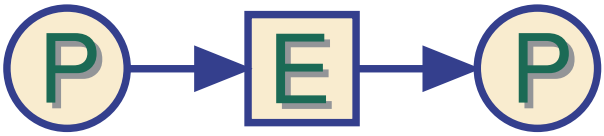
Programming Environment based on Petri nets

Documentation and User Guide Version 1.8



Professor Dr. Eike Best

Dipl. Inform. Bernd Grahlmann

University of Oldenburg

— (Draft) December 1998 —

What is PEP?

The PEP tool (Programming Environment based on Petri Nets) is a comprehensive set of modelling, compilation, simulation and verification components, linked together within an X-based graphical user interface.

- PEP's modelling components facilitate the design of parallel systems by parallel programs (B(PN)^2 and SDL), interacting finite automata, process algebra, or high-level / low-level Petri nets.
- PEP's compilers generate Petri nets from such models.
- PEP's simulators allow automatic or user-driven simulation of high-level / low-level nets and may trigger simulation of the corresponding programs.
- PEP's verification component contains various Petri net indigenous algorithms to check, e.g., reachability properties and deadlock-freeness, as well as model checking algorithms. Presently, PEP contains a prefix-builder on top of which Esparza's model checking algorithm allows simple branching-time logic properties to be checked very efficiently on nets, and hence on all input models of PEP. A model-checker for LTL (linear-time temporal logic) based on the same principles is under construction.
- PEP also has a good interface to Starke's INA package, in which a wide range of structural analysis algorithms such as the computation of place invariants, transition invariants, siphons and traps, as well as stubborn set and symmetrically reduced state space analysis, are implemented. The new version provides interfaces to the verification packages
  - FC2Tools offering verification based on networks of automata,
  - SMV offering BDD based CTL model checking, and
  - SPIN offering LTL model checking with optional partial order reductions.
  - Moreover, an interface to PROD is under construction.
- The PEP tool can be considered as an open platform. Further algorithms can be integrated in the user interface easily. Please, ask for input/output routines for our net formats.

How to read the documentation

This document is intended to be divided into five parts: Installation, General guidelines, Getting Started, Documentation, and Background. The first four parts are not yet complete but the present text already reflects the current state of the tool. The fifths part, the background, is currently not included because we are working on a new version which is appropriate for the new version of PEP. Please, have a look at our Web page: http://theoretica.informatik.uni-oldenburg.de/~pep for new versions of the documentation.

The 'Installation Notes' in section 1 should be followed if you wish to install PEP from scratch on a UNIX platform. They can be skipped if there is already a working system on your machine.

After installation, reading the 'Getting Started' section on page 5 should enable you to get the system running quickly and to play with it for a while. PEP is menu-, windows- and mouse-driven and conforms to the standards of an X based application. The general guidelines in section 2 give some further hints.

Sections 4 – 11 comprise the full documentation. You can either read these sections without needing a machine by your side, or you can use them as reference points in case of need.

Section 4 explains the start-up window of the PEP tool. The functionality of the project window is described in section 5. The use of the PEP file browser is explained in section 6. In sections 7–11, editors for objects of different kinds handled by the tool are documented. The B(PN)^2 programming language editor is explained in section 7. The Petri net editors for high-level nets (M-nets) and for low-level nets (Petri boxes) are described, respectively, in sections 8 and 9. The editor for temporal logic formulae is explained in section 10. The SDL editor is explained in section 11.

At the end of this document an index is given.

Acknowledgements

The development of the PEP system has been started in 1993 - 1994 as part of a common project of the Universities of Hildesheim and Oldenburg sponsored by the DFG (Deutsche Forschungsgemeinschaft).

The project was initiated by Eike Best and Hans Fleischhack, based on theoretical work done by Eike Best and others in the EU Basic Research projects DEMON (Design Methods based On Nets) and CALIBAN (CAusal CalcuLI BAsed on Nets). In particular, Javier Esparza developed the theory behind the unfolding-based model checking algorithm used in PEP.

Bernd Grahlmann led the Hildesheim group (around Eike Best) of the PEP project. He designed and supervised (often also writing and correcting code) the implementation of PEP, and held overall responsibility for the product.

During the first two years, the implementation of the tool was carried out in cooperation with the Oldenburg group (around Hans Fleischhack), mainly in interaction with Matthias Damm, Marc Langnickel, Andree Seidel and the STONE group of students.

Since 1995 the PEP project has been continued in cooperation with Peter H. Starke of the Humboldt-Universität zu Berlin. Lutz Pogrell was responsible for the integration of INA (Integrated Net Analyzer) into PEP. At present, Stefan Haar is continuing this work.

Recently, Eike Best and his group moved to Oldenburg, and as a consequence, the Oldenburg contributions to PEP are currently increasing.

Moreover, we are lucky to receive strong support from the group around Javier Esparza (Technische Universität München), even though they are not an official project partner.

The following persons made contributions to the implementation (in alphabetical order):

Martin Ackermann	Implemented the compiler from a B(PN)^2 program to an M-net.
Ulrich Anhalt	Implemented a graphical editor and an input filter for parallel finite automata (together with Matthias Moeller).
Burkhard Bieber	Implemented the main part of the graphical user interface spending a vast amount of time to make it user-friendly.
Ulf Fildebrandt	Has improved the implementation of the Petri net editors and simulators. Moreover, he integrated new 'programming with nets' features.
Burkhard Graves	Implemented some of the compilers and the model checking algorithm for safe Petri nets, and improved the theory.
Tobias Himstedt	Initiated the implementation of two Petri net editors and simulators.
Johannes Jäger	Improved and extended the C code generator and implemented the McMillan deadlock-freeness checker.
Lars Jenner	Developed and implemented the low-level (via PBC terms) procedure semantics.
Michael Kater	Implemented various text editors, improved the reference component, and contributed to the Linux port. Since 1996 he continued the work of Burkhard Bieber on the graphical user interface. Moreover, he developed and implemented the 3D visualisation component.
Stephan Melzer	Developed and implemented the C code generator and built the linear programming part together with Javier Esparza.

Stephan Merkel	Improved the functionality of some of the compilers.
Matthias Moeller	Implemented a graphical editor and an input filter for parallel finite automata (together with Ulrich Anhalt).
Carola Pohl	Integrated the SPIN verification package and implemented the on-line help system.
Stephan Ptak	Continues the work of Burkhard Bieber and Michael Kater on the graphical user interface.
Stefan Römer	Designed and implemented efficient data structures and operations for Petri net and multiset representation. Was the main designer of the basic net file format. Implemented input/output routines, converter programs, and a wide variety of algorithms (e.g. the prefix builder, the boundedness-checker, and original versions of the formula translator, and the HL-net to LL-net unfolder) and optimised, e.g, the McMillan deadlock checker.
Stefan Schwoon	Extended the general input/output routines for Petri nets, improved the implementation of various compilers and algorithms, and implemented the compiler from SDL to M-nets and from PFA to M-nets.
Andre Stille	Improved the input filter for parallel finite automata.
Thomas Thielke	Implemented a dedicated model checking algorithm for T-systems.
Guido Wimmel	Integrated the SMV verification package.

Further contributions were made by Farzad Eghtessadi, Eberhard Meisel, Steffen Möller, Robert-Christoph Riemann, Hartmut Trüe and Fatma Uzel.

Moreover, the following external program packages have been integrated in the PEP tool:

- The FC2Tools, which were developed at INRIA (Sophia Antipolis) by Robert de Simone, Valérie Roy, Annie Ressouche, Michel Bourdelles, Amar Bouali and others. Please, have a look at their Web page:
  http://www.inria.fr/meije/verification/index.html.
- SMV (Symbolic Model Verifier), which was developed at the Carnegie Mellon University by Edmund Clarke, Ken McMillan and others. Please, have a look at their Web page:
  http://almond.srv.cs.cmu.edu/afs/cs.cmu.edu/project/modck/pub/www/smv.html.
- SPIN, which was developed at Bell Labs (mainly by Gerard Holzmann) in the formal methods and verification group. Please, have a look at their Web page:
  http://netlib.bell-labs.com/netlib/spin/whatispin.html.

The linear programming part uses the lp_solve packet by M.R.C.M. Berkelaar (Eindhoven University of Technology).

Contents

1 Installation of the PEP tool **1**

2 General guidelines **3**

3 Getting started **5**

4 The PEP start-up window **9**
4.1 Description of the start-up window 9
4.2 Menus of the start-up window 10

5 The project window **13**
5.1 Description of the project window 14
5.2 Menus of the project window 15
5.3 Explanation of the options 29
5.3.1 Syntax check 29
5.3.2 B(PN)^2⇒PBC 29
5.3.3 PBC⇒LL Net 30
5.3.4 B(PN)^2⇒M-Net 31
5.3.5 M-Net⇒Petri box 33
5.3.6 Sync HL Net 33
5.3.7 Prefix Builder 35
5.3.8 Boundedness Analyser 36
5.3.9 Reachability analyser 37
5.3.10 LP analyser 37
5.3.11 Model Checker (Safe) 39
5.3.12 Model Checker (SMV) 39
5.3.13 Model Checker (SPIN) 40
5.3.14 B(PN)^2 expander 42
5.3.15 Petri box⇒INA 43
5.3.16 Petri box⇒PROMELA 43
5.3.17 PFA⇒M-net 43
5.3.18 PFA⇒B(PN)^2 44
5.3.19 SDL⇒M-net 45

6 The file browser **47**

7 The B(PN)² editor **49**
7.1 Description of the B(PN)² editor window 49
7.2 Menus of the B(PN)² editor 50
7.2.1 Search 51
7.2.2 Replace 52
7.2.3 Regular expressions 52
7.2.4 Reference Mode 55

8 The high-level Petri net (M-net) editor **57**
8.1 Description of the high-level Petri net editor window 57
8.2 Menus of the high-level net editor 65

9 The low-level Petri net (Petri box) editor **85**
9.1 Description of the low-level Petri net editor window 85
9.2 Menus of the low-level net editor 92

10 The formula editor **105**
10.1 Description of the formula editor window 105
10.2 Menus of the formula editor 107

11 The SDL editor **109**
11.1 Description of the SDL editor window 109
11.2 Menus of the SDL editor 110
11.2.1 Reference Mode 112

List of Figures

2.1 PEP error box . . . 3
2.2 Tcl error box . . . 4
2.3 See stack trace window . . . 4

3.1 The start-up window of the PEP system . . . 5
3.2 Project window belonging to the 'semaphore' project . . . 6

4.1 The start-up window of the PEP system . . . 9
4.2 The new project window . . . 10
4.3 Copy project window . . . 11
4.4 Rename project window . . . 11
4.5 Delete project window . . . 11
4.6 List of open projects . . . 12

5.1 Project window belonging to the 'semaphore' project . . . 13
5.2 Edit project information window. . . . 16
5.3 Boundedness analyser . . . 19
5.4 Reachability analyser . . . 20
5.5 LP reachability analyser . . . 20
5.6 Analyser to check the deadlock-freeness property . . . 21
5.7 Linear programming based checker of the deadlock-freeness property . . . 22
5.8 A deadlock-freeness checker using FC2 . . . 22
5.9 The dialog box of the prefix based model checker . . . 23
5.10 The dialog box of the integrated SPIN model checker . . . 24
5.11 The dialog box of the integrated SMV model checker . . . 25
5.12 Online help system of the PEP tool . . . 27
5.13 Options for the syntax check . . . 29
5.14 Options for the compiler B(PN)^2 $\Rightarrow$ PBC . . . 29
5.15 Options for the compiler PBC $\Rightarrow$ LL-net . . . 30
5.16 Options for the compiler B(PN)^2 $\Rightarrow$ M-net . . . 31
5.17 Options for the compiler M-net $\Rightarrow$ Petri box . . . 33
5.18 Options for the high-level net synchroniser (and 'restricter') . . . 34
5.19 Options for the prefix builder . . . 35
5.20 Options for the boundedness analyser . . . 36
5.21 Options for the reachability analyser . . . 37
5.22 Options for the LP analyser . . . 38

5.23 Options for the model checking algorithm for safe Petri nets 39
5.24 Options for the SMV model checker 40
5.25 Options for the SPIN model checker 41
5.26 Options for the B(PN)^2 expander 43
5.27 Options for the compiler Petri box $\Rightarrow$ INA 43
5.28 Options for the compiler Petri box $\Rightarrow$ PROMELA 43
5.29 Options for the compiler PFA $\Rightarrow$ M-net 44
5.30 Options for the compiler PFA $\Rightarrow$ B(PN)2 44
5.31 Options for the compiler SDL $\Rightarrow$ M-net 45

6.1 File browser 47

7.1 B(PN)^2 editor 49
7.2 Unsaved changes window of the B(PN)^2 editor 51
7.3 Search window within the B(PN)^2 editor 52
7.4 Replace window within the B(PN)^2 editor 52
7.5 Syntax check window within the B(PN)^2 editor 53
7.6 Preferences window of the B(PN)^2 editor 55
7.7 Set abstract name dialog window 56

8.1 High-level net editor 58
8.2 Comment dialog window 59
8.3 Place dialog window 60
8.4 Transition dialog window 61
8.5 Arc dialog window 62
8.6 Block dialog window 63
8.7 Box dialog window 63
8.8 Find dialog window 67
8.9 Node selection dialog window 67
8.10 Hierarchy tree dialog window 68
8.11 Block list dialog window 68
8.12 Other net list dialog window 69
8.13 The 'Nela' window 69
8.14 Add to box dialog window 70
8.15 Instantiate box dialog window 71
8.16 Append suffix dialog window 71
8.17 Parallel composition dialog window 72
8.18 Sequential composition dialog window 72
8.19 Choice composition dialog window 72
8.20 Transition substitution dialog window 73
8.21 Synchronisation dialog window 73
8.22 Restriction dialog window 74
8.23 Expression dialog window 74
8.24 Active transitions dialog window 75

8.25 Transition area dialog window 76
8.26 Execute transitions list dialog window 77
8.27 Select servers dialog window 78
8.28 Change font dialog window 79
8.29 Synchronisation options dialog window 80
8.30 Move options dialog window 81
8.31 Simulation options dialog window 81
8.32 Simulation statistics window 82
8.33 Template directory selection window 82
8.34 Execution dialog window 83
8.35 Execution dialog window 84

9.1 Low-level net editor 86
9.2 Comment dialog window 87
9.3 Place dialog window 88
9.4 Transition dialog window 89
9.5 Arc dialog window 90
9.6 Block dialog window 90
9.7 Box dialog window 91
9.8 Find dialog window 94
9.9 Hierarchy tree dialog window 94
9.10 Block list dialog window 95
9.11 Other net list dialog window 95
9.12 The 'Nela' window 96
9.13 Add to box dialog window 97
9.14 Parallel composition dialog window 97
9.15 Sequential composition dialog window 97
9.16 Choice composition dialog window 98
9.17 Transition substitution dialog window 98
9.18 Active transitions dialog window 99
9.19 Transition area dialog window 100
9.20 Execute transitions list dialog window 100
9.21 Transition dialog window (with time) 102
9.22 Change font dialog window 103
9.23 Move options dialog window 104

10.1 Formula editor 105

11.1 SDL editor 109
11.2 SDL properties editor 111
11.3 Preferences window of the SDL editor 113

List of Tables

7.1 Regular expressions 53

10.1 Logic of the prefix checker 106
10.2 Logic of the SPIN checker 106
10.3 Logic of the SMV checker 107

Chapter 1

Installation of the PEP tool

At the present time we provide **PEP Version 1.8** for Solaris 2.x (abbreviated as `solaris`), Linux (abbreviated as `linux`), Linux with the new glibc (abbreviated as `linux_glibc`), and Irix (abbreviated as `irix`).

Installation of the PEP tool involves the following steps. (In the following we assume a version 1.8a)

1. Download the file:
   `PEP1.8a_`*operating-system*`_prg.tar.gz`
   from our Web-Page
   http://theoretica.informatik.uni-oldenburg.de/~pep/HomePage.html
   Note: *operating-system* must be replaced by one of the abbreviations mentioned above.
2. Uncompress it with `gzip -d PEP1.8a_`*operating-system*`_prg.tar.gz`, obtaining the file `PEP1.8a_`*operating-system*`_prg.tar`.
3. Untar this file with `tar -xvf PEP1.8a_`*operating-system*`_prg.tar.gz`; a directory `pep` with several subdirectories is created (make sure no file or directory called `pep` is present before executing this step). Note: If you have `gtar` you may combine steps 2 and 3 by typing `gtar -xvzf PEP1.8a_`*operating-system*`_prg.tar.gz`.
4. Set an environment variable `PEPHOME` to the name of the new directory.
   If you use `csh` you can include the following line in your `.cshrc` file:
   `setenv PEPHOME` *pathname*`/pep`
   If you use `bash` you can include the following line in your `.profile` file:
   `export PEPHOME="`*pathname*`/pep"`
   You can set `PEPHOME` temporary by typing one of these commands at the shell prompt.
5. Add the subdirectory *pathname*`/pep/bin` to your path.
   If you use `csh` you can include the following line in your `.cshrc` file:
   `set path = ($path` *pathname*`/pep/bin)`
   If you use `bash` you can include the following line in your `.profile` file:
   `export PATH=$PATH:`*pathname*`/pep/bin`
6. Write an email to `pep_help@informatik.uni-oldenburg.de` (Bernd Grahlmann) and ask for a license file. Your mail should contain:
   (a) Some brief information about your interest in the PEP tool and about your institution. The PEP tool is distributed free of charge only for non-commercial use.

(b) The hostid of the computer(s) on which you would like to use the tool. You can get this information by typing `hostid` in your shell. (If you run Solaris `/usr/ucb/hostid` should work.)

7. Insert the file you will receive as *pathname*`/pep/.pep_pswd`.
8. Now you may invoke the peptool from any directory by typing `peptool` at the prompt.

Notes:

1. The easiest way to set the environment variables once and forever, is to edit the script *pathname*`/pep/bin/pep` and to use this for the invocation of the PEP tool.
2. To use the PEP tool you need patched versions of Tcl (Version 7.6) and Tk (Version 4.2). Therefore, we highly recommend to download the correct versions from our Web page. Please have also a look at the two files `INSTALL_tcl7.6_bins.txt` and `INSTALL_tcl7.6.txt`.
3. To use the new 3D visualisation component very specific versions of Java and Liquid Reality are required. Please, have a look at the installation description given in the file `INSTALL_java.txt` on our Web page.
4. You may have problems with older versions of linux. Therefore, we provide new versions of the most important libraries on our Web page.
5. You can invoke the PEP tool and use it in a restricted way even if you have not received a license file.
6. Please communicate any problem you experience to `pep_help@informatik.uni-oldenburg.de`.

Chapter 2

General guidelines

This section describes some guidelines for the usage of the PEP tool. We spent a lot of work to eliminate all bugs and to make the tool user friendly, but nobody and of course no tool is perfect.

- During the work with the PEP tool it may happen, that Tcl error boxes pop up (see figure 2.2). Although most of the possible non-intended usages of the tool are caught automatically yielding a PEP error box like the one shown in figure 2.1, in most of the cases such a usage is the reason for the occurrence of the Tcl error box. In this case you can simply click on the **OK** button and continue with your work. If you discovered a real error or wrong behaviour of the tool, please send an e-mail to `pep_help@informatik.uni-oldenburg.de` containing:

  1. preferably, the output of the menu item **Option▷Show Architecture** of the PEP start-up window.
  2. (if this does not work) the version of the PEP tool in use,
  3. and the kind of operating system,
  4. (in any case) the error message (see figure 2.3) that you will get if you click on the **See Stack Trace** button,
  5. and a short description of the history; what you were doing before the error box occurred.

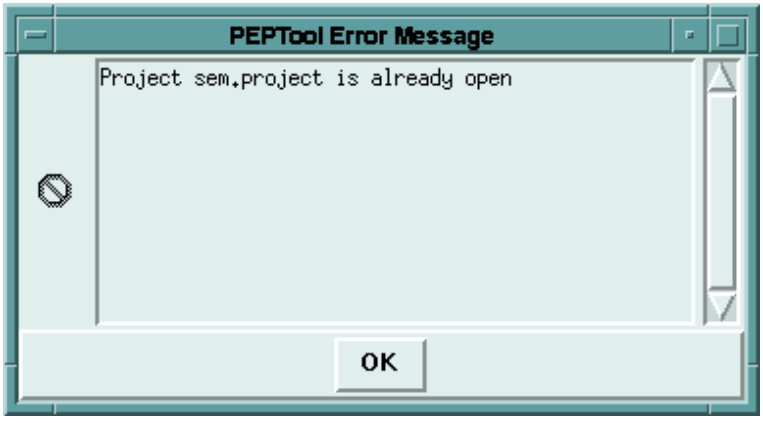


Figure 2.1: PEP error box

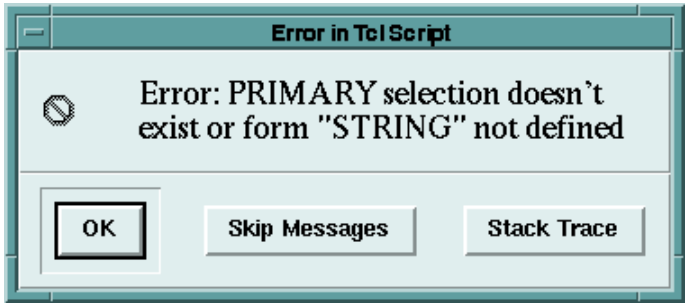


Figure 2.2: Tcl error box

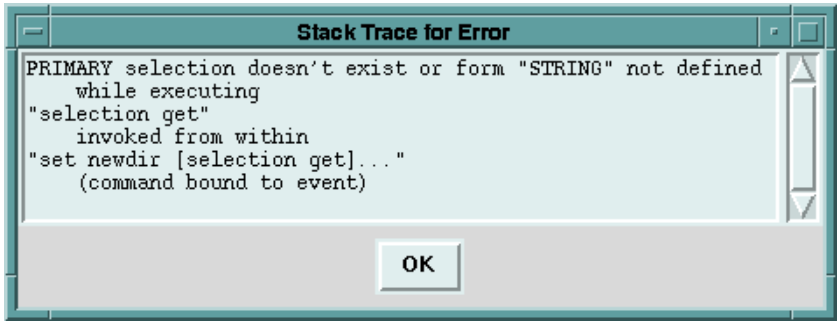


Figure 2.3: See stack trace window

- In general you can iconify the shell during the work with the PEP tool. Some debugging information might be output to the shell if you check the **Display Debug** option in the start-up window. If you are curious you can read them, but it is not necessary for your work with PEP.
- In general you do not have to change the default values of the options for the different algorithms and compilers. Most of them are meant for debugging purposes. Some of them may speed up the calculations, but the defaults are nearly perfect for every example. Only some of them (like **No Scoping** for the compiler **B(PN)^2⇒HL Net** or **Take Current Marking Instead of Initial** for the **Prefix Builder**) are interesting for the usual work with the tool. All the figures explaining these options show the default values.
- If you are working with one of the net editors and the editor does not react in the intended way (e.g., does not create a place if you use the **p**-key) it is very likely that the editor is in a different mode. Try to click at any position of the canvas and hit the **Esc** key afterwards.
- If the tool seems to hang it might be the case that there are error boxes or warning boxes or dialog boxes which have not been closed.

Chapter 3

Getting started

After installing PEP and setting the correct paths, the PEP tool can be invoked from within any directory by typing `peptool` at the command line (or better using the `pep` invocation script). If you are using PEP for the first time, we recommend to invoke it from the `New_FTP_Examples` directory included in the distribution.

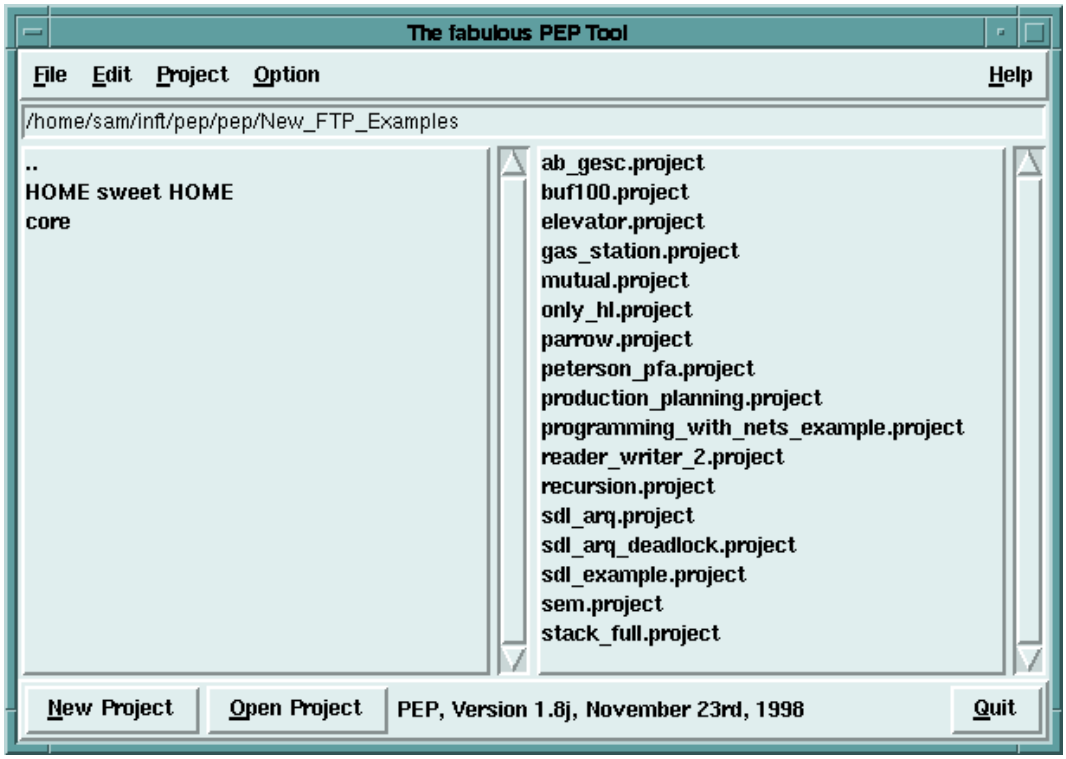


Figure 3.1: The start-up window of the PEP system

The first window that pops up is the start-up window shown in figure 3.1. Its main purpose is to open existing or create new projects. We recommend to (first) open the 'semaphore' project by double-clicking on `sem.project` and to follow the steps described below.

- A window belonging to the 'semaphore' project will appear. Such a window (shown in figure 3.2) is central to the work with a project. It shows the project's objects (as well as model checker and algorithms), and their interdependencies. Figure 3.2 displays twelve objects: a B(PN)^2 program, an HL (high-level) net, an LL (low-level) net, a process

algebra term (called PBC for Petri Box Calculus), C source code, a temporal logic formula, a network of finite automata in FC2 format needed for one of the deadlock-checker, a finite prefix of the branching process (called Prefix) needed for partial order based model checking, an SMV representation of the LL net (needed for the integration of SMV's BDD based CTL model checker), a PROMELA representation of the LL net (needed for the integration of the SPIN LTL checker), and documentation. Objects are saved in separate files in the UNIX file system. Even though the names of these files are not hidden, it is advised not to change the defaults offered by the tool. Possible interdependencies are shown by arrows from one object A to another object B. Arrows can be green, denoting that A currently belongs to B, or red, denoting that currently no dependency exists (for instance, because A has been modified without modifying or recalculating B accordingly.)

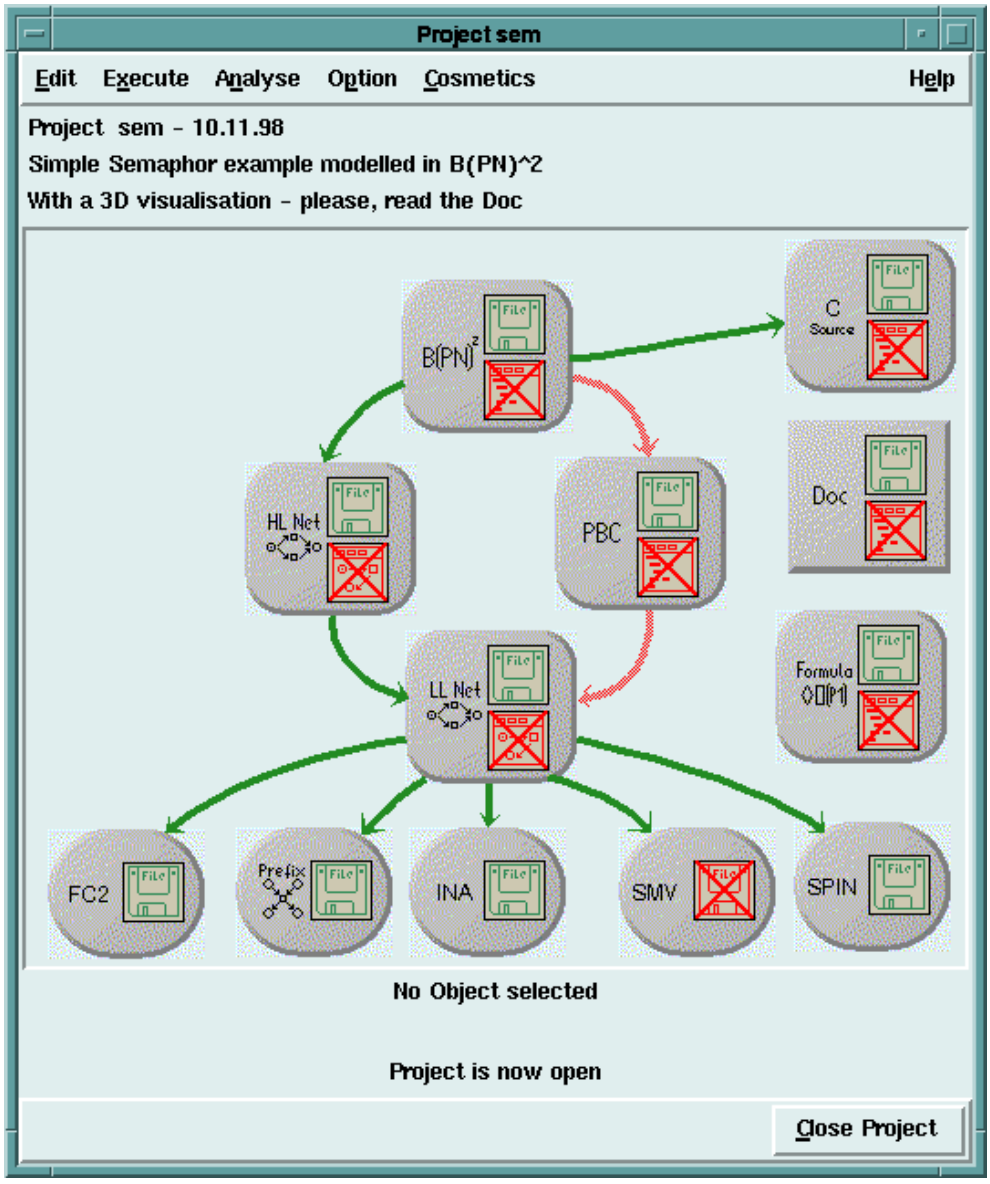


Figure 3.2: Project window belonging to the 'semaphore' project

- Open the editor for the B(PN)^2 program by selecting the menu item **Edit▷Open▷ B(PN)²** or easier by double-clicking on the **B(PN)²** icon. A text editor window con-

taining an example program for the mutual exclusion of two processes will be displayed.

- Compile the M-net semantics of this program by selecting the menu item **Execute▷ B(PN)²⇒HL Net** or easier by double-clicking on the arc from the **B(PN)²** icon to the **HL net** icon.
- Open the editor for the M-net by selecting the menu item **Edit▷Open▷HL Net** or by double-clicking on the **HL Net** icon. The editor displays the (hierarchical) M-net semantics of the B(PN)^2 program. The contents of the two blocks for the two (parallel) processes may be exposed by clicking on the **Expose Contents** button.

  The net editor has an integrated simulator that can be started by selecting the menu items **Action▷Execute** (or easier by clicking on the button) for interactive simulation, or **Action▷Animate** (or easier by clicking on the button) for automatic simulation. If the option **Option▷Highlight Execution** is selected then the corresponding actions in the B(PN)^2 program are highlighted during animation. Furthermore references between actions in the B(PN)^2 program and transitions in the net, and between places in the control flow of the B(PN)^2 program and places in the net, can then be analysed by selecting the menu item **Action▷Show Reference** (or easier by clicking on the button), and then clicking on the transitions, places and blocks of interest. If necessary the B(PN)^2 editor is opened automatically.
- Compile the Petri box semantics of the program by selecting the menu item **Execute▷ HL Net⇒LL Net** or easier by double-clicking on the arc from the **HL net** icon to the **LL net** icon.
- Open the editor for the Petri box by selecting the menu item **Edit▷Open▷LL Net** or easier by double-clicking on the **LL Net** icon. This editor has functionality similar to the high-level net editor. In particular, there is an analogous simulator.
- There are choices for verifying a property. Either in-built properties (is the net a T-system? is it safe? is it deadlock-free? etc.) can be checked by dedicated algorithms, or a property can be entered and one of the model checker can be invoked to check it. To invoke the dedicated algorithms, select the corresponding menu items of the **Analyse** menu.
- Before invoking the prefix model checker it is necessary to generate the finite prefix of the branching process. To this end, select the menu item **Execute▷LL Net⇒Prefix** or double-click on the arc from the **LL net** icon to the **Prefix** icon.
- Similarly, the PROMELA representation of the LL net has to be generated by selecting the menu item **Execute▷LL Net⇒SPIN** (or a double-click on the arc from the **LL net** icon to the **SPIN** icon) before the Spin verifier may be invoked.
- Whereas, the generation of the SMV representation of the LL net is generated transparently whenever the SMV model checker is invoked.
- Properties can be entered in the formula editor which can be opened by selecting the menu point **Edit▷Open▷Formula** or easier by double-clicking on the **Formula** icon. A formula file may contain many formulae of different kind. The one which is not commented out by a **%** is used for the (prefix, SMV or Spin) model checking. We recommend to use first one of the given formulae.
- A model checking window is (e.g., if a prefix formula is selected) opened by selecting the menu point **Analyse▷Model Checker (Safe)** or by double-clicking on the **Prefix**

icon. A click on the **OK** button invokes the model checker. The result is displayed in the main part of the window (and in the bottom of the project window). Depending on the formula it may be possible to simulate a resulting transition sequence either in the LL editor or (in this example) in the B(PN)^2 editor.

Following this general pattern and keeping in mind the standards for window based programs, it will be possible to work with the PEP tool in an intuitive way and to explore the system. In case of doubt you can invoke the online help system in the project window (which is unfortunately not as up-to-date as the present manual). Nevertheless, it is necessary to read the documentation in order to exploit its full functionality.

Chapter 4

The PEP start-up window

4.1 Description of the start-up window

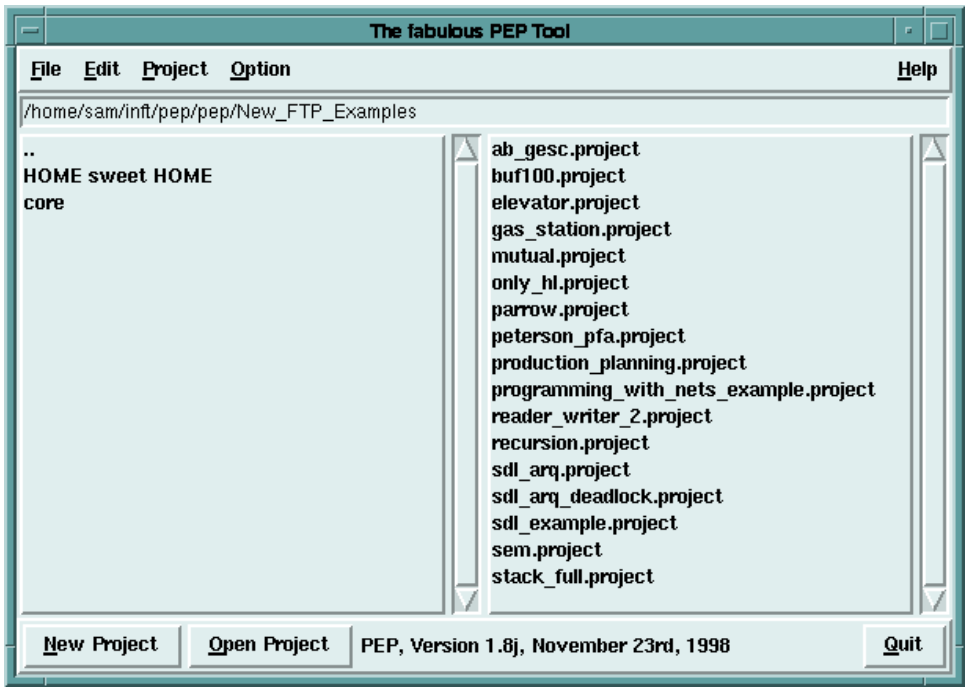


Figure 4.1: The start-up window of the PEP system

After invoking PEP by typing `peptool` at the command line (or better using the `pep` invocation script), the first window that appears is the main window shown in figure 4.1. This window is the starting point for the invocation of further windows, each of them belonging to exactly one project. A project comprises all objects belonging to a modelling task. For instance, you may write a B(PN)^2 program, then generate the corresponding high and low-level nets, (the finite prefix of) the branching process, and finally input temporal logic formulae expressing the properties to be checked. All these objects belong to one project. You may write a different B(PN)^2 program and only create some documentation about it, without computing the nets. This will be a different project. You may also use a net editor to create an (HL or LL) net without using a program first. This will be yet another different project. Or you may even start with an SDL specification or a parallel finite automaton (PFA). Which will give yet two other projects.

The start-up window also serves to open existing or to create new projects. Using the left part of the window, the file directory hierarchy can be traversed by double-clicking on the directory names. In the right part of the window, the existing projects in the current directory are displayed. A project can be opened either by a double-click on its name or by a click on the **Open Project** button when the project name is highlighted.

A new project can be created by clicking on the **New Project** button. The dialog box shown in figure 4.2 pops up. A name for the new project can be entered. If a name without the suffix `.project` is entered this suffix is appended automatically, because projects have to end with `.project`.

Figure 4.2: The new project window

Clicking the **Quit** button closes all opened projects and quits the PEP tool. Be careful, in the current version of PEP all unsaved changes (in the currently opened editors) are discarded!

4.2 Menus of the start-up window

The pull-down menus offer the following functionality:

- **File**
  - ▷ **New Project:** Creates a new project (see figure 4.2).
  - ▷ **Open Project:** Opens the selected project.
  - ▷ **Reread Directory:** Rescans the current directory.
  - ▷ **Change Directory:** Switches to the selected directory.
  - ▷ **Make Directory:** Offers the possibility to create a new directory.
  - ▷ **Quit:** Closes all opened projects and quits the PEP tool without saving unsaved changes of the currently opened editors.
- **Edit**
  - ▷ **Copy Project:** Copies a project. The dialog box shown in figure 4.3 is displayed. If a name without the suffix `.project` is entered this suffix is appended automatically, because projects have to end with `.project`. An option specifies whether or not all its objects are copied. Be careful, if you do not copy all objects then you end up with two projects using the same files!

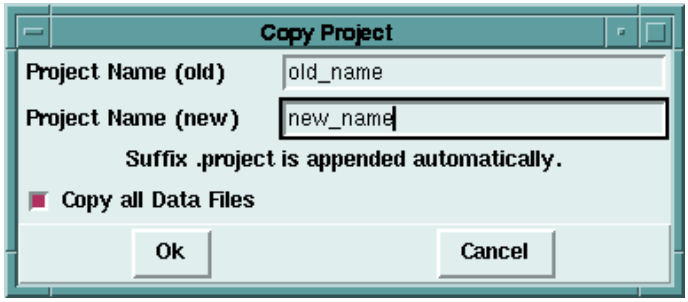


Figure 4.3: Copy project window

▷ **Rename Project:** Renames a project. The dialog box shown in figure 4.4 is displayed. If a name without the suffix `.project` is entered this suffix is appended automatically, because projects have to end with `.project`. An option specifies whether or not all its objects are renamed. Be careful, if you do not rename all objects then you end up with a project using files with differing names!

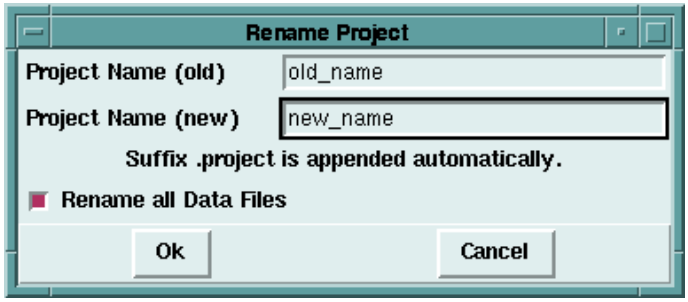


Figure 4.4: Rename project window

▷ **Delete Project:** Deletes a project. The dialog box shown in figure 4.5 is displayed. The name of the project should be entered with or without the postfix `.project`. An option specifies whether or not all its objects are deleted. Be careful, if you do not delete all objects then you end up with owning files that do not belong to any project!

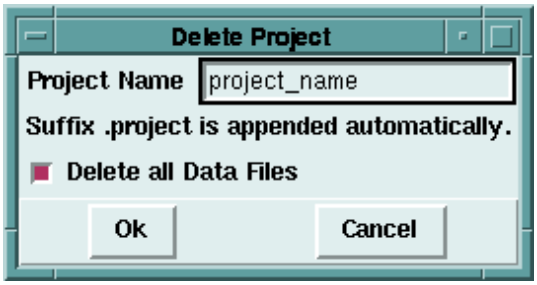


Figure 4.5: Delete project window

• Project

▷ **Project List:** Displays a list of all opened projects (see figure 4.6). A double-click on one of them brings its project window to the foreground.

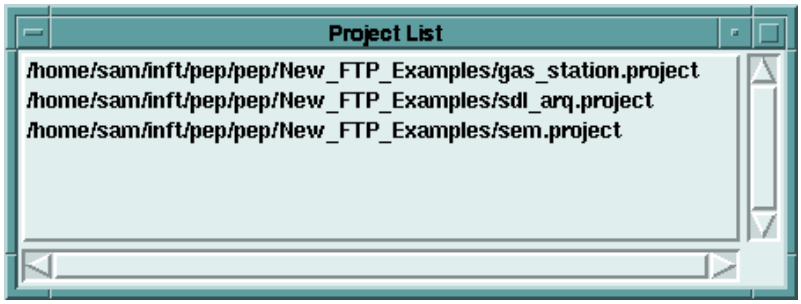


Figure 4.6: List of open projects

- **Options** Although these options are available we do not encourage you to use them. They are mainly interesting for the developers of the PEP tool, or in the case of problems with the installation or to collect information for a bug report.
  - ▷ **Display Debug:** If this option is selected then some debugging information (e.g., all calls to the algorithms invoked by the PEP tool, their results, etc.) is written to the shell during the run of the PEP tool. Usually you can iconify the shell during your work with the PEP tool because it is not needed.
  - ▷ **File Debug:** If this option is selected then the debugging information is continuously written to a file called 'peptool.deb'.
  - ▷ **Show Globals:** Writes Tcl/Tk information to the shell. This information is only interesting for Tcl/Tk programmers.
  - ▷ **Show Commands:** Writes even more Tcl/Tk information to the shell. This information is (again) only interesting for Tcl/Tk programmers.
  - ▷ **Show Auto_path:** Writes the Tcl/Tk auto_path variable to the shell. This information is (again) only interesting for Tcl/Tk programmers.
  - ▷ **Show Open Projects:** Writes a list of all opened projects to the shell.
  - ▷ **Show Com:** Displays some information about communications within the PEP tool in a dialog box. This is only interesting if communication between the different parts of the tool does not work correctly.
  - ▷ **Show Architecture:** Writes interesting information about your Tcl/Tk installation and the architecture of your computer to the shell. This information is quite interesting for us in the case of problems with the installation or as additional information for a bug report.
- **Help**
  - ▷ **About:** Shows the about box for the whole PEP tool.

Chapter 5

The project window

When a project is opened, a project window is displayed on the screen (see figure 5.1).

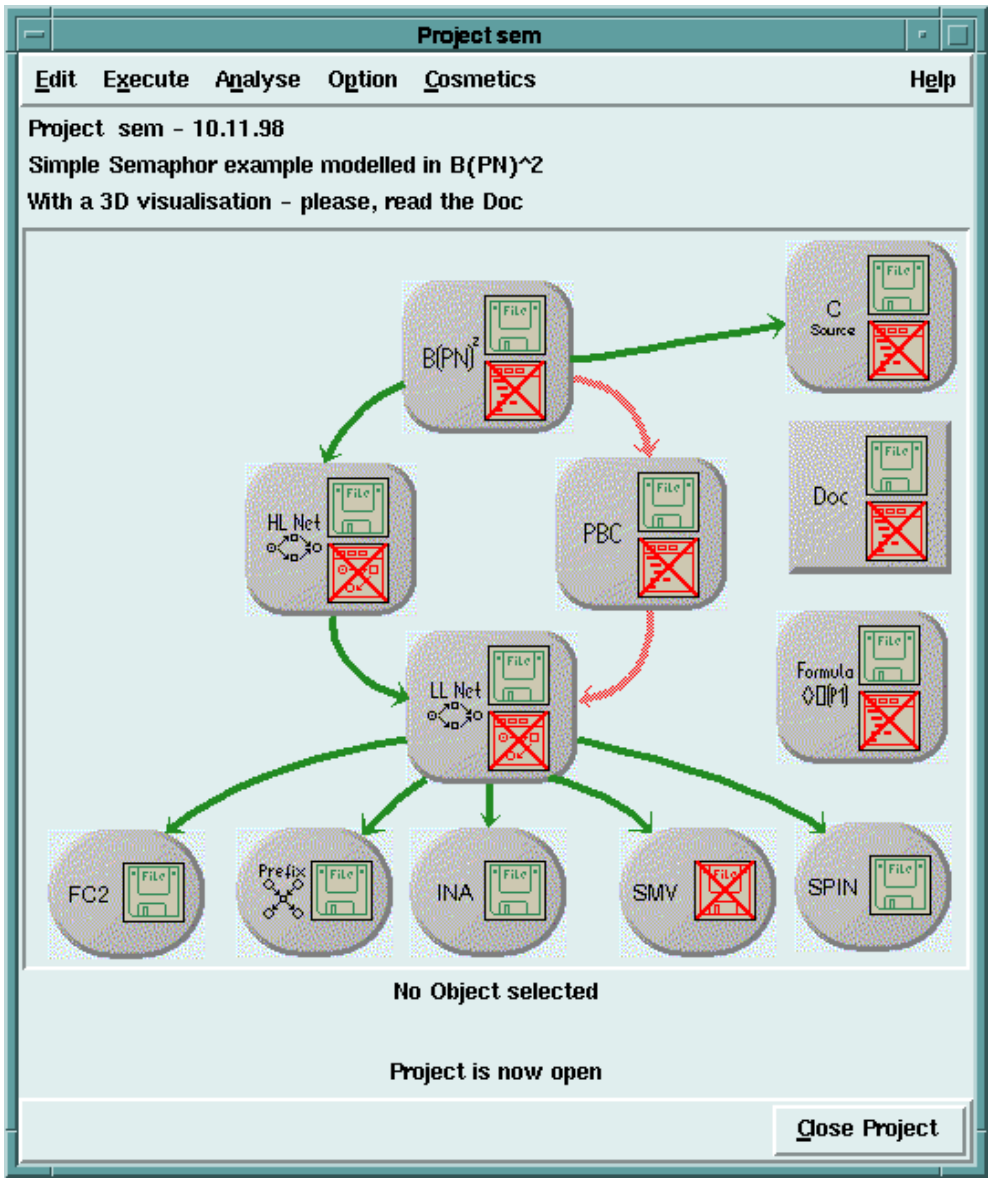


Figure 5.1: Project window belonging to the 'semaphore' project

5.1 Description of the project window

Since editors can be opened, compilers can be invoked, algorithms can be started, etc., the objects of a project may, at any time, be in different states and various relationships with each other. In addition to the objects themselves and their states, therefore, this window shows the current relationship between the objects of a project. It also allows to specify options for the algorithms and compilers.

The following objects may belong to a project:

- a B(PN)^2 program,
- a B(PN)^2 program with expanded procedures,
- an SDL specification,
- a parallel finite automaton (PFA),
- a high-level net (M-net),
- a low-level net (Petri box),
- a process algebra term (PBC term),
- a C Code,
- a finite prefix of the branching process,
- an FC2 (network of communicating automata) representation of the low-level net,
- an INA representation of the low-level net,
- an SMV representation of the low-level net,
- a PROMELA representation of the low-level net,
- a temporal logic formula, and
- a documentation.

The central part of the project window contains (at most) one icon for each object.[1] Depending on the state of the project, one of these objects may be associated with another one. Perhaps an M-net has been generated out of the B(PN)^2 program. These relations are presented in a graphical way. All possible relations are displayed by arcs. If object A currently belongs to object B then the arc from A to B is coloured green, otherwise it is coloured red (if a relation is possible but currently not present).

Each object icon contains one or two smaller icons. The top icon tells the user whether or not this object exists. It is coloured green if the object exists and crossed out red if the object does not exist. If this icon is green then there exists a file for the object. If the cursor is placed over the corresponding icon, the location and the file name are displayed under the main part of the window together with further information about the corresponding object. The second icon (if one exists) tells the user whether or not the corresponding editor (or,

[1] We will later explain how the menu item **Cosmetics▷Show Icons** may be used to hide icons for some of the objects and to customise the project window.

e.g., model checker) is currently opened. If the editor is not opened it can be opened by a double-click on this icon, otherwise it is shuffled up to be completely visible.

Moreover, a double-click with the left mouse button invokes a compiler. In the bottom part of the window, the progress of the main actions (invocation of the editors, algorithms or compilers, and their results) are reported.

An 'Options' menu for a compiler may be opened by a click with the right mouse button on the corresponding arc. A click with the right mouse button on one of the three model checker icons (for 'Prefix', 'SMV' or 'SPIN') similarly allows to change their options. Note that the options for the SPIN verifier may only be changed if PROMELA code has been generated from the low-level net and the formula file contains a syntactically correct LTL formula.

Underneath the pull-down menus, two lines of project information are displayed.

The **Close Project** button can be used to close the project. But be careful, the contents of every opened editor are saved by default before closing! If you want to discard the changes you can use the menu item **Edit▷Close▷Discard Files**.

5.2 Menus of the project window

Further functionality is offered by the pull-down menus.

- **Edit**
  - ▷ **Open** — Note: It is easier to open one of the following editors with a double-click on the corresponding icon.
    - ○ **B(PN)²:** Opens the B(PN)² editor for the B(PN)² program.
    - ○ **Expanded B(PN)²:** Opens the B(PN)² editor for the expanded B(PN)² program, which is the same as the B(PN)² program except that procedures have been expanded.
    - ○ **PBC:** Opens the PBC editor for the process algebra term.
    - ○ **LL Net:** Opens the editor/simulator for the low-level Petri net.
    - ○ **HL Net:** Opens the editor/simulator for the high-level Petri net.
    - ○ **PFA:** Opens the PFA editor for a parallel composition of finite automata.
    - ○ **C Source:** Opens the editor for the generated C code.
    - ○ **Formula:** Opens the editor for the temporal logic formulae.
    - ○ **Document:** Opens the editor for the documentation.
    - ○ **INA:** Opens the INA (Integrated Net Analyser) window for a number of Petri net analyses.
    - ○ **Simulator:** Opens the simulator window which may offer 3D visualisation.

- ◦ **Animator:** Opens the animator window which provides the facilities to create a 3D visualisation (if a 3D VRML model exists).
- ◦ **SDL:** Opens the SDL editor for the SDL specification.
- ◦ **SDL Properties:** Opens a simple editor for additional properties (restrictions) of the SDL specification (which are taken into account for the compilation into a high-level net).

▷ **Edit Project Info:** Opens a two-line editor for project information as shown in figure 5.2.

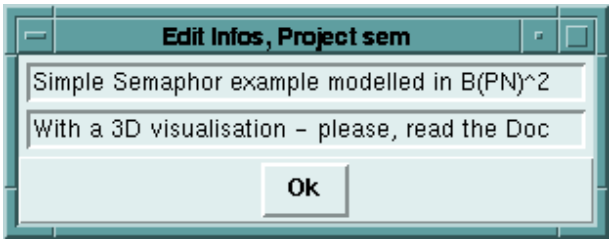


Figure 5.2: Edit project information window.

▷ **Save Project Info:** Saves the two-line project information, the current information concerning the relations between the different objects, and the analysis results, to the project file.

▷ **Save Entire Project:** Saves the contents of all opened editors.

▷ **Close**

- ◦ **Save Files:** Closes the project after saving the contents of all opened editors as well as the two-line project information, the current information concerning the relations between the different objects, and the analysis results, to the project file.
- ◦ **Discard Files:** Closes the project without saving the contents of all opened editors etc. to the project file.

• Execute

▷ **B(PN)^2 ⇒ PBC:** Invokes the compiler from a B(PN)^2 program to a process algebra term. The behaviour of the compiler is controlled by the options shown in figure 5.14 on page 29 and influenced by the option shown in figure 5.13 on page 29.

▷ **PBC ⇒ LL Net:** Invokes the compiler from a process algebra term (PBC) to a low-level Petri net (Petri box). The behaviour of the compiler is controlled by the options shown in figure 5.15 on page 30.

▷ **B(PN)² ⇒ LL Net (via PBC):** Invokes the two compilers mentioned above, from a B(PN)² program to a process algebra term and then to a low-level Petri net (Petri box), in sequence. This is controlled by the options mentioned in the previous two paragraphs.

▷ **B(PN)² ⇒ HL Net:** Invokes the compiler from a B(PN)² program to a high-level net (M-net). The behaviour of the compiler is controlled by the options shown in figure 5.16 on page 31 and influenced by the options shown in figure 5.13 on page 29.

▷ **Expanded B(PN)² ⇒ HL Net:** Invokes the compiler from an expanded B(PN)² program (without procedures) to a high-level net (M-net). The behaviour of the expander is controlled by the options shown in figure 5.16 on page 31 and influenced by the options shown in figure 5.13 on page 29.

▷ **HL Net ⇒ LL Net:** Invokes the unfolder from a high-level net (M-net) to a low-level net. The behaviour of the unfolder is controlled by the options shown in figure 5.17 on page 33.

▷ **B(PN)² ⇒ LL Net (via HL Net):** Invokes the two compilers mentioned above, from a B(PN)² program to a high-level net and then to a low-level Petri net (Petri box), in sequence.

▷ **B(PN)² ⇒ Expanded B(PN)²:** If procedures are used in a B(PN)² program they are expanded on the B(PN)² level. The behaviour of the compiler is controlled by the options shown in figure 5.26 on page 43 and influenced by the options shown in figure 5.13 on page 29.

▷ **B(PN)² ⇒ C Source:** Invokes the C code generator taking a B(PN)² program as input.

▷ **Sync HL Net:** Invokes (an external) algorithm which synchronises (and/or restricts) the high-level net (M-net). The behaviour of the synchroniser is controlled by the options shown in figure 5.18 on page 34.

▷ **LL Net ⇒ Prefix:** Invokes the prefix builder for the calculation of the finite prefix of the branching process. The behaviour of the prefix builder is controlled by the options shown in figure 5.19 on page 35. Be careful, this works correctly only for safe Petri nets, but for reasons of efficiency it is not checked whether or not the net is safe. Every LL net (Petri box) generated out of a B(PN)² program or SDL specification is automatically safe.

▷ **Minimize Net:** Calculates all transitions of the low-level net which are dead under the initial marking and deletes them together with their corresponding arcs. This may reduce the size of the net considerably if the net has been generated out of a B(PN)^2 program or an SDL specification. Be careful, this works correctly only for safe Petri nets, but for reasons of efficiency it is not checked whether or not the net is safe. However, every LL net (Petri box) generated out of a B(PN)^2 program or an SDL specification is automatically safe.

▷ **LL Net ⇒ INA:** Invokes a converter which generates the INA format of the low-level net. The behaviour of the converter is controlled by the options shown in figure 5.27 on page 43.

▷ **LL Net ⇒ SPIN:** Invokes a PROMELA (the input language of SPIN) generator which takes the low-level net as input. The behaviour of the compiler is controlled by the options shown in figure 5.28 on page 43.

▷ **LL Net ⇒ FC2:** Invokes a converter which compiles a low-level net into a network of communicating finite automata in the FC2 format

▷ **PFA ⇒ B(PN)^2:** Invokes the compiler from a PFA to a B(PN)^2 program. The behaviour of the compiler is controlled by the options shown in figure 5.30 on page 44.

▷ **PFA ⇒ HL Net:** Invokes the compiler from a PFA to a high-level net. The behaviour of the compiler is controlled by the options shown in figure 5.29 on page 44.

▷ **SDL ⇒ HL Net:** Generates the high-level net (M-net) semantics of the SDL specification (respecting the given SDL properties). The behaviour of the compiler is controlled by the options shown in figure 5.31 on page 45.

Note: Compiling a B(PN)^2 program into an LL net via an HL net (rather than via a PBC term) creates a number of references, e.g., from the LL net to the program. Thus, e.g., clicking on a transition in the LL net editor (in the 'Ref' mode) will highlight the atomic action(s) in the B(PN)^2 editor. Moreover, if the option **Highlight Execution** is checked, Petri net simulations trigger a simulation of the B(PN)^2 program (or SDL specification, respectively) by highlighting the atomic actions which correspond to the executed transition.

References are explained further in sections 7.2.4, 8.2 and 9.2 (pages 55, 77 and 101, respectively).

- **Analyse**

▷ **T-System:** Checks whether a low-level Petri net is a T-system.

▷ **FC System:** Checks whether a low-level net is free choice.

▷ **Safe System:** Checks whether a low-level net is safe.

▷ **Bounded System:** Checks whether a low-level net is n-bounded. The dialog box shown in figure 5.3 is opened. The upper bound n can be entered and the result of the previous test is displayed. The analysis is controlled by the options shown in figure 5.20 on page 36.

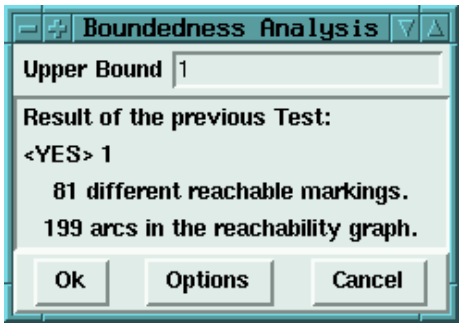


Figure 5.3: Boundedness analyser

▷ **Reachability:** Checks whether a (sub-) marking of a low-level net is reachable. The dialog box shown in figure 5.4 is opened. The names of the places and the number of tokens on these places can be entered and the results of the previous test are displayed. The analysis is controlled by the options shown in figure 5.21 on page 37.

The algorithm calculates the reachability graph and works only for bounded (not necessarily safe) Petri nets. If a net is not bounded this is displayed.
Note: A **YES** means that the net is bounded. Otherwise the text **The net is not bounded** is displayed.

If the marking is reachable the transition sequence yielding this marking from the initial marking is displayed in the main part of the window and may be simulated in the low-level net editor (or if the net was generated automatically in the B(PN)^2 or SDL editor). Otherwise no path is displayed.

▷ **LP Reachability:** Checks whether a marking of a low-level net is reachable or more sophisticated (extended) reachability properties. The dialog box shown in figure 5.5 is opened. Either one list of marked places and one list of not-marked places can be entered or one extended reachability property may be given. As with other analysis windows, the results of the previous test are displayed. The analysis is controlled by the options shown in figure 5.22 on page 38.
Note: In contrast to other algorithms this is a semi-decision algorithm which either answers **No** or **May be yes**. In the latter case a potentially reachable marking is given.

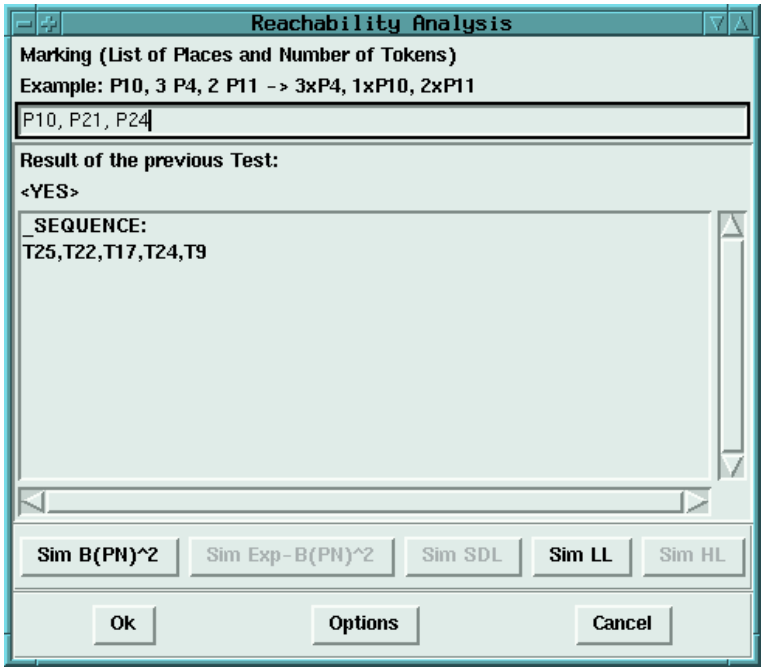


Figure 5.4: Reachability analyser

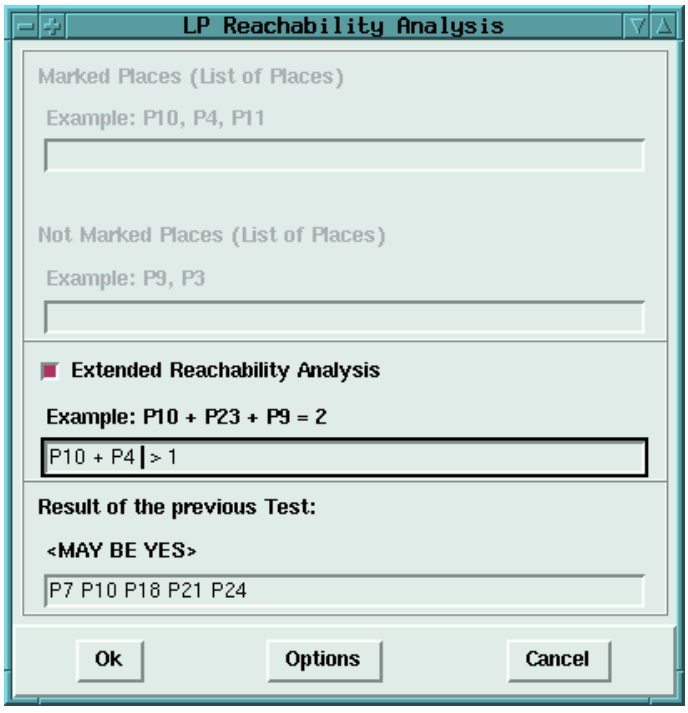


Figure 5.5: LP reachability analyser

▷ **Deadlock-free**

System: Checks whether or not a state can be reached in which no transition is enabled and displays the corresponding transition sequence (which may again be simulated in one of the editors) leading to such a state if one exists. The dialog box shown in figure 5.6 is opened. This algorithm works on the finite prefix of the branching process, which must have been created previously. Thus it works correctly only for safe systems.

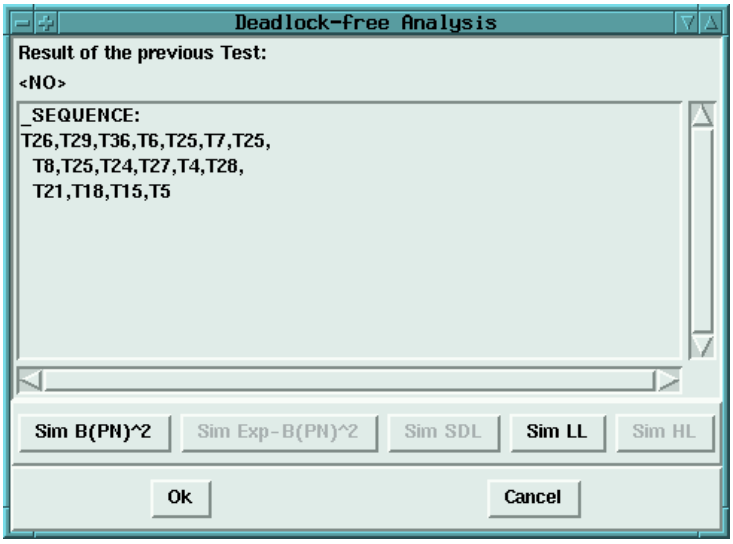


Figure 5.6: Analyser to check the deadlock-freeness property

▷ **LP Deadlock-free**

System: Checks whether a low-level Petri net is deadlock-free. The dialog box shown in figure 5.7 is opened. The analysis is (as the LP reachability analyser) controlled by the options shown in figure 5.22 on page 38. The window (in contrast to 'LP Reachability') subsumes two algorithms. The first one uses the Petri net directly as input and works even for unbounded Petri nets while the second one (option 'Use Prefix instead of LL Net') only works for safe Petri nets and assumes that the prefix has already been calculated. Another difference between the two algorithms is that the prefix-based algorithm gives a clear **Yes** or **No** result, while the net based method is again a semi decision algorithm. In the case of a deadlock, the prefix-based method yields a transition sequence while the net based analyses gives a potentially reachable deadlock marking.

▷ **FC2 Deadlock-free**

System: Checks whether a low-level Petri net is deadlock-free. The dialog box shown in figure 5.8 is opened. This algorithm works on an FC2 (networks of finite automata) representation of the low-level net (which must have been created previously), and (for the time being) works correctly only for safe systems. In the case of a deadlock, a trail is returned which may again be simulated in one of the editors.

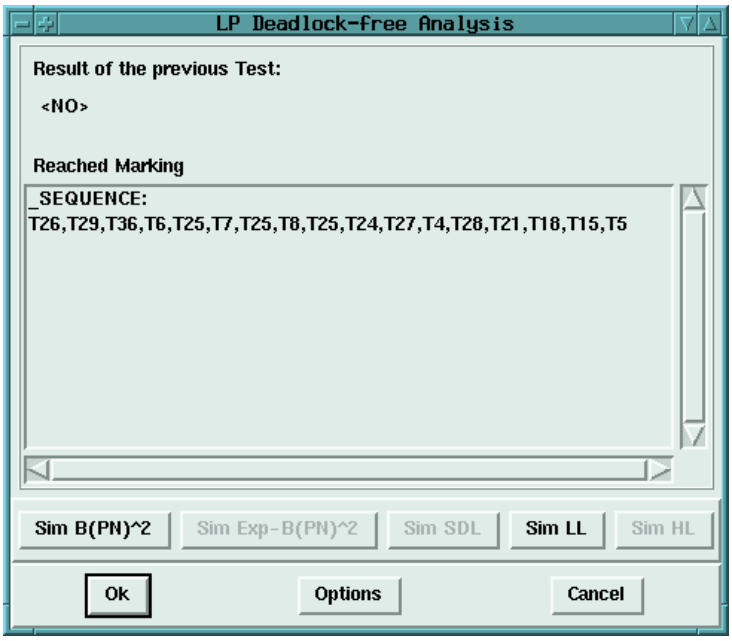


Figure 5.7: Linear programming based checker of the deadlock-freeness property

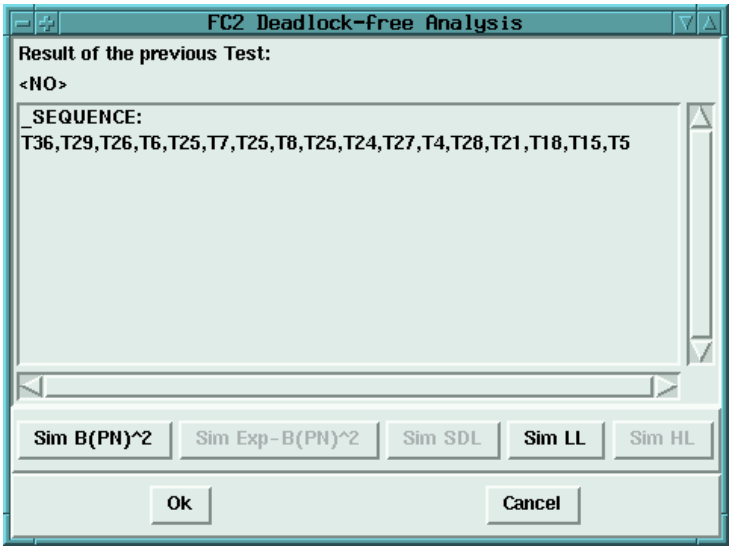


Figure 5.8: A deadlock-freeness checker using FC2

▷ **Model Checker (T-System):**

Performs very fast model checking for T-systems. This algorithm takes a low-level T-system and a formula as input. It uses linear algebra and does not need any pre-calculations such as the finite prefix. The T-system property is checked automatically, because it would work incorrectly for non-T-systems. The result is displayed in the bottom of the project window.

▷ **Model Checker (Safe):**

The dialog box shown in figure 5.9 is opened. This serves to invoke a fast model checking algorithm for safe Petri nets. This algorithm takes the finite prefix of the branching process of a low-level Petri net and a formula as input. The behaviour of the model checker is controlled by the options shown in figure 5.23 on page 39.

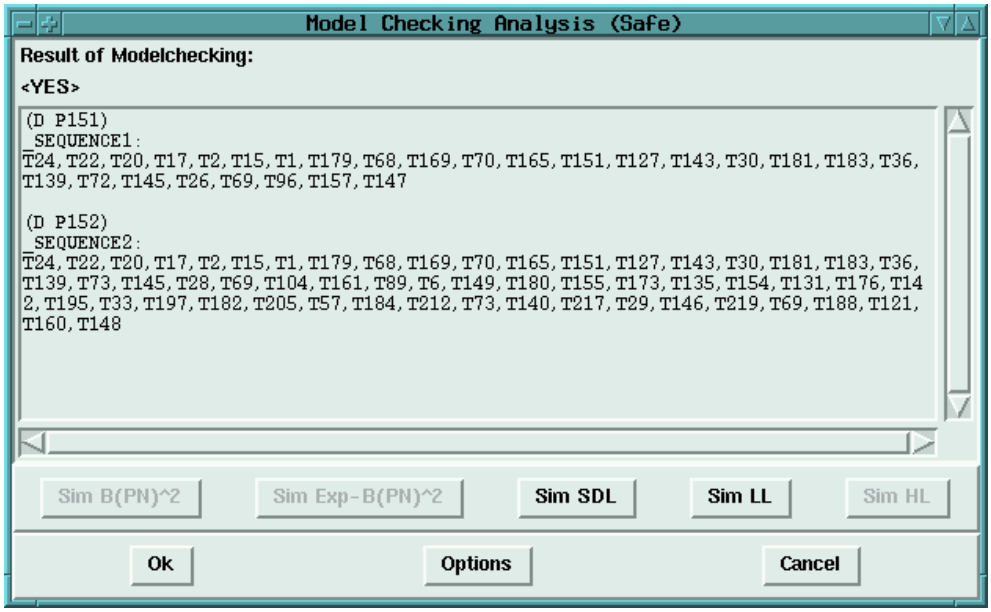


Figure 5.9: The dialog box of the prefix based model checker

For many formulae, this model checker is able to produce a transition sequence (determining the result of the verification) which may the be simulated in one of the editors.
Note: It may be the case that the formula is splitted up into a number of sub-formulae and that the model checker yields a transition sequence (starting with **_SEQUENCE1:**, **_SEQUENCE2:**, ...). You may select one of these sequences for simulation by deleting the number (1, 2, ...).

▷ **Model Checker (SPIN):**

The dialog box shown in figure 5.10 is opened. This serves to invoke a fast model checking algorithm for Petri nets. An option of the compiler from a low-level net into the SPIN formalism controls whether 1-boundedness or 255-boundedness is assumed. This

algorithm (basically) takes the PROMELA representation of a low-level Petri net and a formula as input. The behaviour of the model checker is controlled by the options shown in figure 5.25 on page 41. In contrast to the prefix based model checker, this algorithm 'only' produces error trails (i.e. counter examples). Moreover, this may be a sequence followed by an additional loop.

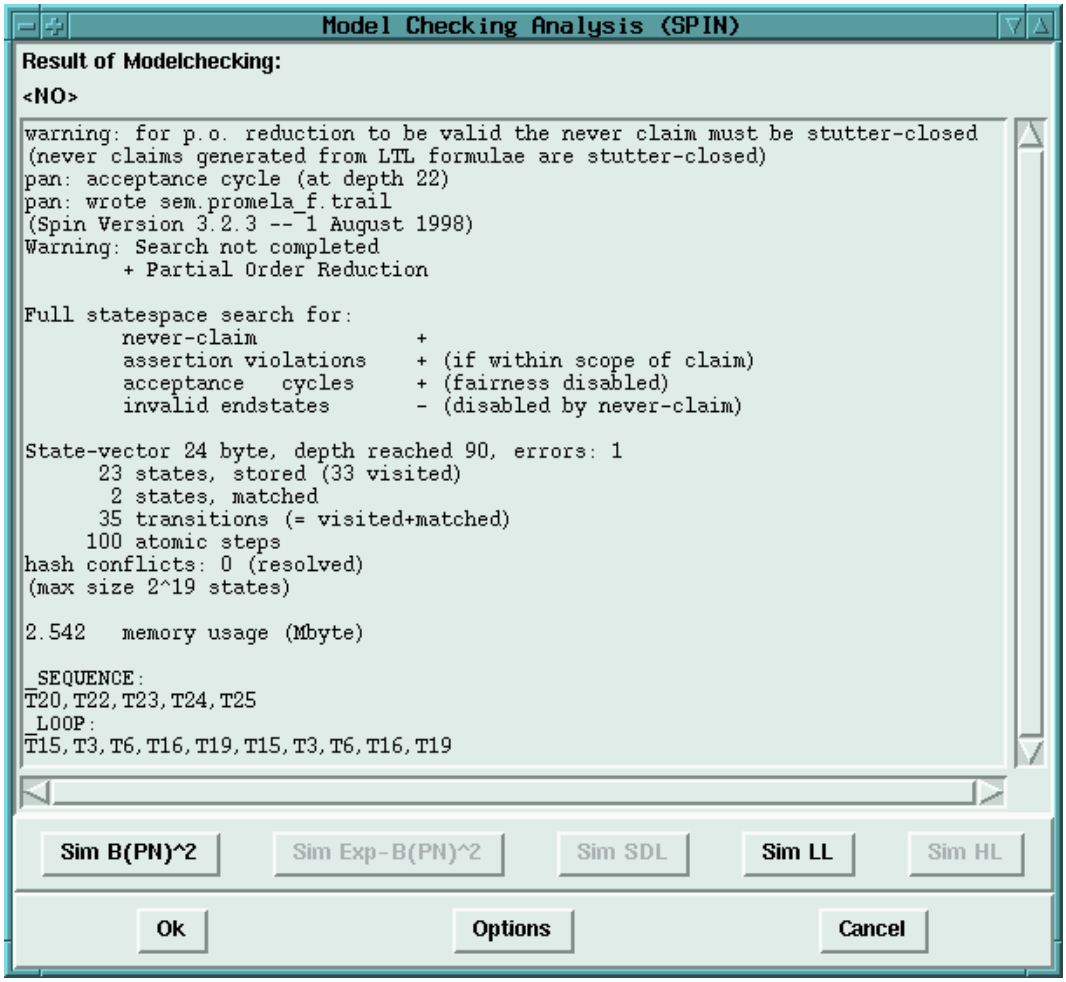


Figure 5.10: The dialog box of the integrated SPIN model checker

▹ **Model Checker (SMV):** The dialog box shown in figure 5.11 is opened. This serves to invoke a fast model checking algorithm for (for the time being) safe Petri nets. This algorithm (basically) takes the SMV representation (which is transparently created) of a low-level Petri net and a formula as input. The behaviour of the model checker is controlled by the options shown in figure 5.24 on page 40. As the SPIN based model checker, this algorithm again 'only' produces error trails (i.e. counter examples) and may also yield a sequence followed by an additional loop.

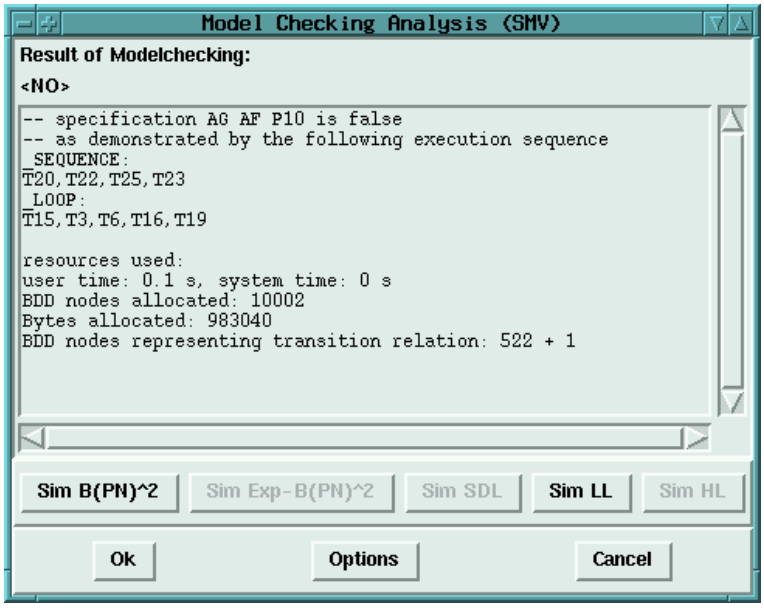


Figure 5.11: The dialog box of the integrated SMV model checker

- **Option**
  - ▹ **Converter Options**
    - ◦ **Syntax Check:** Controls whether or not the syntax of a B(PN)^2 program is checked automatically before a compiler is invoked. The dialog box shown in figure 5.13 on page 29 is displayed.
    - ◦ **B(PN)^2 ⇒ PBC:** Controls the behaviour of the compiler from the B(PN)^2 program to a PBC term. The dialog box shown in figure 5.14 on page 29 is displayed.
      Note: Using this compiler no reference information is generated.
    - ◦ **PBC ⇒ LL Net:** Controls the behaviour of the compiler from the PBC term to the low-level Petri net. The dialog box shown in figure 5.15 on page 30 is displayed.
      Note: Using this compiler no reference information is generated.
    - ◦ **B(PN)^2 ⇒ HL Net:** Controls the behaviour of the compiler from the B(PN)^2 program to the high-level Petri net (M-net). The dialog box shown in figure 5.16 on page 31 is displayed.
    - ◦ **HL Net ⇒ LL Net:** Controls the behaviour of the compiler from the high-level net (M-net) to the low-level Petri net (Petri box). The dialog box shown in figure 5.17 on page 33 is displayed.

- **Sync HL Net:** Controls the behaviour of the (external) algorithm which synchronises (and/or restricts) the HL net (M-net). The dialog box shown in figure 5.18 on page 34 is displayed.
- **Prefix Builder:** Controls the behaviour of the prefix builder (the calculator of the finite prefix of the branching process). The dialog box shown in figure 5.19 on page 35 is displayed.
- **Boundedness Analyser:** Controls the behaviour of the boundedness analyser. The dialog box shown in figure 5.20 on page 36 is displayed.
- **Reachability Analyser:** Controls the behaviour of the reachability analyser. The dialog box shown in figure 5.21 on page 37 is displayed.
- **LP Analyser:** Controls the behaviour of the linear programming based analyser. The dialog box shown in figure 5.22 on page 38 is displayed.
- **Model Checker (Safe):** Controls the behaviour of the model checking algorithm for safe Petri nets. The dialog box shown in figure 5.23 on page 39 is displayed.
- **Model Checker (SMV):** Controls the behaviour of the integrated SMV package which offers BDD based CTL model checking for (for the time being) safe Petri nets. The dialog box shown in figure 5.24 on page 40 is displayed.
- **Model Checker (SPIN):** Controls the behaviour of the integrated Spin verifier which offers LTL model checking for (not necessarily safe) Petri nets. The dialog box shown in figure 5.25 on page 41 is displayed.
- **B(PN)^2 Expander:** Controls the behaviour of the B(PN)^2 expander. The dialog box shown in figure 5.26 on page 43 is displayed.
- **LL Net $\Rightarrow$ INA:** Controls the behaviour of the converter from the low-level net (Petri box) in PEP format into the INA format. The dialog box shown in figure 5.27 on page 43 is displayed.
- **LL Net $\Rightarrow$ SPIN:** Controls the behaviour of the generator which converts a low-level net (Petri box) to a PROMELA representation. The dialog box shown in figure 5.28 on page 43 is displayed.
- **PFA $\Rightarrow$ HL Net:** Controls the behaviour of the compiler from the PFA (parallel finite automaton) into a high-level net (M-net). The dialog box shown in figure 5.29 on page 44 is displayed.
- **PFA $\Rightarrow$ B(PN)^2:** Controls the behaviour of the compiler from the PFA (parallel finite automaton) into a B(PN)^2 program. The dialog box shown in figure 5.30 on page **??** is displayed.

- ◦ **SDL ⇒ HL Net:** Controls the behaviour of the compiler from the SDL specification into a high-level net (M-net). The dialog box shown in figure 5.31 on page 45 is displayed.

▹ **Debug Output**

- ◦ **Show prjx:** Writes information concerning the state of the project to the shell. (E.g. the current relationship between the objects, the information whether or not files exist, or editors are opened, some information about the communication between the different parts of the tool, the sizes and the dates of the files, etc.)

• Help

▹ **Help Viewer:** Invokes the online help system (shown in figure 5.12) for the whole PEP tool.

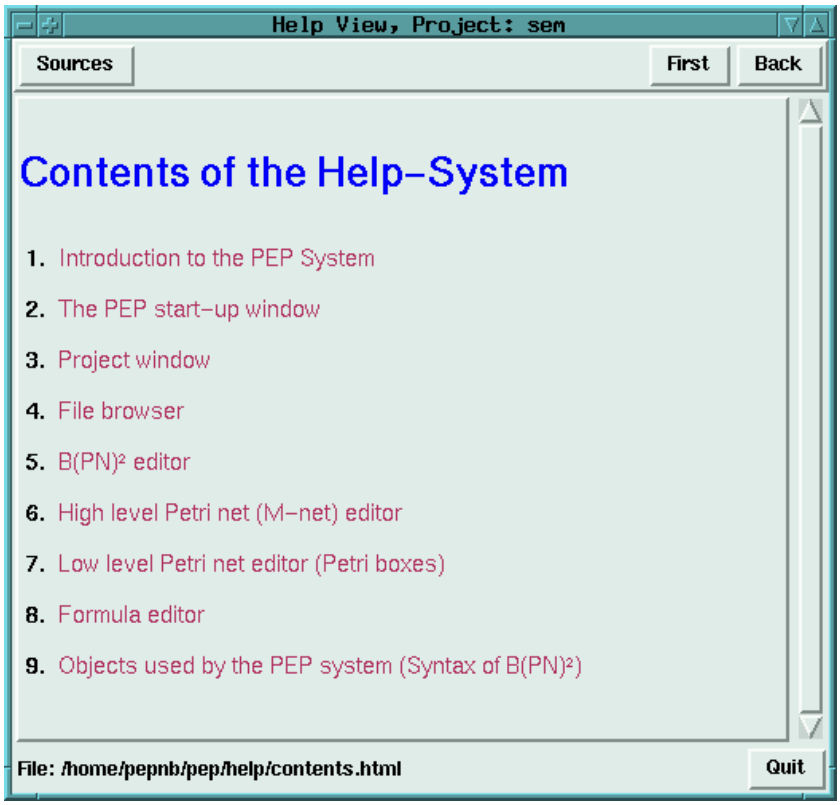


Figure 5.12: Online help system of the PEP tool

The help system is based on HTML pages. It can be used in a similar way as netscape. If a text is highlighted in red the corresponding page is opened by a click with the left mouse button. The name of the file corresponding to the current page is displayed in the bottom of the window. You can go back to the previous page (until you reach the first one) using the **Back** icon. You can go back to the first page using the **First** icon. If you would like to load a page by using a browser you can click on the **Source** icon. The help viewer is closed either if you click

on the **Quit** button or if the project is closed.
Note: Currently the online help system is unfortunately not up-to-date. Thus, we recommend to have a look at the documentation or to follow the intuition.

- **Cosmetics** — The items in this menu provides the user with the possibility to adapt the appearance of the project window to a particular project.
  - ▷ **Edit** — By checking this option the main canvas of the project window becomes editable. The icons may then be moved around by clicking on them with the left mouse button, holding the mouse button and moving the mouse. Moreover, start-, middle, and end-points of the arrows may be moved in the following way: First, an arrow is selected with the left mouse button making the three points visible. Then, the second click (keeping the left mouse button pushed) selects one of these points and moves it. Unchecking the option finally re-enters the usual mode of the PEP tool.
  - ▷ **Defaults**
    - ○ **Load PEP Default:** A PEP default view (displaying all objects and all compilation ways) may be loaded.
    - ○ **Load User Default:** A previously stored user default view (e.g., adapted to a project which starts at the SDL level) may be loaded.
    - ○ **Save as User Default:** A view which has been arranged in such a way that it is well-suited for a number of projects may be stored as a user default view. Note: We are working on providing a collection of default views.
  - ▷ **Show Icons:** In this list the objects (and also model checker or analysis algorithms) can be chosen which are displayed in the project window.
  - ▷ **Show References:** In this list the compilers are selected which are displayed in the project window. Note: If a certain compiler is not selectable, you should check whether the input formalism as well as the output formalism is currently selected for display.

5.3 Explanation of the options

The majority of the options are meant for debugging purposes or for advanced users who are familiar with the algorithms. For normal work it should be sufficient to use the defaults.

Note: For the time being, selecting **Cancel** does not restore the values of switches, but stores them.

5.3.1 Syntax check

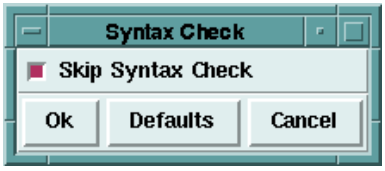


Figure 5.13: Options for the syntax check

- The user can choose to have the syntax of a B(PN)^2 program checked before it is compiled into a PBC term or into an M-net. The compiler into M-nets is able to handle some small extensions that are not accepted by the syntax checker or the other compiler. These extensions will be documented in a later release.

5.3.2 B(PN)^2⇒PBC

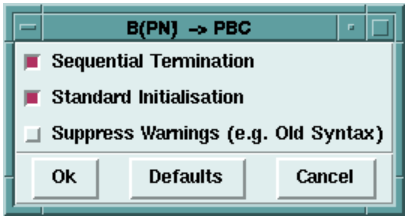


Figure 5.14: Options for the compiler B(PN)^2 ⇒ PBC

- Originally there had been an exit transition for every possible combination of variable bindings. Thus a possibly exponential amount of transitions had been created. It is more space efficient to terminate the variables of a block sequentially.
- The user can specify that variables are initialised with the smallest possible value (resp. *false* for Boolean variables).
- If one writes a program using old syntax, warnings can be suppressed. This option is only for compatibility with older versions of the PEP tool.

5.3.3 PBC⇒LL Net

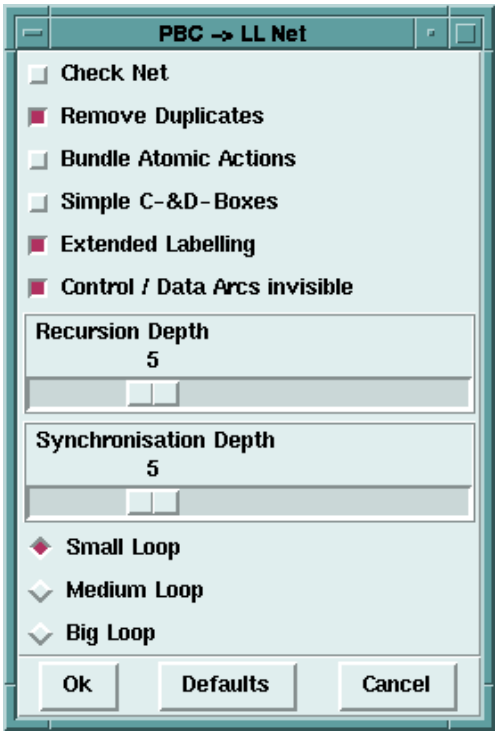


Figure 5.15: Options for the compiler PBC ⇒ LL-net

- The net can be checked for consistency. This is meant only for debugging purposes (e.g. memory allocation is checked).
- Place and transition duplicates (identical pre- and post-sets and meanings) can be removed automatically.
- Transitions corresponding to the same action or the same parallel composition of actions can be displayed at the same position on the screen.
- The user can specify that simple boxes are generated for plain and channel variables, otherwise the initialisation transitions are synchronised with transitions of the control flow which yields nets with a smaller prefix.
- Extended labellings can be generated. A transition corresponding to the action $\langle\ a' = 'a - 1\ \rangle$ will have the meaning $a' = 'a - 1$ which can be displayed optionally in the net editors using **Option▷Display Meanings**.
- Arcs between the data boxes of the net and the control flow part of the net can be specified as invisible to enhance readability of the nets constructed from a program. Invisible arcs can be displayed optionally in the net editor using **Option▷Show Invisible Arcs** (see pages 80 and 103).
- The depth of recursion must be restricted to make sure that the LL nets are finite. Note: The slider can be moved even when **Recursion Depth** is unchecked, but then it has no effect.
- The maximum number of transitions which can be synchronised by one synchronisation should be restricted to make sure that the LL nets are finite.

Note: The slider can be moved even when **Synchronisation Depth** is unchecked, but then it has no effect.

- Different loop semantics are implemented in the compiler. All of them behave in an equivalent way. For larger examples the *small* loop should be chosen for efficiency reasons when calculating the finite prefix of the branching process (or performing other analyses). The *medium* and *large* versions are meant to support future theoretical work.

5.3.4 B(PN)^2⇒M-Net

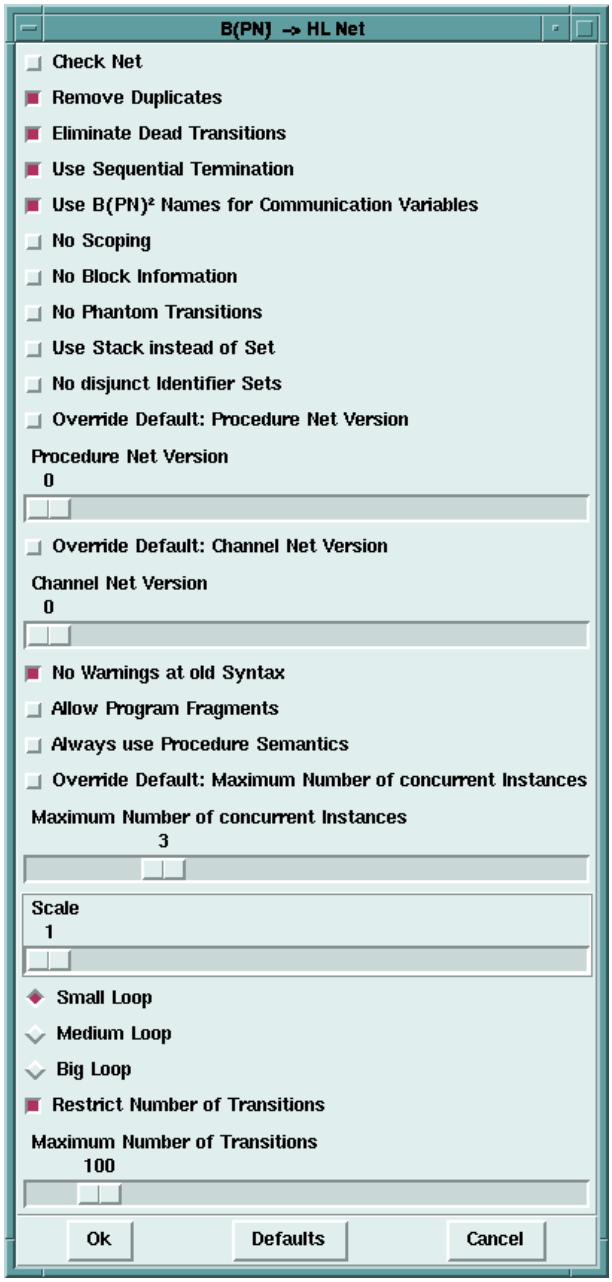


Figure 5.16: Options for the compiler B(PN)^2 ⇒ M-net

- The net can be checked for consistency. This is meant only for debugging purposes.

- Place and transition duplicates can be removed automatically.
- Some transitions can be recognised as initially dead and can be removed automatically.
- Originally there had been an exit transition for every possible combination of variable bindings. Thus a possibly exponential amount of transitions had been created. It is more space efficient to terminate the variables of a block in a sequential way.
- The names of the variables in the B(PN)^2 program can be used for the communication variables in the arc inscriptions. This yields M-nets that are easier to understand, otherwise the communication variables ends with e.g. `@14`.
- If the final scoping (for the outermost block as well as for the inner ones) is omitted then the data boxes and the control flow can be analysed separately. However, it is not possible to simulate such nets.
- Per default, a hierarchical M-net is constructed containing one block for each block of the B(PN)^2 program. The calculation of the hierarchie information may be omitted.
- Per default, the calculation of the M-net yields, in the case of a communication via a channel with capacity zero between two atomic actions, one synchronisation transition (the usual synchronisation result) and two phantom transitions (like virtual transitions representing the original transitions before synchronisation). The calculation of these phantom transitions which are important to enhance visualisation in the net editors may be omitted.
- If procedures are involved, per default, procedure nets which use a stack like structure to store procedure instance identifiers are used. Optionally, a set like structure is provided.
- Per default, the identifier sets of different procedures are made disjoint. This may be changed.
- The compiler supports three different procedure net versions. In order to change the default, the **Override Default** option has to be checked and a procedure net version has to be selected.
- The compiler supports three different channel net versions. In order to change the default, the **Override Default** option has to be checked and a channel net version has to be selected.
- Warnings for the use of an older syntax may be omitted.
- The compiler also supports the translation of program fragments.
- Per default, the compiler only uses the procedure semantics (passing coloured control flow tokens instead of 'only' black tokens) if procedures are used. This may be changed.
- The maximum number of concurrent instances (which is specified in a procedure declaration of a B(PN)^2 program by the **_max_** parameter may be overridden by checking the **Override Default** option and choosing additional restrictions. This may be useful for tests.
- The graphical distances between the elements of the generated net can be changed.
- Different loop semantics are implemented in the compiler. All of them behave in an equivalent way. For larger examples the *small* loop should be chosen for efficiency reasons when calculating the finite prefix of the branching process. The *medium* and *large* versions are meant to support future theoretical work.

- Depending on the power of the machine it might be useful to restrict the maximum number of generated transitions. Otherwise it is possible that one has to wait a while for the end of this calculation or a later one. If the maximum number is reached the compilation is stopped and no net is created. At the present time, no mechanism to stop the execution of an algorithm is implemented. Thus a **kill** command (in the shell) is the only way to stop execution, unless this option is set.
  Note: The slider can be moved even when **Restrict Number of Transitions** is unchecked, but then it has no effect.

Note: Arcs between the data boxes of the net and the control flow part of the net are specified (automatically) as invisible to enhance readability of the nets constructed from a program. Invisible arcs can be displayed optionally in the net editor using **Option▷Show Invisible Arcs** (see pages 80 and 103).

5.3.5 M-Net⇒Petri box

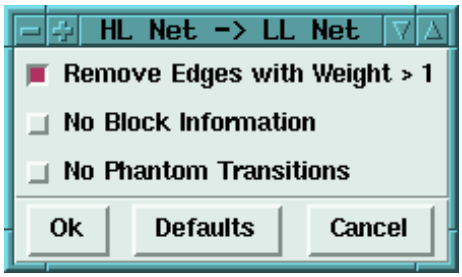


Figure 5.17: Options for the compiler M-net ⇒ Petri box

- Some transitions can be recognised as initially dead and can be removed automatically.
- Per default, a hierarchical M-net is unfolded into a hierarchical Petri box (unfolding blocks of the high-level net into blocks of the low-level net). The unfolding of the hierarchie information may be omitted.
- Per default, the phantom transitions (either created by the compiler from B(PN)^2 into M-nets of entered directly in the high-level net editor) are unfolded into low-level phantom transitions. The unfolding may optionally be restricted to non-phantom transitions.

5.3.6 Sync HL Net

- The net can be checked for consistency. This is meant only for debugging purposes.
- Place and transition duplicates can be removed automatically.
- Some transitions can be recognised as initially dead and can be removed automatically.
- The synchronisation algorithm has a mode which avoids renamming of variables (e.g. by appending `@14`) as far as possible. This mode is also used (by default) in the compiler from a B(PN)^2 program (or an SDL specification) into an M-net. As a consequence, the names of the variables (e.g. in the B(PN)^2 program) are still used for the communication variables in the arc inscriptions. This yields M-nets that are easier to understand.

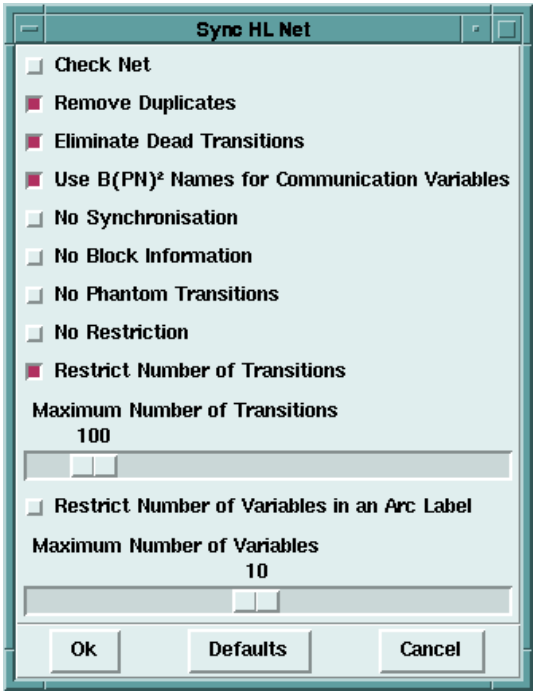


Figure 5.18: Options for the high-level net synchroniser (and 'restricter')

- Per default, synchronisation and restriction are performed. However, it is possible to (optionally) omit synchronisation.
- Per default, if a hierarchical M-net is synchronised this yields again a hierarchical M-net (block information is calculated in the same way as during the construction of the M-net semantics of a B(PN)^2 program or an SDL specification). This forwarding of the hierarchie information may be omitted.
- Per default, the synchronisation of an M-net yields, in the case of a communication via a channel with capacity zero between two transitions, one synchronisation transition (the usual synchronisation result) and two phantom transitions (like virtual transitions representing the original transitions before synchronisation). The calculation of these phantom transitions, which are important to enhance visualisation in the net editors, may be omitted.
- Per default, synchronisation and restriction are performed. However, it is possible to (also optionally) omit restriction.
- Depending on the power of the machine it might be useful to restrict the maximum number of generated transitions. Otherwise it is possible that one has to wait a while for the end of this calculation or a later one. If the maximum number is reached the synchronisation is stopped and no net is created. At the present time, no mechanism to stop the execution of an algorithm is implemented. Thus a **kill** command (in the shell) is the only way to stop execution, unless this option is set.
  Note: The slider can be moved even when **Restrict Number of Transitions** is unchecked, but then it has no effect.
- The maximum number of variables which occur in an arc inscription may be restricted by checking the **Override Default** option and choosing a maximum value. This may be useful for tests.

5.3.7 Prefix Builder

The prefix builder has several options that may increase the speed of the calculations. These options are meant for the developer. Usually the default values are fine. Note that changing critical options may result in incorrect prefices.

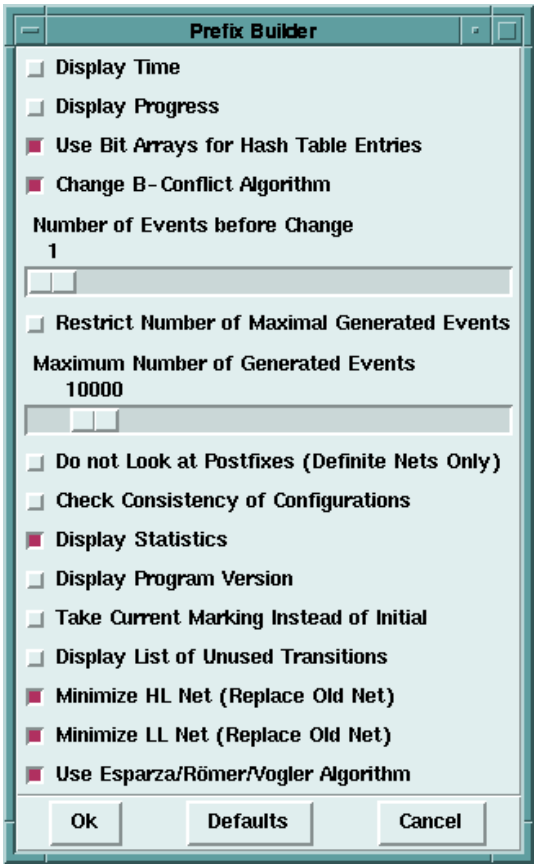


Figure 5.19: Options for the prefix builder

- The time used for the calculation can be written to the shell.
- The progress of the calculation can be written to the shell. This may be useful for large nets.
- Bit arrays can be used internally for hash table entries. This option often increases the speed of the calculation.
- The B-conflict algorithm can be changed. In addition to that, it is possible to change it after a certain number of events. This option often increases the speed of the calculation.
  Note: The slider can be moved even when **Number of Events before Change** is unchecked, but then it has no effect.
- Depending on the power of the machine it might be useful to restrict the maximum number of generated events. (You can set the **Display Progress** option to observe it). However there is no direct correspondence between the number of generated events and the duration of the calculation. At the present time, no mechanism to stop the

execution of an algorithm is implemented. Thus a **kill** command (in the shell) is the only way to stop execution, unless this option is set.
Note: The slider can be moved even when **Maximum Number of Generated Events** is unchecked, but then it has no effect.

- For some nets it makes no difference whether postfixes are considered. This option may increase the speed of the calculation. But be careful, it may result in wrong finite prefices of the branching process! This option is meant only for the developer of the prefix builder.
- The consistency of the calculated configurations can be checked. This option is meant only for debugging purposes.
- The results of the calculations can be written to the shell. Sizes of the original net and the prefix are given. The time used is output.
- The version number of the prefix builder can be written to the shell.
- The current marking can be used for the calculation of the prefix instead of the initial marking. The size of the prefix can be reduced if the net is simulated up to a certain state and this marking is then taken as input for the prefix builder.
- The unused transitions (that is, the initially dead ones that can never occur) can be written to the shell.
- The unused transitions (that is, the initially dead ones that can never occur) can be deleted together with their surrounding arcs. This works not only for the low-level net, but as well for the high-level net (a high-level transition is removed if all low-level transition which resulted from the unfolding of this high-level transition are initially dead). Be careful, the original nets are replaced by the reduced nets!
- The Esparza/Römer/Vogler algorithm is more efficient than the original one and yields minimal prefices.

5.3.8 Boundedness Analyser

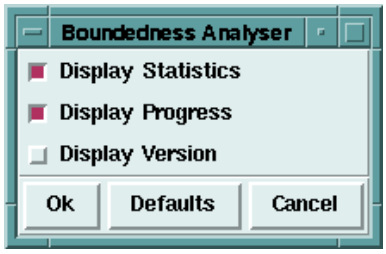


Figure 5.20: Options for the boundedness analyser

- The results of the calculations can be written to the shell. The number of reachable markings and the number of arcs are given.
- Progress information can be written to the shell. This can be useful for large nets.
- The version number of the boundedness analyser can be written to the shell.

5.3.9 Reachability analyser

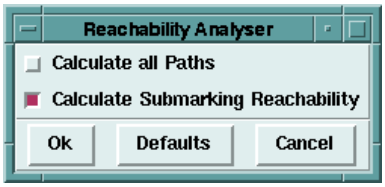


Figure 5.21: Options for the reachability analyser

- Either all paths or a shortest one can be calculated.
- It can be specified that the algorithm should search for a marking of which the given marking is a submarking.

Note: If you check both otions, it is possible that hundreds of paths are calculated and displayed.

5.3.10 LP analyser

The options of the algorithms which are based on linear programming (the reachability analyser and the deadlock-freeness checker) are divided into four sections:

1. Within the first section the general method is specified. The user may choose whether trap and S-invariant properties are considered, or only trap properties, or only S-invariant properties are taken into consideration.
2. The second section comprises a number of general debug options (which require more knowledge of the involved algorithms and underlying linear programming packages).
   - If the CPLEX package is installed it is recommended to use this instead of the lp_solve package which is distributed with the PEP tool.
   - It may be chosen to generate (and solve) a non integer problem.
   - Per default, the Alaiwan/Toudic algorithm is used for traps. This may be changed to the Espeleta/Couvreur method.
   - The default for the scaling factor for trap equations (the coefficient of the objective function) may be overridden checking the **Specify Scaling Factor for trap equations** option.
   - The generated linear programming problem may (optionally) be saved in the **LP** format. This may be useful for later analyses.
   - The generated linear programming problem may (optionally) be saved in the **MPS** format. This may (again) be useful for later analyses.
   - If the generated linear programming problem is saved in one of the previously mentioned formats (**LP** and/or **MPS**) it may make sense to omit the invocation of the LP solver.
   - Optionally, additional debug information may be written to the shell.

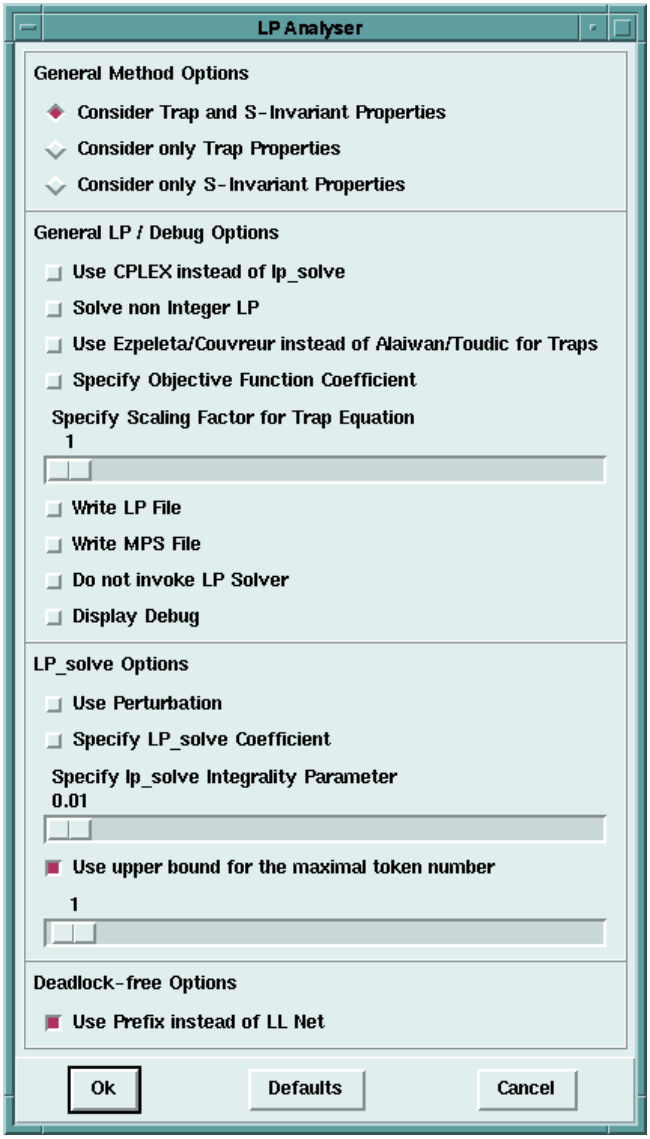


Figure 5.22: Options for the LP analyser

3. The third section only deals with options for the lp_solve package (i.e., they are ignored if the CPLEX package is chosen).

   - Optionally perturbation may be used.
   - The default for the lp_solve integrality parameter may be overridden checking the **Specify lp_solve coefficient** option.
   - A supplementary upper bound for the maximal number of tokens on a place may be specified checking the **Use upper bound for the maximal token number** option. This is particularly useful for Petri boxes which have been automatically generated from B(PN)^2 programs or SDL specifications (because these are 1-safe).

4. The fourth (and last) section comprises one option which only applies to the deadlock-freeness checker. The user may specify whether the analysis is based on the prefix rather than on the low-level net. While the check which takes the low-level net directly as input is a semi decision algorithm, the prefix based analyser yields a 'yes' or 'no'.

5.3.11 Model Checker (Safe)

- The model checker needs certain isomorphisms for the verification. They can be written to a file. It can be chosen whether the model checker decides if it is necessary to calculate them (because they do not yet exist, or because the file containing the prefix is newer than the files containing the isomorphisms) or if they are calculated anyway.
- In many cases, a firing sequences may give a proof (e.g. an example how a certain marking may be reached) for the result of the model checking. If this option is selected even transition sequences for sub formulae (of the normal form of the given formula) are displayed. The PEP tool allows the simulation of these sequences in the low-level net editor (and the B(PN)^2 or SDL editor depending on the established references).
- Either all transition sequences (which might be thousands) or just the one belonging to the shortest maximal configuration (which is not necessarily the minimal sequence) may be displayed.
- The result of the model checking may be automatically added to the documentation file of the project in order to support later evaluations.

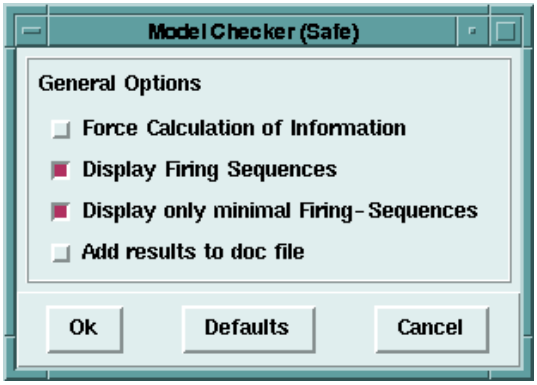


Figure 5.23: Options for the model checking algorithm for safe Petri nets

5.3.12 Model Checker (SMV)

- The compiler from a low-level Petri net into the SMV formalism supports a number of different translation methods. We highly recommend to use the fourth or fifth method, where the fifth method performs better for almost all examples. Methods one, two, three, six and seven are supported for experimental use. They may yield wrong verification results.
- Verification may usually be speeded up by using additional S-invariant information. The first two methods may be used with safe Petri nets of all kind because (non overlapping or overlapping) place invariants are calculated by an additional program. If the Petri box was generated automatically from a B(PN)^2 program (via an HL net) or from an SDL specification we recommend to use B(PN)^2 or SDL related S-invariants which are provided (as a side effect) by the compilers.
- The result of the model checking may be automatically added to the documentation file of the project in order to support later evaluations.

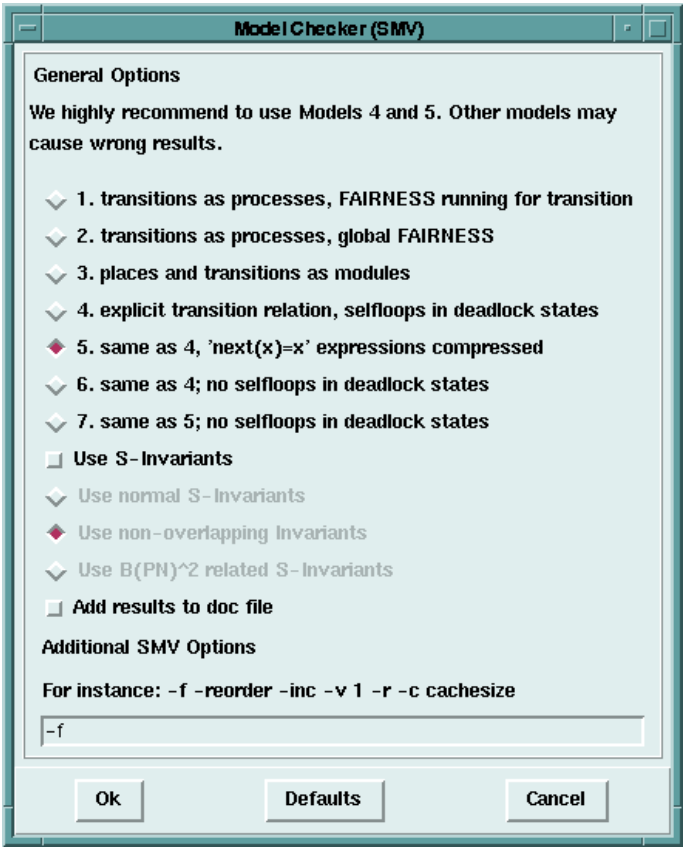


Figure 5.24: Options for the SMV model checker

- Finally, additional SMV options which are passed (as such) to the SMV verifier may be given. We refer to the SMV manual for the details.

5.3.13 Model Checker (SPIN)

For the integrated Spin verifier we provide exactly the same options as Xspin (the graphical interface of the Spin verifier) supports. Thus, in case of doubt, we refer to the documentation of the Spin package.
Note: Some options are only provided for consistency with the Xspin tool. In our particular case (of automatically generated PROMELA code which is for the time being not meant to be edited by the user of PEP) they do not make sense or do not influence the verification.

- In the upper part of the options window, a number of restrictions and additional compile-time or run-time options may be given:
  - The maximum of the memory usage may be specified. A click on the **Explain** button displays additional explanation.
  - An estimation of the state space size may be given. A click on the **Explain** button (again) displays additional explanation.
  - A maximum search depth may (additionally) restrict the verification. A click on the **Explain** button (again) displays additional explanation.
  - A chooser allows to add extra compile-time directives.
  - A chooser allows to add extra run-time options.

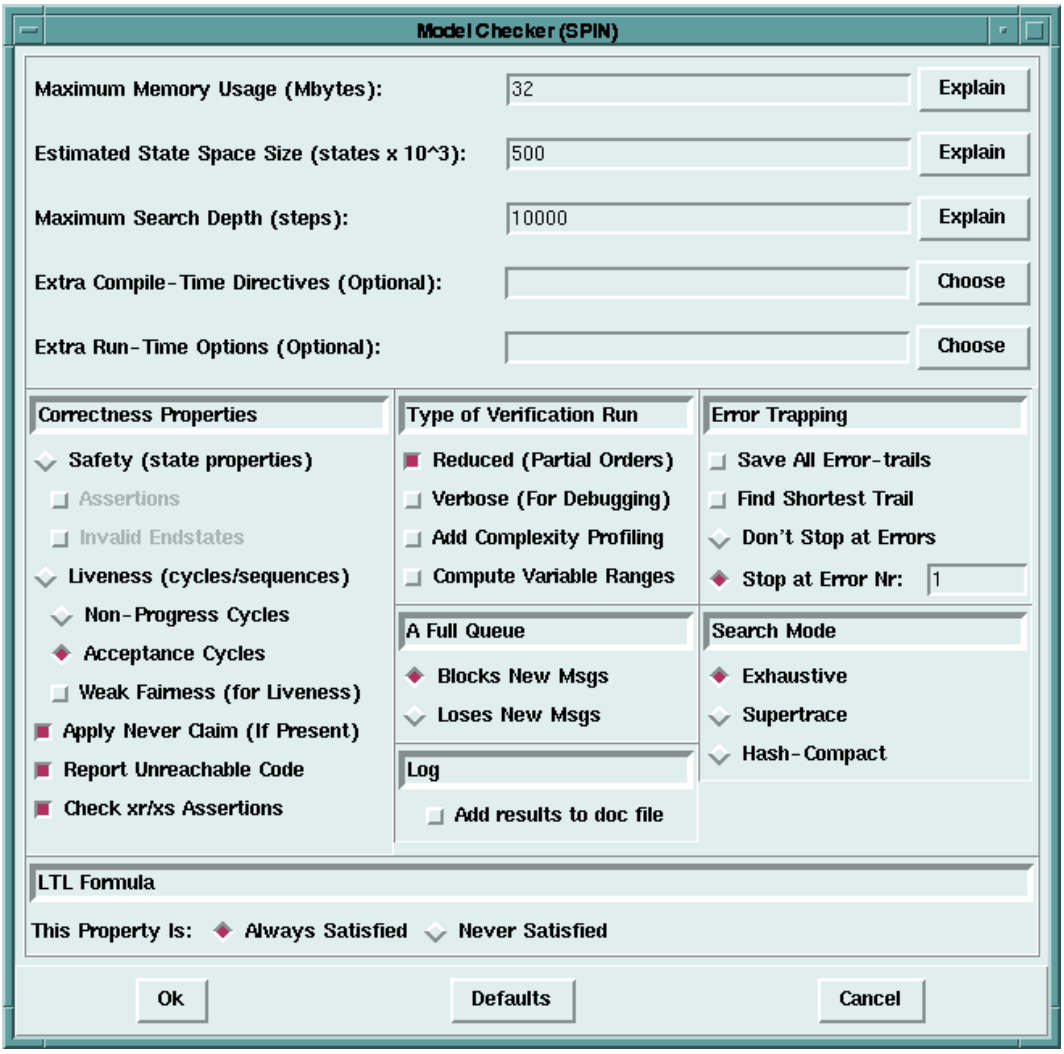


Figure 5.25: Options for the SPIN model checker

- In the lower left part of the window correctness properties (which are then checked by the Spin verifier) are specified:
  - Safety (i.e. state) properties (assertions or invalid endstates) may be checked. For the time being the automatically generated PROMELA code does not use these features.
  - Liveness (i.e. cycles/sequences) properties (non-progress or acceptance cycles) may be checked. Optionally weak fairness may apply. For the time being the automatically generated PROMELA code does not use these features.
  - The Spin verifier is integrated in such a way that the formula (edited in the formula editor) is transparently tranformed into a never claim which is then supposed to be combined with the PROMELA representation of the low-level net. Thus, the **Apply Never Claim** option should definitely be checked.
  - The Spin verifier offers the additional feature to report unreachable code. This may provide additional insight.
  - **xr/xs** assertions may (optionally) be checked. For the time being the automatically generated PROMELA code does not use these features.

- The type of the verification run may be specified by four options:
  - The search algorithm (used in the Spin verifier) may use partial order reductions. Unfortunately, this (for the time being) does not give any speed up for the automatically generated PROMELA code.
  - The Spin verifier may (optionally) produce a lot more output (mainly for debugging purposes).
  - Complexity profiling may be added to the produced executable C code. This is again only relevant for debugging purposes.
  - Computing variable ranges may provide interesting information for the low-level net under consideration.
- The choice whether or not a full queue blocks a message or not is not relevant for the automatically generated PROMELA code.
- The result of the model checking may be automatically added to the documentation file of the project in order to support later evaluations.
- Error trapping is influenced by two options and one choice:
  - The user may choose to have all error-trails saved (and not just at most one).
  - The Spin verifier provides an options which tries (but not necessarily succeeds) to find a shortest error-trail.
  - The verification may either stop after a certain number of errors (i.e. counter examples) have been found or exploit the whole state space (taking additionally given restrictions into account).
- There are three choices for the search mode.
  1. An exhaustive search guarantees (if it terminates successfully) the correctness of the result, but may be infeasible for larger systems.
  2. A supertrace method may overcome memory restriction, but on the other hand may yield wrong results (a potential incorrectness is however reported by PEP).
  3. The hash-compact method provides an alternative approach to overcome memory restrictions.
- The lowest choice chooses whether the property given in the formula file should always or never be satisfied.

5.3.14 B(PN)^2 expander

- The user can specify that variables are initialised with the smallest possible value (resp. *false* for Boolean variables).
- The maximal nesting depth (resulting from recursions and blocks) must be restricted to make sure that the resulting program (and thus the resulting nets) is finite.

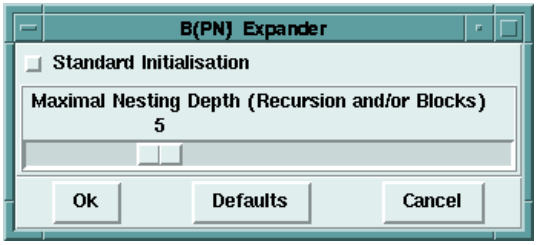


Figure 5.26: Options for the B(PN)^2 expander

5.3.15 Petri box⇒INA

- The converter may generate reference information (relating INA identifiers to, e.g., names of transitions of the low-level net in the PEP format). These references are, for instance, the basis for a highlighting of transition invariants in a net editor.
- Optionally, a coloured net representation in INA format (instead of a low-level representation) may be generated.

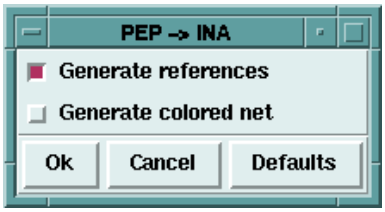


Figure 5.27: Options for the compiler Petri box ⇒ INA

5.3.16 Petri box⇒PROMELA

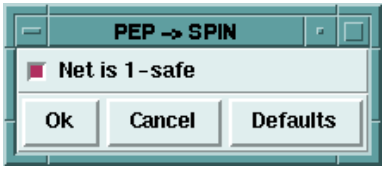


Figure 5.28: Options for the compiler Petri box ⇒ PROMELA

- The converter per default assumes that the net is 1-safe. Optionally, a 255-boundedness may underly the conversion.

5.3.17 PFA⇒M-net

- If the final scoping (for the outermost block as well as for the inner ones) is omitted then the data boxes and the control flow can be analysed separately. However, it is not possible to simulate such nets.
- Per default, a hierarchical M-net is constructed containing one block for each automaton of the PFA. The calculation of the hierarchie information may be omitted.

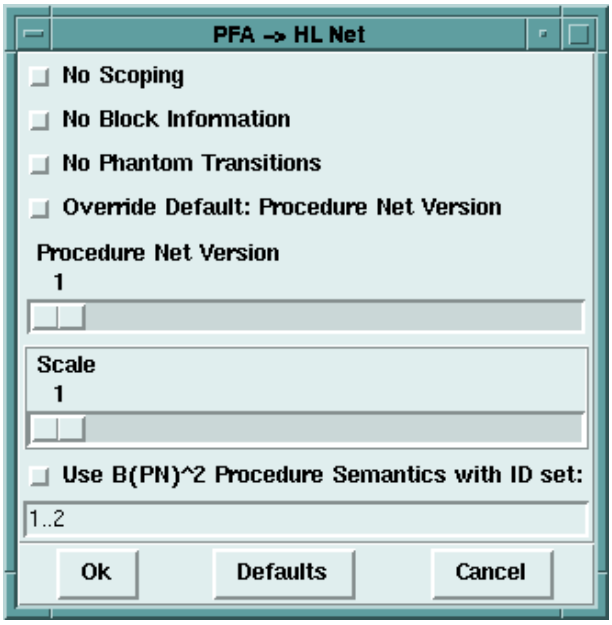


Figure 5.29: Options for the compiler PFA ⇒ M-net

- Per default, the calculation of the M-net yields, in the case of a communication via a channel with capacity zero between two edges (atomic actions), one synchronisation transition (the usual synchronisation result) and two phantom transitions (like virtual transitions representing the original transitions before synchronisation). The calculation of these phantom transitions which are important to enhance visualisation in the net editors may be omitted.
- The compiler supports three different procedure net versions. In order to change the default, the **Override Default** option has to be checked and a procedure net version has to be selected.
- Per default, the compiler does not uses the procedure semantics (passing coloured control flow tokens instead of 'only' black tokens). This may be changed checking the **Use B(PN)² Procedure Semantics** option and specifying a procedure instance identifier set.
- The graphical distances between the elements of the generated net can be changed.

5.3.18 PFA⇒B(PN)²

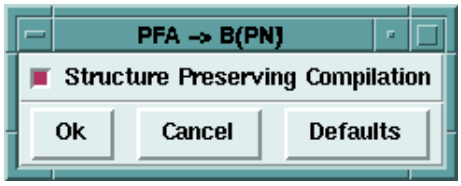


Figure 5.30: Options for the compiler PFA ⇒ B(PN)2

- The compiler comprises two different algorithms for the translation of a PFA into a B(PN)² program. By default, the structure of the automata is preserved (e.g., each

choice or loop is translated using the **do-od** construct). This translation method should be preferred. However, some PFA may not be translated this way. Thus, an optional translation method yielding one big **do-od** construct (for each automaton) comprising all possible alternatives (given by the edges) may be chosen.

5.3.19 SDL⇒M-net

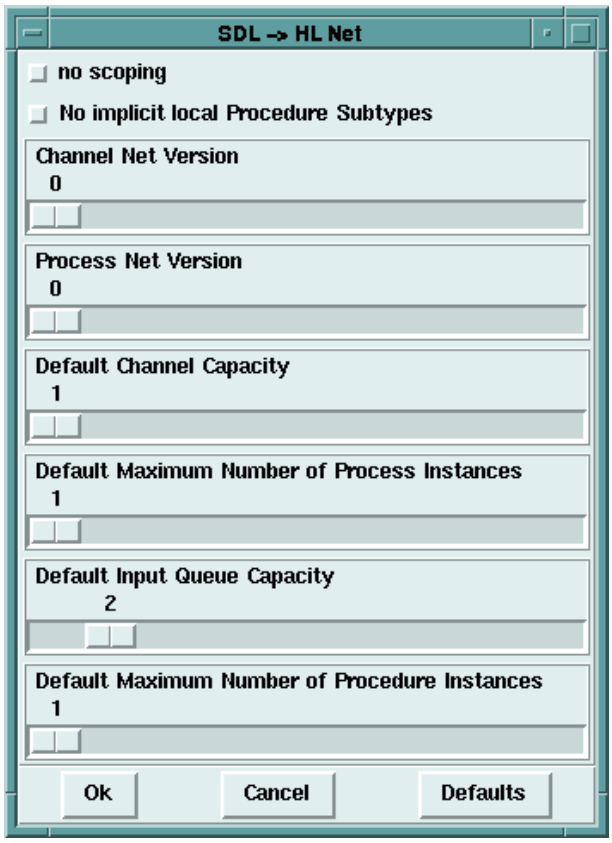


Figure 5.31: Options for the compiler SDL ⇒ M-net

- If the final scoping (for the outermost block as well as for the inner ones) is omitted then the data boxes and the control flow can be analysed separately. However, it is not possible to simulate such nets.
- An implicit generation of local procedure subtypes may be prohibited.
- The compiler supports two different channel net versions. In order to change the default, the **Override Default** option has to be checked and a channel net version has to be selected.
- The compiler supports four different process net versions. In order to change the default, the **Override Default** option has to be checked and a process net version has to be selected.
- The default capacity of channels (which may be overridden by restrictions specified in the SDL property file) is set by a slider.

- The default maximum number of concurrent process instances of a certain process type (which may be overridden by adding restrictions to the SDL property file) is (also) set by a slider.
- The default capacity of input queues (which may be overridden by adding restrictions to the SDL property file) is (also) set by a slider.
- The default maximum number of concurrent procedure instances of a certain procedure (which may be overridden by adding restrictions to the SDL property file) is (also) set by a slider.
- The maximum number of concurrent instances (which is specified in a procedure declaration of a B(PN)^2 program by the **_max_** parameter) may be overridden by checking the **Override Default** option and choosing an additional restriction. This may be useful for tests.

Chapter 6

The file browser

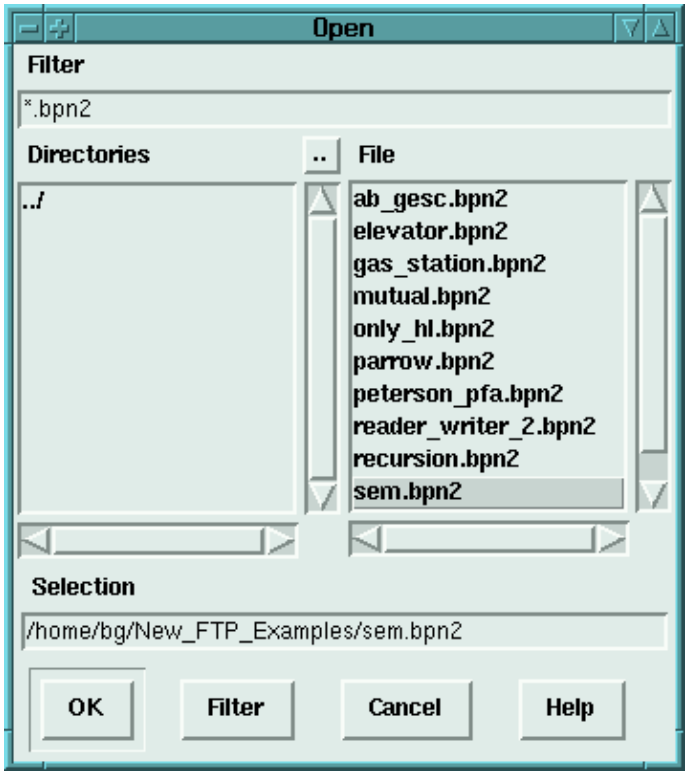


Figure 6.1: File browser

Every time a user wants to open a file, a file browser (cf. figure 6.1 is used for choosing it.

- In the left part of the window the directory hierarchy can be traversed by double-clicking on the directory names.
- In the right part the existing files in the current directory are displayed.
- The list can be restricted by a filter. For example all B(PN)^2 programs can be listed by selecting `*.bpn2` as a filter, provided you follow the naming conventions and do not name a B(PN)^2 program, say, `bill.txt`.
- The current selection is displayed at the bottom of the window.

- A file can be opened either by a double-click on its name or by a click on the **OK** button when the file is selected.
- The file browser can be closed by a click on the **Cancel** button.
- The **Help** button invokes a help page.

Chapter 7

The B(PN)^2 editor

The B(PN)^2 editor (cf. figure 7.1) is a text editor with some B(PN)^2 specific features.

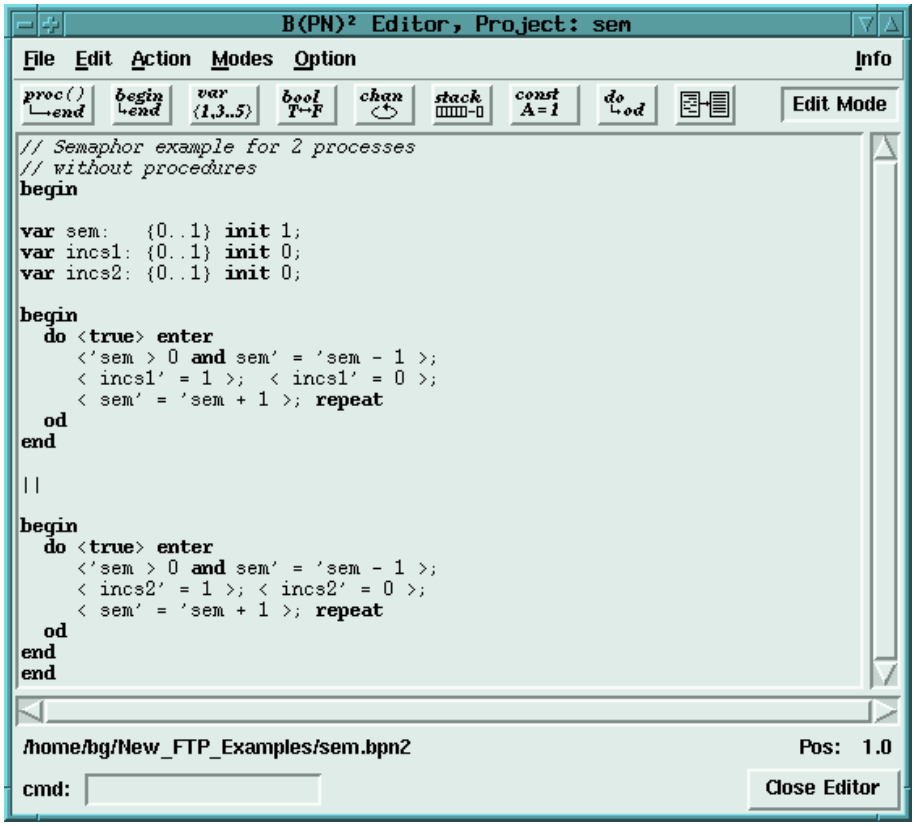


Figure 7.1: B(PN)^2 editor

7.1 Description of the B(PN)^2 editor window

- In the main part of the editor window a B(PN)^2 program can be edited.
- If the keyboard offers this possibility the **cut**, **copy** and **paste** keys can be used.
- Under the editing canvas of the editor the name of the currently edited file is displayed.
- The current cursor position is shown in the lower right part of the window.

- In the lower left part an additional area allows to type in commands (such as `w` to save the file) directly.
- All but the last icons underneath the pull-down menu are macros for typing part of a B(PN)² program.
  - The **proc()–end** button pastes the declaration of a procedure to the current cursor position.
  - The **begin–end** button pastes a *begin-end* block to the current cursor position.
  - The **var {1,3..5}** button pastes the declaration of a plain variable to the current cursor position.
  - The **bool** button pastes the declaration of a Boolean variable to the current cursor position.
  - The **chan** button pastes the declaration of a channel variable to the current cursor position.
  - The **stack** button pastes the declaration of a stack variable to the current cursor position.
  - The **const** button pastes the declaration of a constant to the current cursor position.
  - The **do–od** button pastes a *do-od* construct to the current cursor position.
- The right most icon invokes an algorithm to display keywords in boldface and comments in italic.

7.2 Menus of the B(PN)² editor

• File

▷ **Open:** Opens the file browser to select a B(PN)² program.

▷ **Reload:** Reloads the program (e.g. if you made a mistake and want to go back to the previous version).

▷ **Save:** Saves the contents of the B(PN)² editor to a file. The name of the file is displayed underneath the main part of the window.

▷ **Save as:** Opens the file browser to save the contents of the B(PN)² editor to a file.

▷ **Import File:** Opens the file browser to select the name of a file to be loaded into the B(PN)² editor at the current cursor position.

▷ **Export Clipboard:** Opens the file browser to select the name of a file. The contents of the clipboard are saved to this file. You can either use the **copy** button on your keyboard or the menu entry **Edit▷Copy** to copy a selection to the clipboard. Be careful, the clipboard is not local to the PEP tool, it can be changed e.g. if you hit the **copy** key on your keyboard even if you work with a totally different program!

Figure 7.2: Unsaved changes window of the B(PN)^2 editor

▷ **Quit:** Closes the B(PN)^2 editor. If there are unsaved changes a dialog box (see figure 7.2) asks whether or not these changes should be saved before the editor is closed.

- **Edit**

▷ **Cut:** Cuts the current selection (a part of the text marked using the left mouse button). It is saved to the clipboard and can be used for further **paste** or **export** operations.

▷ **Copy:** Copies the current selection to the clipboard.

▷ **Paste:** Pastes the contents of the clipboard to the current cursor position.

▷ **Search:** Opens the dialog box (shown in figure 7.3) that offers a comfortable searching facility described in the next paragraph.

▷ **Replace:** Opens the dialog box (shown in figure 7.4) that offers a comfortable search and replace facility described in the second next paragraph.

▷ **Add Comment:** Comments out the current selection by inserting **//** in front of each line of the selection.

▷ **Delete Comment:** Undoes a comment out of the current selection by deleting the **//** in front of each line of the selection.

7.2.1 Search

- The user can enter the search term.
- The direction of the search can be specified.
- The behaviour of the search algorithm is controlled by some options:
  - Either the search stops at the end of the text or it is continued at the beginning.
  - The search algorithm accepts regular expressions described on page 52.

Figure 7.3: Search window within the B(PN)² editor

7.2.2 Replace

- The user can enter a term that is to be replaced.
- The user can enter the term that will be inserted instead.
- The direction of the search can be chosen.
- The behaviour of the search algorithm is controlled by some options:
  - Either the search stops at the end of the text or it is continued at the beginning.
  - The search algorithm accepts regular expressions (described below).

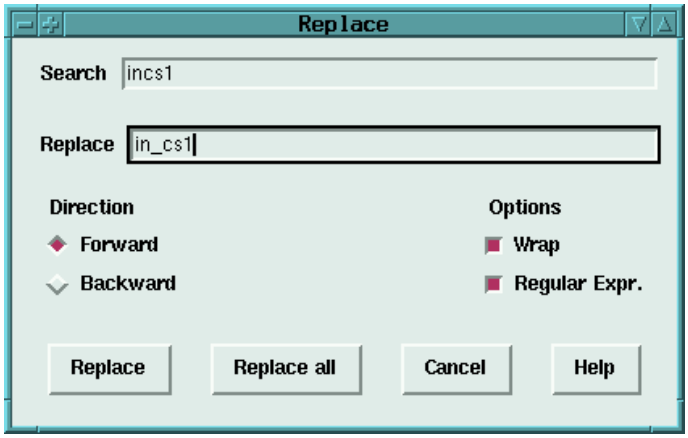


Figure 7.4: Replace window within the B(PN)² editor

7.2.3 Regular expressions

A powerful way to express a search string is by specifying a regular expression. It has a general pattern specification syntax including the ability to extract substrings from the matching string. A pattern is a sequence of a literal character, a matching character, a repetition clause, an alternation clause, or a sub-pattern grouped by parentheses. The syntax (which is not necessarily the most convenient one, but which is predetermined by Tcl/Tk) is summarised in table 7.1. If you want to search a string containing one of the characters in the left column of this table you have to preface it with a backslash.

.	Matches any character.
*	Matches zero or more characters. (Repetition operator)
+	Matches one or more characters. (Repetition operator)
?	Matches zero or one character.
( )	Groups a sub-pattern. Subsequent repetition operators or enclosing alternation operators apply to the whole sub-pattern.
|	Alternation operator.
[ ]	Delimits a set of characters. Ranges are specified as *x*–*y*. (E.g. [ae-g] searches for a, e, f or g). If the first character (after the opening bracket) in the set is ^, then there is a match if the remaining characters (within the brackets) are not present. (E.g. [^ae-g] searches for characters that are not equal to a, e, f or g).
^	Represents the beginning of a line. (Only when it is the first character of the search string.)
$	Represents the end of a line. (Only when it is the last character of the search string.)

Table 7.1: Regular expressions

- **Action**

  ▷ **B(PN)² Syntax Check:** Invokes an algorithm checking whether or not the B(PN)² program is syntactically correct. If you already edited the program after the last saving, a dialog box asks you whether the current version should be saved first or the check should be cancelled. If there is no syntax error a small box displays success. Otherwise the dialog box shown in figure 7.5 explains the errors and allows their correction. Each line gives a description of one error. A double-click on one of the lines positions the cursor of the B(PN)² editor at the position where the error occurred.

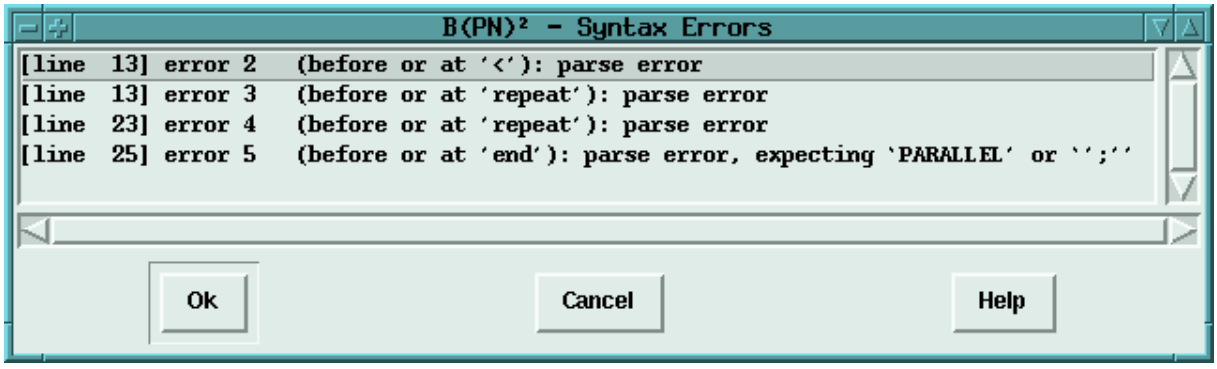


Figure 7.5: Syntax check window within the B(PN)² editor

▷ **B(PN)² To TEX:** Generates a TEX representation of the B(PN)² program (including boldface fonts for keywords, ...) and opens a file selection box which allows to save the TEX code in a file.

- **Modes** The B(PN)² editor may be used in three different modes:

  - ▷ **Enter Edit Mode:** The 'usual' editing mode which allows to enter (or change) a B(PN)² program is chosen. The icons which allow to paste templates (e.g. for a variable declaration) are only shown in this mode.

  - ▷ **Enter Reference Mode:** A reference mode (which is explained in more detail below) is entered. The general purpose of this mode is to highlight (or paste) counterparts of a B(PN)² selection (e.g., an atomic action) in another editor (e.g., the corresponding transition(s) in the high-level Petri net editor). The exact behaviour of this highlighting (or pasting) depends on the decisions taken in the **Options** menu.

  - ▷ **Enter Simulation Mode:** A simulation mode which allows the simulation of a low-level Petri net transition sequence on the B(PN)² level is entered. This is particularly interesting for the simulation of analysis and/or verification results.
    Note: The editor enters this mode automatically if the **Sim B(PN)²** button of one of the analysis/verification windows is pushed.

- **Option** The behaviour of the B(PN)² editor in the **Reference Mode** is controlled by a number of options:

  - ▷ **Show Reference in HL:** If this option is checked references are highlighted in the high-level net editor.

  - ▷ **Show Reference in LL:** If this option is checked references are highlighted in the low-level net editor.

  - ▷ **Show Reference in Formula:** If this option is checked references are pasted in the formula editor (at the current cursor position).

  - ▷ **Show Reference in PFA:** If this option is checked references are highlighted in the graphical editor for the parallel finite automata.
    Note: This option is disabled in the current version of PEP.

  - ▷ **Show Reference partial:** If this option is checked also counterparts (e.g., transitions) which are not only related to the current selection but also to other parts of the B(PN)² program are highlighted/pasted.

  - ▷ **Show Reference exact:** If this option is checked only counterparts (e.g., transitions) which are exactly related to the current selection (e.g., all – and no more than – the currently selected atomic actions) are highlighted/pasted.

▷ **Show all Values:** This option is relevant for place related references. If the option is checked the type of the places under consideration is not taken into account. Otherwise the value given in the **Value** field has to be matched. This is particularly interesting for places which hold the value of a variable.

▷ **Preferences:** The dialog box shown in figure 7.6 is opened which allows to specify the delay between two action occurrences in the simulation mode.

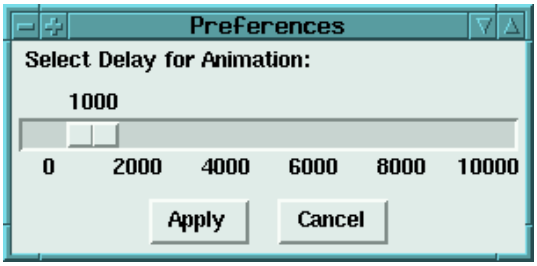


Figure 7.6: Preferences window of the B(PN)^2 editor

7.2.4 Reference Mode

As mentioned above, the B(PN)^2 editor has a reference mode. This offers a facility for determining, for instance, which place in a Petri box (or in the M-net) corresponds to a given point in the control flow (or a value of a certain variable) of the associated B(PN)^2 program, or which transition(s) in the Petri box correspond(s) to one or more action(s) in the program.

When the B(PN)^2 editor is in the reference mode the parts of the program for which references exist (i.e. those parts of the program which have counter parts on the net level) are coloured in blue each time the mouse cursor is positioned on them. An atomic action may then be selected in three different ways:

1. Selecting the atomic action with the middle mouse button asks for the transition(s) corresponding to the atomic action itself. In the B(PN)^2 editor the action becomes coloured in green in order to show the kind of selection.

2. A selection with the left mouse button refers to the place(s) which correspond(s) to the point in the control flow directly before (or at) the atomic action (this colours the action yellow).

3. Finally, a click with the right mouse button starts an ask for the place(s) which correspond(s) to the point in the control flow directly after the atomic action (this colours the action red, or orange if the action was already coloured yellow).

For the **begin** of a block almost the same rules apply as for atomic actions: left means before; and right means after.

For the time being, identifiers of variables, channels or stacks in a declaration may only be selected with the middle mouse button. Such a selection refers to the place(s) which, e.g., hold(s) the value of the variable.

Using this selection method selections (or multi-selections) may be constructed. Moreover, an entry of a value in the **Value** field (above the canvas of the editor) may restrict place related references (if the option **Show all Values** is not checked).

Once, the desired selection (or multi-selection) has been established a click on the **Send** button performs the highlight (or paste) operation. In order to prepare a new selection, a click on the **Clear** button restarts with an empty selection.

Likewise (using the option **Show Reference in Formula**), a temporal logic formulae can be made to refer to points of the control flow or actions (using macros like `LIVE`) of the program, even though technically speaking, the temporal logic refers only to places of a net.

Note: Currently there are two tricky points:

1. If a transition results from a synchronisation of two atomic actions (e.g., $\langle\ C!{=}1\ \rangle$ and $\langle\ C?{=}1\ \rangle$) this transition carries references to both of the actions and an additional reference to the channel C. As a consequence, in order to request this transition both actions and the channel declaration have to be selected if the **Show Reference exact** option is checked. Unfortunately, this is not sufficient for the time being, because a selection of a variable, channel or stack declaration yields references of a different kind. Thus, we recommend to use the **Show Reference partial** mode and to select only the two actions.

2. As mentioned above, an entry in the **Value** field restricts place related references (if the option **Show all Values** is not checked). Using this option appropriately requires deep knowledge of the underlying Petri net semantics. For instance, a variable declaration **var** $x : \{2\}$ may produce a place with type $\{2\}$ (if no procedures are involved), with type $\{\bullet\} \times \{2\}$ (if it is a global variable and procedures are involved), or with type $\{1\} \times \{2\}$ (if it is a local variable within a procedure with an instance identifier set $\{1\}$). As a consequence, either a value `2`, or a value `(2,dot)`, or a value `(2,1)` has to be requested, respectively.

Anyway, a good way to become familiar with references of all these kinds between a B(PN)² program and a net is to use the menu entry **Show Reference** (in combination with the option **Highlight Execution** in one of the net editors (see pages 77 and 101).

An additional purposes of the **Reference Mode** is to relate abstract names to atomic actions or variable, channel and stack declarations. These abstract names are the basis for relating a program simulation to a 3D visualisation using the new 3D component. Once you have a singleton selection (made with the middle mouse button) you can click on the **Set Name** button which opens the dialog window shown in figure 7.7 where an abstract name can be entered.

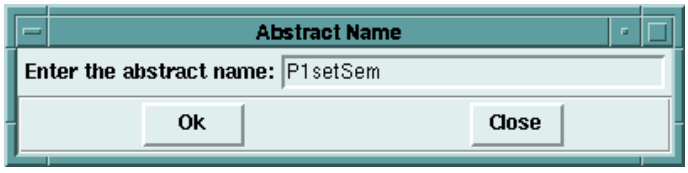


Figure 7.7: Set abstract name dialog window

• Info

▷ **Info:** Shows the about box for the text editors integrated in the PEP tool.
Note: It is worth doing it if you are an Enterprise fan.

Chapter 8

The high-level Petri net (M-net) editor

The high-level net editor (shown in figure 8.1) is a Petri net editor with an integrated simulator offering some M-net specific features.

8.1 Description of the high-level Petri net editor window

- The main part of the editor window can be used as a drawing canvas for creating and editing M-nets.
- The functionality of the menu bar is described in the next section.
- An additional icon bar provides a comfortable access to some of the functionality offered by the entries of the menu bar. We will give the details when we describe the corresponding menu items in the next section.
- The name of the currently edited M-net is displayed underneath the menu bar (and an additional icon bar) on the left hand side.
- The speed of a simulation may be changed using the slider in the middle underneath the menu bar.
- The current mode of the net editor is displayed underneath the menu bar on the right hand side.
- If the cursor is, for instance, positioned on one of the icons of the additional icon bar a brief information about the provided functionality is given in the area in the lower left corner under the canvas.
- If the editor is in *Edit Mode* the following can be done:
  - Places can be created by pressing the **p**-key at the desired position of the drawing canvas.
  - Transitions can be created by pressing the **t**-key at the desired position of the drawing canvas.
  - Arcs can be created by pressing the **a**-key first and selecting the place (transition) and the transition (place, resp.) with the left mouse button afterwards.
    Note: Clicking to quickly, may cause errors.

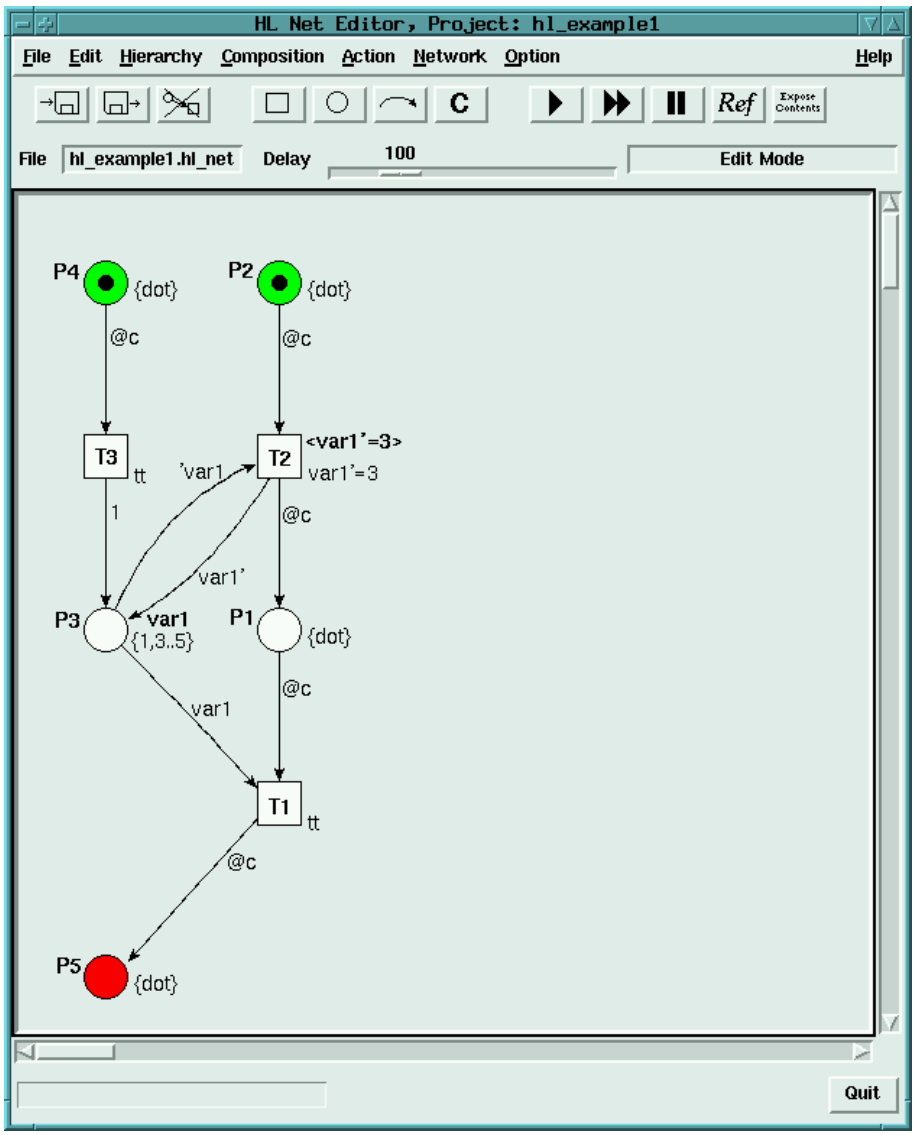


Figure 8.1: High-level net editor

- Comments can be created by pressing the **c**-key at the desired position of the drawing canvas. The dialog box shown in figure 8.2 is opened to edit the text.
- An object (place, transition, arc, block, box, comment or annotation) can be selected by a click with the left mouse button.
- A selected object can be moved by holding the left button down while moving the mouse. (For arcs, this moves the point you clicked at. You may turn such an arc into a straight one by selecting it with the left mouse button and clicking on the little red square, which appears, with the middle mouse button.)
- A multi-selection of places and transitions can be extended by a click on another place or transition while the **shift** key is held down.
  - A multi-selection can be deleted by pressing the **del** key.
  - A multi-selection can be moved by holding down the left mouse button (after its last extension) while moving the mouse.
- An area can be selected by clicking with the middle mouse button at the upper left corner of the area, holding the button down until the lower right corner of the

Figure 8.2: Comment dialog window

area is reached, and finally clicking with the middle mouse button somewhere in the area. (If you have a mouse with only two buttons you can probably emulate this by clicking both buttons at the same time).

* An area can be deleted by pressing the **del** key.
* An area can be moved by the left mouse button (after clicking somewhere in the area without hitting an object).

Note: In principle, general multi-selections containing objects of all kinds (including blocks or boxes) and areas are supported. However, the current version of the editor has still problems with combined multi-selections.

– The place dialog window shown in figure 8.3 can be opened by a double-click with the left mouse button on a place.

  Thus the name, the meaning, the set of references, the set of blocks (the place belongs to), the box (the place belongs to), the type, the initial as well as the current marking and the status of the place can be changed:

  * The name must be unique w.r.t. the other places of the net; this is checked by the editor.
  * If a net is compiled out of a B(PN)^2 program (or an SDL specification) the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding atomic action).
  * If a net is compiled out of a B(PN)^2 program (or an SDL specification) a set of references is automatically generated for some of the places. These references are used by some of the internal programs of the PEP tool. Although, the set of references may be edited, we highly recommend not to do this.
  * If a net is compiled out of a B(PN)^2 program which contains nested blocks, a place may belong to one or more hierarchical net objects also called blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical high-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
  * Boxes build a second hierarchy level. They are used for a 'programming with nets' approach. We recommend to have a look at the documentation of the

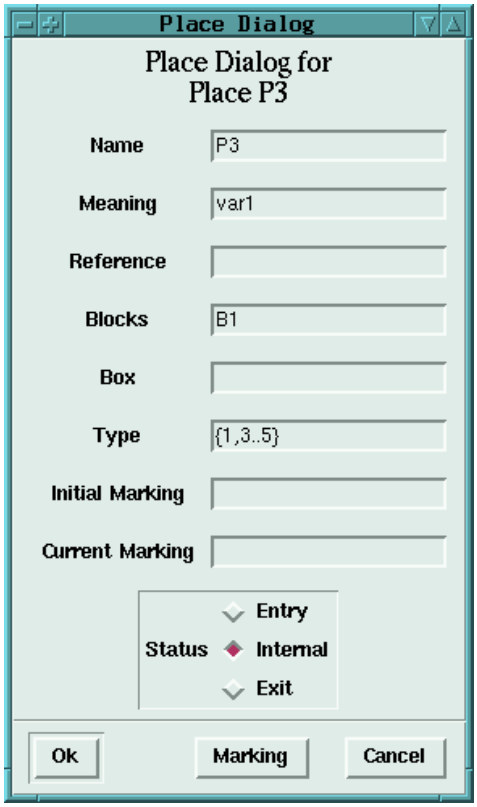


Figure 8.3: Place dialog window

'`programming_with_nets_example`' project in the PEP distribution in order to get a first intuition of this new method.

- Types can be given as, e.g., `{dot}`, `{tt,ff}` or `{1,3..5}`, with `tt` representing *true* and `ff` *false* respectively. Cartesian products are allowed, e.g. `{1..4} X {dot,1..2}`.
- A marking must be a multi-set over `Type`. E.g., `3*1,4` specifies a marking `1 1 1 4`. Tuples can be specified, e.g. `(1,dot),(2,tt,2),(3,1,dot,dot)`.
- The status of a place may either be **entry**, **internal** or **exit**.

Moreover, pressing the **Marking** button opens an additional window which gives the current marking of the place. This is particularly useful during the simulation of huge nets because it allows to trace the most interesting places.

– The transition dialog window shown in figure 8.4 can be opened by a double-click with the left mouse button on a transition.

Thus the name, the meaning, the set of references, the set of action terms, the value term, the set of synchronisation or phantom transitions which is associated with a phantom or synchronisation transition, respectively, the set of blocks (the transition belongs to), and the box (the transition belongs to) of the transition can be changed.

- The name must be unique w.r.t. the other transitions of the net; this is checked by the editor.
- If a net is compiled out of a B(PN)^2 program (or an SDL specification) the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding atomic action).

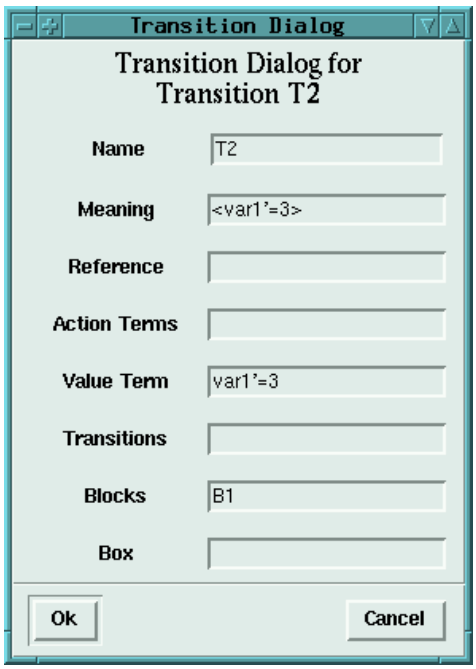


Figure 8.4: Transition dialog window

* If a net is compiled out of a B(PN)^2 program (or an SDL specification) a set of references is automatically generated for some of the transitions. These references are used by some of the internal programs of the PEP tool. Although, the set of references may be edited, we highly recommend not to do this.
* Action terms can be given as, e.g., `var1('var1,var1')`, `^var1('var1,1)`, `var1('var1,var1',idvar1)`, `chan1!(chan1!,dot)` or `^stack1?(stack1?)` (see **??** for explanation). You can get some hints if you compile a program into an HL net with the option **no scoping**.
* Value terms can be given as, e.g., `var1'=ff & var2'=3`. Sometimes it is necessary to specify (explicitly) the range of a variable by prefixing the value term with, e.g., `var1{1,3..5} & var2{tt,ff}:`. Projections of tuples (as used in earlier versions of PEP) are no longer used because we now provide a more comfortable notation using tuples directly as arc inscriptions. In case of doubt, you can get some hints if you look at the compilation result of a B(PN)^2 program (or SDL specification) into an HL net.
  Note: Within a value term a variable `var1'` is not at all related to a variable `var1` or `'var1`.
* For the translation of synchronous communication we now distinguish transitions of two kinds. The synchronisation of two atomic actions (e.g., $\langle\ C!{=}1\ \rangle$ and $\langle\ C?{=}1\ \rangle$) now yields one synchronisation transition (the usual synchronisation result which is a normal transition) and two phantom transitions (each of them visualises one of the original counter parts of the synchronisation). As a consequence, for such a synchronisation transition a list of corresponding phantom transitions which are associated with it may be specified (actually a transition is automatically coloured in cyan as soon as a list of phantom transitions is specified). Similarly, a list of synchronisation transitions may be given for a phantom transition.

  * If a net is compiled out of a B(PN)^2 program which contains nested blocks, a transition may belong to one or more hierarchical net objects also called blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical high-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
  * Boxes build a second hierarchy level. They are used for a 'programming with nets' approach. We recommend to have a look at the documentation of the '`programming_with_nets_example`' project in the PEP distribution in order to get a first intuition of this new method.

– The arc dialog window shown in figure 8.5 can be opened by a double-click with the left mouse button on an arc. Thus, the arc inscription and its visibility can be changed.

Figure 8.5: Arc dialog window

  * The unique name is generated automatically and cannot be changed.
  * A multi-set of (tuples of) variables and/or constants (such as `var1`, `dot`, `(var1,2)` or `(var2,2,dot)`) can be specified. If multisets are used, for instance, `var1,3*var2` is an abbreviation for `var1,var2,var2,var2`.
  * A click on the **Visibility** icon changes the visibility of the arc depending on the **Show invisible Arcs** option of the editor.

– The block dialog window shown in figure 8.6 can be opened by a double-click with the left mouse button on a block.

  Thus the name, the meaning, the set of blocks (the block belongs to), the box (the block belongs to), and the visibility of the block can be changed.

  * The name must be unique w.r.t. the other blocks of the net; this is checked by the editor.
  * If a net is compiled out of a B(PN)^2 program the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding B(PN)^2 block name).
  * If a net is compiled out of a B(PN)^2 program which contains nested blocks, a block may belong to one or more other blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical high-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
  * Again, a block may (on a second hierarchy level) belong to a box.

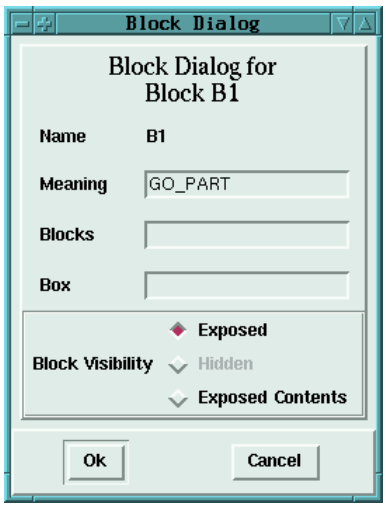


Figure 8.6: Block dialog window

  * Depending on the visibility of more upper level blocks, the visibility of a block may be changed:
    - **Exposed** means that the icon for the block is visible, but its contents are not.
    - **Hidden** means that neither the icon of the block nor its contents are visible,
    - **Exposed Contents** means that the contents of the block are visible, but its icon is not.

– The box dialog window shown in figure 8.7 can be opened by a double-click with the left mouse button on a box.

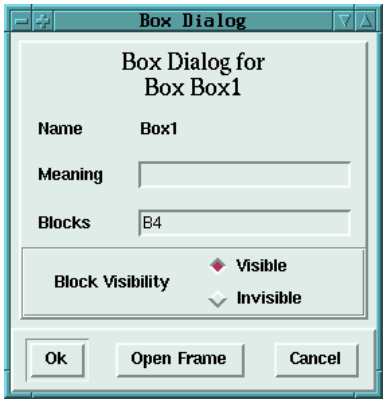


Figure 8.7: Box dialog window

Thus the name, the meaning, the set of blocks (the box belongs to), and the visibility of the box can be changed:

  * The name must be unique w.r.t. the other boxes of the net; this is checked by the editor.
  * If desired, a meaning may be specified for a box in order to provide some intuition.

  - It may be specified, that a box belongs to one or more blocks.
    Note: If the **Take in Box** function is used this relation is updated automatically.
  - A box may either be visible or not.
  - Moreover, an own frame can be opened for a box.
    Note: Unfortunately, this feature is still in a beta testing state.

- The whole screen is repainted if the **r**-key is pressed.

8.2 Menus of the high-level net editor

The pull-down menus offer further functionality.

- **File**
  - ▷ **Open:** Opens the file browser to select an M-net. The same functionality is provided by the button.
  - ▷ **Reload:** Reloads the M-net.
  - ▷ **Save:** Saves the contents of the net editor to the file whose name is displayed underneath the menu bar. The same functionality is provided by the button.
  - ▷ **Save as:** Opens the file browser to save the contents of the net editor to another file.
  - ▷ **Save Selection as:** Opens the file browser to save all currently selected objects (places, transitions, arcs, blocks and boxes – with all their annotations – and comments) to another file.
  - ▷ **Export to PostScript**
    - ○ **All Objects:** A file browser is opened to choose the name of a file to which all objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) are exported in PostScript format. The net examples in the appendix have been produced in this way.
    - ○ **Selection:** A file browser is opened to choose the name of a file to which all currently selected objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) are exported in PostScript format.
    - ○ **Entire Screen:** A file browser is opened to choose the name of a file to which all objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) that are currently visible on the screen are exported in PostScript format.
  - ▷ **Quit:** Closes the net editor. If there are unsaved changes a dialog box asks whether or not these changes should be saved.
    Note: Some changes like moving places or changing a current marking are not recognised. If you would like to keep these changes you should use the menu entry **Save** first.
- **Edit**
  - ▷ **New Place:** Switches the mode of the net editor to place insertion. The shape of the cursor changes to an internal place with type `{dot}`. The user can create such a new place by clicking at the desired position of the drawing canvas. The same functionality is provided by the button.

▷ **New Transition:** Switches the mode of the net editor to transition insertion. The shape of the cursor changes to a transition. The user can create such a new transition (with empty action term `{}` and value term `true`) by clicking at the desired position of the drawing canvas. The same functionality is provided by the button.

▷ **New PTransition:** Switches the mode of the net editor to phantom transition insertion. The shape of the cursor changes to a transition. The user can create such a new phantom transition (with empty action term `{}` and value term `true`) by clicking at the desired position of the drawing canvas.

▷ **New Block:** Switches the mode of the net editor to block insertion. The shape of the cursor changes to a block. The user can create such a new block (without meaning, an empty block list, without an associated box and an 'exposed' visibility flag) by clicking at the desired position of the drawing canvas.

▷ **New Box:** Switches the mode of the net editor to box insertion. The shape of the cursor changes to a box. The user can create such a new box (without meaning, an empty block list and a set visibility flag) by clicking at the desired position of the drawing canvas.

▷ **New Arc:** Switches the mode of the net editor to arc insertion. The user can determine first the start node and then the end node of the arc by clicking at the desired place (transition) and transition (place, resp.). The new arc has the inscription `a` and its 'invisible' flag is not set. The same functionality is provided by the button.

▷ **New PArc:** Switches the mode of the net editor to phantom arc insertion. The user can determine first the start node and then the end node of the phantom arc by clicking at the desired place (phantom transition) and phantom transition (place, resp.). Phantom arcs do not carry any inscription; nor they have an 'invisible' flag.

▷ **Create Comment:** Opens a window where a comment may be entered which is then positioned (almost arbitrarily) at the canvas. The same functionality is provided by the button.
Note: We recommend to use the **c**-key because this has the advantage that the comment is inserted at the current cursor position.

▷ **Cut:** Cuts the current selection (i.e. deletes the selected objects and stores it in the internal clipboard for re-insertion using the **paste** item).

▷ **Copy:** Copies the current selection (i.e. stores it in the internal clipboard for insertion using the **paste** item).

▷ **Paste:** Pastes the content of the clipboard (previously set by a **cut** or **copy** operation) as a selection to the canvas.

▷ **Find:** The dialog box shown in figure 8.8 is opened. It allows to find places, (also phantom) transitions, blocks or boxes by name.

Figure 8.8: Find dialog window

▷ **Select some Nodes:** The selection box shown in figure 8.9 is opened. It allows to select a place, a (e.g., phantom) transition, a block or a box by name.

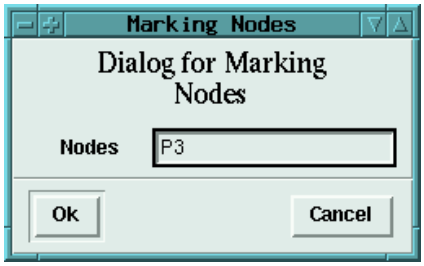


Figure 8.9: Node selection dialog window

▷ **Select all:** Selects the whole net. This is particularly important if, for instance, the whole net should be synchronised.
Note: A selection can be de-selected by pressing the **Esc** key.

▷ **Delete Elements:** Deletes the currently selected elements.

▷ **Delete Net:** Clears the editor (i.e., deletes the current net but not the file). The same functionality is provided by the button.

▷ **Repaint:** Repaints the whole canvas. This may sometimes be necessary because some repainting is still not done automatically.

• Hierarchy

▷ **Hierarchy Tree:** The dialog box shown in figure 8.10 is opened. It displays the hierarchy of the boxes and blocks involved in the current net and allows to change their visibility. A click on the **Exposed Contents all** button shows the contents of all boxes and blocks. The same functionality is provided by the

button.

▷ **Block List:** The dialog box shown in figure 8.11 is opened. It displays a list of all the blocks involved in the current net and allows to change their visibility. A click on the **Exposed Contents all** button shows the contents of all blocks. In contrast to the **Exposed Contents all** button of the hierarchy tree dialog box (or the

button) only the contents of all blocks are exposed.

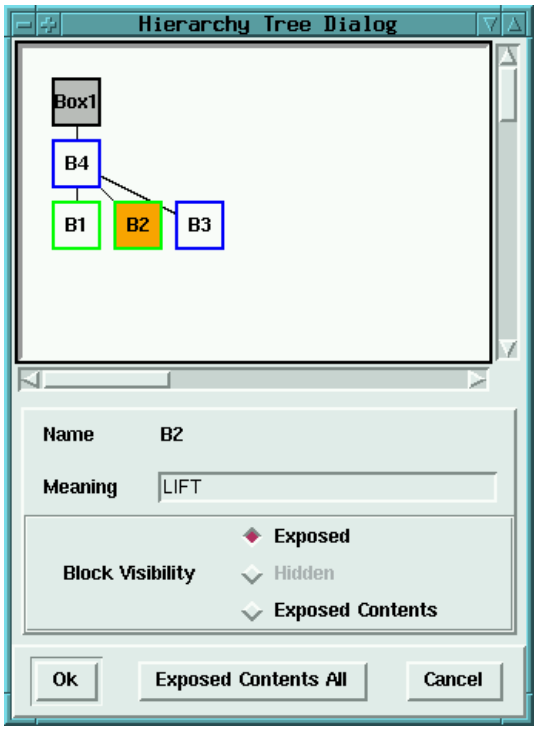


Figure 8.10: Hierarchy tree dialog window

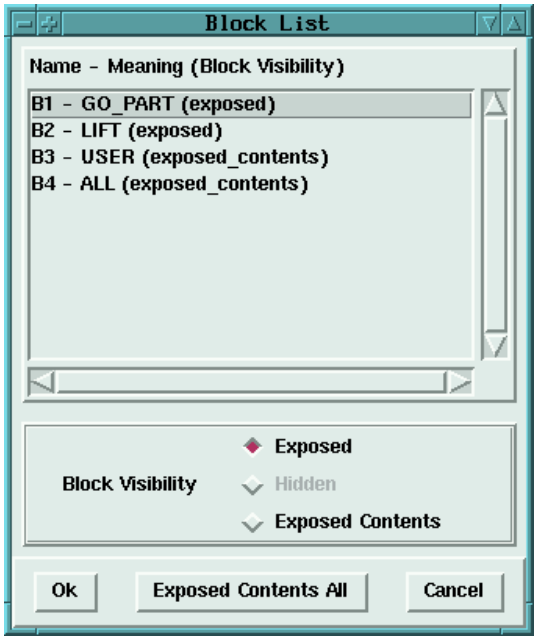


Figure 8.11: Block list dialog window

▷ **Other Nets List:** The dialog box shown in figure 8.12 is opened. In principle, this window allows to handle different nets simultaneously. Unfortunately, this functionality is clearly in a beta testing state. Thus, we do not recommend to use this feature.

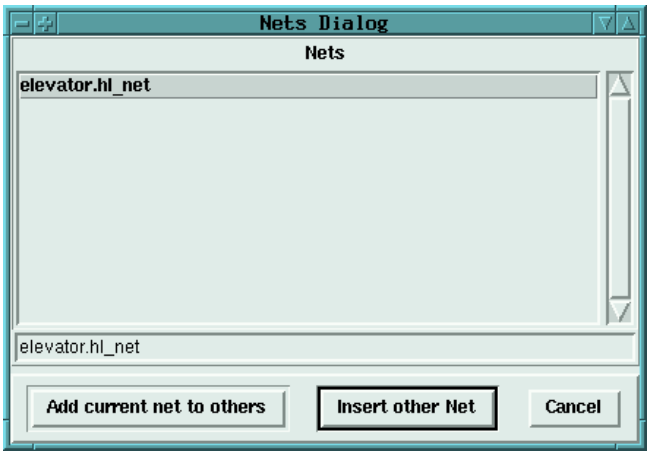


Figure 8.12: Other net list dialog window

• Composition

▷ **Nela:** The dialog box shown in figure 8.13 is opened. It offers a comfortable interface to (most of) the functionality provided in the rest of the menu.

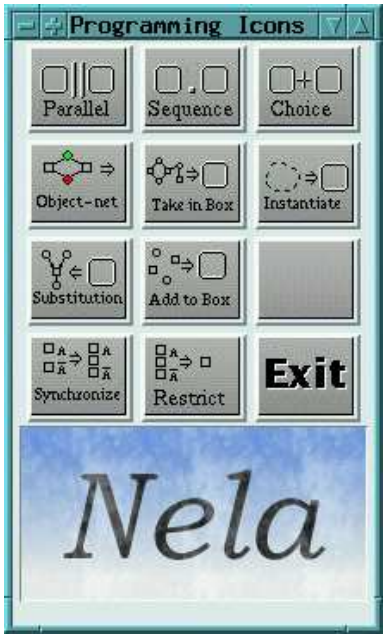


Figure 8.13: The 'Nela' window

▷ **Take in Box:** The current selection is taken as the basis for a new box. This means, that recursively all objects belonging to a block to which one of the selected (or already added) objects belong are also added to the box. This offers a simple

method of constructing boxes which may then be used for composition operations. The same functionality is provided by the following icon of the 'Nela' window:

▷ **Add to Box:** The current selection is taken as the basis for an extension of an existing box. This means, that recursively all objects belonging to a block to which one of the selected (or already added) objects belong are also added to the box which is chosen using the dialog window shown in figure 8.14. The same functionality is provided by the following icon of the 'Nela' window:

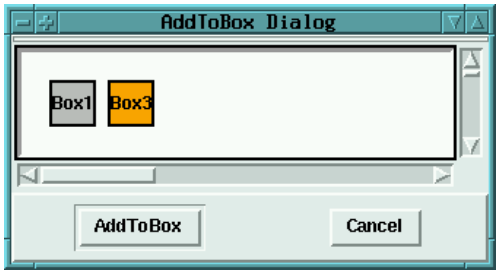


Figure 8.14: Add to box dialog window

▷ **Instantiate Box:** One of the purposes of the new 'programming with nets' approach is to have the ability to load the template nets which are usually used during the construction of a program semantics (for instance, a variable net) and to instantiate the macros (for instance, for the name of the variable). This menu item opens the corresponding instantiation dialog window shown in figure 8.15.

In the canvas (the lower part) of the window a box which is to be instantiated can be chosen. A selection has the effect that a list of the macro names used in the template box becomes available in the upper part of the instantiation window. Now, the macros may be chosen step by step with the left mouse button. After each selection you have to click with the left mouse button in the middle part of the window (where the current value of the macro appears) in order to start the editing. It is important that each change is confirmed by hitting the **return**-key in the typing area. Finally, if all instantiations are defined, the actual instantiation is invoked by clicking in the **Instantiate** button.

The same window is opened by the following icon of the 'Nela' window:

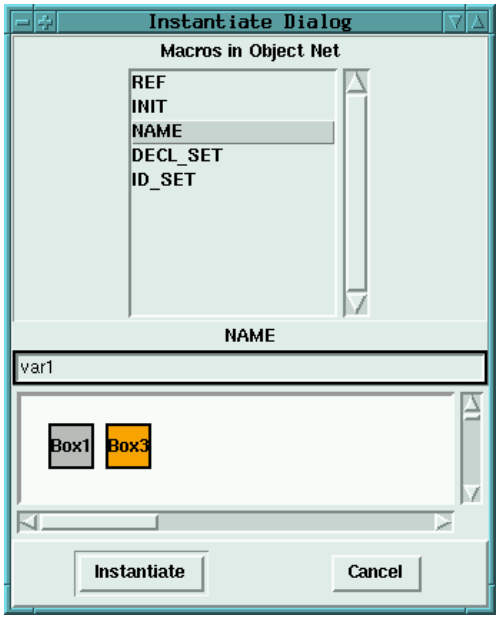


Figure 8.15: Instantiate box dialog window

▷ **Append Suffix:** The dialog window shown in figure 8.16 is opened. It allows to add a suffix (which is entered in the area in the upper part of the window) to the names of all objects belonging to the box chosen in the main part (the canvas) of the window.

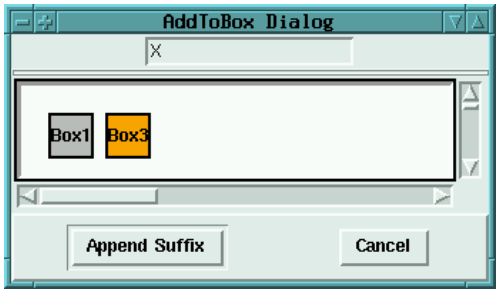


Figure 8.16: Append suffix dialog window

▷ **Parallel:** The dialog window shown in figure 8.17 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in parallel. The same functionality is provided by the following icon of the 'Nela' window:

▷ **Sequence:** The dialog window shown in figure 8.18 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in sequence. The same functionality is provided by the following icon of the 'Nela' window:

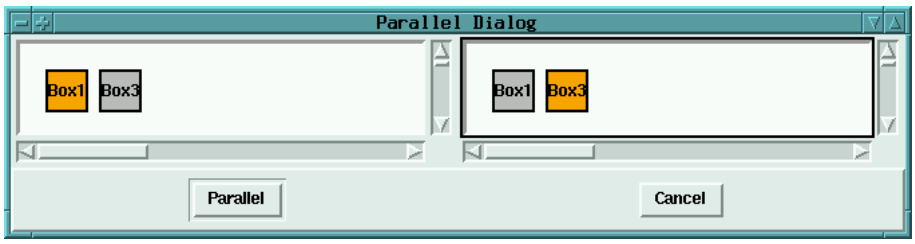


Figure 8.17: Parallel composition dialog window

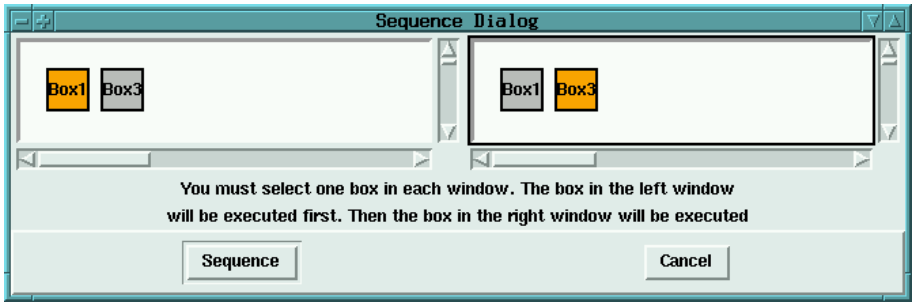
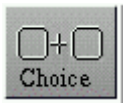


Figure 8.18: Sequential composition dialog window

▷ **Choice:** The dialog window shown in figure 8.19 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in choice. The same functionality is provided by the following icon of the 'Nela' window: 

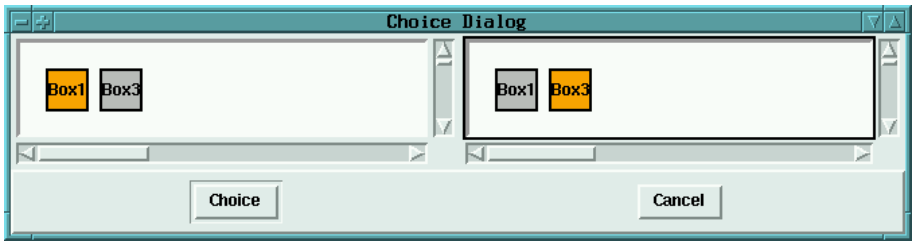
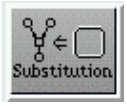


Figure 8.19: Choice composition dialog window

▷ **Transition Substitution:** The dialog window shown in figure 8.20 is opened. It allows to substitute a transition (which is chosen in the list in the upper part of the window) by the box chosen in the main part (the canvas) of the window. The same dialog window is opened by clicking on the following icon of the 'Nela' window: 

▷ **Synchronisation:** The dialog window shown in figure 8.21 is opened. It allows to synchronise the current selection (not the whole currently displayed net) with respect to the action symbols

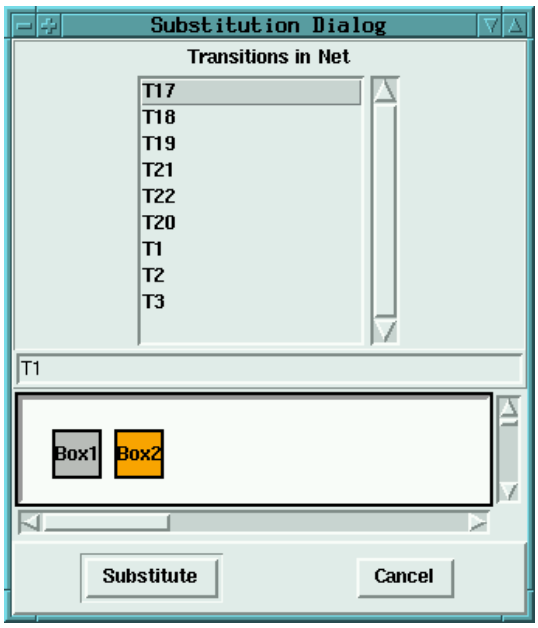
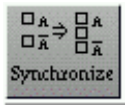


Figure 8.20: Transition substitution dialog window

listed in the upper editing area of the dialog box while action symbols given in the lower editing area are excluded. The same dialog window is opened by clicking on the following icon of the 'Nela' window:



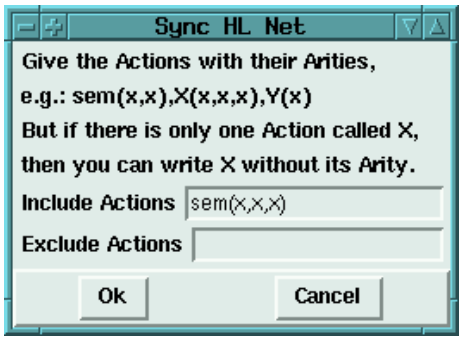


Figure 8.21: Synchronisation dialog window

▷ **Restriction:** The dialog window shown in figure 8.22 is opened. It allows to restrict the current selection (not the whole currently displayed net) with respect to the action symbols listed in the upper editing area of the dialog box while action symbols given in the lower editing area are excluded. The same dialog window is opened by clicking on the following icon of the 'Nela' window:

▷ **Expression:** The dialog window shown in figure 8.23 is opened. It allows to create a high-level net which is the semantics of an

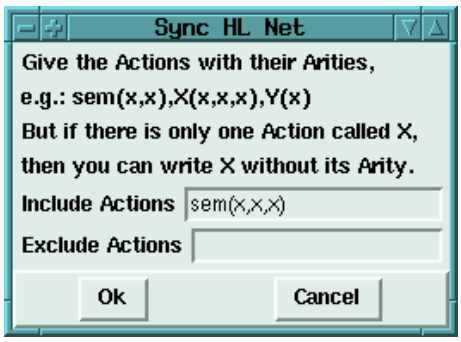


Figure 8.22: Restriction dialog window

expression entered in the editing area of the dialog box. It may optionally be specified that procedure semantics is assumed for the creation. The same dialog window is opened by clicking on the following icon of the 'Nela' window:

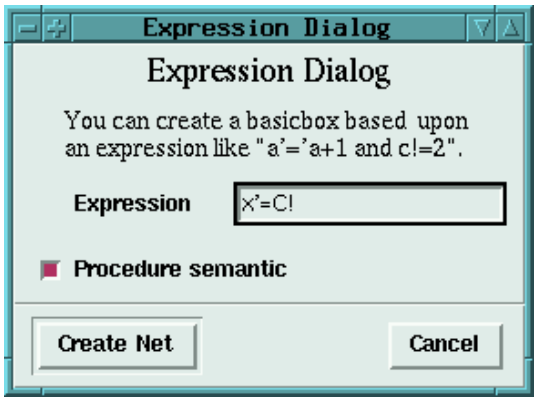
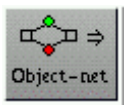


Figure 8.23: Expression dialog window

▷ **Load Module:** A file selection box is opened (in the current directory) which allows to load a high-level net as a module. The same functionality is provided by the following icon of the 'Nela' window:



▷ **Load Templatenet:** A file selection box is opened (in the directory which contains the template nets used by the compiler from a B(PN)^2 program into a high-level net if procedures are involved) which allows to load such a template net as a module.

- **Action**

▷ **Show Active Transition:** Colours all currently enabled transitions (synchronisation as well as phantom transitions) yellow. The

mode can be exited by (clicking in the canvas and) pressing the **Esc** key.
Note: If an enabled transition belongs to a block whose contents are currently not visible, then the icon of the block is coloured in yellow. Clicking on the (yellow) block icon fires the transition.

▷ **Active Transition Dialog:** The dialog window shown in figure 8.24 is opened.

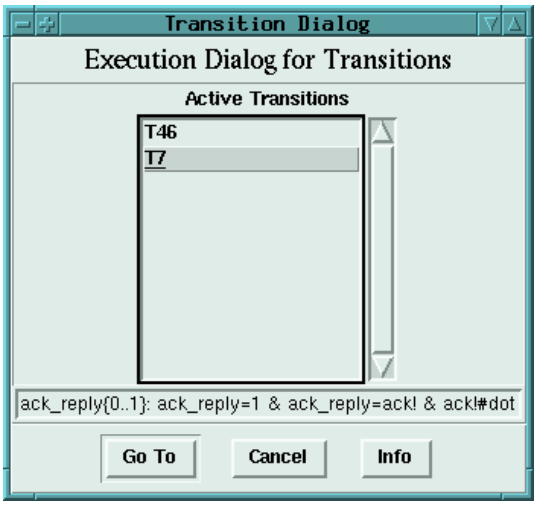


Figure 8.24: Active transitions dialog window

It allows to get more information concerning the currently active transitions listed in the main part of the dialog box. After selecting one of the transitions (by a mouse-click) you may have this transition (or the corresponding block icon) centred and coloured in orange in the canvas of the net editor by pressing the **Go To** button.

Moreover, you may open an additional dialog window shown in figure 8.25 by pressing the **Info** button. The main part of this dialog box displays the list of pre-places (on the left hand side) and the list of post-places (on the right hand side). After selecting one place of such a list (which automatically colours this place in orange and focuses the canvas of the net editor to this place), a couple of information is given: the corresponding arc inscription, the meaning of the place, the type of the place, its initial and its current marking.

▷ **All Transition Dialog:** A similar (to the active transition dialog) window is opened. It gives a list of all the transitions (in alphabetical order) and offers the same functionality.

▷ **Execute Transition:** If a transition sequence has been stored (either by **Execute Transition List** or by selecting **Sim HL**

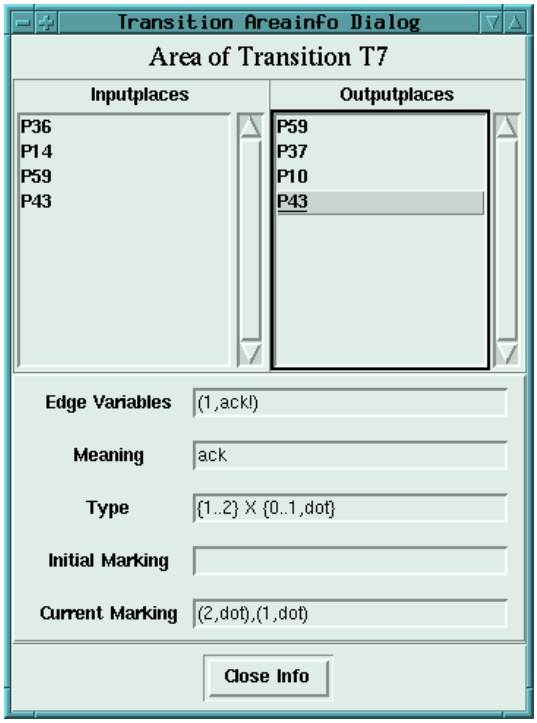


Figure 8.25: Transition area dialog window

in one of the analysis or verification windows) the next transition in the list is executed. Otherwise, the mode of the net editor is switched so that all currently enabled transitions (or corresponding blocks if they are exposed) are coloured yellow. They can be executed by a mouse click. The behaviour depends on the option **Random Bindings**. The mode can be exited by pressing the **Esc** key. The same functionality is provided by the following icon: ▶

▷ **Execute Transition List:** The dialog window shown in figure 8.26 is opened. It allows you to enter a transition sequence and (optionally) an additional 'loop' transition sequence. Clicking on the **OK** button stores the given sequences. Subsequent invocations of either **Execute Transition** (e.g., by clicking on the ▶ button) or of **Animate** (e.g., by clicking on the ▶▶ button) will start to execute these lists.

▷ **Animate:** If a transition sequence has been stored (either by **Execute Transition List** or by selecting **Sim HL** in one of the analysis or verification windows) the transitions in the list are executed. Otherwise, the mode of the editor is switched so that an automatic simulation is started. This menu entry is only enabled if the option **Random Bindings** is switched on. During the simulation all currently enabled tran-

Figure 8.26: Execute transitions list dialog window

sitions are coloured yellow. One of them is chosen randomly together with a possible binding for the variables, and is executed. The surrounding arcs are coloured green to show the token flow. The mode can be exited by pressing the **Esc** key (or clicking on the ▮▮ button). The same functionality is provided by the following icon: ▶▶

▷ **Reset to Initial Marking:** Resets the marking to the initial marking from which a new simulation of the net can be started.

▷ **Clear Transition List:** Clears the transition list(s) entered using the **Execute Transition List** item or by a click on a **Sim HL** button of one of the analysis or verification windows. As a consequence, an animation may continue in random binding mode.

▷ **Show Reference:** If the M-net belongs to a B(PN)^2 program (or an SDL specification), the correspondence between, for instance, a place of the net and a point in the control flow of the B(PN)^2 program (or SDL specification) or between a transition in the net and actions in the B(PN)^2 program (or, e.g., a task in the SDL specification) can be demonstrated. The mode of the net editor is switched so that the corresponding text segment is highlighted in aquamarine if the user clicks on a transition, a place or a block. The mode can be exited by pressing the **Esc** key. The same functionality is provided by the following icon: *Ref*

▷ **Convert to New Net Format:** This menu item provides a possibility to convert old high-level Petri net files (using the old '`pr1`' notation for tuples) into the new format.

• **Network** The new version of the high-level Petri net editor offers (beta) support for distributing the net simulation over a network of computers.

▷ **Select Servers:** The dialog window shown in figure 8.27 is opened. It allows you to add/delete servers to/from the list of computers on which the simulation is performed. The upper list gives the current status. Typing the name of a new server in the lower typing area and clicking on the **Add** button adds a server to the list; while selecting a server in the list and pressing the **Delete** button removes this server from the list. The current configuration may be saved as a default for the current user (as `.pep_server` in the home directory) or as a general PEP default (as `.pep_server` in the `PEPHOME` directory (this requires of course write permission).
Note: we tried to avoid mal functions caused by wrong entries as far as possible. However, we cannot ensure correct behaviour in all cases. Thus, in case of a problem it might help to remove the mentioned `.pep_server` files manually.

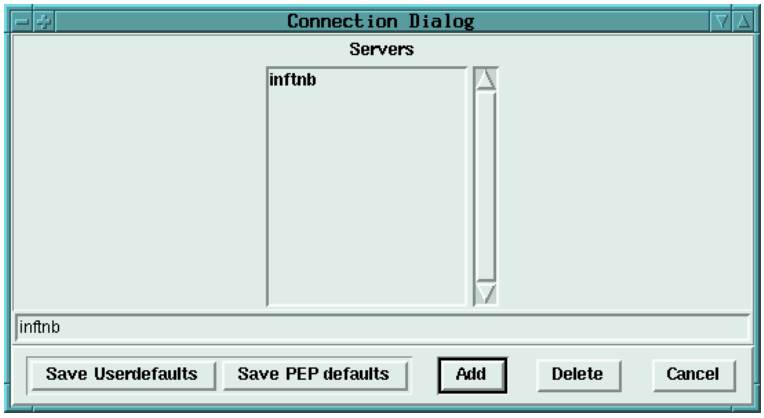


Figure 8.27: Select servers dialog window

▷ **Establish Connections:** A connection to the servers given in the list is established.

• Option

▷ **Display**

◦ **Names:** Chooses whether or not the names of places, (synchronisation as well as phantom) transitions, blocks and boxes are displayed.

◦ **Meanings:** Chooses whether or not the meanings of places, (synchronisation as well as phantom) transitions, blocks and boxes are displayed.

◦ **Action Terms:** Chooses whether or not the action terms of transitions are displayed.

◦ **Value Terms:** Chooses whether or not the value terms of transitions are displayed.

- ◦ **Types:** Chooses whether or not the types of places are displayed.
- ◦ **Arc Annotations:** Chooses whether or not the arc annotations are displayed.
- ◦ **Arc during Moving:** Chooses whether or not the display of arcs is updated during a move of connected places and/or transitions. This may slow down move operations considerably.

▷ **Change Font:** The dialog window shown in figure 8.28 is opened.

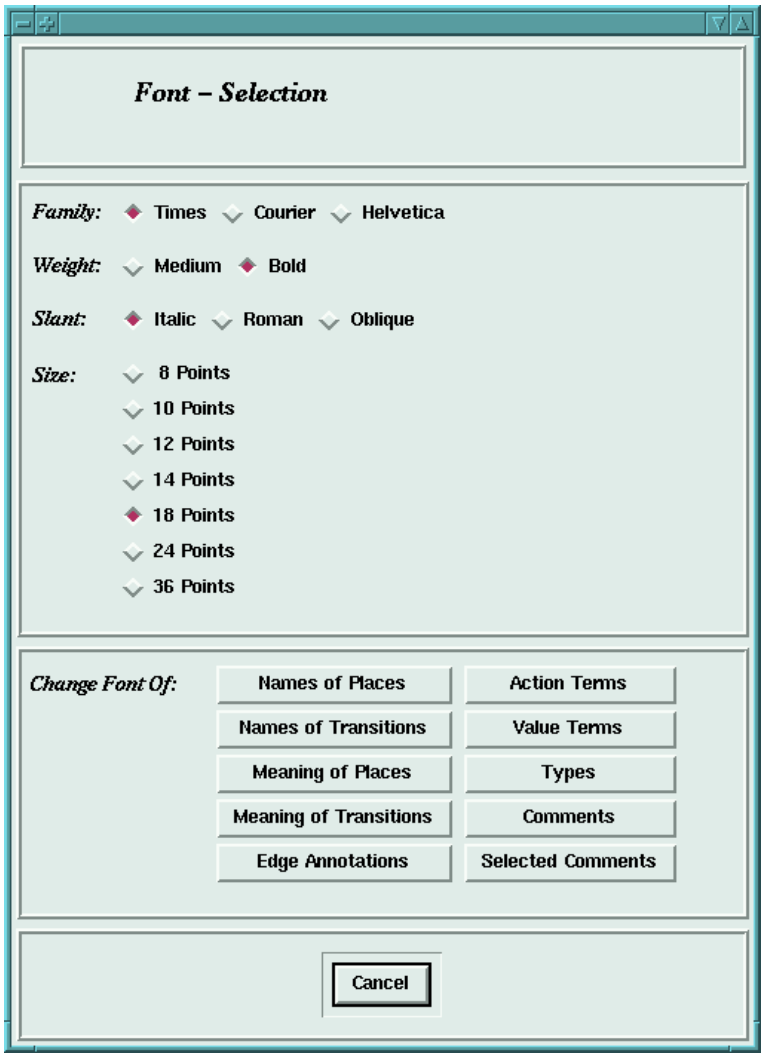


Figure 8.28: Change font dialog window

It allows you to select a font for names of places, names of transitions, meanings of places, meanings of transitions, edge annotations, action terms, value terms, types, all comments or selected comments. In the middle part of the window a font is selected. An example of a text in the selected font (or a message that the chosen font does not exist) is shown in

the upper part of the dialog window. A click on one of the icons in the lower part of the window changes the font for the corresponding selection. Clicking on the **Cancel** button closes the dialog window.

▷ **Show**

◦ **Invisible Arcs:** Chooses whether or not invisible arcs are displayed. An invisible flag is set by the compilers 'B(PN)^2 ⇒ M-net' and 'SDL ⇒ M-net' for arcs between the data boxes of the net and the control flow part of the net to enhance the readability of the constructed nets (see page 33).

◦ **Synchronisation Transitions:** Chooses whether or not the synchronisation transitions (together with their adjacent arcs) are shown. In case of a lot of synchronous communication this may turn the visualisation of a net into something almost unreadable.

◦ **Phantom Transitions:** Chooses whether or not phantom transitions are shown (together with their adjacent phantom arcs).

▷ **Synchronisation Options:** The dialog window shown in figure 8.29 is opened.

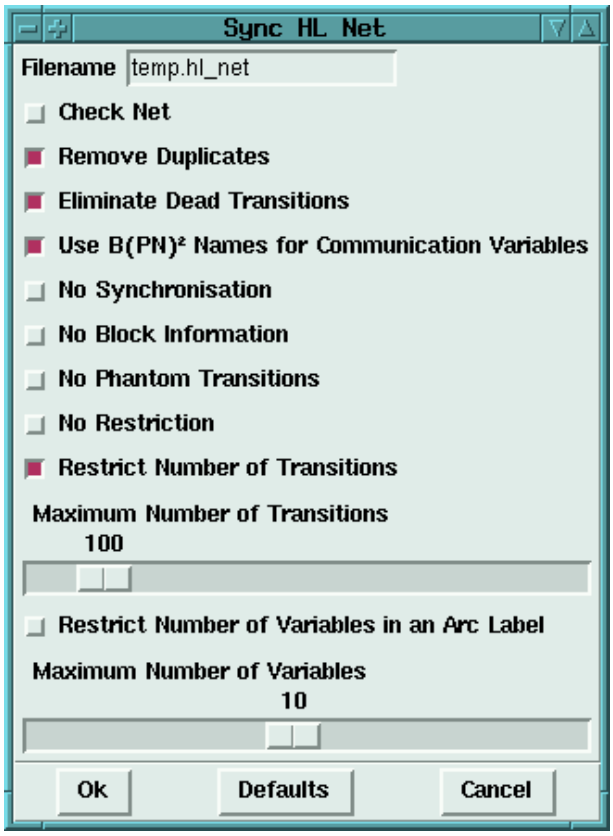


Figure 8.29: Synchronisation options dialog window

It allows to set the options for the integrated synchronisation algorithm. The options have already been explained in subsection 5.3.6 on page 33.

▷ **Move Options:** The dialog window shown in figure 8.30 is opened. It allows to specify whether or not nodes are moved if the block (or box) they belong to is moved and whether or not selections are moved together.

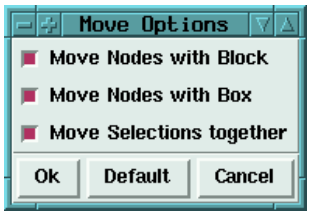


Figure 8.30: Move options dialog window

▷ **Simulation Options:** The dialog window shown in figure 8.31 is opened. It allows to fine tune the behaviour of the simulation algorithm. In particular, an additional pre analysis may be switched off; an additional in analysis may be switched off; caching of already generated binding information may be forbidden; it may be specified that all bindings are calculated; and the automatically generated protocol for the time needed for the simulation may be switched off.

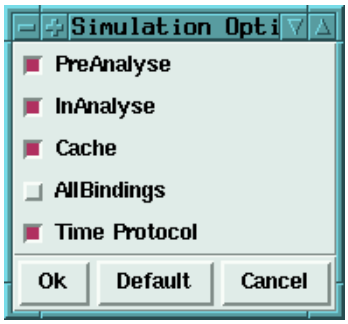


Figure 8.31: Simulation options dialog window

▷ **Simulation Statistics:** The dialog window shown in figure 8.32 is opened. It gives a list of the fired transitions together with the time needed for the simulation. In the right column the overall simulation time is counted.

▷ **Template Directory:** The dialog window shown in figure 8.33 is opened. It allows to change the name of the directory where template nets (cf. the explanation of the **Load Templatenet** entry of the **Composition** dialog) are located.

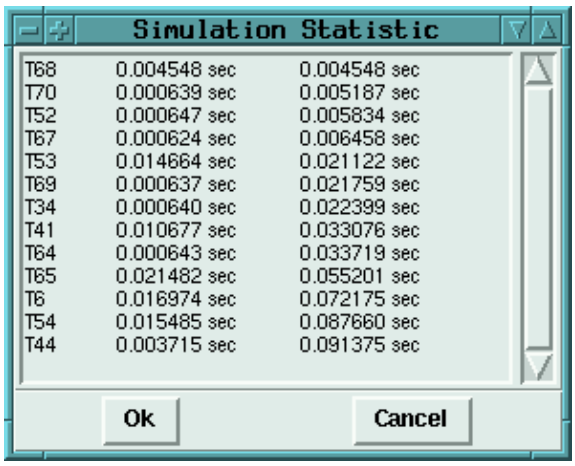


Figure 8.32: Simulation statistics window

Figure 8.33: Template directory selection window

▷ **Highlight Execution:** If the M-net belongs to a B(PN)^2 program (or SDL specification), the correspondence between a transition in the net and actions in the B(PN)^2 program (or SDL specification) can be demonstrated. The mode of the net editor is switched so that if a transition is executed, the corresponding text segment is highlighted in aquamarine in the B(PN)^2 (or SDL) editor.

▷ **Output Meaning:** If the M-net belongs to a B(PN)^2 program (or SDL specification), it may be useful to output the meaning of executed transitions (which gives the relation to the actions of the B(PN)^2 program – or SDL specification) to the shell.

▷ **Reference Format:** This options is for debugging purposes only and should not be touched! It changes the format of internal communication between the net editor and the PEP tool.

▷ **Random Bindings:** Chooses whether a random binding is calculated and taken if a transition is executed, or whether the dialog box shown in figure 8.34 is opened to allow the user to test and choose a binding.

This dialog window may be used in two different

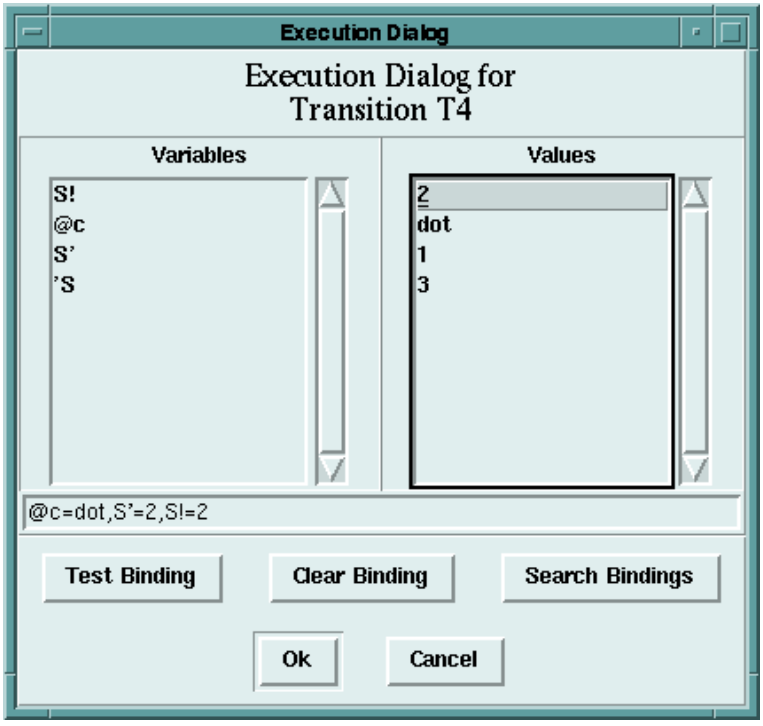


Figure 8.34: Execution dialog window

ways:

1. You may compose a binding (which is in turn displayed in the typing area in the bottom part of the window) step by step by selecting first a variable in the left list and then a value for the variable in the list of possible values (on the right hand side).
2. You may let the net editor calculate a list of all possible (enabling) bindings by pressing the **Search Bindings** button. Likewise, the list of bindings is displayed in an additional area in the lower part of the window (cf. figure 8.34). From this list you may choose a binding (which is again displayed in the typing area in the bottom part of the window).

Once you have created a binding (for all the variables), you may check whether or not you composed a valid (i.e. enabling) binding using the **Test Bindings** button. If the binding is valid you may fire the transition using this binding by pressing the **OK** button. Pressing the **Clear Bindings** button clears the binding.

▷ **Zoom:** Switches amongst different zoom factors.

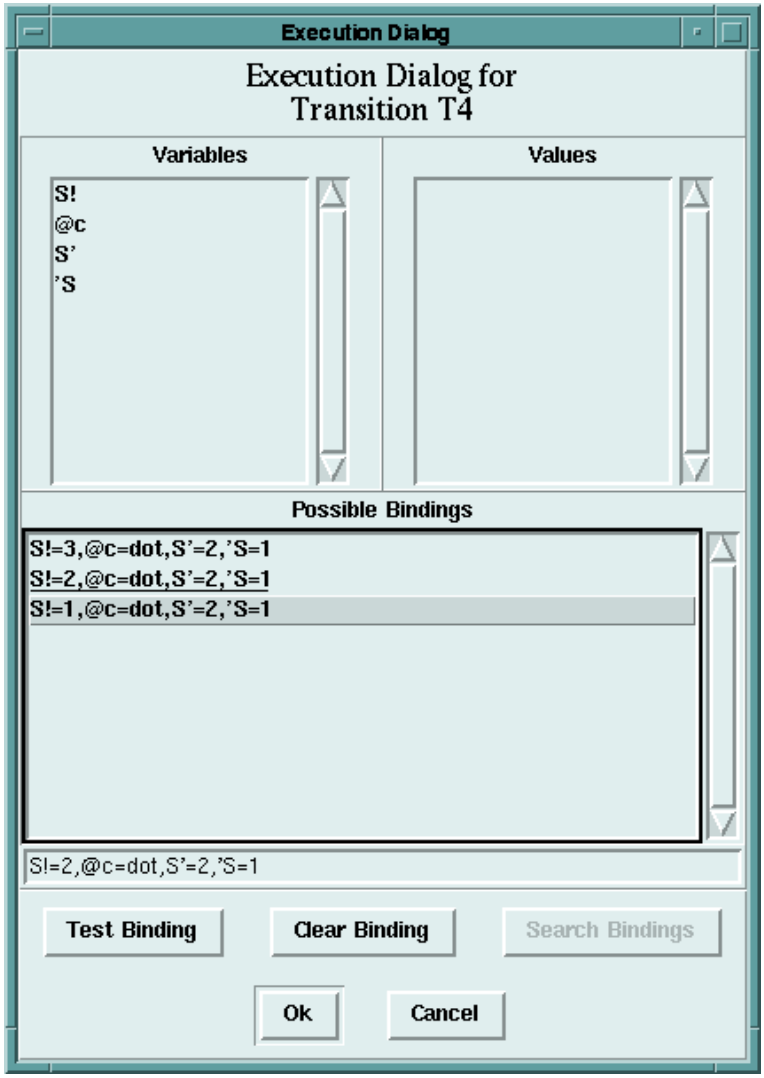


Figure 8.35: Execution dialog window

Note: The status of a place is always displayed by appropriate colours. Entry places are coloured green, exit places are coloured red and internal places are uncoloured.

- **Help**

  ▷ **Info:** Shows an about box.

Chapter 9

The low-level Petri net (Petri box) editor

LL nets are special HL nets, essentially by having all places type `{dot}` and all transition value terms `true`. The windows and dialogs of the LL net editor (which is shown in figure 9.1) are correspondingly simplified (restricted) versions of those of the HL net editor. As a consequence, reading the section describing the high-level Petri net editor should basically be sufficient to work with the low-level net editor as well. However, we have chosen to duplicate the explanations (at least partially) in order to provide a self-contained description of the low-level net editor for those who do not intend to work with the high-level net editor.

9.1 Description of the low-level Petri net editor window

The low-level net editor offers Petri box specific features and has an integrated simulator.

- The main part of the editor window can be used as a drawing canvas for creating and editing Petri boxes.
- The functionality of the menu bar is described in the next section.
- An additional icon bar provides a comfortable access to some of the functionality offered by the entries of the menu bar. We will give the details when we describe the corresponding menu items in the next section.
- The name of the currently edited Petri box is displayed underneath the menu bar (and an additional icon bar) on the left hand side.
- The speed of a simulation may be changed using the slider in the middle underneath the menu bar.
- The current mode of the net editor is displayed underneath the menu bar on the right hand side.
- If the cursor is, for instance, positioned on one of the icons of the additional icon bar a brief information about the provided functionality is given in the area in the lower left corner under the canvas.
- If the editor is in *Edit Mode* the following can be done:

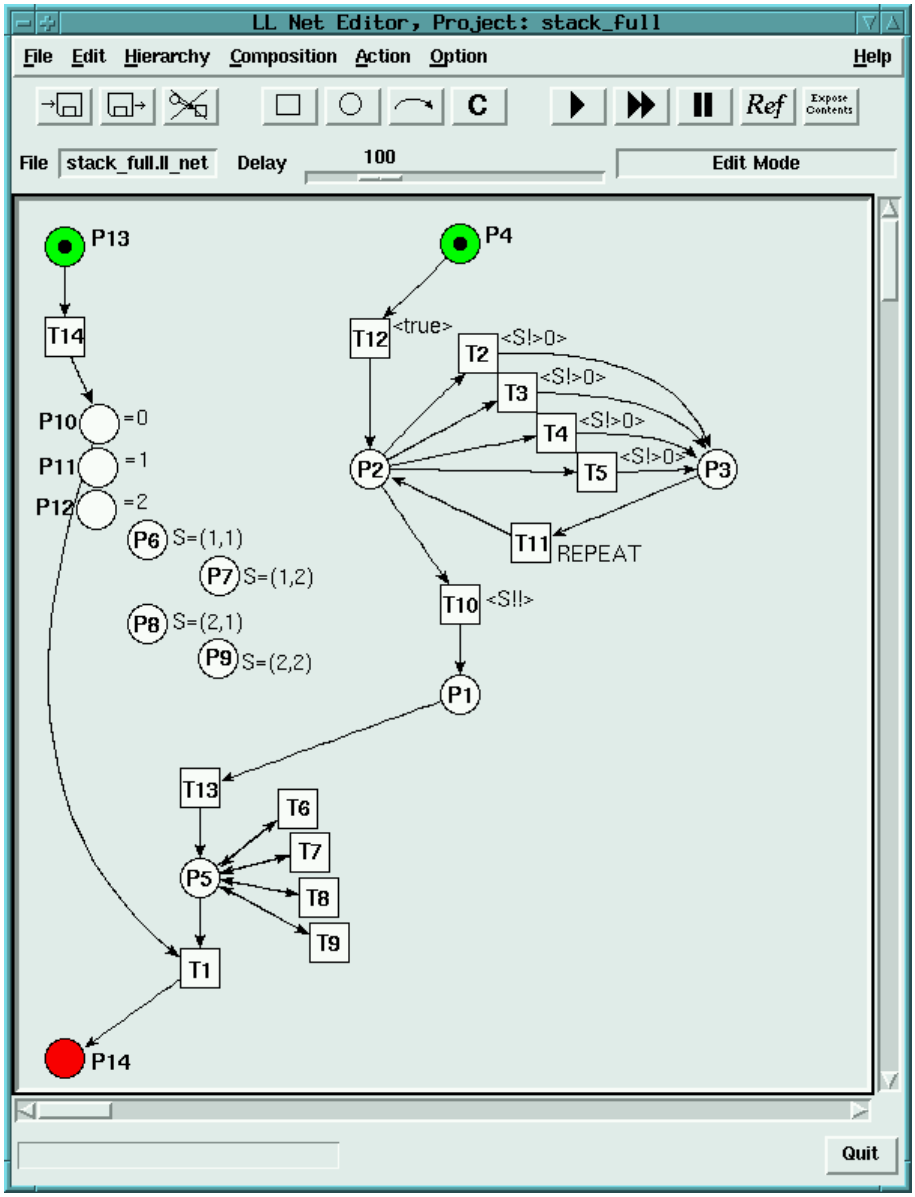


Figure 9.1: Low-level net editor

- Places can be created by pressing the **p**-key at the desired position of the drawing canvas.
- Transitions can be created by pressing the **t**-key at the desired position of the drawing canvas.
- Arcs can be created by pressing the **a**-key first and selecting the place (transition) and the transition (place, resp.) with the left mouse button afterwards.
  Note: Clicking to quickly, may cause errors.
- Comments can be created by pressing the **c**-key at the desired position of the drawing canvas. The dialog box shown in figure 9.2 is opened to edit the text.
- An object (place, transition, arc, block, box, comment or annotation) can be selected by a click with the left mouse button.
- A selected object can be moved by holding the left button down while moving the mouse. (For arcs, this moves the point you clicked at. You may turn such an

Figure 9.2: Comment dialog window

arc into a straight one by selecting it with the left mouse button and clicking on the little red square, which appears, with the middle mouse button.)

- A multi-selection of places and transitions can be extended by a click on another place or transition while the **shift** key is held down.
  - A multi-selection can be deleted by pressing the **del** key.
  - A multi-selection can be moved by holding down the left mouse button (after its last extension) while moving the mouse.
- An area can be selected by clicking with the middle mouse button at the upper left corner of the area, holding the button down until the lower right corner of the area is reached, and finally clicking with the middle mouse button somewhere in the area. (If you have a mouse with only two buttons you can probably emulate this by clicking both buttons at the same time).
  - An area can be deleted by pressing the **del** key.
  - An area can be moved by the left mouse button (after clicking somewhere in the area without hitting an object).

  Note: In principle, general multi-selections containing objects of all kinds (including blocks or boxes) and areas are supported. However, the current version of the editor has still problems with combined multi-selections.
- The place dialog window shown in figure 9.3 can be opened by a double-click with the left mouse button on a place.

  Thus the name, the meaning, the set of references, the set of blocks (the place belongs to), the box (the place belongs to), the initial as well as the current marking and the status of the place can be changed:
  - The name must be unique w.r.t. the other places of the net; this is checked by the editor.
  - If a net is compiled out of a B(PN)^2 program (or an SDL specification) the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding atomic action).
  - If a net is compiled out of a B(PN)^2 program (or an SDL specification) a set of references is automatically generated for some of the places. These references

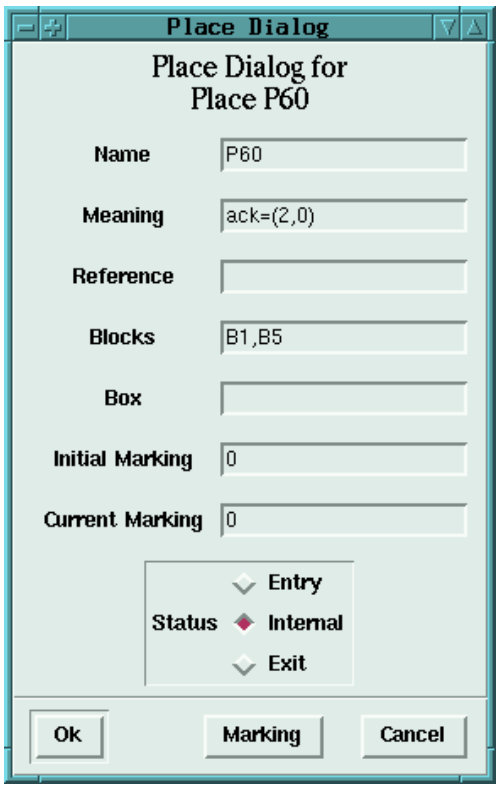


Figure 9.3: Place dialog window

are used by some of the internal programs of the PEP tool. Although, the set of references may be edited, we highly recommend not to do this.

* If a net is compiled out of a B(PN)^2 program which contains nested blocks, a place may belong to one or more hierarchical net objects also called blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical low-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
* Boxes build a second hierarchy level. They are used for a 'programming with nets' approach. We recommend to have a look at the documentation of the '`programming_with_nets_example`' project in the PEP distribution in order to get a first intuition of this new method.
* The marking of a place gives the number of black tokens which are initially or currently stored on this place.
* The status of a place may either be **entry**, **internal** or **exit**.

Moreover, pressing the **Marking** button opens an additional window which gives the current marking of the place. This is particularly useful during the simulation of huge nets because it allows to control the most interesting places.

– The transition dialog window shown in figure 9.4 can be opened by a double-click with the left mouse button on a transition.

  Thus the name, the meaning, the set of references, the set of synchronisation or phantom transitions which is associated with a phantom or synchronisation transition, respectively, the set of blocks (the transition belongs to), and the box (the transition belongs to) of the transition can be changed.

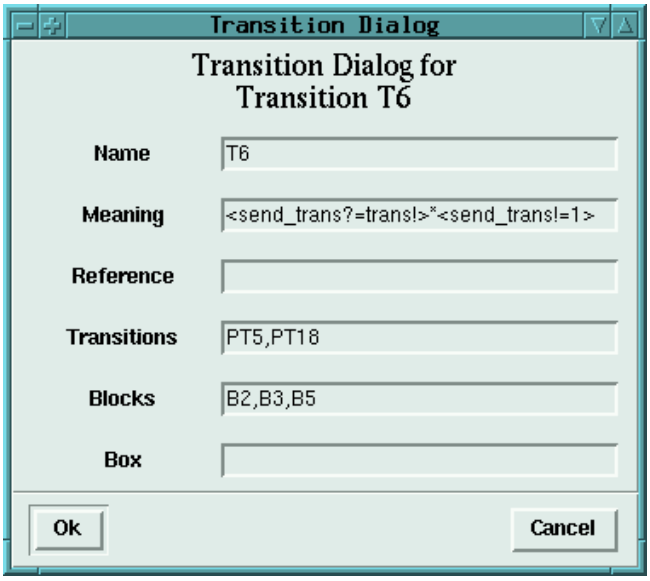


Figure 9.4: Transition dialog window

* The name must be unique w.r.t. the other transitions of the net; this is checked by the editor.
* If a net is compiled out of a B(PN)^2 program (or an SDL specification) the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding atomic action).
* If a net is compiled out of a B(PN)^2 program (or an SDL specification) a set of references is automatically generated for some of the transitions. These references are used by some of the internal programs of the PEP tool. Although, the set of references may be edited, we highly recommend not to do this.
* For the translation of synchronous communication we now distinguish transitions of two kinds. The synchronisation of two atomic actions (e.g., $\langle\ C!{=}1\ \rangle$ and $\langle\ C?{=}1\ \rangle$) now yields one synchronisation transition (the usual synchronisation result which is a normal transition) and two phantom transitions (each of them visualises one of the original counter parts of the synchronisation). As a consequence, for such a synchronisation transition a list of corresponding phantom transitions which are associated with it may be specified (actually a transition is automatically coloured in cyan as soon as a list of phantom transitions is specified). Similarly, a list of synchronisation transitions may be given for a phantom transition.
* If a net is compiled out of a B(PN)^2 program which contains nested blocks, a transition may belong to one or more hierarchical net objects also called blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical low-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
* Boxes build a second hierarchy level. They are used for a 'programming with nets' approach. We recommend to have a look at the documentation of the '`programming_with_nets_example`' project in the PEP distribution in order to get a first intuition of this new method.

– The arc dialog window shown in figure 9.5 can be opened by a double-click with the left mouse button on an arc.

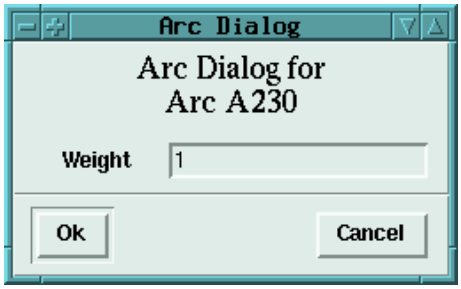


Figure 9.5: Arc dialog window

Thus the arc weight and its visibility can be changed:

* The unique name is generated automatically and cannot be changed.
* The arc weight is given.
* A click on the **Visibility** icon changes the visibility of the arc depending on the **Show invisible Arcs** option of the editor.

– The block dialog window shown in figure 9.6 can be opened by a double-click with the left mouse button on a block.

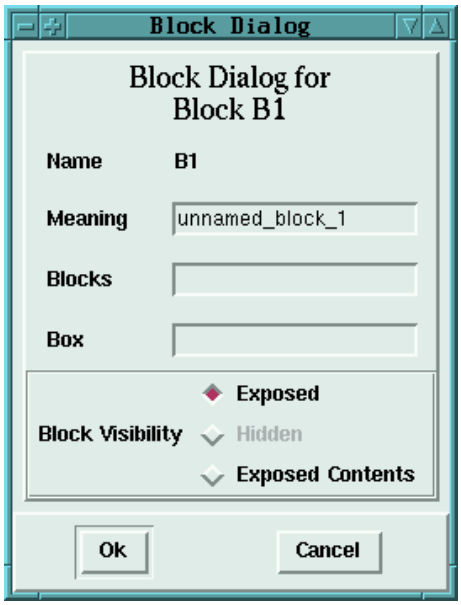


Figure 9.6: Block dialog window

Thus the name, the meaning, the set of blocks (the block belongs to), the box (the block belongs to), and the visibility of the block can be changed.

* The name must be unique w.r.t. the other blocks of the net; this is checked by the editor.
* If a net is compiled out of a B(PN)^2 program the meaning is meant to help the user understanding the relation (e.g., by giving the corresponding B(PN)^2 block name).

  - If a net is compiled out of a B(PN)^2 program which contains nested blocks, a block may belong to one or more other blocks. Moreover, such a 'belong to' relation may as well be established using the dialog box if a hierarchical low-level Petri net is edited by hand. Of course, all names given must refer to existing blocks.
  - Again, a block may (on a second hierarchy level) belong to a box.
  - Depending on the visibility of more upper level blocks, the visibility of a block may be changed:
    - **Exposed** means that the icon for the block is visible, but its contents are not.
    - **Hidden** means that neither the icon of the block nor its contents are visible,
    - **Exposed Contents** means that the contents of the block are visible, but its icon is not.

- The box dialog window shown in figure 9.7 can be opened by a double-click with the left mouse button on a box.

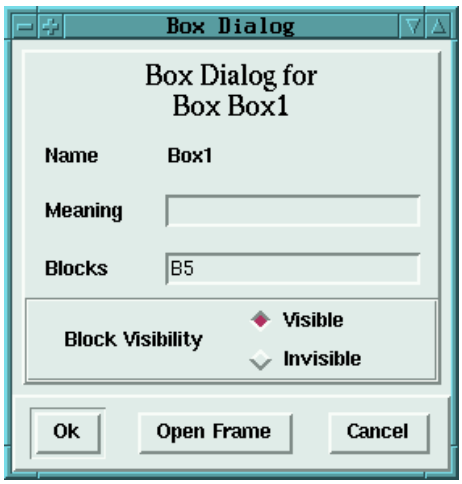


Figure 9.7: Box dialog window

Thus the name, the meaning, the set of blocks (the box belongs to), and the visibility of the box can be changed:

  - The name must be unique w.r.t. the other boxes of the net; this is checked by the editor.
  - If desired, a meaning may be specified for a box in order to to provide some intuition.
  - It may be specified, that a box belongs to one or more blocks.
    Note: If the **Take in Box** function is used this relation is updated automatically.
  - A box may either be visible or not.
  - Moreover, an own frame can be opened for a box.
    Note: Unfortunately, this feature is still in a beta testing state.

- The whole screen is repainted if the **r**-key is pressed.

9.2 Menus of the low-level net editor

The pull-down menus offer further functionality.

- **File**
  - ▷ **Open:** Opens the file browser to select a Petri box. The same functionality is provided by the button.
  - ▷ **Reload:** Reloads the Petri box.
  - ▷ **Save:** Saves the contents of the net editor to the file whose name is displayed underneath the menu bar. The same functionality is provided by the button.
  - ▷ **Save as:** Opens the file browser to save the contents of the net editor to another file.
  - ▷ **Save Selection as:** Opens the file browser to save all currently selected objects (places, transitions, arcs, blocks and boxes – with all their annotations – and comments) to another file.
  - ▷ **Export to PostScript**
    - ○ **All Objects:** A file browser is opened to choose the name of a file to which all objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) are exported in PostScript format. The net examples in the appendix have been produced in this way.
    - ○ **Selection:** A file browser is opened to choose the name of a file to which all currently selected objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) are exported in PostScript format.
    - ○ **Entire Screen:** A file browser is opened to choose the name of a file to which all objects (places, transitions, arcs, blocks, boxes and currently displayed kinds of annotations and comments) that are currently visible on the screen are exported in PostScript format.
  - ▷ **Quit:** Closes the net editor. If there are unsaved changes a dialog box asks whether or not these changes should be saved.
    Note: Some changes like moving places or changing a current marking are not recognised. If you would like to keep these changes you should use the menu entry **Save** first.
- **Edit**
  - ▷ **New Place:** Switches the mode of the net editor to place insertion. The shape of the cursor changes to an internal (unmarked) place. The user can create such a new place by clicking at the desired position of the drawing canvas. The same functionality is provided by the button.

▷ **New Transition:** Switches the mode of the net editor to transition insertion. The shape of the cursor changes to a transition. The user can create such a new transition by clicking at the desired position of the drawing canvas. The same functionality is provided by the button.

▷ **New PTransition:** Switches the mode of the net editor to phantom transition insertion. The shape of the cursor changes to a transition. The user can create such a new phantom transition by clicking at the desired position of the drawing canvas.

▷ **New Block:** Switches the mode of the net editor to block insertion. The shape of the cursor changes to a block. The user can create such a new block (without meaning, an empty block list, without an associated box and an 'exposed' visibility flag) by clicking at the desired position of the drawing canvas.

▷ **New Box:** Switches the mode of the net editor to box insertion. The shape of the cursor changes to a box. The user can create such a new box (without meaning, an empty block list and a set visibility flag) by clicking at the desired position of the drawing canvas.

▷ **New Arc:** Switches the mode of the net editor to arc insertion. The user can determine first the start node and then the end node of the arc by clicking at the desired place (transition) and transition (place, resp.). The new arc has weight `1` and its 'invisible' flag is not set. The same functionality is provided by the button.

▷ **New PArc:** Switches the mode of the net editor to phantom arc insertion. The user can determine first the start node and then the end node of the phantom arc by clicking at the desired place (phantom transition) and phantom transition (place, resp.). Phantom arcs do not carry any inscription; nor they have an 'invisible' flag.

▷ **Create Comment:** Opens a window where a comment may be entered which is then positioned (almost arbitrarily) at the canvas. The same functionality is provided by the button.
Note: We recommend to use the **c**-key because this has the advantage that the comment is inserted at the current cursor position.

▷ **Cut:** Cuts the current selection (i.e. deletes the selected objects and stores it in the internal clipboard for re-insertion using the **paste** item).

▷ **Copy:** Copies the current selection (i.e. stores it in the internal clipboard for insertion using the **paste** item).

▷ **Paste:** Pastes the content of the clipboard (previously set by a **cut** or **copy** operation) as a selection to the canvas.

▷ **Find:** The dialog box shown in figure 9.8 is opened. It allows to find places, (also phantom) transitions, blocks or boxes by name.

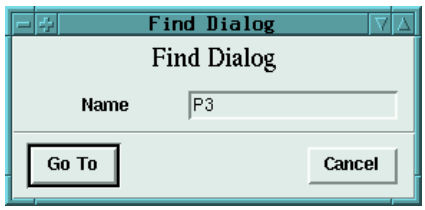


Figure 9.8: Find dialog window

▷ **Select all:** Selects the whole net. This is particularly important if, for instance, the whole net should be synchronised.
Note: A selection can be de-selected by pressing the **Esc** key.

▷ **Delete Elements:** Deletes the currently selected elements.

▷ **Delete Net:** Clears the editor (i.e., deletes the current net but not the file). The same functionality is provided by the button.

▷ **Repaint:** Repaints the whole canvas. This may sometimes be necessary because some repainting is still not done automatically.

• Hierarchy

▷ **Hierarchy Tree:** The dialog box shown in figure 9.9 is opened. It displays the hierarchy of the boxes and blocks involved in the current net and allows to change their visibility. A click on the **Exposed Contents all** button shows the contents of all boxes and blocks. The same functionality is provided by the button.

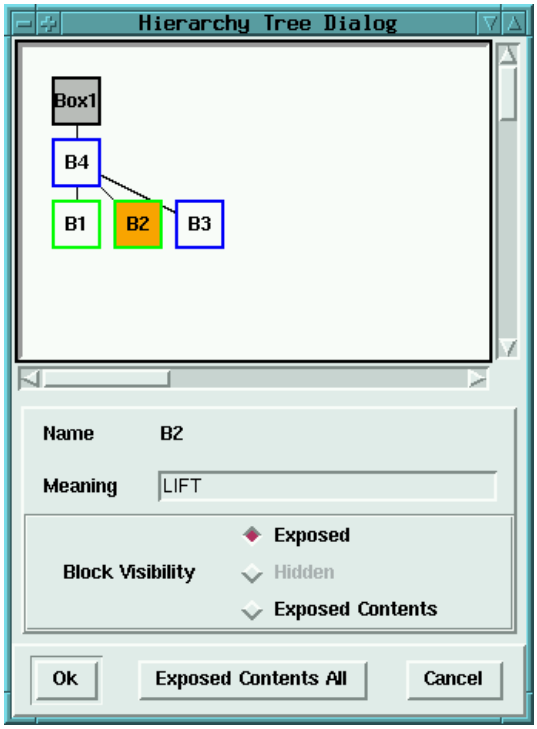


Figure 9.9: Hierarchy tree dialog window

▷ **Block List:** The dialog box shown in figure 9.10 is opened. It displays a list of all the blocks involved in the current net and allows to change their visibility. A click on the **Exposed Contents all** button shows the contents of all blocks. In contrast to the **Exposed Contents all** button of the hierarchy tree dialog box (or the

button) only the contents of all blocks are exposed.

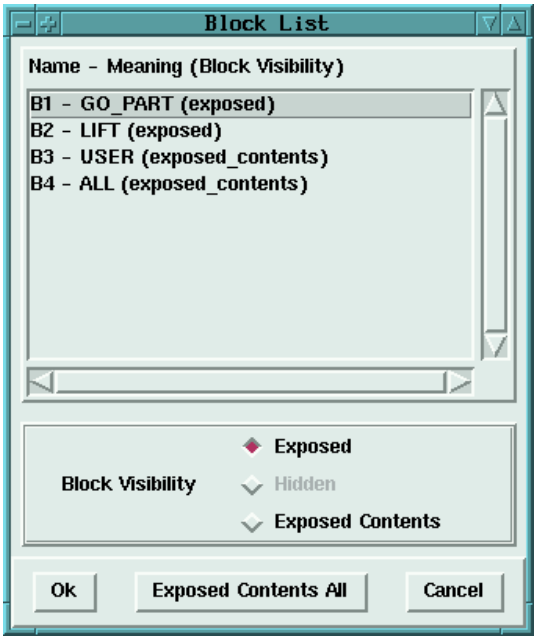


Figure 9.10: Block list dialog window

▷ **Other Nets List:** The dialog box shown in figure 9.11 is opened. In principle, this window allows to handle different nets simultaneously. Unfortunately, this functionality is clearly in a beta testing state. Thus, we do not recommend to use this feature.

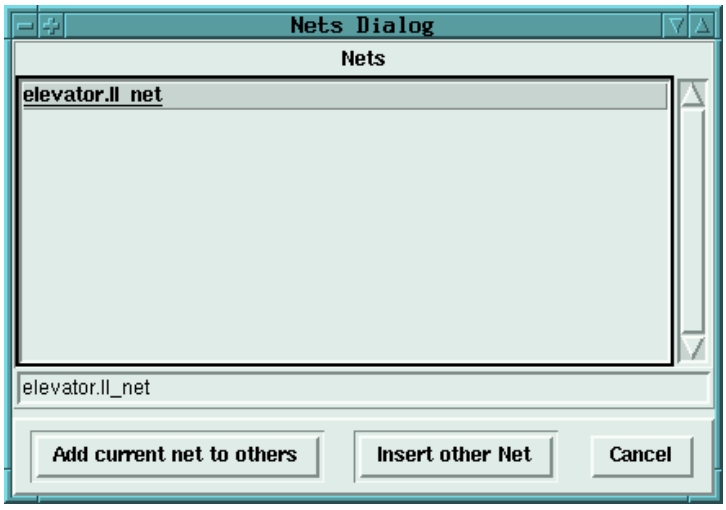


Figure 9.11: Other net list dialog window

• Composition

▷ **Nela:** The dialog box shown in figure 9.12 is opened. It offers a comfortable interface to (most of) the functionality provided in the rest of the menu.

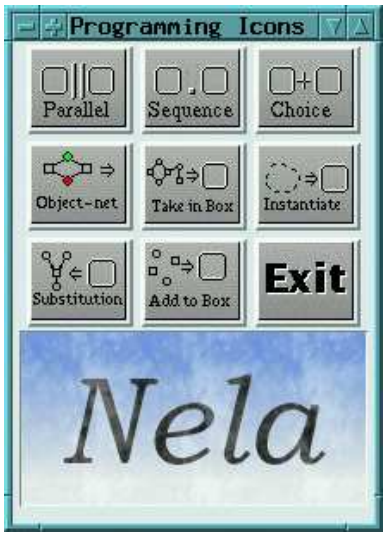


Figure 9.12: The 'Nela' window

▷ **Take in Box:** The current selection is taken as the basis for a new box. This means, that recursively all objects belonging to a block to which one of the selected (or already added) objects belong are also added to the box. This offers a simple method of constructing boxes which may then be used for composition operations. The same functionality is provided by the following icon of the 'Nela' window:

▷ **Add to Box:** The current selection is taken as the basis for an extension of an existing box. This means, that recursively all objects belonging to a block to which one of the selected (or already added) objects belong are also added to the box which is chosen using the dialog window shown in figure 9.13. The same functionality is provided by the following icon of the 'Nela' window:

▷ **Instantiate Box:** Instantiation of low-level boxes does not work yet. Thus, we highly recommend not to invoke this menu item and not to use the corresponding icon of the 'Nela' window:

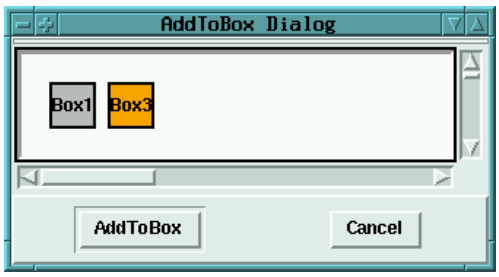


Figure 9.13: Add to box dialog window

▷ **Parallel:** The dialog window shown in figure 9.14 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in parallel. The same functionality is provided by the following icon of the 'Nela' window:

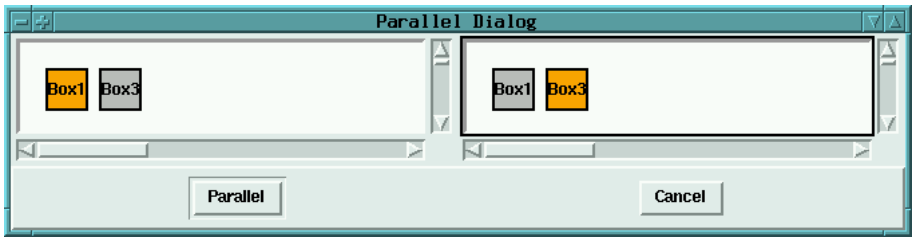


Figure 9.14: Parallel composition dialog window

▷ **Sequence:** The dialog window shown in figure 9.15 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in sequence. The same functionality is provided by the following icon of the 'Nela' window:

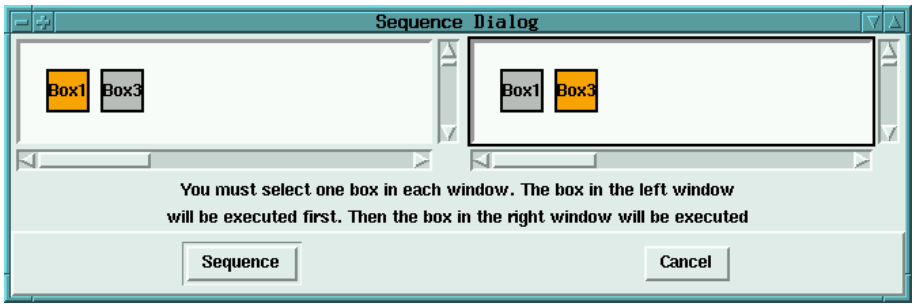
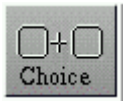


Figure 9.15: Sequential composition dialog window

▷ **Choice:** The dialog window shown in figure 9.16 is opened. It allows to compose two boxes (which are chosen in the left and right part, respectively, of the dialog window) in choice.

The same functionality is provided by the following icon of the 'Nela' window: 

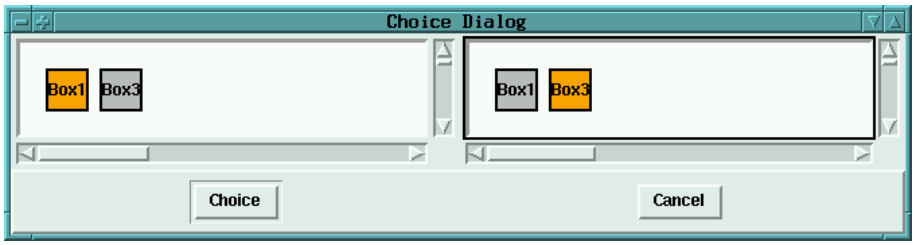
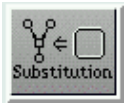


Figure 9.16: Choice composition dialog window

▹ **Transition Substitution:** The dialog window shown in figure 9.17 is opened. It allows to substitute a transition (which is chosen in the list in the upper part of the window) by the box chosen in the main part (the canvas) of the window. The same dialog window is opened by clicking on the following icon of the 'Nela' window: 

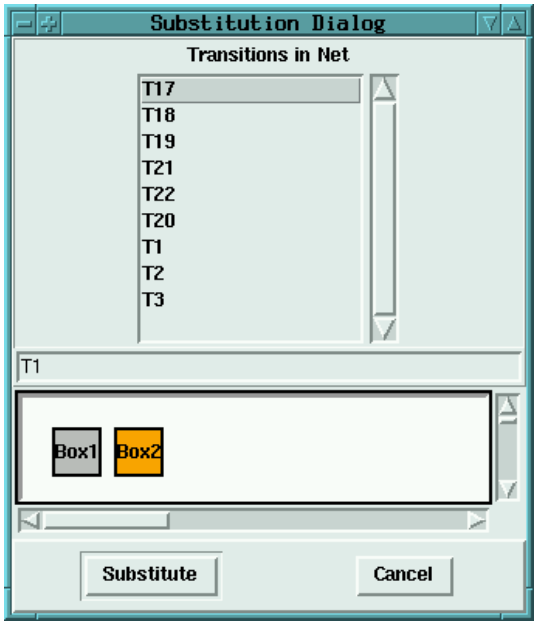
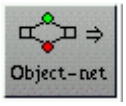


Figure 9.17: Transition substitution dialog window

▹ **Load Module:** A file selection box is opened (in the current directory) which allows to load a low-level net as a module. The same functionality is provided by the following icon of the 'Nela' window: 

- **Action**

  ▷ **Show Active Transition:** Colours all currently enabled transitions (synchronisation as well as phantom transitions) yellow. The mode can be exited by (clicking in the canvas and) pressing the **Esc** key.
  Note: If an enabled transition belongs to a block whose contents are currently not visible, then the icon of the block is coloured in yellow. Clicking on the (yellow) block icon fires the transition.

  ▷ **Active Transition Dialog:** The dialog window shown in figure 9.18 is opened.

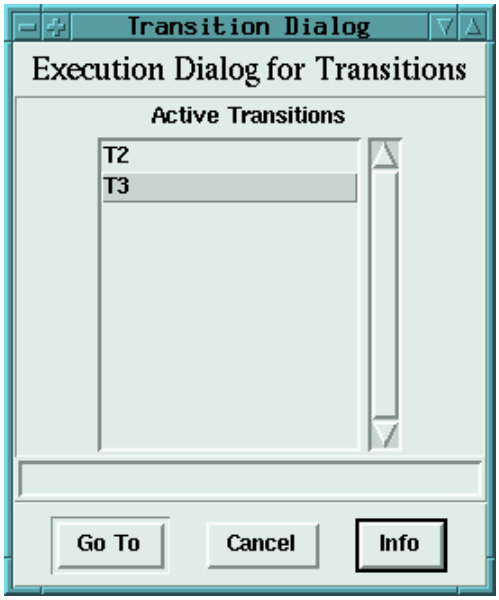


Figure 9.18: Active transitions dialog window

It allows to get more information concerning the currently active transitions listed in the main part of the dialog box. After selecting one of the transitions (by a mouse-click) you may have this transition (or the corresponding block icon) centred and coloured in orange in the canvas of the net editor by pressing the **Go To** button.

Moreover, you may open an additional dialog window shown in figure 9.19 by pressing the **Info** button. The main part of this dialog box displays the list of pre-places (on the left hand side) and the list of post-places (on the right hand side). After selecting one place of such a list (which automatically colours this place in orange and focuses the canvas of the net editor to this place), a couple of information is given: the corresponding arc weight, the meaning of the place, its initial and its current marking.

▷ **All Transition Dialog:** A window which is similar to the active transition dialog is opened. It gives a list of all the transitions (in alphabetical order) and offers the same functionality.

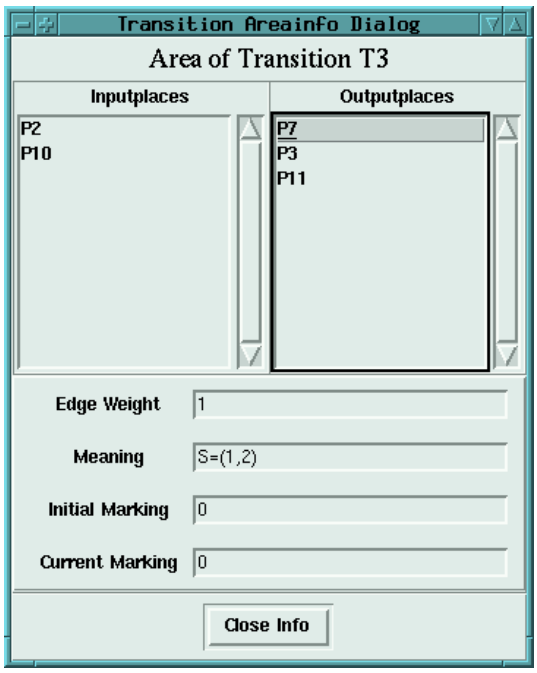


Figure 9.19: Transition area dialog window

▷ **Execute Transition:** If a transition sequence has been stored (either by **Execute Transition List** or by selecting **Sim LL** in one of the analysis or verification windows) the next transition in the list is executed. Otherwise, the mode of the editor is switched so that all currently enabled transitions (or the corresponding blocks if they are exposed) are coloured yellow. They can be executed by a mouse click. The mode can be exited by pressing the **Esc** key. The same functionality is provided by the following icon:

▷ **Execute Transition List:** The dialog window shown in figure 9.20 is opened. It allows you to enter a transition sequence and (optionally) an additional 'loop' transition sequence. Clicking on the **OK** button stores the given sequences. Subsequent invocations of either **Execute Transition** (e.g., by clicking on the button) or of **Animate** (e.g., by clicking on the button) will start to execute these lists.

Figure 9.20: Execute transitions list dialog window

▷ **Animate:** If a transition sequence has been stored (either by **Execute Transition List** or by selecting **Sim LL** in one of the analysis or verification windows) the transitions in the list are executed. Otherwise, the mode of the editor is switched so that an automatic simulation is started. During the simulation all currently enabled transitions are coloured yellow. One of them is chosen randomly and is executed. The surrounding arcs are coloured green to show the token flow. The mode can be exited by pressing the **Esc** key (or clicking on the button). The same functionality is provided by the following icon:

▷ **Reset to Initial Marking:** Resets the marking to the initial marking from which a new simulation of the net can be started.

▷ **Clear Transition List:** The transition list(s) entered using the **Execute Transition List** item or by clicking on a **Sim LL** button of one of the analysis or verification windows is cleared. As a consequence, an animation may continue in random mode.

▷ **Show Reference:** If the Petri box belongs to a B(PN)^2 program (or an SDL specification), the correspondence between, for instance, a place of the net and a point in the control flow of the B(PN)^2 program (or SDL specification) or between a transition in the net and actions in the B(PN)^2 program (or, e.g., a task in the SDL specification) can be demonstrated. The mode of the net editor is switched so that the corresponding text segment is highlighted in aquamarine if the user clicks on a transition, a place or a block. The mode can be exited by pressing the **Esc** key. The same functionality is provided by the following icon:

• Option

▷ **Display**

◦ **Names:** Chooses whether or not the names of places, (synchronisation as well as phantom) transitions, blocks and boxes are displayed.

◦ **Meanings:** Chooses whether or not the meanings of places, (synchronisation as well as phantom) transitions, blocks and boxes are displayed.

◦ **Arc Annotations:** Chooses whether or not the arc weights are displayed.

◦ **Arc during Moving:** Chooses whether or not the display of arcs is updated during a move of connected places and/or transitions. This may slow down move operations considerably.

◦ **Time:** The new version of the low-level net editor allows the user to specify an earliest and a latest firing time for a transition if this option is checked. These additional attributes may be entered in the extended transition dialog shown in figure 9.21.
Note: Although the editor allows to specify these time related attributes and also saves them to file, the integrated simulator does not take this information into account.

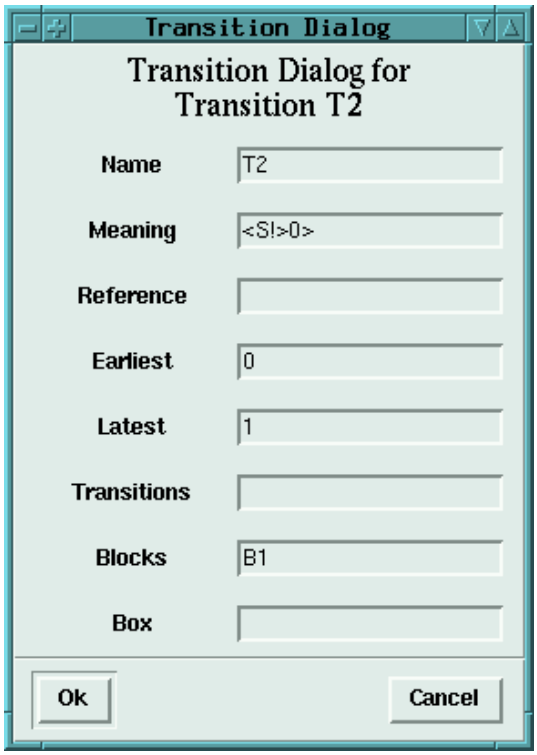


Figure 9.21: Transition dialog window (with time)

▷ **Change Font:** The dialog window shown in figure 9.22 is opened. It allows you to select a font for names of places, names of transitions, meanings of places, meanings of transitions, all comments or selected comments. In the middle part of the window a font is selected. An example of a text in the selected font (or a message that the chosen font does not exist) is shown in the upper part of the dialog window. A click on one of the icons in the lower part of the window changes the font for the corresponding selection. Clicking on the **Cancel** button closes the dialog window.

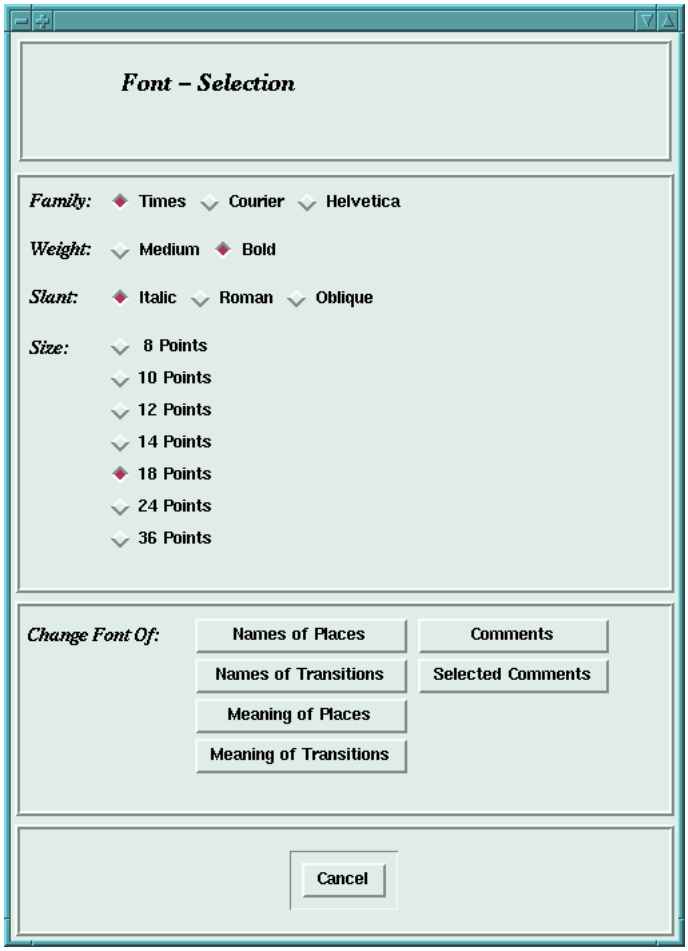


Figure 9.22: Change font dialog window

▷ **Show**

- ○ **Synchronisation Transitions:** Chooses whether or not the synchronisation transitions (together with their adjacent arcs) are shown. In case of a lot of synchronous communication this may turn the visualisation of a net into something almost unreadable.

- ○ **Phantom Transitions:** Chooses whether or not the phantom transitions are shown (together with their adjacent phantom arcs).

- ○ **Invisible Arcs:** Chooses whether or not invisible arcs are displayed. An invisible flag is set by the compilers 'B(PN)^2 ⇒ M-net' and 'SDL ⇒ M-net' (and forwarded by the unfolder 'M-net ⇒ Petri box') for arcs between the data boxes of the net and the control flow part of the net to enhance the readability of the constructed nets (see page 33).

▷ **Move Options:** The dialog window shown in figure 9.23 is opened. It allows to specify whether or not nodes are moved

if the block (or box) they belong to is moved and whether or not selections are moved together.

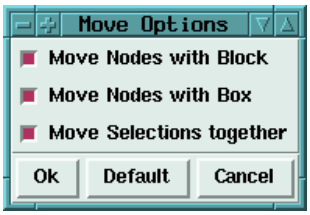


Figure 9.23: Move options dialog window

▷ **Highlight Execution:** If the Petri box belongs to a B(PN)^2 program (or SDL specification), the correspondence between a transition in the net and actions in the B(PN)^2 program (or SDL specification) can be demonstrated. The mode of the net editor is switched so that if a transition is executed, the corresponding text segment is highlighted in aquamarine in the B(PN)^2 (or SDL) editor.

▷ **Reference Format:** This options is for debugging purposes only and should not be touched! It changes the format of internal communication between the net editor and the PEP tool.

▷ **Zoom:** Switches amongst different zoom factors.

Note: The status of a place is always displayed by appropriate colours. Entry places are coloured green, exit places are coloured red and internal places are uncoloured.

• Help

▷ **Info:** Shows an about box.

Chapter 10

The formula editor

The formula editor (cf. figure 10.1) is a text editor with some temporal logic formula specific features. It allows to enter properties to be checked by one of the integrated model checker.

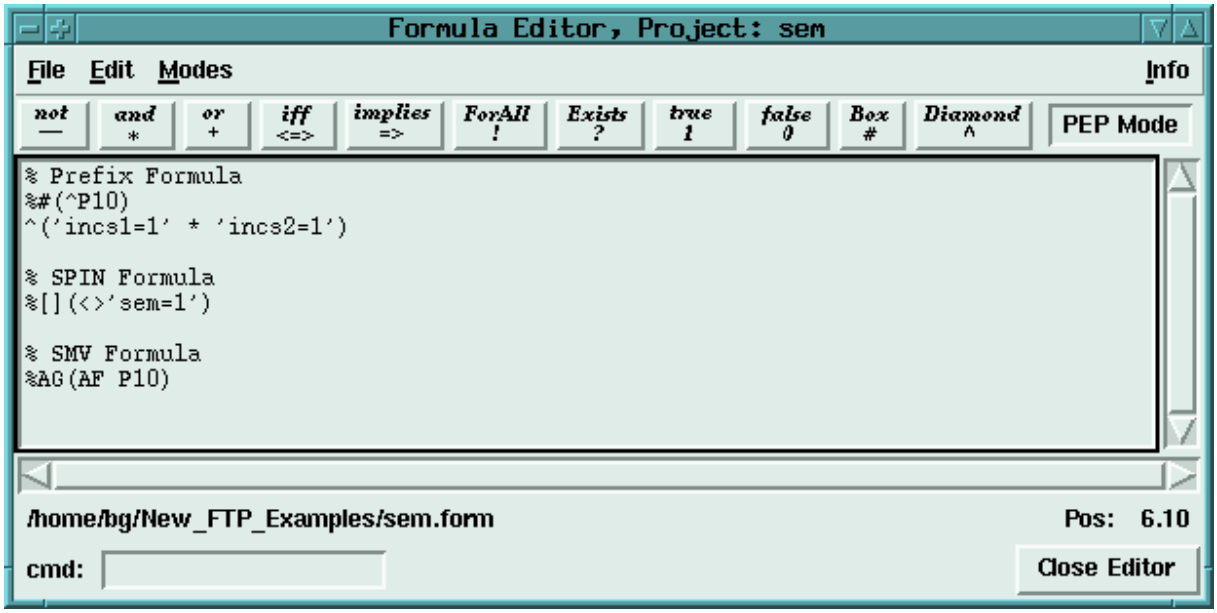


Figure 10.1: Formula editor

10.1 Description of the formula editor window

- In the main part of the editor window formulae can be edited.
- Under the editing canvas of the editor the name of the currently edited file is displayed.
- The current cursor position is shown in the lower right part of the window.
- In the lower left part an additional area allows to type in commands (such as `w` to save the file) directly.
- If the keyboard offers this possibility the **cut**, **copy** and **paste** keys can be used.

- The icons underneath the pull-down menu (except the far right one) allow to type parts of a formula by clicking on them. Thus the user need not be familiar with the abbreviations of the operators in the different supported logics.

  The following tables (table 10.1 for the logic supported by the prefix checker, table 10.2 for the integrated SPIN verifier, and table 10.3 for the integrated SMV verifier) list, for each of the supported model checkers and for each button, the mathematical symbol together with the characters that are inserted if you click on the button.

Button	Inserted characters	Mathematical symbol (S_4)
not	–	¬
and	*	∧
or	+	∨
iff	<=>	⇔
implies	=>	⇒
Forall	!	∀
Exists	?	∃
True	1	true
False	0	false
Box	#	□ – always
Diamond	^	◇ – possibly

Table 10.1: Logic of the prefix checker

Button	Inserted characters	Mathematical symbol (LTL)
always	[]	□ – always
eventually	<>	◇ – eventually
strong until	U	U
Dual of until	V	V
and	&&	∧
or	||	∨
not	!	¬
implies	– >	⇒
iff	< – >	⇔
True	true	true
False	false	false

Table 10.2: Logic of the SPIN checker

- Formulae involve place names of a net (e.g., `# Place-name`). Furthermore variable bindings can be used (e.g., `# 'Variable-name = Value'`). They are automatically transformed into place names. Brackets can be used to structure formulae (i.e. to overrule priorities of the operators). You can keep different formulae within one formula file if you prefix the currently unused formulae with an % (see figure 10.1).

Note: Using characters as in the middle column of the tables is a compromise because formulae must also be interpretable as ASCII text.

Button	Inserted characters	Mathematical symbol (CTL)
all path	**A**	**A**
exist path	**E**	**E**
next	**X**	**X**
future	**F**	**F**
globally	**G**	**G**
until	**U**	**U**
not	!	$\neg$
and	&	$\wedge$
or	|	$\vee$
implies	$->$	$\Rightarrow$
iff	$<->$	$\Leftrightarrow$

Table 10.3: Logic of the SMV checker

10.2 Menus of the formula editor

- **File**
  - ▷ **Open:** Opens the file browser to select a formula file.
  - ▷ **Reload:** Reloads the formula file.
  - ▷ **Save:** Saves the contents of the formula editor to the file whose name is displayed underneath the main part of the window.
  - ▷ **Save as:** Opens the file browser to save the contents of the formula editor to another file.
  - ▷ **Import File:** Opens the file browser to select the name of a file to be loaded in the formula editor at the current cursor position.
  - ▷ **Export Clipboard:** Opens the file browser to select the name of another file. The contents of the clipboard are saved to this file. You can either use the **copy** button on your keyboard or the menu entry **Edit▷Copy** to copy the selection to the clipboard.
  - ▷ **Quit:** Closes the formula editor. If there are unsaved changes a dialog box asks whether or not these changes should be saved.
- **Edit**
  - ▷ **Cut:** Cuts the current selection. It is saved to the clipboard and can be used for further **paste** operations.
  - ▷ **Copy:** Copies the current selection to the clipboard.
  - ▷ **Paste:** Pastes the contents of the clipboard to the current cursor position.
  - ▷ **Search:** Opens a dialog box as shown in figure 7.3 on page 51.

- ▷ **Replace:** Opens a dialog box as shown in figure 7.4 on page 52.

- **Modes** These menu items are used to switch the icon bar above the editing canvas.
  - ▷ **Enter PEP Mode:** Displays the icons for the logic supported by the prefix checker.
  - ▷ **Enter SPIN Mode:** Displays the icons for the logic supported by the integrated SPIN verifier.
  - ▷ **Enter SMV Mode:** Displays the icons for the logic supported by the integrated SMV verifier.
- **Info**
  - ▷ **Info:** Shows the about box for the text editors integrated in the PEP tool.

Chapter 11

The SDL editor

The SDL editor (shown in figure 11.1) is a text editor with some SDL specific features. It is very similar to the B(PN)^2 editor. We will, however, give a complete description in order to avoid that users who 'only' want to work with SDL have to read the documentation of the B(PN)^2 editor.

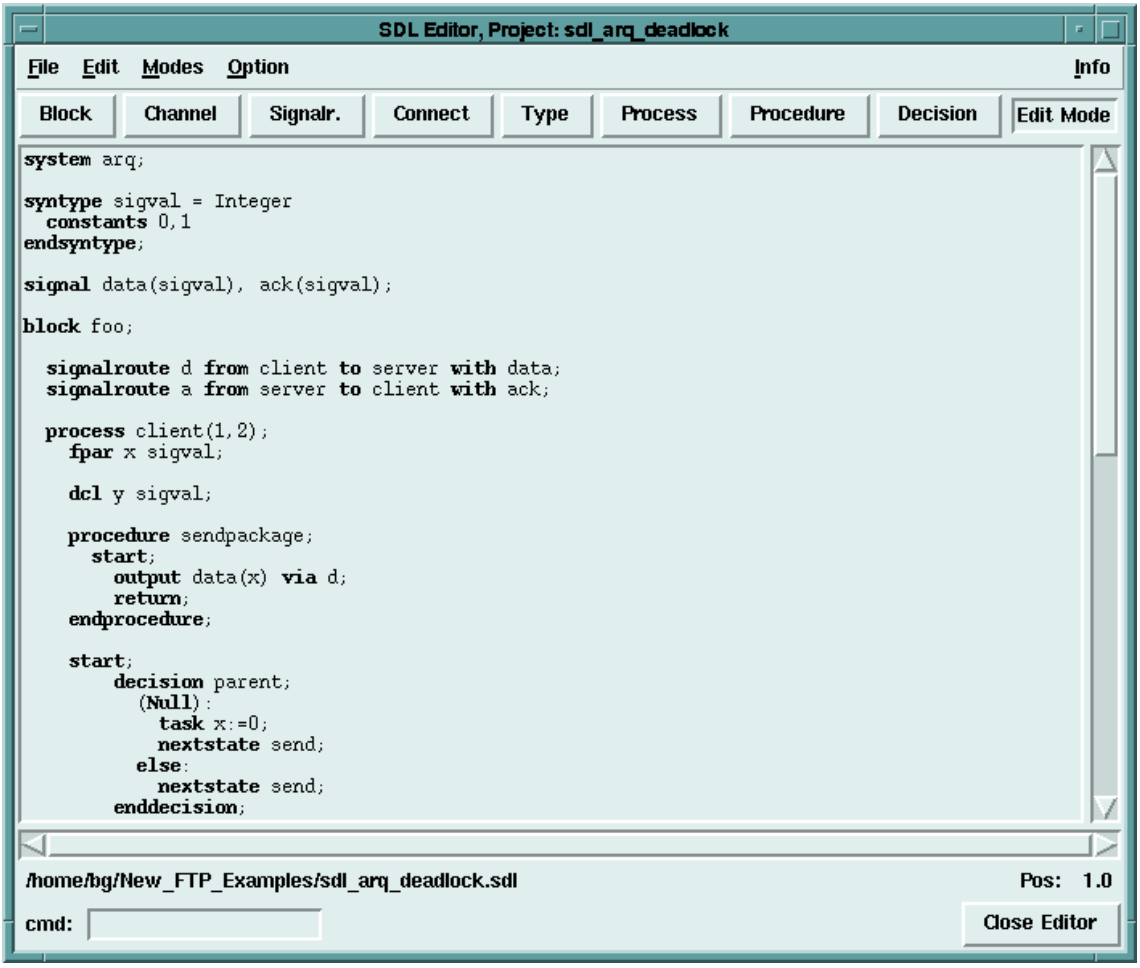


Figure 11.1: SDL editor

11.1 Description of the SDL editor window

- In the main part of the editor window an SDL specification can be edited.

- Under the editing canvas of the editor the name of the currently edited file is displayed.
- The current cursor position is shown in the lower right part of the window.
- In the lower left part an additional area allows to type in commands (such as `w` to save the file) directly.
- If the keyboard offers this possibility the **cut**, **copy** and **paste** keys can be used.
- All but the last icons underneath the pull-down menu are macros for typing part of an SDL specification.
  - The **Block** button pastes the declaration of a block to the current cursor position.
  - The **Channel** button pastes the declaration of a channel to the current cursor position.
  - The **Signalr.** button pastes the declaration of a signal route to the current cursor position.
  - The **Connect** button pastes the declaration of a connector to the current cursor position.
  - The **Type** button pastes the declaration of a type to the current cursor position.
  - The **Process** button pastes the declaration of a process to the current cursor position.
  - The **Procedure** button pastes the declaration of a procedure to the current cursor position.
  - The **Decision** button pastes a decision construct to the current cursor position.

11.2 Menus of the SDL editor

• File

▷ **Open:** Opens the file browser to select an SDL specification.

▷ **Reload:** Reloads the SDL specification (e.g. if you made a mistake and want to go back to the previous version).

▷ **Save:** Saves the contents of the SDL editor to a file. The name of the file is displayed underneath the main part of the window.

▷ **Save as:** Opens the file browser to save the contents of the SDL editor to a file.

▷ **Edit SDL Properties:** Opens the editor (shown in figure 11.2) which allows you to enter additional restrictions (such as for the maximum number of concurrently active procedure instances or the capacities of input queues of certain processes or the capacities of channels).

▷ **Import File:** Opens the file browser to select the name of a file to be loaded into the SDL editor at the current cursor position.

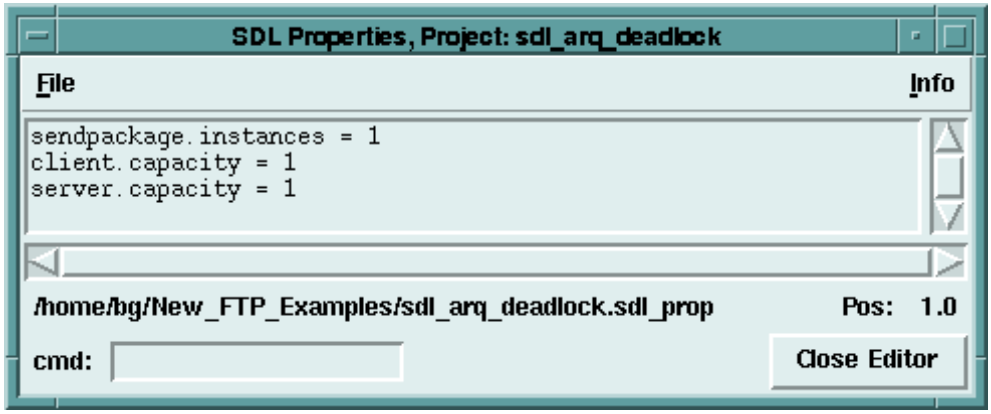


Figure 11.2: SDL properties editor

▷ **Export Clipboard:** Opens the file browser to select the name of a file. The contents of the clipboard are saved to this file. You can either use the **copy** button on your keyboard or the menu entry **Edit▷Copy** to copy a selection to the clipboard. Be careful, the clipboard is not local to the PEP tool, it can be changed e.g. if you hit the **copy** key on your keyboard even if you work with a totally different program!

▷ **Quit:** Closes the SDL editor. If there are unsaved changes a dialog box (see figure 7.2 on page 51) asks whether or not these changes should be saved before the editor is closed.

- **Edit**

▷ **Cut:** Cuts the current selection (a part of the text marked using the left mouse button). It is saved to the clipboard and can be used for further **paste** or **export** operations.

▷ **Copy:** Copies the current selection to the clipboard.

▷ **Paste:** Pastes the contents of the clipboard to the current cursor position.

▷ **Search:** Opens the dialog box (already shown in figure 7.3 on page 52) that offers a comfortable searching facility described on page 52.

▷ **Replace:** Opens the dialog box (already shown in figure 7.4 on page 52) that offers a comfortable searching facility described on page 52.

- **Modes** The SDL editor may be used in three different modes:

▷ **Enter Edit Mode:** The 'usual' editing mode which allows to enter (or change) an SDL specification is chosen. The icons which allow to paste templates (e.g. for a process declaration) are only shown in this mode.

▷ **Enter Reference Mode:** A reference mode (which is explained in more detail below) is entered. The general purpose of this mode is to highlight

(or paste) counterparts of an SDL selection (e.g., a task) in another editor (e.g., the corresponding transition(s) in the high-level Petri net editor). The exact behaviour of this highlighting (or pasting) depends on the decisions taken in the **Options** menu.

▷ **Enter Simulation Mode:** A simulation mode which allows the simulation of a low-level Petri net transition sequence on the SDL level is entered. This is particularly interesting for the simulation of analysis and/or verification results.
Note: The editor enters this mode automatically if the **Sim SDL** button of one of the analysis/verification windows is pushed.

• **Option** The behaviour of the SDL editor in the **Reference Mode** is controlled by a number of options:

▷ **Show Reference in HL:** If this option is checked references are highlighted in the high-level net editor.

▷ **Show Reference in LL:** If this option is checked references are highlighted in the low-level net editor.

▷ **Show Reference in Formula:** If this option is checked references are pasted in the formula editor (at the current cursor position).

▷ **Show Reference partial:** If this option is checked also counterparts (e.g., transitions) which are not only related to the current selection but also to other parts of the SDL specification are highlighted/pasted.

▷ **Show Reference exact:** If this option is checked only counterparts (e.g., transitions) which are exactly related to the current selection (e.g., all – and no more than – the currently selected task) are highlighted/pasted.

▷ **Show all Values:** This option is relevant for place related references. If the option is checked the type of the places under consideration is not taken into account. Otherwise the value given in the **Value** field has to be matched. This is particularly interesting for places which hold the value of a variable.

▷ **Preferences:** The dialog box shown in figure 11.3 is opened which allows to specify the delay between two action occurrences in the simulation mode.

11.2.1 Reference Mode

As mentioned above, the SDL editor has a reference mode. This offers a facility for determining, for instance, which place in a Petri box (or in the M-net) corresponds to a given

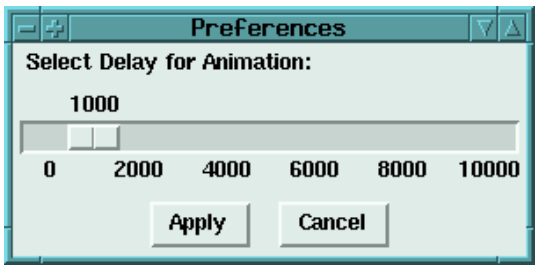


Figure 11.3: Preferences window of the SDL editor

point in the control flow (or a value of a certain variable) of the associated SDL specification, or which transition(s) in the Petri box correspond(s) to a task of the specification.

When the SDL editor is in the reference mode the parts of the specification for which references exist (i.e. those parts of the specification which have counter parts on the net level) are coloured in blue each time the mouse cursor is positioned on them. A task, for instance, may then be selected in three different ways:

1. Selecting the task with the middle mouse button asks for the transition(s) corresponding to the task itself. In the SDL editor the task becomes coloured in green in order to show the kind of selection.

2. A selection with the left mouse button refers to the place(s) which correspond(s) to the point in the control flow directly before (or at) the task (this colours the action yellow).

3. Finally, a click with the right mouse button starts an ask for the place(s) which correspond(s) to the point in the control flow directly after the task (this colours the task red, or orange if the task was already coloured yellow).

For the time being, identifiers of variables, channels or signal routes in a declaration may only be selected with the middle mouse button. Such a selection refers to the place(s) which, e.g., hold(s) the value of the variable.

Using this selection method selections (or multi-selections) may be constructed. Moreover, an entry of a value in the **Value** field (above the canvas of the editor) may restrict place related references. Once, the desired selection (or multi-selection) has been established a click on the **Send** button performs the highlight (or paste) operation. In order to prepare a new selection, a click on the **Clear** button restarts with an empty selection.

After selecting, e.g., a task in the SDL specification, two of the entries in the **Reference** menu can be used to paste the name(s) of the corresponding transition(s) to the formula editor. In this way temporal logic formulae can be made to refer to points of the control flow or tasks (using macros like `LIVE`) of the specification, even though technically speaking, the temporal logic refers only to places of a net.

Note: Currently there is one tricky point:

- As mentioned above, an entry in the **Value** field restricts place related references (if the option **Show all Values** is not checked). Using this option appropriately requires deep knowledge of the underlying Petri net semantics. For instance, a variable declaration **dcl** $x : \{2\}$ may produce a place with type $\{2, \bullet, 1\}$ (if it belongs to a process with a process identifier set containing only the process identifier 1). As a consequence, a value `(2,dot,1)` has to be requested.

Anyway, a good way to become familiar with references of all these kinds between an SDL specification and a net is to use the menu entry **Show Reference** (in combination with the option **Highlight Execution** in one of the net editors (see pages 77 and 101).

• Info

▷ **Info:** Shows the about box for the text editors integrated in the PEP tool.
Note: It is worth doing it if you are an Enterprise fan.

Index

About, 12
Action
  Active Transition Dialog, 75, 99
  All Transition Dialog, 75, 99
  Animate, 76, 101
  B(PN)^2 Syntax Check, 53
  B(PN)^2 To TEX, 53
  Clear Transition List, 77, 101
  Convert to New Net Format, 77
  Execute Transition, 75, 100
  Execute Transition List, 76, 100
  Reset to Initial Marking, 77, 101
  Show Active Transition, 74, 99
  Show Reference, 77, 101
  Zoom, 83, 104
Algorithm
  Invocation, 14
Algorithms
  Bounded System, 19
  Deadlock-free System, 21
  FC System, 19
  FC2 Deadlock-free System, 21
  LP Deadlock-free System, 21
  LP Reachability, 19
  Minimize Net, 18
  Model Checker (Safe), 23
  Model Checker (SMV), 24
  Model Checker (SPIN), 23
  Model Checker (T-System), 23
  Options, 14
  Reachability, 19
  Safe System, 19
  T-System, 18
Analyse
  Bounded System, 19
  Deadlock-free System, 21
  FC System, 19
  FC2 Deadlock-free System, 21
  LP Deadlock-free System, 21
  LP Reachability, 19
  Model Checker (Safe), 23
  Model Checker (SMV), 24
  Model Checker (SPIN), 23
  Model Checker (T-System), 23
  Reachability, 19
  Safe System, 19
  T-System, 18

B(PN)^2 Editor, 49
  Action
    B(PN)^2 Syntax Check, 53
    B(PN)^2 To TEX, 53
  Edit
    Add Comment, 51
    Copy, 51
    Cut, 51
    Delete Comment, 51
    Paste, 51
    Replace, 51
    Search, 51
  File
    Export Clipboard, 50
    Import File, 50
    Open, 50
    Quit, 51
    Reload, 50
    Save, 50
    Save as, 50
  Info
    Info, 56
  Modes, 54
    Enter Edit Mode, 54
    Enter Reference Mode, 54
    Enter Simulation Mode, 54
  Option, 54
    Preferences, 55
    Show all Values, 55
    Show Reference exact, 54
    Show Reference in Formula, 54
    Show Reference in HL, 54
    Show Reference in LL, 54
    Show Reference in PFA, 54

Show Reference partial, 54
B(PN)² Expander
Options, 26, 42
B(PN)² References, 55
B(PN)²⇒C Source
Invocation, 17
B(PN)²⇒Expanded B(PN)²
Invocation, 17
B(PN)²⇒HL Net
Invocation, 17
B(PN)²⇒LL Net (via PBC)
Invocation, 17
B(PN)²⇒LL Net (via HL)
Invocation, 17
B(PN)²⇒M-Net
Options, 25, 31
B(PN)²⇒PBC
Invocation, 16
Options, 25, 29
Bounded System
Invocation, 19
Boundedness Analyser
Options, 26, 36

Close
Discard Files, 16
Save Files, 16
Close Project, 15
Compiler
B(PN)²⇒C Source, 17
B(PN)²⇒Expanded B(PN)², 17
B(PN)²⇒HL Net, 17
B(PN)²⇒LL Net (via PBC), 17
B(PN)²⇒LL Net (via HL), 17
B(PN)²⇒PBC, 16
Expanded B(PN)²⇒HL Net, 17
HL Net⇒LL Net, 17
Invocation, 14
LL Net⇒FC2, 18
LL Net⇒INA, 18
LL Net⇒Prefix, 17
LL Net⇒SPIN, 18
Options, 14
PBC⇒LL Net, 16
PFA⇒B(PN)², 18
PFA⇒HL Net, 18
SDL⇒HL Net, 18
Sync HL Net, 17
Copy
Project, 10
Cosmetics, 28
Defaults, 28
Edit, 28
Load PEP Default, 28
Load User Default, 28
Save as User Default, 28
Show Icons, 28
Show References, 28

Deadlock-free System
Invocation, 21
Defaults, 28
Delete
Project, 11
Directory
Change, 10
Make, 10
Reread, 10

Edit, 28
Add Comment, 51
Close
Discard Files, 16
Save Files, 16
Copy, 51, 66, 93, 107, 111
Project, 10
Create Comment, 66, 93
Cut, 51, 66, 93, 107, 111
Delete
Project, 11
Delete Comment, 51
Delete Elements, 67, 94
Delete Net, 67, 94
Edit Project Info, 16
Find, 67, 93
New Arc, 66, 93
New Block, 66, 93
New Box, 66, 93
New PArc, 66, 93
New Place, 65, 92
New PTransition, 66, 93
New Transition, 66, 93
Open, 15
Animator, 16
B(PN)² Editor, 15
C Source, 15
Document, 15
Expanded B(PN)², 15

Formula, 15
HL Net, 15
INA, 15
LL Net, 15
PBC, 15
PFA, 15
SDL, 16
SDL Properties, 16
Simulator, 15
Paste, 51, 66, 93, 107, 111
Rename
Project, 11
Repaint, 67, 94
Replace, 51, 108, 111
Save
Entire Project, 16
Save Project Info, 16
Search, 51, 107, 111
Select all, 67, 94
Select some Nodes, 67
Edit Project Info, 16
Enter PEP Mode, 108
Enter SMV Mode, 108
Enter SPIN Mode, 108
Execute
B(PN)^2⇒C Source, 17
B(PN)^2⇒Expanded B(PN)^2, 17
B(PN)^2⇒HL Net, 17
B(PN)^2⇒LL Net (via PBC), 17
B(PN)^2⇒LL Net (via HL), 17
B(PN)^2⇒PBC, 16
Expanded B(PN)^2⇒HL Net, 17
HL Net⇒LL Net, 17
LL Net⇒FC2, 18
LL Net⇒INA, 18
LL Net⇒Prefix, 17
LL Net⇒SPIN, 18
Minimize Net, 18
PBC⇒LL Net, 16
PFA⇒B(PN)^2, 18
PFA⇒HL Net, 18
SDL⇒HL Net, 18
Sync HL Net, 17
Expanded B(PN)^2⇒HL Net
Invocation, 17

FC System
Invocation, 19
FC2 Deadlock-free System
Invocation, 21
File
Change Directory, 10
Edit SDL Properties, 110
Export Clipboard, 50, 107, 111
Export to PostScript
All Objects, 65, 92
Entire Screen, 65, 92
Selection, 65, 92
Import File, 50, 107, 110
Make Directory, 10
New
Project, 10
Open, 50, 65, 92, 107, 110
Project, 10
Quit, 10, 51, 65, 92, 107, 111
Reload, 50, 65, 92, 107, 110
Reread Directory, 10
Save, 50, 65, 92, 107, 110
Save as, 50, 65, 92, 107, 110
Save Selection as, 65, 92
File Browser, 47
Formula Editor, 105
Edit
Copy, 107
Cut, 107
Paste, 107
Replace, 108
Search, 107
File
Export Clipboard, 107
Import File, 107
Open, 107
Quit, 107
Reload, 107
Save, 107
Save as, 107
Info
Info, 108
Modes
Enter PEP Mode, 108
Enter SMV Mode, 108
Enter SPIN Mode, 108

Help
About, 12
Help Viewer, 27
Info, 84, 104
Help Viewer, 27

Hierarchy
Add to Box, 70, 96
Append Suffix, 71
Block List, 67, 95
Choice, 72, 97
Expression, 73
Hierarchy Tree, 67, 94
Instantiate Box, 70, 96
Load Module, 74, 98
Load Templatenet, 74
Nela, 69, 96
Other Nets List, 69, 95
Parallel, 71, 97
Restriction, 73
Sequence, 71, 97
Synchronisation, 72
Take in Box, 69, 96
Transition Substitution, 72, 98
HL Net Editor
Action
Active Transition Dialog, 75
All Transition Dialog, 75
Animate, 76
Clear Transition List, 77
Convert to New Net Format, 77
Execute Transition, 75
Execute Transition List, 76
Reset to Initial Marking, 77
Show Active Transition, 74
Show Reference, 77
Zoom, 83
Edit
Copy, 66
Create Comment, 66
Cut, 66
Delete Elements, 67
Delete Net, 67
Find, 67
New Arc, 66
New Block, 66
New Box, 66
New PArc, 66
New Place, 65
New PTransition, 66
New Transition, 66
Paste, 66
Repaint, 67
Select all, 67
Select some Nodes, 67
File
Open, 65
Quit, 65
Reload, 65
Save, 65
Save as, 65
Save Selection as, 65
Help
Info, 84
Hierarchy
Add to Box, 70
Append Suffix, 71
Block List, 67
Choice, 72
Expression, 73
Hierarchy Tree, 67
Instantiate Box, 70
Load Module, 74
Load Templatenet, 74
Nela, 69
Other Nets List, 69
Parallel, 71
Restriction, 73
Sequence, 71
Synchronisation, 72
Take in Box, 69
Transition Substitution, 72
Network
Establish Connections, 78
Select Servers, 78
Option
Change Font, 79
Highlight Execution, 82
Move Options, 81
Output Meaning, 82
Random Bindings, 82
Reference Format, 82
Simulation Options, 81
Simulation Statistics, 81
Synchronisation Options, 80
Template Directory, 81
HL Net⇒LL Net
Invocation, 17
Options, 25, 33

Info, 56, 114
Info, 56, 108, 114
Information

Object Relationship, 14
Objects, 14
Invocation, 9

LL Net Editor
Action
Active Transition Dialog, 99
All Transition Dialog, 99
Animate, 101
Clear Transition List, 101
Execute Transition, 100
Execute Transition List, 100
Reset to Initial Marking, 101
Show Active Transition, 99
Show Reference, 101
Zoom, 104
Edit
Copy, 93
Create Comment, 93
Cut, 93
Delete Elements, 94
Delete Net, 94
Find, 93
New Arc, 93
New Block, 93
New Box, 93
New PArc, 93
New Place, 92
New PTransition, 93
New Transition, 93
Paste, 93
Repaint, 94
Select all, 94
File
Open, 92
Quit, 92
Reload, 92
Save, 92
Save as, 92
Save Selection as, 92
Help
Info, 104
Hierarchy
Add to Box, 96
Block List, 95
Choice, 97
Hierarchy Tree, 94
Instantiate Box, 96
Load Module, 98
Nela, 96
Other Nets List, 95
Parallel, 97
Sequence, 97
Take in Box, 96
Transition Substitution, 98
Option
Change Font, 102
Highlight Execution, 104
Move Options, 103
Reference Format, 104
LL Net⇒FC2
Invocation, 18
LL Net⇒INA
Invocation, 18
Options, 43
LL net⇒INA
Options, 26
LL Net⇒Prefix
Invocation, 17
LL Net⇒SPIN
Invocation, 18
Options, 43
LL net⇒SPIN
Options, 26
Load PEP Default, 28
Load User Default, 28
LP Analyser
Options, 26, 37
LP Deadlock-free System
Invocation, 21
LP Reachability
Invocation, 19

M-Net⇒Petri Box
Options, 33
M-net⇒Petri box
Options, 25
Menus
Project Window, 15
Start-up Window, 10
Minimize Net
Invocation, 18
Model Checker (Safe)
Invocation, 23
Options, 26, 39
Model Checker (SMV)
Invocation, 24
Options, 26, 39

Model Checker (SPIN)
Invocation, 23
Options, 26, 40
Model Checker (T-System)
Invocation, 23
Modes, 54, 111
Enter Edit Mode, 54, 111
Enter PEP Mode, 108
Enter Reference Mode, 54, 111
Enter Simulation Mode, 54, 112
Enter SMV Mode, 108
Enter SPIN Mode, 108
Preferences, 55, 112
Show all Values, 55, 112
Show Reference exact, 54, 112
Show Reference in Formula, 54, 112
Show Reference in HL, 54, 112
Show Reference in LL, 54, 112
Show Reference in PFA, 54
Show Reference partial, 54, 112

Network
Establish Connections, 78
Select Servers, 78
New
Project, 10

Objects
Relationship, 14
Open
Animator, 16
B(PN)^2 Editor, 15
C Source, 15
Document, 15
Editors, 14
Expanded B(PN)^2, 15
Formula, 15
HL Net, 15
INA, 15
LL Net, 15
PBC, 15
PFA, 15
Project, 9, 10
SDL, 16
SDL Properties, 16
Simulator, 15
Option, 54, 112
Change Font, 79, 102
Display
Action Terms, 78
Arc Annotations, 79, 101
Arc during Moving, 79, 102
Meanings, 78, 101
Names, 78, 101
Time, 102
Types, 79
Value Terms, 78
Highlight Execution, 82, 104
Move Options, 81, 103
Output Meaning, 82
Random Bindings, 82
Reference Format, 82, 104
Show
Invisible Arcs, 80, 103
Phantom Transitions, 80, 103
Synchronisation Transitions, 80, 103
Simulation Options, 81
Simulation Statistics, 81
Synchronisation Options, 80
Template Directory, 81
Options, 12
Converter Options
B(PN)^2 Expander, 26, 42
B(PN)^2⇒M-Net, 25, 31
B(PN)^2⇒PBC, 25, 29
Boundedness Analyser, 26, 36
HL Net⇒LL Net, 25, 33
LL Net⇒INA, 26, 43
LL Net⇒SPIN, 26, 43
LP Analyser, 26, 37
M-Net⇒Petri Box, 33
M-net⇒Petri box, 25
Model Checker (Safe), 26, 39
Model Checker (SMV), 26, 39
Model Checker (SPIN), 26, 40
PBC⇒LL Net, 25, 30
Petri box⇒INA, 26, 43
Petri box⇒PROMELA, 43
Petri box⇒SPIN, 26
PFA⇒B(PN)^2, 26, 44
PFA⇒HL Net, 26, 43
PFA⇒M-net, 26, 43
Prefix Builder, 26, 35
Reachability Analyser, 26, 37
SDL⇒HL Net, 27, 45
SDL⇒M-net, 27, 45
Sync HL Net, 26, 33

Syntax Check, 25, 29
Debug Output
Show prjx, 27
Display Debug, 12
File Debug, 12
Project, 29
Show Architecture, 12
Show Auto_path, 12
Show Com, 12
Show Commands, 12
Show Globals, 12
Show Open Projects, 12

PBC⇒LL Net
Invocation, 16
Options, 25, 30
Petri box⇒INA
Options, 26
INA⇒INA
Options, 43
PROMELA⇒PROMELA
Options, 43
Petri box⇒SPIN
Options, 26
PFA⇒B(PN)^2
Invocation, 18
Options, 26, 44
PFA⇒HL Net
Invocation, 18
Options, 26, 43
PFA⇒M-net
Options, 26, 43
Prefix Builder
Options, 26, 35
Project
Copy, 10
Delete, 11
New, 10
Objects, 14
Open, 9, 10
Project List, 11
Rename, 11
Window, 13
Project List, 11

Quit, 10

Reachability
Invocation, 19
Reachability Analyser
Options, 26, 37
Reference, 18, 106
Rename
Project, 11

Safe System
Invocation, 19
Save
Entire Project, 16
Save as User Default, 28
Save Project Info, 16
SDL Editor, 109
Edit
Copy, 111
Cut, 111
Paste, 111
Replace, 111
Search, 111
File
Edit SDL Properties, 110
Export Clipboard, 111
Import File, 110
Open, 110
Quit, 111
Reload, 110
Save, 110
Save as, 110
Modes, 111
Enter Edit Mode, 111
Enter Reference Mode, 111
Enter Simulation Mode, 112
Option, 112
Preferences, 112
Show all Values, 112
Show Reference exact, 112
Show Reference in Formula, 112
Show Reference in HL, 112
Show Reference in LL, 112
Show Reference partial, 112
SDL References, 112
SDL⇒HL Net
Invocation, 18
Options, 27, 45
SDL⇒M-net
Options, 27, 45
Show Icons, 28
Show References, 28
Start-up Window, 9

Sync HL Net
    Invocation, 17
    Options, 26, 33
Syntax Check
    Options, 25, 29

T-System
    Invocation, 18