Chapter 12. Notes on XSPIN

"The ability to simplify means to eliminate the unnecessary so that the necessary may speak."

—(Hans Hofmann, 1880–1966)

XSPIN is the graphical interface to SPIN that for many users is the first introduction to the tool. It can be a considerable benefit, though, if the user is familiar with the basic operation of SPIN before switching to XSPIN, especially when more subtle design problems are encountered.

The interface operates independently from SPIN itself. It synthesizes and executes SPIN commands in the background, in response to user selections and button clicks. Nonetheless, this front−end tool supplies a significant added value by providing graphical displays of, for instance, message flows and time sequence diagrams. XSPIN also provides a clean overview of the many options in SPIN that are available for performing simulations and verifications.

To run XSPIN, you first of all need to be able to run SPIN itself, which means that you minimally will need access to an ANSI compatible C preprocessor and compiler. XSPIN is written in Tcl/Tk, so if you have a local installation of the Tcl/Tk toolset,[1] you can run it directly from its source. You do not need to install Tcl/Tk to run the tool, though. It is also available as a stand−alone binary executable that is available as part of the standard SPIN distribution (see Appendix D).

[1] Tcl/Tk can be downloaded free of charge from www.tcl.tk.

This chapter gives a brief overview of the main options that are available through XSPIN. The interface is intuitive enough that most questions can be answered by simply running the tool and by exploring its options and help menus interactively.

Starting a Session With XSPIN

Assuming all the software has been installed properly, XSPIN can be started on both UNIX systems and Windows PC systems from the shell command line with the name of a file containing a PROMELA specification as an argument, for instance, as follows:

```
$ xspin leader
```

On a Windows system the program can also be selected from the start menu, or by double−clicking the source file named `xspin.tcl`.

When XSPIN starts, it first checks that good versions of SPIN and Tcl/Tk are available. It prints the version numbers in a command−log window, and optionally opens and loads an initial PROMELA file.

Throughout the use of XSPIN, a log of all actions performed is maintained in a special command−log window that appears at the bottom of the XSPIN display. Syntax errors, unexpected events, time−consuming actions such as background compilations and verification runs, can be tracked in the log.

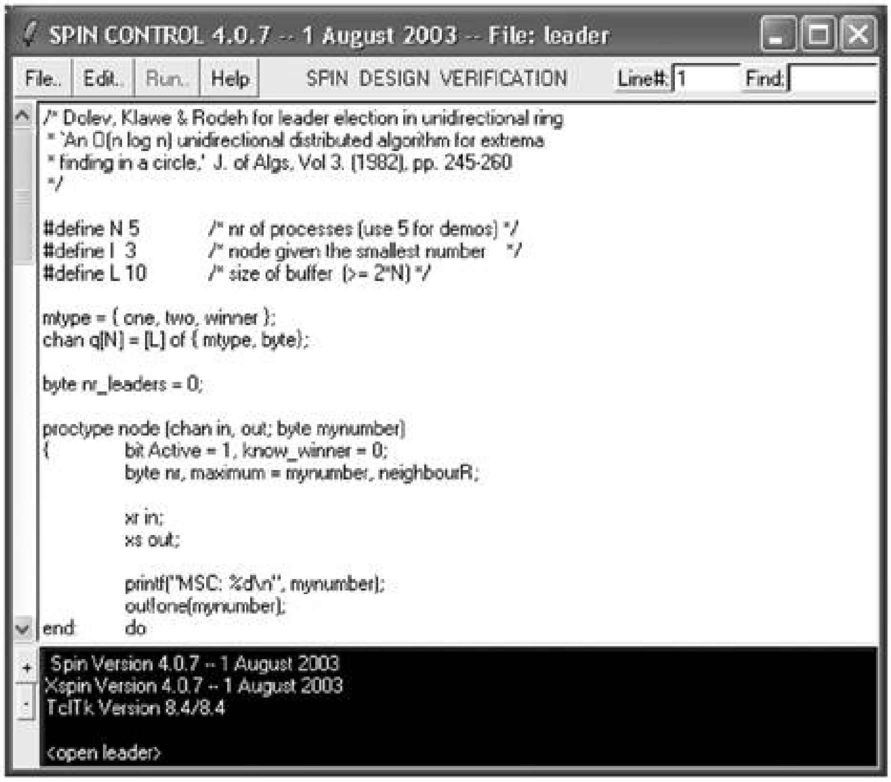
The main display of XSPIN is a text window that displays the file being used, just like a graphical text editor would do. The file name, if any, is shown in the title bar. The view in the text window can be changed in four different ways:

- With the scroll bar on the left−hand side of the text window.
- By typing a line number (followed by a <return>) in the box in the title bar marked `Line#:`.
- By typing a regular expression pattern (followed by a <return>) in the box marked `Find:`.
- By moving a three−button mouse with the middle mouse button down, or a two−button mouse with both buttons down.

Moving around in the text with the mouse buttons down (the last method above) is the most convenient, and it works in most of the text displays that are provided by XSPIN.

There are four other buttons in the title bar of the XSPIN display: `File..`, `Edit..`, `Run..`, and `Help`, as shown in Figure 12.1. We will discuss each of these separately in the following sections.

Figure 12.1. XSPIN Main Window


SPIN CONTROL 4.0.7 -- 1 August 2003 -- File: leader
File.. Edit.. Run.. Help SPIN DESIGN VERIFICATION Line#: 1 Find:

```
/* Dolev, Klawe & Rodeh for leader election in unidirectional ring
 * `An O(n log n) unidirectional distributed algorithm for extrema
 * finding in a circle,' J. of Algs, Vol 3. (1982), pp. 245-260
 */

#define N 5          /* nr of processes (use 5 for demos) */
#define I  3         /* node given the smallest number    */
#define L 10         /* size of buffer  (>= 2*N) */

mtype = { one, two, winner };
chan q[N] = [L] of { mtype, byte};

byte nr_leaders = 0;

proctype node (chan in, out; byte mynumber)
{       bit Active = 1, know_winner = 0;
        byte nr, maximum = mynumber, neighbourR;

        xr in;
        xs out;

        printf("MSC: %d\n", mynumber);
        out!one(mynumber);
end:    do
```

```
Spin Version 4.0.7 -- 1 August 2003
Xspin Version 4.0.7 -- 1 August 2003
TclTk Version 8.4/8.4

<open leader>
```

The File Menu

The file menu gives a choice of seven actions: `New`, `UnSelect`, `ReOpen`, `Open`, `Save As`, `Save`, and `Quit`.

`New` clears the contents of the main window, but does not reset the file name or affect any other parameter.

`UnSelect` removes any selection highlights that the user or a background program may have placed in the main window.

`ReOpen` reloads the contents of the current file in the main text window, discarding any changes made since the last `Save` operation.

`Open` prompts the user with a standard file dialogue, listing all files in the current directory. Double−clicking any of the files will cause XSPIN to open it and place it in its text window. Of course, this only makes sense for PROMELA specifications. Double−clicking a directory name will cause the browse window to descend into that directory and display the files listed there. Double−clicking the up−arrow icon will cause the browse window to move up to the parent directory, and display the files listed there.

`Save As..` provides a file browse dialogue, allowing the user to select a file name in which the current contents of the text window should be saved. During each session, XSPIN always maintains a private copy of the current contents of the text window in a file called `pan_in` to avoid unintentional changes in the original source of the PROMELA specification. The source file itself is only (re)written with an explicit `Save` command.

`Quit` terminates the session of XSPIN, removing any temporary files that were created during simulation or verification runs. No warning is issued if the file being edited was changed since the last time it was saved.

The Edit Menu

The edit menu contains the three standard entries for performing `Cut`, `Copy`, and `Paste` operations on selected text in the main window. Text can be selected as usual, by sweeping with the left mouse button down, or by double−clicking text strings. Cut, copy, and paste operations are also available with control−key combinations: control−X for cut, control−C for copy, and control−V for paste. Be careful though, there is no undo operation implemented in XSPIN.

The Help Menu

The help menu gives a quick online review of the main usage options of SPIN and XSPIN, and contains an explanation of the proper setting of the main parameters for verification and simulation runs. The menu also provide hints for reducing search complexity. The entries in this menu will be self−explanatory.

The Run Menu

The run menu has eight entries for performing syntax checks, property−based slicing, setting simulation or verification parameters, running simulations or verifications, and viewing the internal automata structures computed by SPIN. We discuss each of these menu choices next.

Syntax Check

XSPIN runs a syntax check by asking SPIN to execute the command:

```
$ spin -a -v pan_in
```

in the background, using its private file copy of the PROMELA text in the main window. Results, if any, are displayed in the standard command log window and in a separate popup window that can be closed again with a mouse−click on its `Ok` button. Wherever possible, error text is highlighted in the main XSPIN window for ease of reference.

Property−Based Slicing

To run the slicing algorithm, which also provides a thorough syntax check of the PROMELA source, XSPIN executes the following command:

```
$ spin −A pan_in
```

The slicing algorithm tries to locate all logical properties that are part of the model, for instance, as expressed in assertions and in a `never` claim, and it uses this information to identify those parts of the model that cannot possibly affect the correctness or incorrectness of those properties. In the absence of properties, the algorithm can still do useful things, by identifying portions of the model that are redundant no matter which properties are specified.

The results that are produced are displayed both in the command log window and in a separate popup window. Included in the output are also any warnings about potentially wasteful constructs, such as variables that were declared as integers but that assume only boolean values. If no redundancies can be found in the model, SPIN will report this as well, so this option is also generally useful as a somewhat more thorough check of the model.

Set Simulation Parameters

The simulation options panel allows the user to select the types of displays that will be generated during simulation runs. In the upper right−hand corner of the panel the choice between random, guided, and interactive simulation can be made. When XSPIN is used, random simulation is by default done with a predefined seed value of one. This seed value can be changed freely to obtain different types of runs, but once the seed value is fixed, all experiments are fully reproducible. If the entry box for the seed value is left blank, the current system time is used as a seed value, which of course does not guarantee reproducibility. The guided simulation option requires the presence of a file named `pan.trail`, which is normally produced in a verification run when SPIN finds a counterexample to a correctness property. The number of steps that should be skipped before the display of the sequence is initiated can be specified, the default value being zero.

Two further options are selectable in the right−hand side column of the simulation options panel. For send statements, the user has a choice of semantics. Either a send operation that targets a full message channel (queue) blocks, or can be defined to be non−blocking. If non−blocking, messages sent to a full channel are lost. Up to three channel numbers (queue numbers) can be specified in the three entry boxes at the bottom of the right−hand column. If channel numbers are entered, send and receive operations that target the corresponding channels will not be displayed in graphical MSC (message sequence chart) displays.

On the left−hand side of the panel four types of outputs can be requested. By default, only two of these will be selected. A most useful type of display is the MSC (message sequence chart) panel. Normally, the execution steps in this display are tagged with identifying numbers. By moving the mouse cursor over one of the steps, the source text will show and the main text window will scroll to the corresponding statement. Alternatively, the user can also choose to have the source text shown for each step in the display. For very long runs, the message sequence chart can be compacted somewhat by selecting the condensed spacing option.

Normally, the message sequence chart will display only send and receive actions, connecting matching pairs with arrows. The output from print statements can be added to a message sequence chart by starting any newline terminated string to be printed with the prefix `"MSC:"` followed by a space, for instance, as in:

```
printf("MSC: this will appear in the MSC\n");
```

The default background color for text boxes that are created in this manner is yellow. The color of the box can also be changed by starting the text string to be printed with a special two−character control sequence. For instance,

```
printf("MSC: ~W uses a white box\n");
printf("MSC: ~G uses a green box\n");
printf("MSC: ~R uses a red box\n");
printf("MSC: ~B uses a blue box\n");
```

The prefix `MSC:` and an optional two−character control codes for setting colors do not appear in the output itself.

As a special feature of print statements, if the statement

```
printf("MSC: BREAK\n");
```

causes XSPIN to suspend the simulation run temporarily, simulating a breakpoint in the run. The run can be restarted from the main simulation output panel.

The time−sequence panel can show verbose output of the simulation run. Normally, the output is shown in the interleaving order from the execution, but it can also be split out into separate windows so that a separate trace for each executing process can be obtained. If the number of running processes is large, this option can require a lot of real estate on the screen, so it is not always helpful. The main window can also be replicated, by selecting the "One Trace per Process" option, so that each text window can track the execution of one specific process. There are actually two ways to track individual process executions: as an basic listing of all steps performed, and with a moving highlight in the source text that moves from step to step. The latter mode is selected with the "One Window per Process" option.

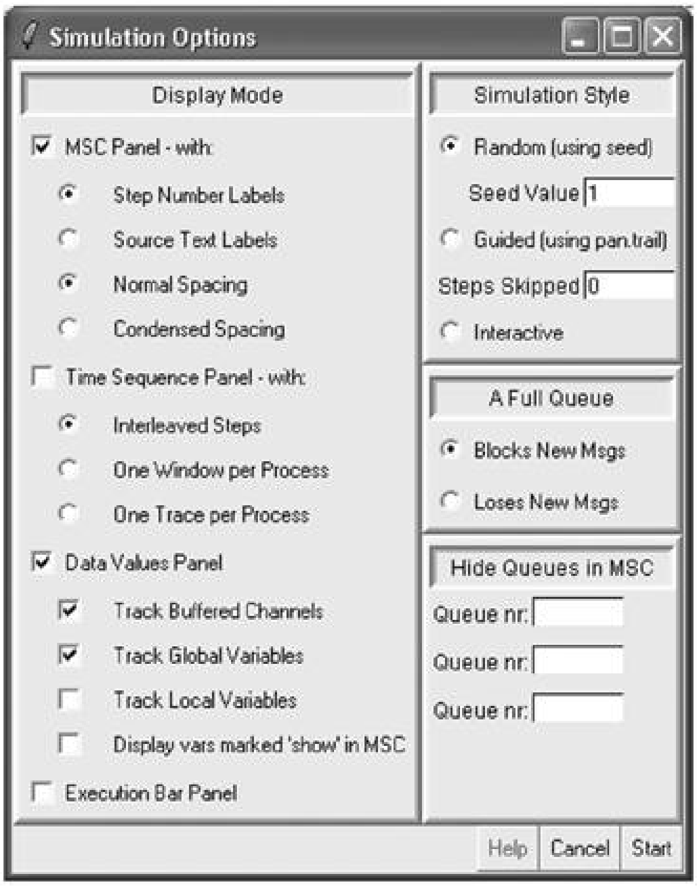
The data values panel shows the most recently assigned values for variables in the model. By default, only global variable values are shown. The output for local variable values can be added by selecting the corresponding box on the options panel. Note that only variables that change value appear here, not variables that still retain their initial value from a declaration. If the amount of output produced for larger models becomes too large, an alternative option is available to display only the values of selected variables. To select such variables, the keyword `show` can be added in the PROMELA model before the variable declaration, for instance, as in:

```
show byte nr_leaders = 0;
```

By selecting the box marked `Display vars marked 'show'`, the output in the Message Sequence Chart panel will now include an entry for each value change of (only) these specially marked variables.

The execution bar panel, selectable at the bottom of the left−hand column in the simulation options panel (see Figure 12.2), gives a dynamically updated bar−chart of the number of statement executions in each running process.

Figure 12.2. The Simulations Options Panel



Display Mode

[✓ MSC Panel - with:]

- [✓ Step Number Labels]
- Source Text Labels
- [✓ Normal Spacing]
- Condensed Spacing

Time Sequence Panel - with:

- [✓ Interleaved Steps]
- One Window per Process
- One Trace per Process

[✓ Data Values Panel]

- [✓ Track Buffered Channels]
- [✓ Track Global Variables]
- Track Local Variables
- Display vars marked 'show' in MSC

Execution Bar Panel

Simulation Style

- [✓ Random (using seed)]
- Seed Value: 1
- Guided (using pan.trail)
- Steps Skipped: 0
- Interactive

A Full Queue

- [✓ Blocks New Msgs]
- Loses New Msgs

Hide Queues in MSC

- Queue nr:
- Queue nr:
- Queue nr:

Selecting the `Start` button will bring up the initial contents of the display panels that were selected, such as the message sequence chart display shown in Figure 12.3, and executes a first single step in the simulation. By selecting `Single Step` we can now step through a simulation, one statement execution at a time. By selecting `Run`, SPIN will take over and run the simulation. The label on this button then changes to `Suspend` as shown in Figure 12.4. The button toggles back and forth between these two modes each time it is clicked.

Figure 12.3. Message Sequence Chart Display (portion)

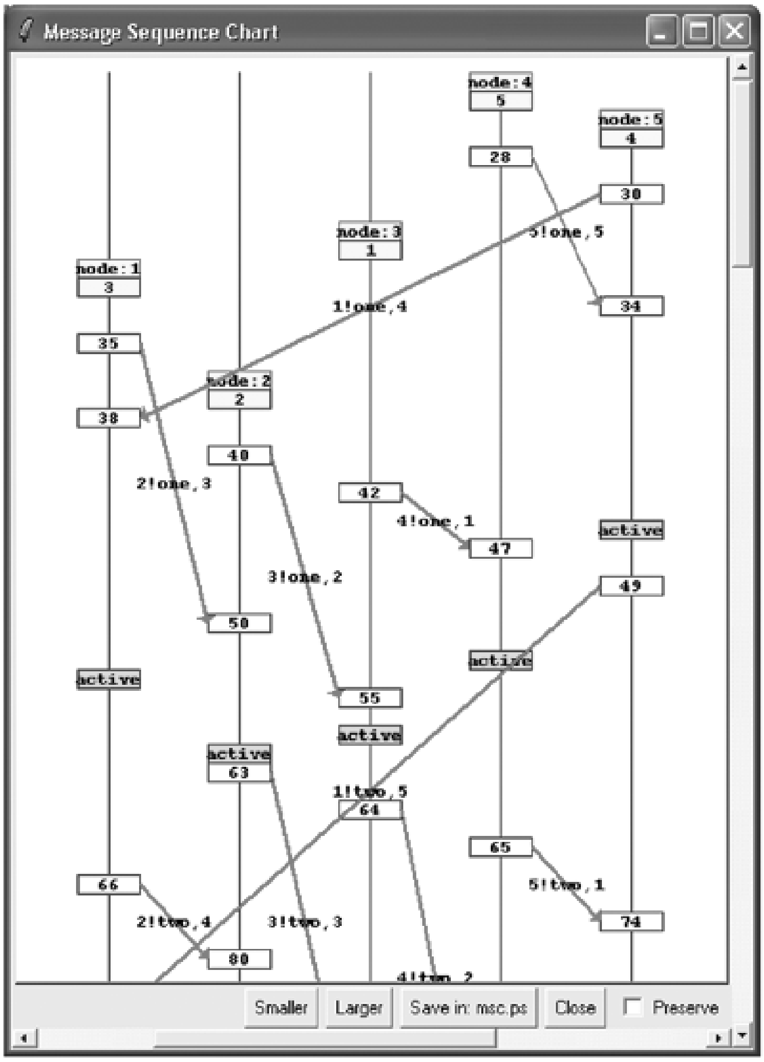
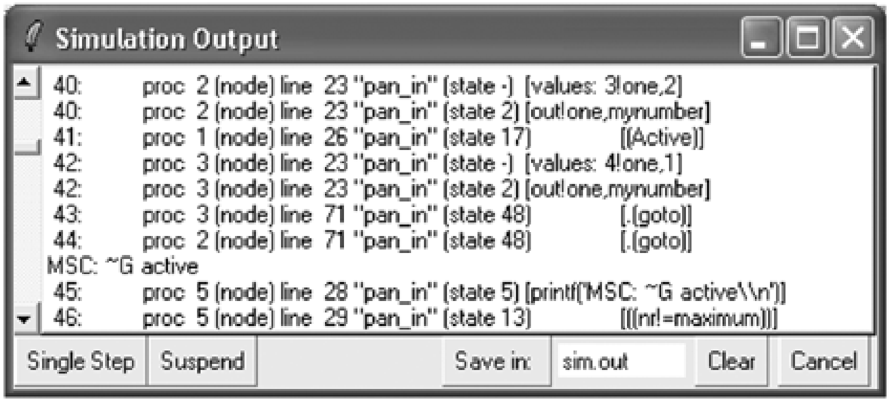


Figure 12.4. Main Simulation Output Panel



```
 40:        proc  2 (node) line  23 "pan_in" (state -)  [values: 3!one,2]
 40:        proc  2 (node) line  23 "pan_in" (state 2) [out!one,mynumber]
 41:        proc  1 (node) line  26 "pan_in" (state 17)             [(Active)]
 42:        proc  3 (node) line  23 "pan_in" (state -)  [values: 4!one,1]
 42:        proc  3 (node) line  23 "pan_in" (state 2) [out!one,mynumber]
 43:        proc  3 (node) line  71 "pan_in" (state 48)             [.(goto)]
 44:        proc  2 (node) line  71 "pan_in" (state 48)             [.(goto)]
MSC: ~G active
 45:        proc  5 (node) line  28 "pan_in" (state 5) [printf('MSC: ~G active\\n')]
 46:        proc  5 (node) line  29 "pan_in" (state 13)             [((nr!=maximum))]
```

(Re)Run Simulation

When the simulation parameters have been set once, they persist for the remainder of the session, or until the setting is changed. New simulation runs can now be initiated directly with these settings by selecting this menu option from the `Run` menu. The option is grayed out, and remains unselectable, until the simulation parameters panel has been displayed at least once.

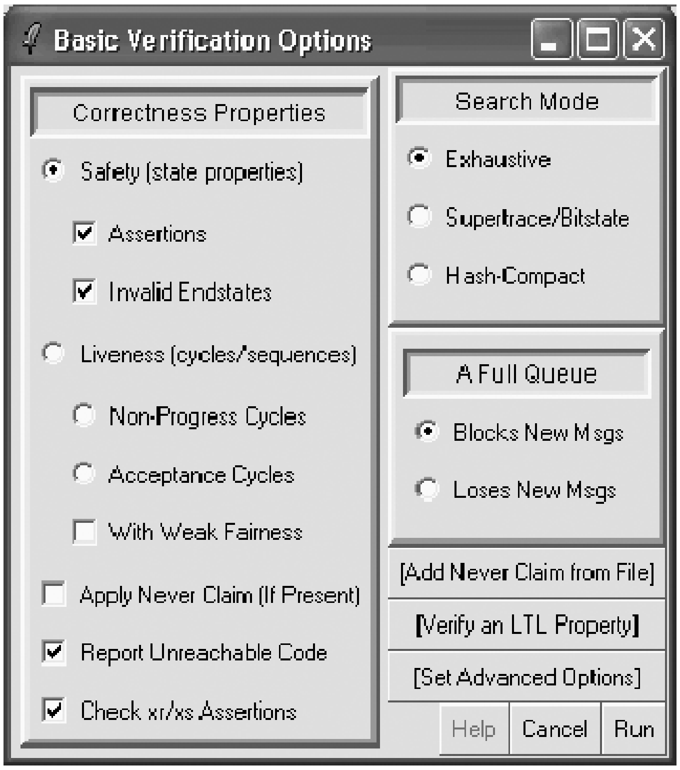
Set Verification Parameters

The verification parameters panel gives visual control over most of the options that SPIN provides for performing automated verifications. The initial settings of all parameters are chosen in such a way that they provide a reasonable starting point for most applications. A first verification run, therefore, can in most cases safely be performed by hitting the `Run` button in the lower right corner of the panel, without altering the default settings.

When a verification run completes, XSPIN attempts to provide hints about ways to proceed, based on the results obtained. No hints are provided when a clean run is performed, that is, a complete exhaustive search that did not reveal any errors. The default hint in cases like these would be to consider whether or not the properties that were proven are the correct ones, and whether or not other properties still remain to be proven.

The default settings define a search for safety properties only. Proving liveness properties (properties of infinite behaviors as manifested by execution cycles) requires a separate verification run with the appropriate options selected in the `Correctness Properties` section of the verification parameters panel, shown in Figure 12.5.

Figure 12.5. Basic Verification Options


Basic Verification Options

Correctness Properties
- [✓ Safety (state properties)]
  - [✓ Assertions]
  - [✓ Invalid Endstates]
- Liveness (cycles/sequences)
  - Non-Progress Cycles
  - Acceptance Cycles
  - With Weak Fairness
- Apply Never Claim (If Present)
- [✓ Report Unreachable Code]
- [✓ Check xr/xs Assertions]

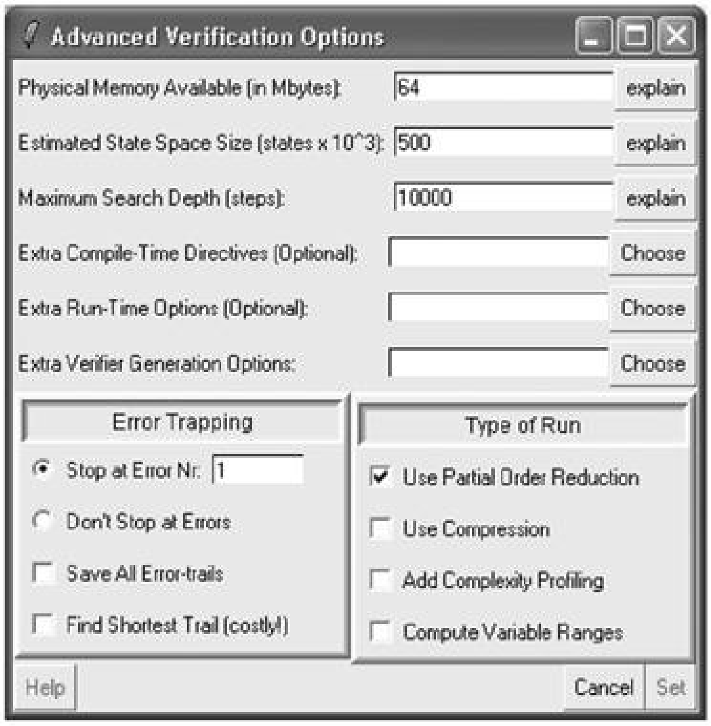
Search Mode
- [✓ Exhaustive]
- Supertrace/Bitstate
- Hash-Compact

A Full Queue
- [✓ Blocks New Msgs]
- Loses New Msgs

Note that if the PROMELA specification contains syntax errors, these errors will show up in the XSPIN log when the `Run` button is selected. The run itself is canceled in that case. It is useful to keep an eye on such error reports, and to be aware of the types of things that XSPIN or SPIN perform in the background.

Three main search modes are selectable in the upper right–hand corner of the panel: exhaustive verification, bitstate approximation, or hash–compact. Some of the more rarely used settings for performing verifications are delegated to a special `Advanced Options` panel, shown in Figure 12.6, that can be selected in the lower right–hand corner of the basic verification options panel. Especially in the beginning, this panel can safely be ignored.

Figure 12.6. Advanced Verification Options



Advanced Verification Options

Physical Memory Available (in Mbytes): 64

Estimated State Space Size (states x 10^3): 500

Maximum Search Depth (steps): 10000

Extra Compile-Time Directives (Optional):

Extra Run-Time Options (Optional):

Extra Verifier Generation Options:

Error Trapping

- [✓ Stop at Error Nr: 1]
- Don't Stop at Errors
- Save All Error-trails
- Find Shortest Trail (costly!)

Type of Run

- [✓ Use Partial Order Reduction]
- Use Compression
- Add Complexity Profiling
- Compute Variable Ranges

Also selectable via the basic verification options panel is the LTL Property panel, which we will discuss in more detail shortly. Finally, if a `never` claim is present in a file, but not already included within the PROMELA model itself, it can be picked up by the verifier for a single verification run by selecting the `Add Never Claim from File` option. This option is, for instance, the simplest method for including `never` claims that are generated with the Timeline editing tool that is discussed in Chapter 13 (p. 283).

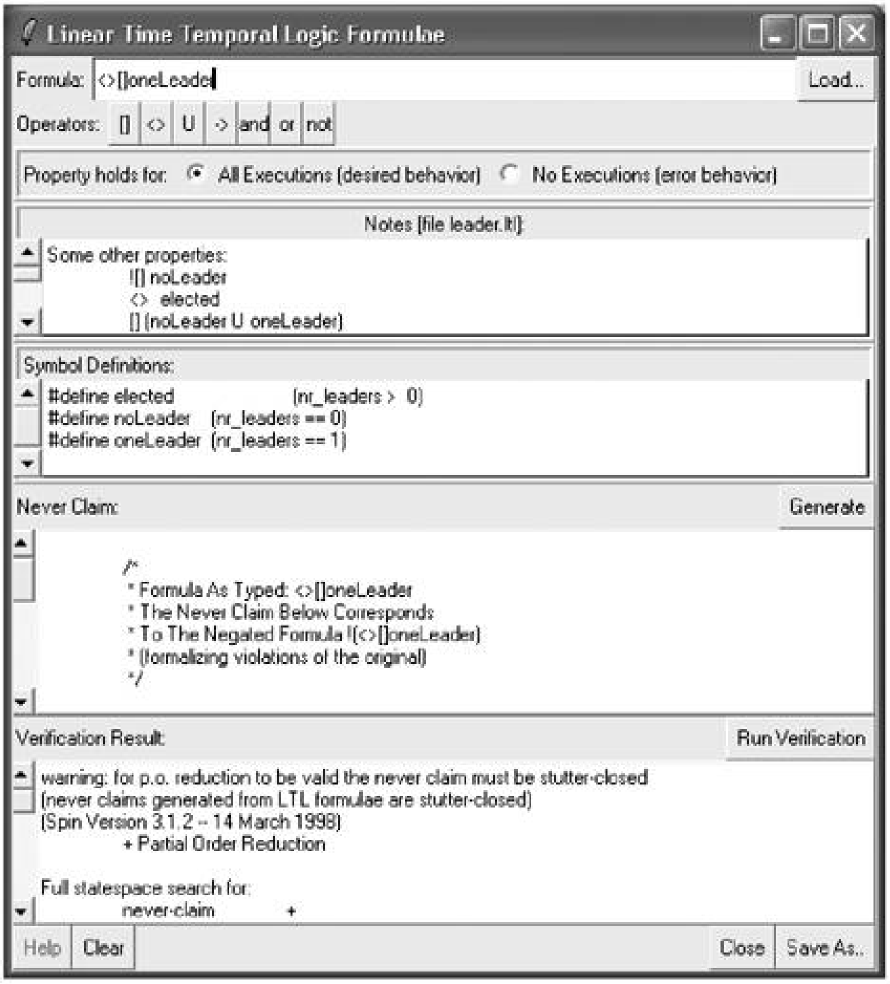
(Re)Run Verification

When the verification parameters panel has been displayed at least once, this menu entry becomes selectable. It will initiate a new verification run, preserving all parameter settings that were chosen earlier. This can be useful, for instance, when small changes in the PROMELA model are made to remedy problems uncovered in earlier verification runs.

LTL Property Manager

Selecting this entry from the `Run` menu brings up a panel for entering an LTL formula to be used in a verification attempt, shown in Figure 12.7. By clicking on the button labeled `Generate`, or by typing a return character in the formula entry box, the formula is converted into a `never` claim. Both this claim and the main PROMELA specification are now submitted to SPIN when the `Run Verification` button is selected.

Figure 12.7. The LTL Property Manager


Linear Time Temporal Logic Formulae
Load...
Operators: [] <> U -> and or not
Generate
Run Verification
Help
Clear
Close
Save As..

Formula: <>[]oneLeader

Property holds for: [✓ All Executions (desired behavior)] No Executions (error behavior)

Notes [file leader.ltl]:

```
Some other properties:
            ![] noLeader
            <>  elected
            [] (noLeader U oneLeader)
```

Symbol Definitions:

```
#define elected          (nr_leaders >  0)
#define noLeader   (nr_leaders == 0)
#define oneLeader  (nr_leaders == 1)
```

Never Claim:

```
        /*
         * Formula As Typed: <>[]oneLeader
         * The Never Claim Below Corresponds
         * To The Negated Formula !(<>[]oneLeader)
         * (formalizing violations of the original)
         */
```

Verification Result:

```
warning: for p.o. reduction to be valid the never claim must be stutter-closed
(never claims generated from LTL formulae are stutter-closed)
(Spin Version 3.1.2 -- 14 March 1998)
        + Partial Order Reduction

Full statespace search for:
        never-claim         +
```

Templates of standard forms of LTL formulae can be loaded into the LTL property window with the `Load` option in the upper right corner of the display. Four templates are predefined for invariance properties, response properties, precedence properties, and objective properties. They have the following general form:

```
[] p                # invariance
p -> <> q           # response
p -> (q U r)        # precendence
p -> <> (q || r)    # objective
```

Each of these generic types of properties can (and will generally have to) be prefixed by temporal operators such as `[]`, `<>`, `[]<>`, or `<>[]`. The property type named `objective` can be read to mean that `p` (a state property) is an enabling condition that determines when the requirement becomes applicable. Once enabled, the truth of state property `q` can signify the fullfillment of the requirement, while the truth of `r` can be treated as a discharging condition that voids the requirement.

LTL properties consist of temporal and logical operators and user−defined propositional symbols. For propositional symbols any name that starts with a lowercase character can be used. It is customary to use single character names, such as the `p`, `q`, and `r` that we used in the template formulae.

The LTL property can be cast as either a positive (desired) or a negative (undesired) property of the model by selecting the corresponding field below the formula entry box.

A positive property is negated by the convertor to convert it into a `never` claim. A negative property is not negated.

Each propositional symbol that is used in the formula must be defined as a boolean expression in the `Symbol Definitions` panel.

These definitions will be remembered as part of the property definition, together with any annotations and verification results if the formula is saved as a template with the `Save As` option in the lower right−hand corner of the display. Always make sure to enclose symbol definitions in round braces to secure a proper interpretation of operator precedence rules. For instance:

```
#define p (a > b)
#define q (len(q) < 5)
```

where `a` and `b` are global variables in the PROMELA model, and `q` is a global channel.

Valid temporal operators are:

```
[] always (with no space between [ and ])
<> eventually (no space between < and >)
U (strong) until, and
V the dual of the until operator: (p V q) == !(!p U !q)
```

All operators are left−associative, including `U` and `V`. The `V` operator is rarely used in user−defined formulae, but it is often used internally by SPIN when it normalizes formulae.

Boolean operators can also be used inside LTL formulae, using standard PROMELA syntax.

```
&& logical and
!  logical negation
|| logical or
```

Arithmetic operators are not allowed within an LTL formula, but can be used within the macro definitions of the propositional symbols.

Two shorthands are available for defining logical implication and equivalence.

```
->  logical implication
<-> logical equivalence
```

The formula `(p -> q)` is short for `(!p || q)` and `(p <-> q)` is short for `(p -> q) && (q -> p)`.

Recall that logical implication and logical equivalence are boolean and not temporal operators, and that therefore no passage of time is implied by the use of a subformula such as `(p -> q)`. (On this point, see also the section on Using Temporal Logic in Chapter 6.)

The names of operands in an LTL formula must be alphanumeric names, always beginning with a lowercase letter. The preferred style is to use only single–character names, such as `p`, `q`, and `r`. Prudence, further, dictates that the right–hand side of each of the corresponding symbol definitions is enclosed in round braces, to protect against unintended effects of operator precedence rules. For instance, instead of defining

```
#define p     a < b
```

the preferred style us to use

```
#define p     (a < b)
```

Remote reference operations can also only be used indirectly, via a symbol definition, which again are normally enclosed in round braces for security, as in, for instance:

```
#define q     (main[5]@label)
```

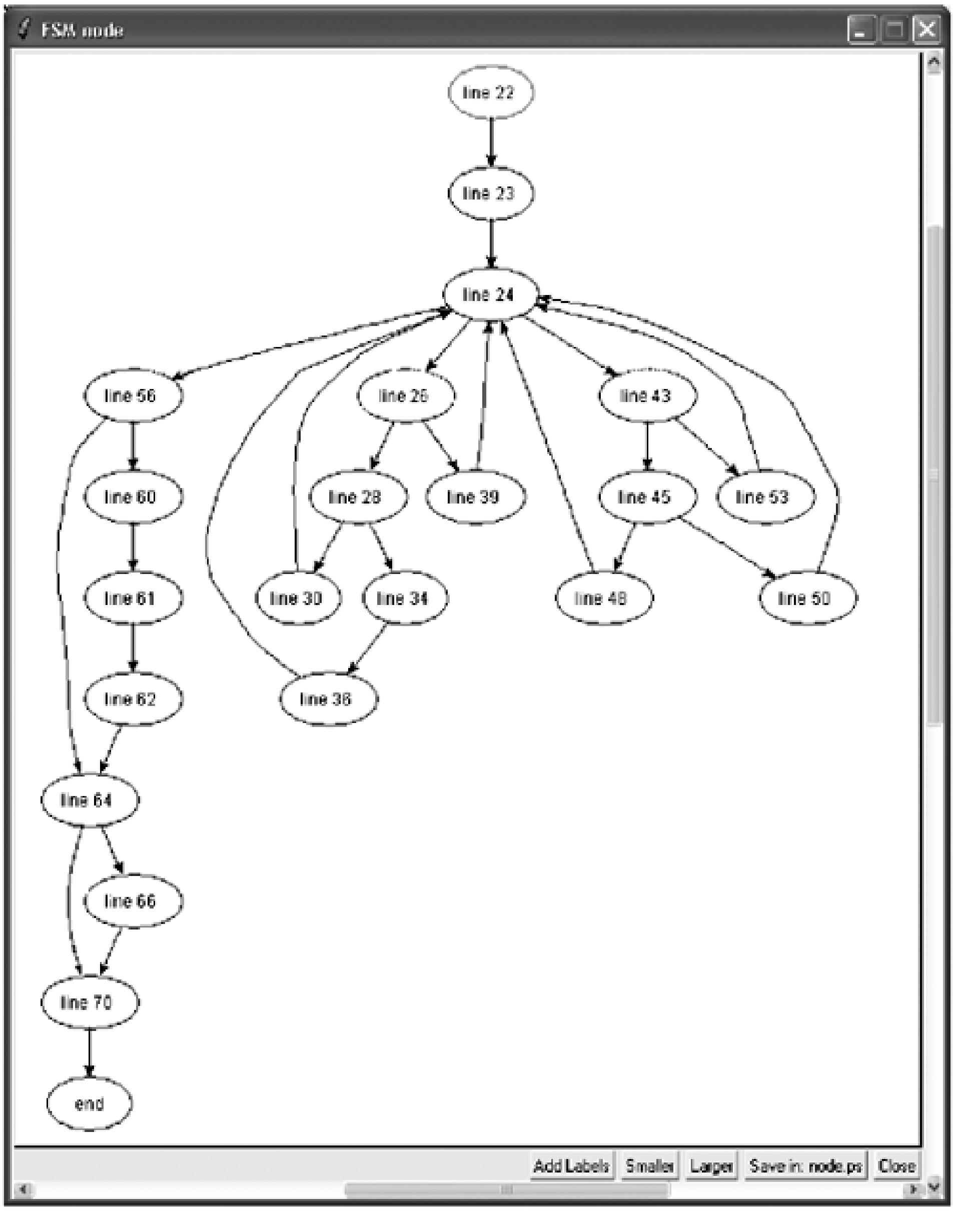
The Automaton View Option

The automaton view option, finally, allows for the selection of one of the `proctype`s that are part of the PROMELA specification, so that it can be displayed in automaton form.

When this option is selected, XSPIN first generates and compiles an executable verifier. It then use the output from PAN's run−time option `-d` to synthesize the automaton view. It is highly recommended to have a copy of the graph layout tool `dot` installed on the system.[2] If it is present, XSPIN will use it to compute the layout for the automaton graph, as illustrated in Figure 12.8. If absent, a cruder approximation of the layout will be used.

[2] Dot can be downloaded from http://www.research.att.com/sw/tools/graphviz/.

Figure 12.8. The Automata View



The view in Figure 12.8 shows all states in `proctype node` from the leader election protocol example. For simplicity, we have turned off the display of the statement labels in this display. They can be restored by selecting the button labeled `Add Labels` at the bottom of the display.

Each state in the graph by default shows the line number in the source file that correspond to that state. Moving the cursor over a state causes the corresponding line to be highlighted in the main text window, and changes the line number text for the internally assigned state number. The display of the line number is restored when the cursor is moved away from the state. The graphical display that is generated is for information only; it cannot be edited.

In Summary

XSPIN synthesizes commands that are issued to the SPIN model checker based on user selections and preferences. For each run performed by SPIN in the background, XSPIN also intercepts the output and presents it in a slightly more pleasing visual way through its graphical interface.

An arbitrary number of assertions, `progress`, `accept`, and `end` state labels can be defined in a model, but at all times there can be only one `never` claim. If `never` claims are derived from LTL formulae, the LTL property manager makes it easy to build a library of formulae, each of which can be stored in a separate file and checked against the model. The results of each run are stored automatically in the file that contains the corresponding LTL property. These files have the default extension `.ltl`.