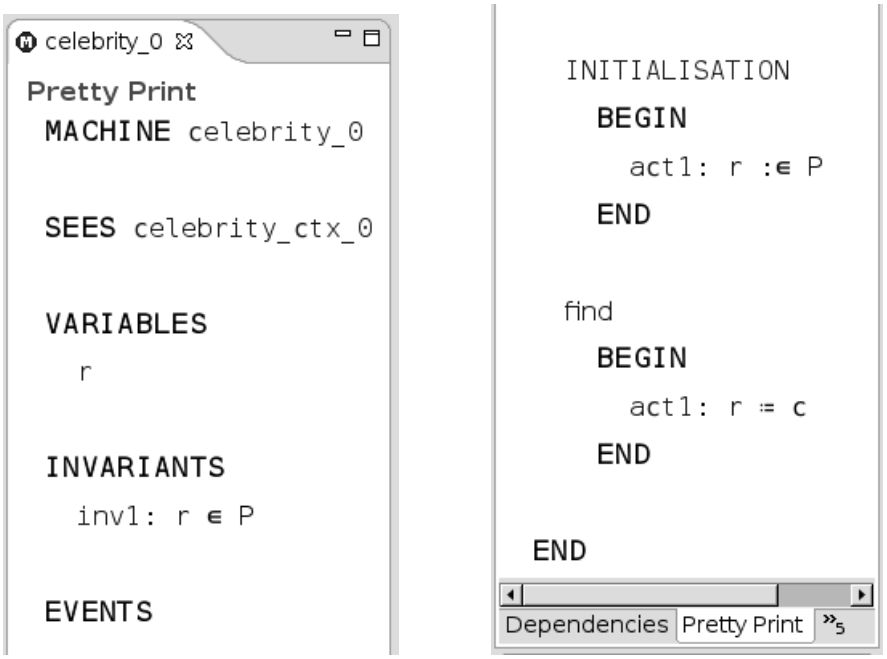
It is possible to add comments to variables, invariants, theorems, events, guards, and actions. For doing so, select the corresponding modeling element and enter the ”Properties” window as indicated below where it is shown how one can add comments on a certain guard:



Multiline comments can be added in the editing area labeled ”Comments”.

3.7 Pretty Print

The pretty print of a machine looks like an input file. It is produced as an output of the editing process:



3.8 Dependencies: Refining a Machine

A machine can be refined by other ones. This can be done directly by selecting the machine to be refined in the ”Project Explorer” window. A right click on the mouse yields the following contextual menu:

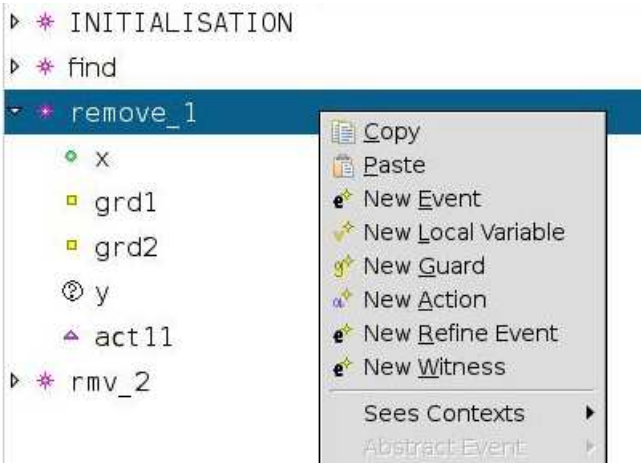
You then press button ”Refine”. A wizard will ask you to enter the name of the refined machine. The abstract machine is entirely copied in the refined machine: this is very convenient as, quite often, the refined machine has lots of elements in common with its abstraction.

3.9 Adding more Dependencies

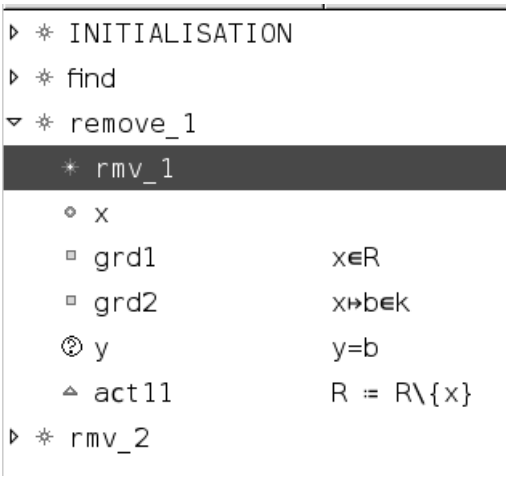
3.9.1 Abstract Event

The absraction of an event is denoted by a ”Refine Event” element. Most of the time the concrete and abstract events bear the same name. But, it is always possible to change the name of a concrete event or the name of its abstraction.

If you want to specify the abstraction of an event, first select the ”Event” tab of the editor and right click on the event name. The following contextual menu will pop up:



You have then to choose the "New Refine Event" option. The abstract event can then be entered by adding the name of the abstract event: here rmv_1.



3.9.2 Splitting an Event

An abstract event can be split into two or more concrete events by just saying that these events refine the former (as explained in previous section).

3.9.3 Merging Events

Two or more abstract events can be merged into a single concrete event by saying that the latter refines all the former. This is done by using several times the approach explained in the previous case. The constraints is that the abstract events to be merged must have exactly the same actions (including the labels of these actions). A proof obligation is generated which states that the guard of the concrete event implies the disjunction of the guards of the abstract events

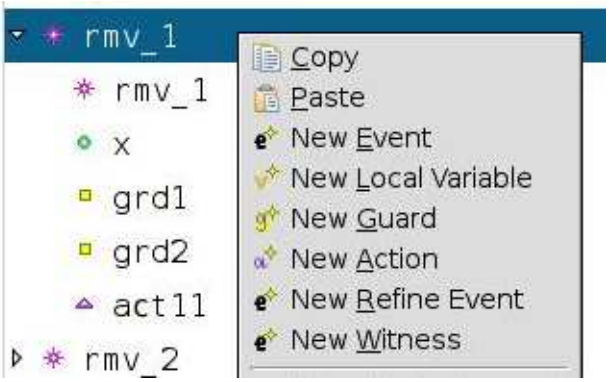
3.9.4 Witnesses

When an abstract event contains some parameters, the refinement proof obligation involves proving an existentially quantified statement. In order to simplify the proof, the user is required to give witnesses for those abstract parameters which are not present in the refinement (those appearing in the refinement are implicitly taken as witnesses for their corresponding abstract counterparts). Here is an example of an abstract event (left) and its refinement (right):

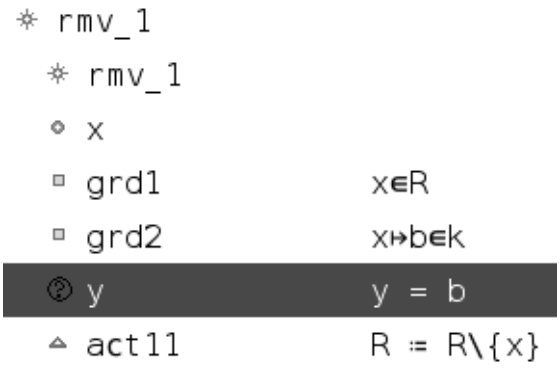
```
rmv_1
  ANY
    x
    y
  WHERE
    grd1: x∈Q
    grd2: y∈Q
    grd11: x↦y∈k
  THEN
    act11: Q ≔ Q\{x}
  END
```

```
rmv_1
  REFINES
    rmv_1
  ANY
    x
  WHERE
    grd1: x∈R
    grd2: x↦b∈k
  THEN
    act11: R ≔ R\{x}
  END
```

The parameter x, being common to both the abstraction and the refinement, does not require a witness, whereas one is needed for abstract parameter y. In order to define the witness, one has first to select the "Events" tab of the editor for the concrete machine where the concrete event (here rmv_1) is selected. After a right click, a menu appears in the window as indicated:



You press button "New Witness" and then you enter the parameter name (here y) and a predicate involving y (here $y = b$) as indicated below



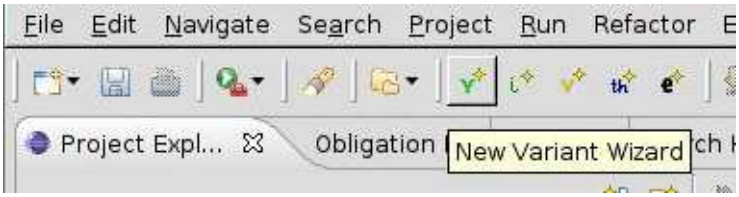
Most of the time, the predicate is an equality as in the previous example, meaning that the parameter is defined in a deterministic way. But it can also be any predicate, in which case the parameter is defined in a non-deterministic way.

3.9.5 Variant

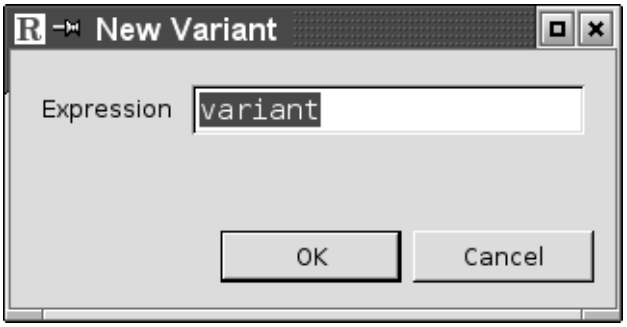
New events can be defined in a concrete machine. Such events have no abstract counterparts. They must refine the implicit ”empty” abstract event which does nothing.

Some of the new events can be selected to decrease a variant so that they do not take control for ever. Such events are said to be CONVERGENT. In order to make a new event CONVERGENT, select it in the ”Events” tab and open the ”Properties” window. You can edit the ”conv.” area. There are three options: ORDINARY (the default), CONVERGENT, or ANTICIPATED. The latter corresponds to a new event which is not yet declared to be CONVERGENT but will be in a subsequent refinement.

In order to define the variant, use the variant wizard as shown below:



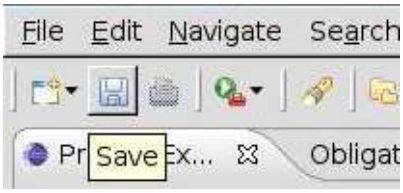
After pressing that button, the following wizard will pop up



You can enter the variant and then press ”OK”. The variant is either a natural number expression or a finite set expression.

4 Saving a Context or a Machine

Once a machine or context is (possibly partly only) edited, you can save it by using the following button:



4.1 Automatic Tool Invocations

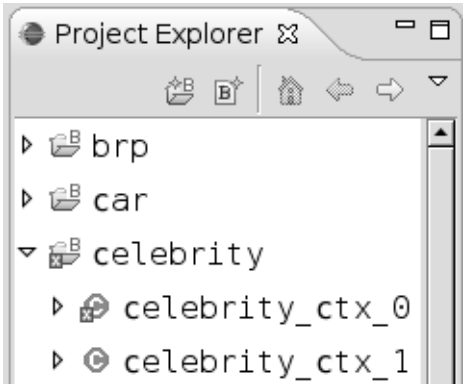
Once a ”Save” is done, three tools are called automatically, these are:

- the Static Checker,
- the Proof Obligation Generator,
- the Auto-Prover.

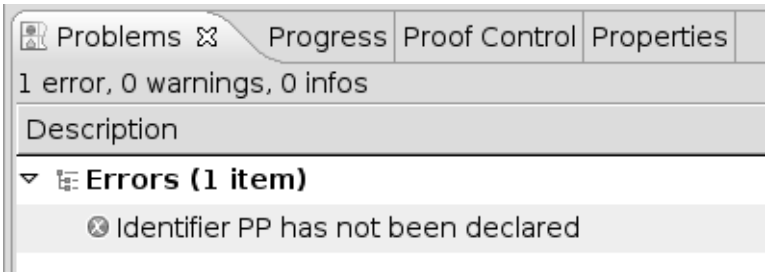
This can take a certain time. A ”Progress” window can be opened at the bottom right of the screen to see which tools are working (most of the time, it is the auto-prover).

4.2 Errors. The Problems Window

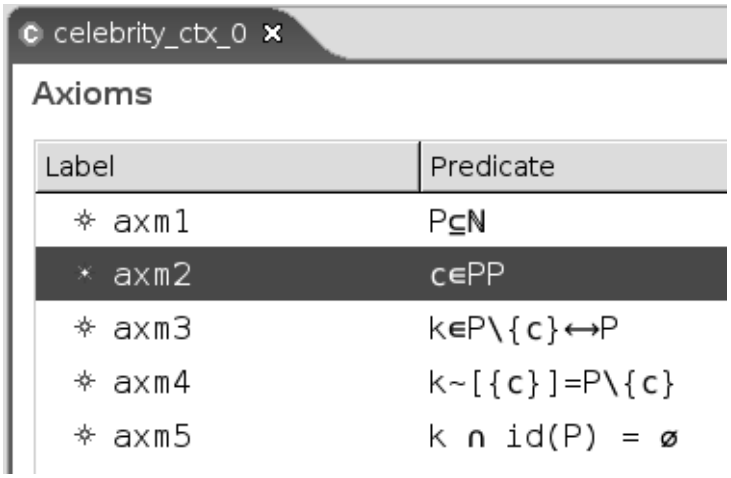
When the Static Checker discovers an error in a project, a little ”x” is added to this project and to the faulty component in the ”Project Explorer” window as shown in the following screen shot:



The error itself is shown by opening the ”Problems” window.

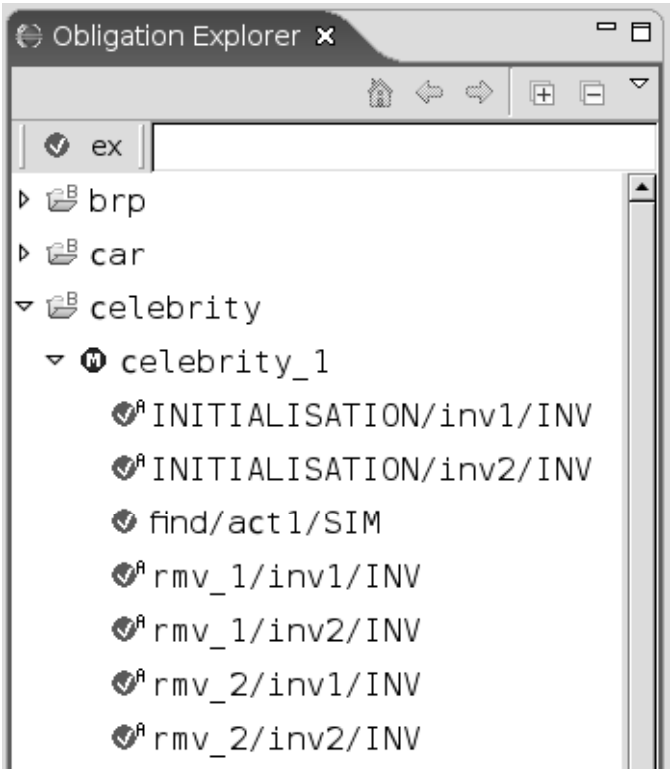


By double-clicking on the error statement, you are transferred automatically into the place where the error has been detected so that you can correct it easily as shown below:



5 The Proof Obligation Explorer

The ”Proof Obligation Explorer” window has the same main entries as the ”Project Explorer” window, namely the projects and its components (context and machines). When expanding a component in the Proof Obligation Explorer you get a list of its proof obligations as generated automatically by the Proof Obligation Generator:



As can be seen in this screen shot, machine "celebrity_1" of project "celebrity" is expanded. We find seven proof obligations. Each of them has got a compound name as indicated in the tables below. A green logo situated on the left of the proof obligation name states that it has been proved (an A means it has been proved automatically). By clicking on the proof obligation name, you are transferred into the Proving Perspective which we are going to describe in subsequent sections.

Next is a table describing the names of context proof obligations:

Well-definedness of an Axiom	axm / WD	axm is the axiom name
Well-definedness of a Theorem	thm / WD	thm is the theorem name
Theorem	thm / THM	thm is the theorem name

Next is a table showing the name of machine proof obligations:

Well-definedness of an Invariant	inv / WD	inv is the invariant name
Well-definedness of a Theorem	thm / WD	thm is the theorem name
Well-definedness of an event Guard	evt / grd / WD	evt is the event name grd is the action name
Well-definedness of an event Action	evt / act / WD	evt is the event name act is the action name
Feasibility of a non-det. event Action	evt / act / FIS	evt is the event name act is the action name
Theorem	thm / THM	thm is the theorem name
Invariant Establishment	INIT. / inv / INV	inv is the invariant name
Invariant Preservation	evt / inv / INV	evt is the event name inv is the invariant name

Next are the proof obligations concerned with machine refinements:

Guard Strengthening	evt / grd / GRD	evt is the concrete event name grd is the abstract guard name
Guard Strengthening (merge)	evt / MRG	evt is the concrete event name
Action Simulation	evt / act / SIM	evt is the concrete event name act is the abstract action name
Equality of a preserved Variable	evt / v / EQL	evt is the concrete event name v is the preserved variable

Next are the proof obligations concerned with the new events variant:

Well definedness of Variant	VWD	
Finiteness for a set Variant	FIN	
Natural number for a numeric Variant	evt / NAT	evt is the new event name
Decreasing of Variant	evt / VAR	evt is the new event name

Finally, here are the proof obligations concerned with witnesses:

Well definedness of Witness	evt / p / WWD	evt is the concrete event name p is parameter name or a primed variable name
Feasibility of non-det. Witness	evt / p / WFIS	evt is the concrete event name p is parameter name or a primed variable name

Remark: At the moment, the deadlock freeness proof obligation generation is missing. If you need it, you can generate it yourself as a theorem saying the the disjunction of the abstract guards imply the disjunction of the concrete guards.

6 The Proving Perspective

The Proving Perspective is made of a number of windows: the proof tree, the goal, the selected hypotheses, the proof control, the proof information, and the searched hypotheses. In subsequent sections, we study these windows, but before that let us see how one can load a proof.

6.1 Loading a Proof

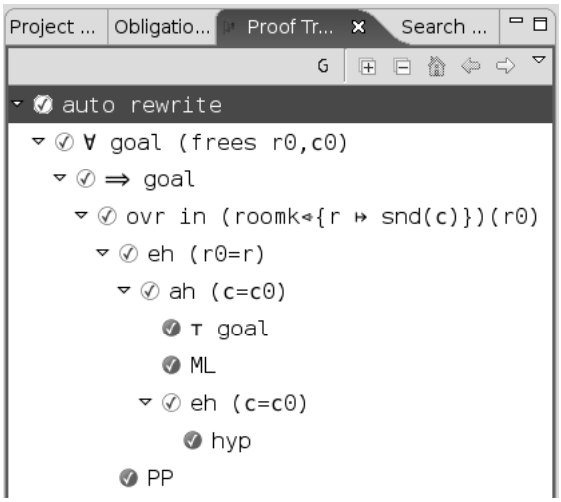
In order to load a proof, enter the Proof Obligation window, select the project, select and expand the component, finally select the proof obligation: the corresponding proof will be loaded. As a consequence:

- the proof tree is loaded in the Proof Tree window. As we shall see in section 6.2, each node of the proof tree is associated with a sequent.
- In case the proof tree has some ”pending” nodes (whose sequents are not discharged yet) then the sequent corresponding to the first pending node is decomposed: its goal is loaded in the Goal window (section 6.3), whereas parts of its hypotheses (the ”selected” ones) are loaded in the Selected Hypotheses window (section 6.3).
- In case the proof tree has no pending node, then the sequent of the root node is loaded as explained previously.

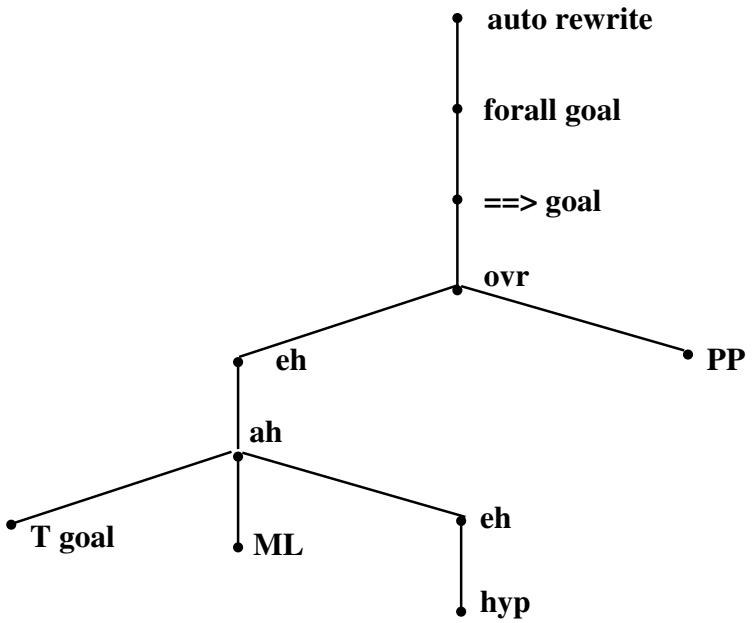
6.2 The Proof Tree

6.2.1 Description of the Proof Tree

The proof tree can be seen in the corresponding window as shown in the following screen shot:



Each line in the proof tree corresponds to a node which is a sequent. A line is right shifted when the corresponding node is a direct descendant of the node of the previous line. Here is an illustration of the previous tree:



Each node is labelled with a comment explaining how it can be discharged. By selecting a node in the proof tree, the corresponding sequent is decomposed and loaded in the Goal and Selected Hypotheses windows as explained in section 6.1.

6.2.2 Decoration

The leaves of the tree are decorated with three kinds of logos:

- a green logo with a "$\surd$" in it means that this leaf is discharged,
- a red logo with a "?" in it means that this leaf is not discharged,
- a blue logo with a "R" in it means that this leaf has been reviewed.

Internal nodes in the proof tree are decorated in the same (but lighter) way.

6.2.3 Navigation within the Proof Tree

On top of the proof tree window one can see three buttons:

- the "G" buttons allows you to see the goal of the sequent corresponding to the node
- the "+" button allows you to fully expand the proof tree
- the "-" allows you to fully collapse the tree: only the root stays visible.

6.2.4 Hiding

The little triangle next to each node in the proof tree allows you to expand or collapse the subtree starting at that node.

6.2.5 Pruning

The proof tree can be pruned from a node: it means that the subtree starting at that node is eliminated. The node in question becomes a leaf and is red decorated. This allows you to resume the proof from that node. After selecting a sequent in the proof tree, pruning can be performed in two ways:

- by right-clicking and then selecting "Prune",
- by pressing the "Scissors" button in the proof control window (section 6.4).

Note that after pruning, the post-tactic is not applied to the new current sequent: if needed you have to press the "post-tactic" button in the Proof Control window (section 6.4). This happens in particular when you want to redo a proof from the beginning: you prune the proof tree from the root node and then you have to press the "post-tactic" button in order to be in exactly the same situation as the one delivered automatically initially.

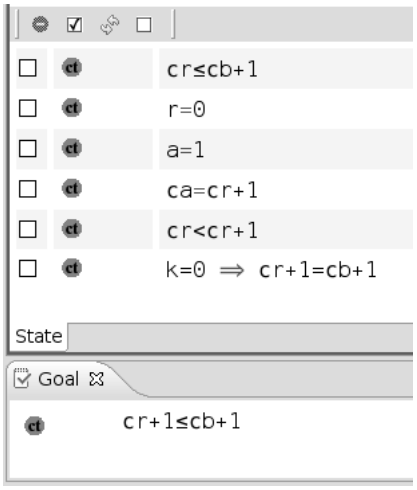
When you want to redo a proof from a certain node, it might be advisable to do it after copying the tree so that in case your new proof fails you can still resume the previous situation by pasting the copied version (see next section).

6.2.6 Copy/Paste

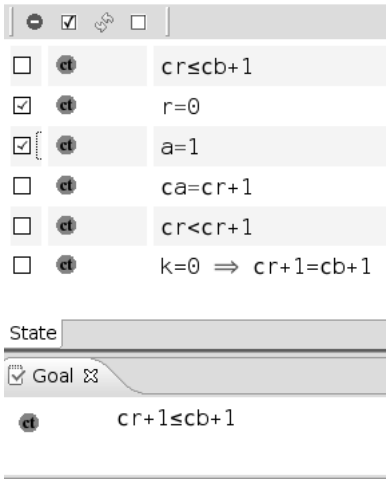
By selecting a node in the proof tree and then clicking on the right key of the mouse, you can copy the part of the proof tree starting at that sequent: it can later be pasted in the same way. This allows you to reuse part of a proof tree in the same (or even another) proof.

6.3 Goal and Selected Hypotheses

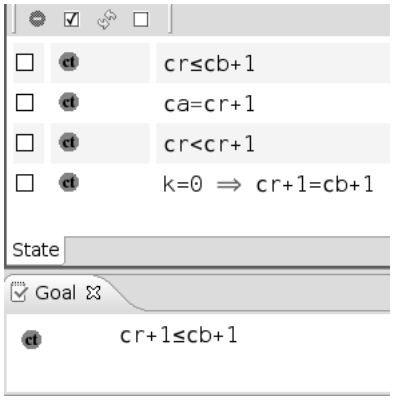
The ”Goal” and ”Selected Hypotheses” windows display the current sequent you have to prove at a given moment in the proof. Here is an example:



A selected hypothesis can be deselected by first clicking in the box situated next to it (you can click on several boxes) and then by pressing the red (-) button at the top of the selected hypothesis window:



Here is the result:



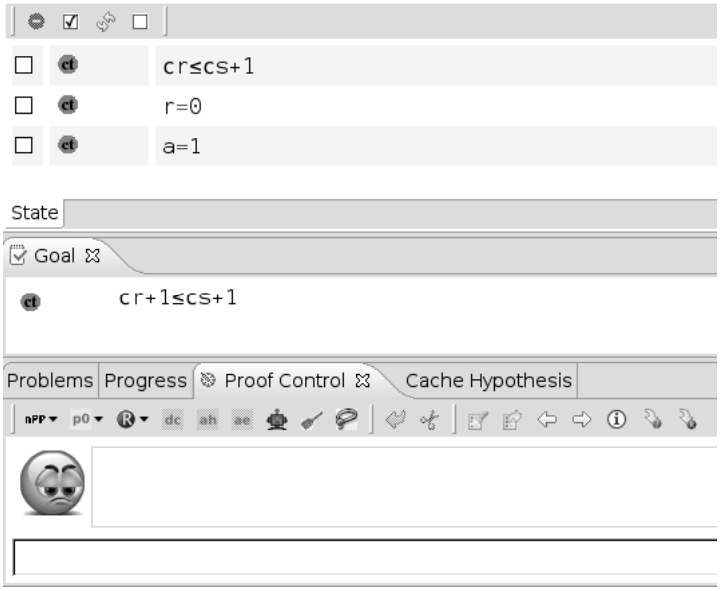
Notice that the deselected hypotheses are not lost: you can get them back by means of the Searched Hypotheses window (section 6.7).

The three other buttons next to the red (-) button allow you to do the reverse operation, namely keeping some hypotheses. The (ct) button next to the goal allows you to do a proof by contradiction: pressing it makes the negation of the goal being a selected hypothesis whereas the goal becomes ”false”. The (ct) button next to a selected hypothesis allows you to do another kind of proof by contradiction: pressing it makes the negation of the hypothesis the goal whereas the negated goal becomes an hypothesis.

6.4 The Proof Control Window

The Proof Control window contains the buttons which you can use to perform an interactive proof. Next is a screen shot where you can see successively from top to bottom:

- some selected hypotheses,
- the goal,
- the ”Proof Control” window,
- a small editing area within which you can enter parameters used by some buttons of the Proof Control window
- the smiley (section 6.5)



The Proof Control window offers a number of buttons which we succinctly describe from left to right:

- (p0): the prover PP attempts to prove the goal (other cases in the list)
- (R) review: the user forces the current sequent to be discharged, it is marked as being reviewed (it's logo is blue-colored)
- (dc) proof by cases: the goal is proved first under the predicate written in the editing area and then under its negation,
- (ah) lemma: the predicate in the editing area is proved and then added as a new selected hypothesis,
- (ae) abstract expression: the expression in the editing area is given a fresh name,
- the auto-prover attempts to discharge the goal. The auto-prover is the one which is applied automatically on all proof obligations (as generated automatically by the proof obligation generator after a "save") without any intervention of the user. With this button, you can call yourself the auto-prover within an interactive proof.
- the post-tactic is executed (see section 6.8),
- lasso: load in the Selected Hypotheses window those unseen hypotheses containing identifiers which are common with identifiers in the goal and selected hypotheses,
- backtrack form the current node (i.e., prune its parent),
- scissors: prune the proof tree from the node selected in the proof tree,
- show (in the Search Hypotheses window) hypotheses containing the character string as in the editing area,
- show the Cache Hypotheses window,
- load the previous non-discharged proof obligation,
- load the next undischarged proof obligation,

- (i) show information corresponding to the current proof obligation in the corresponding window. This information correspond to the elements that took directly part in the proof obligation generation (events, invariant, etc.),
- goto the next pending node of the current proof tree,
- load the next reviewed node of the current proof tree.

6.5 The Smiley

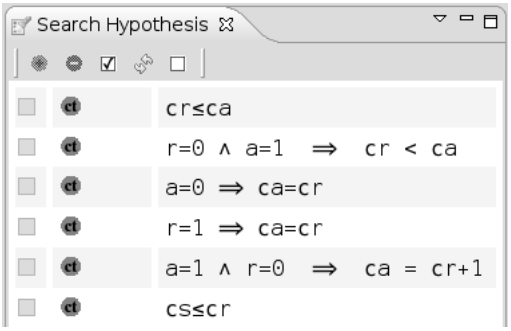
The smiley can take three different colors: (1) red, meaning that the proof tree contains one or more non-discharged sequents, (2) blue, meaning that all non-discharged sequents of the proof tree have been reviewed, (3) green, meaning that all sequents of the proof tree are discharged.

6.6 The Operator "Buttons"

In the goal and in the selected, searched, or cache hypotheses some operators are colored in red. It means that they are "buttons" you can press. When doing so, the meaning (sometimes several) is shown in a menu where you can select various options. The operation performed by these options is described in sections 6.9.1 and 6.9.2.

6.7 The Search Hypotheses Window

A window is provided which contains the hypotheses having a character string in common with the one in the editing area. For example, if we search for hypotheses involving the character string "cr", then after pressing the "search hypothesis" button in the proof control window, we obtain the following:



Such hypotheses can be moved to the "Selected Hypotheses" window (button (+)) or removed from the "Search Hypotheses" window (button (-)). As for the selected hypotheses, other buttons situated next to the previous ones, allow you to move or remove all hypotheses. By pressing the (ct) button the negation of the corresponding hypothesis becomes the new goal.

6.8 The Automatic Post-tactic

In this section, we present the various re-writing or inference rules which are applied automatically as a systematic post-tactic after each proof step. Note that the post-tactic can be disabled by using the "***P***" button situated on the right of the proof control window.

The post-tactic is made of two different rules: rewrite rules, which are applied on any sub-formula of the goal or selected hypotheses (section 6.8.1) and inference rules which are applied on the current sequent (section 6.8.2).

6.8.1 Rewrite rules

The following rewrite rules are *applied automatically* in a systematic fashion from left to right either in the goal or in the selected hypotheses. They all correspond to predicate logical equivalences or expression equalities and result in simplifications. They are sorted according to their main purpose.

Conjunction

$$P \wedge \ldots \wedge \top \wedge \ldots \wedge Q \;==\; P \wedge \ldots \wedge Q$$

$$P \wedge \ldots \wedge \bot \wedge \ldots \wedge Q \;==\; \bot$$

$$P \wedge \ldots \wedge Q \wedge \ldots \wedge \neg Q \wedge \ldots \wedge R \;==\; \bot$$

$$P \wedge \ldots \wedge Q \wedge \ldots \wedge Q \wedge \ldots \wedge R \;==\; P \wedge \ldots \wedge Q \wedge \ldots \wedge R$$

Disjunction

$$P \vee \ldots \vee \top \vee \ldots \vee Q \;==\; \top$$

$$P \vee \ldots \vee \bot \vee \ldots \vee Q \;==\; P \vee \ldots \vee Q$$

$$P \vee \ldots \vee Q \vee \ldots \vee \neg Q \vee \ldots \vee R \;==\; \top$$

$$P \vee \ldots \vee Q \vee \ldots \vee Q \vee \ldots \vee R \;==\; P \vee \ldots \vee Q \vee \ldots \vee R$$

Implication

$$\top \Rightarrow P \;==\; P$$

$$\bot \Rightarrow P \;==\; \top$$

$$P \Rightarrow \top \;==\; \top$$

$$P \Rightarrow \bot \;==\; \neg P$$

$$P \Rightarrow P \;==\; \top$$

Equivalence

$$P \Leftrightarrow \top \;==\; P$$

$$\top \Leftrightarrow P \;==\; P$$

$$P \Leftrightarrow \bot \;==\; \neg P$$

$$\bot \Leftrightarrow P \;==\; \neg P$$

$$P \Leftrightarrow P \;==\; \top$$

Negation

$$\neg \top \;==\; \bot$$

$$\neg \bot \;==\; \top$$

$$\neg \neg P \;==\; P$$

$$E \neq F \;==\; \neg E = F$$

$$E \notin F \;==\; \neg E \in F$$

$$E \not\subset F \;==\; \neg E \subset F$$

$$E \not\subseteq F \;==\; \neg E \subseteq F$$

$$\neg a \leq b \;==\; a > b$$

$$\neg a \geq b \;==\; a < b$$

$$\neg a > b \;==\; a \leq b$$

$$\neg a < b \;==\; a \geq b$$

$$\neg (E = \text{FALSE}) \;==\; (E = \text{TRUE})$$

$$\neg (E = \text{TRUE}) \;==\; (E = \text{FALSE})$$

$$\neg (\text{FALSE} = E) \;==\; (\text{TRUE} = E)$$

$$\neg (\text{TRUE} = E) \;==\; (\text{FALSE} = E)$$

Quantification

$$\forall x \cdot (P \wedge Q) \;==\; (\forall x \cdot P) \;\wedge\; (\forall x \cdot Q)$$

$$\exists x \cdot (P \vee Q) \;==\; (\exists x \cdot P) \;\vee\; (\exists x \cdot Q)$$

$$\forall \ldots, z, \ldots \cdot P(z) \;==\; \forall z \cdot P(z)$$

$$\exists \ldots, z, \ldots \cdot P(z) \;==\; \exists z \cdot P(z)$$

Equality

$$E = E \;==\; \top$$

$$E \neq E \;==\; \bot$$

$$E \mapsto F = G \mapsto H \;==\; E = G \;\wedge\; F = H$$

$$\text{TRUE} = \text{FALSE} \;==\; \bot$$

$$\text{FALSE} = \text{TRUE} \;==\; \bot$$

Set Theory

$$S \cap \ldots \cap \varnothing \cap \ldots \cap T \;==\; \varnothing$$

$$S \cap \ldots \cap T \cap \ldots \cap T \cap \ldots \cap U \;==\; S \cap \ldots \cap T \cap \ldots \cap U$$

$$S \cup \ldots \cup \varnothing \cup \ldots \cup T \;==\; S \cup \ldots \cup T$$

$$S \cup \ldots \cup T \cup \ldots \cup T \cup \ldots \cup U \;==\; S \cup \ldots \cup T \cup \ldots \cup U$$

$$\varnothing \subseteq S \;==\; \top$$

$$S \subseteq S \;==\; \top$$

$$E \in \varnothing \;==\; \bot$$

$$B \in \{A, \ldots, B, \ldots, C\} \;==\; \top$$

$$\{A, \ldots, B, \ldots, B, \ldots, C\} \;==\; \{A, \ldots, B, \ldots, C\}$$

$$E \in \{\, x \mid P(x) \,\} \;==\; P(E)$$

$$S \setminus S \;==\; \varnothing$$

$$S \setminus \varnothing \;==\; S$$

$$\varnothing \setminus S \;==\; \varnothing$$

$$r^{-1-1} \;==\; r$$

$$\mathrm{dom}(\{x \mapsto a, \ldots, y \mapsto b\} \;==\; \{x, \ldots, y\}$$

$$\mathrm{ran}(\{x \mapsto a, \ldots, y \mapsto b\} \;==\; \{a, \ldots, b\}$$

$$(f \leftarrow \ldots \leftarrow \{E \mapsto F\})(E) \;==\; F$$

$E \in \{F\} \;==\; E = F$ where F is a single expression

$$\{E\} = \{F\} \quad == \quad E = F \quad \text{where } E \text{ and } F \text{ are single expressions}$$

$$\{x \mapsto a, \ldots, y \mapsto b\}^{-1} \quad == \quad \{a \mapsto x, \ldots, b \mapsto y\}$$

$$Ty = \varnothing \quad == \quad \bot$$

$$\varnothing = Ty \quad == \quad \bot$$

$$t \in Ty \quad == \quad \top$$

In the three previous rewrite rules, Ty denotes a type expression, that is either a basic type ($\mathbb{Z}$, BOOL, any carrier set), or $\mathbb{P}$(type expression), or type expression$\times$type expression, and t denotes an expression of type Ty.

$$U \setminus U \setminus S \quad == \quad S$$

$$S \cup \ldots \cup U \cup \ldots \cup T \quad == \quad U$$

$$S \cap \ldots \cap U \cap \ldots \cap T \quad == \quad S \cap \ldots \cap T$$

$$S \setminus U \quad == \quad \varnothing$$

In the four previous rules, S and T are supposed to be of type $\mathbb{P}(U)$.

$$r \,;\varnothing \quad == \quad \varnothing$$

$$\varnothing \,; r \quad == \quad \varnothing$$

$$f(f^{-1}(E)) \quad == \quad E$$

$$f^{-1}(f(E)) \quad == \quad E$$

Arithmetic

$$E + \ldots + 0 + \ldots + F \quad == \quad E + \ldots + F$$

$$E - 0 \quad == \quad E$$

$$0 - E \quad == \quad -E$$

$$-(-E) \quad == \quad E$$

$$E * \ldots * 1 * \ldots * F \quad == \quad E * \ldots * F$$

$$E * \ldots * 0 * \ldots * F \quad == \quad 0$$

$$(-E) * \ldots * (-F) \quad == \quad E * \ldots * F \qquad \text{(if an even number of -)}$$

$$(-E) * \ldots * (-F) \quad == \quad -(E * \ldots * F) \qquad \text{(if an odd number of -)}$$

$$E/1 \;==\; E$$

$$0/E \;==\; 0$$

$$(-E)/(-F) \;==\; E/F$$

$$E^1 \;==\; E$$

$$E^0 \;==\; 1$$

$$1^E \;==\; 1$$

$-(i) \;==\; (-i)$ where i is a literal

$-((-i)) \;==\; i$ where i is a literal

$i = j \;==\; \top$ or $\bot$ (computation) where i and j are literals

$i \leq j \;==\; \top$ or $\bot$ (computation) where i and j are literals

$i < j \;==\; \top$ or $\bot$ (computation) where i and j are literals

$i \geq j \;==\; \top$ or $\bot$ (computation) where i and j are literals

$i > j \;==\; \top$ or $\bot$ (computation) where i and j are literals

$$E \leq E \;==\; \top$$

$$E < E \;==\; \bot$$

$$E \geq E \;==\; \top$$

$$E > E \;==\; \bot$$

6.8.2 Automatic inference rules

The following inference rules are *applied automatically* in a systematic fashion at the end of each proof step. They have the following possible effects:

- they discharge the goal,
- they simplify the goal and add a selected hypothesis,
- they simplify the goal by decomposing it into several simpler goals,
- they simplify a selected hypothesis,
- they simplify a selected hypothesis by decomposing it into several simpler selected hypotheses.

Axioms

$$\frac{}{\mathbf{H}, \mathbf{P} \vdash \mathbf{P}} \text{ HYP}$$

$$\frac{}{\mathbf{H}, \mathbf{Q} \vdash \mathbf{P} \vee \ldots \vee \mathbf{Q} \vee \ldots \vee \mathbf{R}} \text{ HYP\_OR}$$

$$\frac{}{\mathbf{H},\ \mathbf{P},\ \neg \mathbf{P} \vdash \mathbf{Q}} \text{ CNTR}$$

$$\frac{}{\mathbf{H}, \bot \vdash \mathbf{P}} \text{ FALSE\_HYP}$$

$$\frac{}{\mathbf{H} \vdash \top} \text{ TRUE\_GOAL}$$

Simplification

$$\frac{\mathbf{H},\ \mathbf{P} \vdash \mathbf{Q}}{\mathbf{H},\ \mathbf{P},\ \mathbf{P} \vdash \mathbf{Q}} \text{ DBL\_HYP}$$

Conjunction

$$\frac{\mathbf{H}, \mathbf{P}, \mathbf{Q} \vdash \mathbf{R}}{\mathbf{H},\ \mathbf{P} \wedge \mathbf{Q} \vdash \mathbf{R}} \text{ AND\_L}$$

$$\frac{\mathbf{H} \vdash \mathbf{P} \qquad \mathbf{H} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \wedge \mathbf{Q}} \text{ AND\_R}$$

Implication

$$\frac{\mathbf{H},\ \mathbf{Q},\ \mathbf{P} \wedge \ldots \wedge \mathbf{R} \Rightarrow \mathbf{S} \vdash \mathbf{T}}{\mathbf{H},\ \mathbf{Q},\ \mathbf{P} \wedge \ldots \wedge \mathbf{Q} \wedge \ldots \wedge \mathbf{R} \Rightarrow \mathbf{S} \vdash \mathbf{T}} \text{ IMP\_L1}$$

$$\frac{\mathbf{H}, \mathbf{P} \vdash \mathbf{Q}}{\mathbf{H} \vdash \mathbf{P} \Rightarrow \mathbf{Q}} \text{ IMP\_R}$$

$$\frac{\mathbf{H}, \mathbf{P} \Rightarrow \mathbf{Q}, \mathbf{P} \Rightarrow \mathbf{R} \vdash \mathbf{S}}{\mathbf{H},\ \mathbf{P} \Rightarrow \mathbf{Q} \wedge \mathbf{R} \vdash \mathbf{S}} \text{ IMP\_AND\_L}$$

$$\frac{\mathbf{H}, \mathbf{P} \Rightarrow \mathbf{R}, \mathbf{Q} \Rightarrow \mathbf{R} \vdash \mathbf{S}}{\mathbf{H},\ \mathbf{P} \vee \mathbf{Q} \Rightarrow \mathbf{R} \vdash \mathbf{S}} \text{ IMP\_OR\_L}$$

Negation

$$\frac{\mathbf{H},\ E \in \{a, \ldots, c\} \ \vdash \ \mathbf{P}}{\mathbf{H},\ E \in \{a, \ldots, b, \ldots, c\}, \neg (E = b) \ \vdash \ \mathbf{P}}$$

$$\frac{\mathbf{H},\ E \in \{a, \ldots, c\} \ \vdash \ \mathbf{P}}{\mathbf{H},\ E \in \{a, \ldots, b, \ldots, c\}, \neg (b = E) \ \vdash \ \mathbf{P}}$$

Quantification

$$\frac{\mathbf{H},\ \mathbf{P(x)} \ \vdash \ \mathbf{Q}}{\mathbf{H},\ \exists \mathbf{x} \cdot \mathbf{P(x)} \ \vdash \ \mathbf{Q}} \quad \text{XST\_L} \quad (\mathbf{x} \text{ not free in } \mathbf{H} \text{ and } \mathbf{Q})$$

$$\frac{\mathbf{H} \ \vdash \ \mathbf{P(x)}}{\mathbf{H} \ \vdash \ \forall \mathbf{x} \cdot \mathbf{P(x)}} \quad \text{ALL\_R} \quad (\mathbf{x} \text{ not free in } \mathbf{H})$$

Equality

$$\frac{\mathbf{H(E)} \ \vdash \ \mathbf{P(E)}}{\mathbf{H(x)},\ x = E \ \vdash \ \mathbf{P(x)}} \quad \text{EQL\_LR}$$

$$\frac{\mathbf{H(E)} \ \vdash \ \mathbf{P(E)}}{\mathbf{H(x)},\ E = x \ \vdash \ \mathbf{P(x)}} \quad \text{EQL\_RL}$$

In these two rules x is a variable which is not free in E

6.9 Interactive Tactics

In this section, the rewriting rules and inference rules, which you can use to perform an interactive proof, are presented. Each of these rules can be invoked by pressing ”buttons” which corresponds to emphasized (red) operators in the goal or the hypotheses. A menu is proposed when there are several options.

6.9.1 Interactive Rewriting Rules

Most of the rewriting rules correspond to distributive laws. For associative operators in connection with such laws as in:

$$P \wedge (Q \vee \ldots \vee R)$$

it has been decided to put the "button" on the first associative/commutative operator (here $\vee$). Pressing that button will generate a menu: the first option of this menu will be to distribute all associative/commutative operators, the second option will be to distribute only the first associative/commutative operator. In the following presentation, to simplify matters, we write associative/commutative operators with two parameters only, but it must always be understood implicitly that we have a sequence of them. For instance, we shall never write $Q \vee \ldots \vee R$ but $Q \vee R$ instead. Rules are sorted according to their purpose.

Conjunction

$$P \wedge (Q \vee R) \quad == \quad (P \wedge Q) \vee (P \wedge R)$$

Disjunction

$$P \vee (Q \wedge R) \quad == \quad (P \vee Q) \wedge (P \vee R)$$

$$P \vee Q \vee \ldots \vee R \quad == \quad \neg P \Rightarrow (Q \vee \ldots \vee R)$$

Implication

$$P \Rightarrow Q \quad == \quad \neg Q \Rightarrow \neg P$$

$$P \Rightarrow (Q \Rightarrow R) \quad == \quad P \wedge Q \Rightarrow R$$

$$P \Rightarrow (Q \wedge R) \quad == \quad (P \Rightarrow Q) \ \wedge \ (P \Rightarrow R)$$

$$P \vee Q \ \Rightarrow \ R \quad == \quad (P \Rightarrow R) \ \wedge \ (Q \Rightarrow R)$$

Equivalence

$$P \Leftrightarrow Q \quad == \quad (P \Rightarrow Q) \ \wedge \ (Q \Rightarrow P)$$

Negation

$$\neg (P \wedge Q) \quad == \quad \neg P \vee \neg Q$$

$$\neg (P \vee Q) \quad == \quad \neg P \wedge \neg Q$$

$$\neg (P \Rightarrow Q) \quad == \quad P \wedge \neg Q$$

$$\neg \forall x \cdot P \quad == \quad \exists x \cdot \neg P$$

$$\neg \exists x \cdot P \quad == \quad \forall x \cdot \neg P$$

$$\neg (S = \varnothing) \quad == \quad \exists x \cdot x \in S$$

Set Theory

Most of the rules are concerned with simplifying set membership predicates.

$$E \mapsto F \in S \times T \quad == \quad E \in S \wedge F \in T$$

$$E \in \mathbb{P}(S) \quad == \quad E \subseteq S$$

$$S \subseteq T \quad == \quad \forall x \cdot (x \in S \Rightarrow x \in T)$$

In the previous rule, x denotes several variables when the type of S and T is a Cartesian product.

$$E \in S \cup T \quad == \quad E \in S \vee E \in T$$

$$E \in S \cap T \quad == \quad E \in S \wedge E \in T$$

$$E \in S \setminus T \quad == \quad E \in S \wedge \neg (E \in T)$$

$$E \in \{A, \ldots, B\} \quad == \quad E = A \vee \ldots \vee E = B$$

$$E \in \mathrm{union}(S) \quad == \quad \exists s \cdot s \in S \wedge E \in s$$

$$E \in (\bigcup x \cdot x \in S \wedge P \mid T) \quad == \quad \exists x \cdot x \in S \wedge P \wedge E \in T$$

$$E \in \mathrm{inter}(S) \quad == \quad \forall s \cdot s \in S \Rightarrow E \in s$$

$$E \in (\bigcap x \cdot x \in S \wedge P \mid T) \quad == \quad \forall x \cdot x \in S \wedge P \Rightarrow E \in T$$

$$E \in \mathrm{dom}(r) \quad == \quad \exists y \cdot E \mapsto y \in r$$

$$F \in \mathrm{ran}(r) \quad == \quad \exists x \cdot x \mapsto F \in r$$

$$E \mapsto F \in r^{-1} \quad == \quad F \mapsto E \in r$$

$$E \mapsto F \in S \lhd r \quad == \quad E \in S \wedge E \mapsto F \in r$$

$$E \mapsto F \in r \rhd T \quad == \quad E \mapsto F \in r \wedge F \in T$$

$$E \mapsto F \in S \lhd\!\!\!- r \quad == \quad E \notin S \wedge E \mapsto F \in r$$

$$E \mapsto F \in r \rhd\!\!\!- T \quad == \quad E \mapsto F \in r \wedge F \notin T$$

$$F \in r[w] \quad == \quad \exists x \cdot x \in w \wedge x \mapsto F \in r$$

$$E \mapsto F \in (p ; q) \quad == \quad \exists x \cdot E \mapsto x \in p \wedge x \mapsto F \in q$$

$$p \lhd\!\!\!+ q \quad == \quad (\mathrm{dom}(q) \lhd p) \cup q$$

$$E \mapsto F \in \mathsf{id}(S) \quad == \quad E \in S \wedge F = E$$

$$E \mapsto (F \mapsto G) \in p \otimes q \quad == \quad E \mapsto F \in p \wedge E \mapsto G \in q$$

$$(E \mapsto G) \mapsto (F \mapsto H) \in p \parallel q \quad == \quad E \mapsto F \in p \wedge G \mapsto H \in q$$

$$r \in S \mathrel{\leftarrow\!\!\!\leftrightarrow} T \quad == \quad r \in S \leftrightarrow T \ \land\ \mathrm{dom}(r) = S$$

$$r \in S \mathrel{\leftrightarrow\!\!\!\twoheadrightarrow} T \quad == \quad r \in S \leftrightarrow T \ \land\ \mathrm{ran}(r) = T$$

$$r \in S \mathrel{\leftarrow\!\!\!\leftrightarrow\!\!\!\twoheadrightarrow} T \quad == \quad r \in S \leftrightarrow T \ \land\ \mathrm{dom}(r) = S \ \land\ \mathrm{ran}(r) = T$$

$$f \in S \pfun T \quad == \quad f \in S \leftrightarrow T \ \land\ \forall x, y, z \cdot x \mapsto y \in f \ \land\ x \mapsto z \in f \ \Rightarrow\ y = z$$

$$f \in S \rightarrow T \quad == \quad f \in S \pfun T \ \land\ \mathrm{dom}(f) = S$$

$$f \in S \pinj T \quad == \quad f \in S \pfun T \ \land\ f^{-1} \in T \pfun S$$

$$f \in S \rightarrowtail T \quad == \quad f \in S \pinj T \ \land\ \mathrm{dom}(f) = S$$

$$f \in S \psurj T \quad == \quad f \in S \pfun T \ \land\ \mathrm{ran}(f) = T$$

$$f \in S \twoheadrightarrow T \quad == \quad f \in S \psurj T \ \land\ \mathrm{dom}(f) = S$$

$$f \in S \bij T \quad == \quad f \in S \rightarrowtail T \ \land\ \mathrm{ran}(f) = T$$

$$(p \cup q)^{-1} \quad == \quad p^{-1} \cup q^{-1}$$

$$(p \cap q)^{-1} \quad == \quad p^{-1} \cap q^{-1}$$

$$(s \lhd r)^{-1} \quad == \quad r^{-1} \rhd s$$

$$(s \ndres r)^{-1} \quad == \quad r^{-1} \nrres s$$

$$(r \rhd s)^{-1} \quad == \quad s \lhd r^{-1}$$

$$(r \nrres s)^{-1} \quad == \quad s \lhd r^{-1}$$

$$(p \,;\, q)^{-1} \quad == \quad q^{-1} \,;\, p^{-1}$$

$$(s \cup t) \lhd r \quad == \quad (s \lhd r) \cup (t \lhd r)$$

$$(s \cap t) \lhd r \quad == \quad (s \lhd r) \cap (t \lhd r)$$

$$(s \cup t) \ndres r \quad == \quad (s \ndres r) \cap (t \ndres r)$$

$$(s \cap t) \ndres r \quad == \quad (s \ndres r) \cup (t \ndres r)$$

$$s \lhd (p \cup q) \quad == \quad (s \lhd p) \cup (s \lhd q)$$

$$s \lhd (p \cap q) \quad == \quad (s \lhd p) \cap (s \lhd q)$$

$$s \ndres (p \cup q) \quad == \quad (s \ndres p) \cup (s \ndres q)$$

$$s \ndres (p \cap q) \quad == \quad (s \ndres p) \cap (s \ndres q)$$

$$r \rhd (s \cup t) \;==\; (r \rhd s) \cup (r \rhd t)$$

$$r \rhd (s \cap t) \;==\; (r \rhd s) \cap (r \rhd t)$$

$$r \rhd\!\!> (s \cup t) \;==\; (r \rhd\!\!> s) \cap (r \rhd\!\!> t)$$

$$r \rhd\!\!> (s \cap t) \;==\; (r \rhd\!\!> s) \cup (r \rhd\!\!> t)$$

$$(p \cup q) \rhd s \;==\; (p \rhd s) \cup (q \rhd s)$$

$$(p \cap q) \rhd s \;==\; (p \rhd s) \cap (q \rhd s)$$

$$(p \cup q) \rhd\!\!> s \;==\; (p \rhd\!\!> s) \cup (q \rhd\!\!> s)$$

$$(p \cap q) \rhd\!\!> s \;==\; (p \rhd\!\!> s) \cap (q \rhd\!\!> s)$$

$$r[S \cup T] \;==\; r[S] \cup r[T]$$

$$(p \cup q)[S] \;==\; p[S] \cup q[S]$$

$$\mathrm{dom}(p \cup q) \;==\; \mathrm{dom}(p) \cup \mathrm{dom}(q)$$

$$\mathrm{ran}(p \cup q) \;==\; \mathrm{ran}(p) \cup \mathrm{ran}(q)$$

$$S \subseteq T \;==\; (U \setminus T) \subseteq (U \setminus S)$$

In the previous rule, the type of S and T is $\mathbb{P}(U)$.

$$S = T \;==\; S \subseteq T \;\wedge\; T \subseteq S$$

In the previous rule, S and T are sets

$$p \,;\, (q \cup r) \;==\; (p \,;\, q) \cup (p \,;\, r)$$

$$(q \cup r) \,;\, p \;==\; (q \,;\, p) \cup (r \,;\, p)$$

$$(p \,;\, q)[s] \;==\; q[p[s]]$$

$$(s \lhd p) \,;\, q \;==\; s \lhd (p \,;\, q)$$

$$(s \lhd\!\!\! - p) \,;\, q \;==\; s \lhd\!\!\! - (p \,;\, q)$$

$$p \,;\, (q \rhd s) \;==\; (p \,;\, q) \rhd s$$

$$p \,;\, (q \rhd\!\!> s) \;==\; (p \,;\, q) \rhd\!\!> s$$

$$U \setminus (S \cap T) \;==\; (U \setminus S) \cup (U \setminus T)$$

$$U \setminus (S \cup T) \;==\; (U \setminus S) \cap (U \setminus T)$$

$$U \setminus (S \setminus T) \;==\; (U \setminus S) \cup T$$

In the three previous rules, S and T are supposed to be of type $\mathbb{P}(U)$.

6.9.2 Interactive Inference rules

Disjunction

$$\frac{\mathbf{H},\ \mathbf{P}\ \vdash\ \mathbf{R} \quad \ldots \quad \mathbf{H},\ \mathbf{Q}\ \vdash\ \mathbf{R}}{\mathbf{H},\ \mathbf{P} \vee \ldots \vee \mathbf{Q}\ \vdash\ \mathbf{R}}\ \text{CASE}$$

Implication

$$\frac{\mathbf{H}\ \vdash\ \mathbf{P} \qquad \mathbf{H},\ \mathbf{P},\ \mathbf{Q}\ \vdash\ \mathbf{R}}{\mathbf{H},\ \mathbf{P} \Rightarrow \mathbf{Q}\ \vdash\ \mathbf{R}}\ \text{MH}$$

$$\frac{\mathbf{H}\ \vdash\ \neg\mathbf{Q} \qquad \mathbf{H},\ \neg\mathbf{Q},\ \neg\mathbf{P}\ \vdash\ \mathbf{R}}{\mathbf{H},\ \mathbf{P} \Rightarrow \mathbf{Q}\ \vdash\ \mathbf{R}}\ \text{HM}$$

Equivalence

$$\frac{\mathbf{H(Q)}, \mathbf{P} \Leftrightarrow \mathbf{Q}\ \vdash\ \mathbf{G(Q)}}{\mathbf{H(P)},\ \mathbf{P} \Leftrightarrow \mathbf{Q}\ \vdash\ \mathbf{G(P)}}\ \text{EQV}\quad \textbf{postponed}$$

Set Theory

$$\frac{\mathbf{H},\ G = E,\ \mathbf{P}(F)\ \vdash\ \mathbf{Q} \qquad \mathbf{H},\ \neg\,(G = E),\ \mathbf{P}(f(G))\ \vdash\ \mathbf{Q}}{\mathbf{H},\ \mathbf{P}((f \lhd \{E \mapsto F\})(G))\ \vdash\ \mathbf{Q}}\ \text{OV\_L}$$

$$\frac{\mathbf{H},\ G = E\ \vdash\ \mathbf{Q}(F) \qquad \mathbf{H},\ \neg\,(G = E)\ \vdash\ \mathbf{Q}(f(G))}{\mathbf{H}\ \vdash\ \mathbf{Q}((f \lhd \{E \mapsto F\})(G))}\ \text{OV\_R}$$

$$\frac{\mathbf{H},\ G \in \mathrm{dom}(g),\ \mathbf{P}(g(G))\ \vdash\ \mathbf{Q} \qquad \mathbf{H},\ \neg\, G \in \mathrm{dom}(g),\ \mathbf{P}(f(G))\ \vdash\ \mathbf{Q}}{\mathbf{H},\ \mathbf{P}((f \lhd g)(G))\ \vdash\ \mathbf{Q}}\ \text{OV\_L}$$

$$\frac{\mathbf{H},\ G \in \mathrm{dom}(g)\ \vdash\ \mathbf{Q}(g(G)) \qquad \mathbf{H},\ \neg\, G \in \mathrm{dom}(g)\ \vdash\ \mathbf{Q}(f(G))}{\mathbf{H}\ \vdash\ \mathbf{Q}((f \lhd g)(G))}\ \text{OV\_R}$$

In the four previous rules the $\lhd$ operator must appear at the "top level"

$$\frac{\mathbf{H}\ \vdash\ f^{-1} \in A \pfun B \qquad \mathbf{H}\ \vdash\ \mathbf{Q}(f[S] \cap f[T])}{\mathbf{H}\ \vdash\ \mathbf{Q}(f[S \cap T])}\ \cap\text{DIS\_R}$$

$$\frac{\mathbf{H}\ \vdash\ f \in A \pfun B \qquad \mathbf{H}\ \vdash\ \mathbf{Q}(f^{-1}[S] \setminus f^{-1}[T])}{\mathbf{H}\ \vdash\ \mathbf{Q}(f^{-1}[S \setminus T])}\ \setminus\text{DIS\_R}$$

In the two previous rules, the occurrence of f^{-1} must appear at the "top level". Moreover A and B denote some type. Similar left distribution rules exist

$$\frac{\mathbf{H}\ \vdash\ \mathrm{WD}(\mathbf{Q}(\{f(E)\})) \qquad \mathbf{H}\ \vdash\ \mathbf{Q}(\{f(E)\})}{\mathbf{H}\ \vdash\ \mathbf{Q}(f[\{E\}])}\ \text{SIM\_R}$$

In the previous rule, the occurrence of f must appear at the "top level". A similar left simplification rule exists.

$$\frac{\mathbf{H}\ \vdash\ \mathrm{WD}(\mathbf{Q}(g(f(x)))) \qquad \mathbf{H}\ \vdash\ \mathbf{Q}(g(f(x)))}{\mathbf{H}\ \vdash\ \mathbf{Q}((f\,;g)(x))}\ \text{SIM\_R}$$

In the previous rule, the occurrence of $f\,;g$ must appear at the "top level". A similar left simplification rule exists.

7 Syntax of the Mathematical Language

In this section,we present the syntax of the mathematical language. It comprises two main syntactic constructs, namely $< predicate >$ (section 7.1) and $< expression >$ (section 7.2).

7.1 Syntax for Predicates

$$
\begin{array}{lcl}
< predicate > & ::= & \{< quantifier >\} < unquantified\_predicate > \\
\\
< quantifier > & ::= & `\forall' < ident\_list > `\cdot' \\
 & & `\exists' < ident\_list > `\cdot' \\
\\
< ident\_list > & ::= & < ident > \{, < ident >\} \\
\\
< unquantified\_predicate > & ::= & < simple\_predicate > [`\Rightarrow' < simple\_predicate >] \\
 & & < simple\_predicate > [`\Leftrightarrow' < simple\_predicate >] \\
\\
< simple\_predicate > & ::= & < literal\_predicate > \{`\wedge' < literal\_predicate >\} \\
 & & < literal\_predicate > \{`\vee' < literal\_predicate >\} \\
\\
< literal\_predicate > & ::= & \{`\neg'\} < atomic\_predicate > \\
\\
< atomic\_predicate > & ::= & `\bot' \\
 & & `\top' \\
 & & `\text{finite}'(< expression >) \\
 & & < pair\_expr >< relop >< pair\_expr > \\
 & & (< predicate >) \\
\\
< relop > & ::= & `=' \mid `\neq' \mid `\in' \mid `\notin' \mid `\subset' \mid `\not\subset' \mid `\subseteq' \mid `\not\subseteq' \\
 & & `<' \mid `\leq' \mid `>' \mid `\geq'
\end{array}
$$

7.2 Syntax for Expressions

$$
\begin{array}{lcl}
< expression > & ::= & `\lambda' < ident\_pattern > `\cdot' < predicate > ` \mid' < expression > \\
 & & ` \bigcup' < ident\_list > `\cdot' < predicate > ` \mid' < expression > \\
 & & ` \bigcup' < expression > ` \mid' < predicate > \\
 & & ` \bigcap' < ident\_list > `\cdot' < predicate > ` \mid' < expression > \\
 & & ` \bigcap' < expression > ` \mid' < predicate > \\
 & & < pair\_expr > \\
 & & \\
< ident\_pattern > & ::= & < ident\_pattern > \{` \mapsto' < ident\_pattern >\} \\
 & & `(' < ident\_pattern > `)' \\
 & & < ident > \\
 & & \\
< pair\_expr > & ::= & < rel\_set\_expr > \{` \mapsto' < rel\_set\_expr >\} \\
 & & \\
< rel\_set\_expr > & ::= & < set\_expr > \{< rel\_set\_op >< set\_expr >\} \\
 & & \\
< rel\_set\_op > & ::= & ` \leftrightarrow' \mid ` \nleftrightarrow' \mid ` \leftrightarrow' \mid ` \nleftrightarrow' \mid ` \pfun' \mid ` \fun' \\
 & & ` \rightarrowtail' \mid ` \fun' \mid ` \pfun' \mid ` \fun' \mid ` \rightarrowtail' \\
 & & \\
< set\_expr > & ::= & < interval\_expr > \{`\cup' < interval\_expr >\} \\
 & & < interval\_expr > \{`\times' < interval\_expr >\} \\
 & & < interval\_expr > \{` \ltimes' < interval\_expr >\} \\
 & & < interval\_expr > \{`\circ' < interval\_expr >\} \\
 & & < interval\_expr > `\|' < interval\_expr > \\
 & & [< domain\_mod >] < rel\_expr > \\
 & & \\
< domain\_mod > & ::= & < interval\_expr > (` \lhd' \mid ` \ndres') \\
 & & \\
< rel\_expr > & ::= & < interval\_expr > `\otimes' < interval\_expr > \\
 & & < interval\_expr > \{`;' < interval\_expr >\}[< range\_mod >] \\
 & & < interval\_expr > \{`\cap' < interval\_expr >\}[`\backslash' < interval\_expr > \mid < range\_mod >] \\
 & & \\
< range\_mod > & ::= & (` \rhd' \mid ` \nrres') < interval\_expr > \\
 & & \\
< interval\_expr > & ::= & < arith\_expr > [`..' < arith\_expr >] \\
 & & \\
< arith\_expr > & ::= & [`-'] < term > \{(` +' \mid `-') < term >\} \\
 & & \\
< term > & ::= & < factor > \{(` *' \mid ` \div' \mid ` \bmod ') < factor >\} \\
 & & \\
< factor > & ::= & < image > [`^{-1}' < image >] \\
 & & \\
< image > & ::= & < primary > \{`[' < expression > `]' \mid `(' < expression > `)'\}
\end{array}
$$

$< primary >$ ::= $< simple\_expr > \{`^{-1}'\}$

$< simple\_expr >$ ::= ‘bool’ ‘(’ $< predicate >$ ‘)’
$< unary\_op >$ ‘(’ $< expression >$ ‘)’
‘(’ $< expression >$ ‘)’
‘{’ $< ident\_list >$ ‘·’ $< predicate >$ ‘ |’ $< expression >$ ‘}’
‘{’ $< expression >$ ‘ |’ $< predicate >$ ‘}’
‘{’ $[< expression > \{`,' < expression >\}]$ ‘}’
‘$\mathbb{Z}$’ | ‘$\mathbb{N}$’ | ‘$\mathbb{N}_1$’ | ‘BOOL’ | ‘TRUE’ | ‘FALSE’ | ‘$\varnothing$’
$< ident >$
$< integer\_literal >$

$< unary\_op >$ ::= ‘card’ | ‘$\mathbb{P}$’ | ‘$\mathbb{P}_1$’ | ‘union’ | ‘inter’ | ‘dom’ | ‘ran’
‘prj_1’ | ‘prj_2’ | ‘id’ | ‘min’ | ‘max’

8 Appendix: ASCII Representations of the Mathematical Symbols

8.1 Atomic Symbols

ASCII	Symbol
true	$\top$
false	$\bot$
INT	$\mathbb{Z}$

ASCII	Symbol
NAT	$\mathbb{N}$
NAT1	$\mathbb{N}_1$
BOOL	BOOL

ASCII	Symbol
TRUE	TRUE
FALSE	FALSE
{}	$\varnothing$

8.2 Unary Operators

ASCII	Symbol
not	$\neg$
finite	finite
card	card
POW	$\mathbb{P}$
POW1	$\mathbb{P}_1$

ASCII	Symbol
union	union
inter	inter
dom	dom
ran	ran
prj1	prj_1

ASCII	Symbol
prj2	prj_2
id	id
min	min
max	max
-	$-$

8.3 Assignment Operators

ASCII	Symbol
:=	$:=$

ASCII	Symbol
:|	$:\mid$

ASCII	Symbol
::	$:\in$

8.4 Binary Operators

ASCII	Symbol
&	$\wedge$
or	$\vee$
=>	$\Rightarrow$
<=>	$\Leftrightarrow$
=	$=$
/=	$\neq$
:	$\in$
/:	$\notin$
<<:	$\subset$
/<<:	$\not\subset$
<:	$\subseteq$
/<:	$\not\subseteq$
<	$<$
<=	$\leq$
>	$>$
>=	$\geq$

ASCII	Symbol
|->	$\mapsto$
<->	$\leftrightarrow$
<<->	$\nleftrightarrow$
<->>	$\leftrightarrow\!\!\!\!\rightarrow$
<<->>	$\nleftrightarrow\!\!\!\!\rightarrow$
+->	$\rightarrow\!\!\!\!\!+$
-->	$\rightarrow$
>+>	$\rightarrow\!\!\!\!\!+$
>->	$\rightarrowtail$
+->>	$\rightarrow\!\!\!\!\!+\!\!\!\!\rightarrow$
-->>	$\twoheadrightarrow$
>->>	$\rightarrowtail\!\!\!\!\rightarrow$
/\	$\cap$
\/	$\cup$
\	$\setminus$
**	$\times$

ASCII	Symbol
<+	$\lhd\!\!\!+$
||	$\parallel$
><	$\otimes$
;	;
<|	$\lhd$
<<|	$\trianglelefteq$
|>	$\rhd$
|>>	$\trianglerighteq$
+	$+$
-	$-$
*	$*$
/	$\div$
mod	mod
..	..
^	$\hat{}$
~	$^{-1}$

8.5 Quantifiers

ASCII	Symbol
!	$\forall$
#	$\exists$
%	λ

ASCII	Symbol
UNION	$\bigcup$
INTER	$\bigcap$

ASCII	Symbol
|	$\mid$
.	$\cdot$

8.6 Bracketing

ASCII	Symbol
(	(
)	)

ASCII	Symbol
[	[
]	]

ASCII	Symbol
{	{
}	}