Installation process and main functionalities of the Spin model checker

Miguel J. Hornos[1], Juan Carlos Augusto[2]

[1] Software Engineering Department, University of Granada, Spain

[2] Department of Computer Science, Middlesex University, London, United Kingdom

mhornos@ugr.es, j.augusto@mdx.ac.uk

Abstract. This document explains how to download and install all the software needed to properly run the Spin model checker and its user-friendly graphical user interface, called iSpin. It also offers a short tutorial which presents the main functionalities of Spin and explains briefly some basic concepts which are important to be able to use Spin through iSpin.

Keywords. Model Checking, Spin, iSpin, Installation, Simulation, Verification.

1. Installation process

Although there are several steps to be carried out for installing Spin [1][2], it is a very easy process. This tool is multiplatform, since we can install it in Unix/Linux systems, Windows PCs or Macs. The interested reader can found in [3] a detailed explanation on how to install Spin in any of these systems, as well as links to download the required files.

Since we suppose that most users will want to install Spin on Windows, we are going to highlight briefly the main steps to do it. The unique prerequisite to install it is to have some C compiler preinstalled, because Spin generates each verifier (software optimized for the specific model being checked) as C source code that requires compilation before a verification can be performed. Hence, if your system does not have any C compiler, the first you must do it is to install one. Although the Spin webpage advices Cygwin [4] (and you can install it as an option), we have got better results with MinGW [5].

The second step is to download the pc_spin*.zip file containing the latest available version (indicated by the numbers which replace to * in the file name) of Spin for Windows from [6], and unfold it in C:\ispin. This distribution includes the new graphical user interface for Spin, namely iSpin [7].

As iSpin is implemented using the Tcl programming language and the Tk graphical user interface toolkit, the third step is to install Tcl/Tk on your system, downloading the self-extracting and freely available package from [8].

The next step is to update the PATH environment variable of your system. In Windows 7 or Windows Vista you find that by displaying the *Start* menu and right clicking on *Computer* option, then choosing the *Properties* tab, then *Advanced System Settings*, and then *Environment Variables*, where you should edit the PATH variable to add

";C:\MinGW\bin;C:\ispin;C:\Tcl\bin;" (without writing the quotation marks), supposing you have installed MinGW and choose the default directories in each case. Thus, you tell your computer where to find the important files needed to run Spin.

Additionally, if you want to visualize graphically the automata that Spin can generate, you need to install Graphviz [9], which is also freely available and self-extracting. Afterwards, you will have to update the PATH environment variable adding the appropriate directory (e.g. C:\Program Files\Graphviz 2.28\bin;).

Finally, to run Spin using iSpin, go to the directory C:\ispin where Spin was installed and double click on the file ispin.tcl.

2. Main available functionalities with iSpin

This graphical user interface (iSpin) only works with Spin version 6.0 or later. If you are using a previous version, then you will need to use its preceding graphical user interface, called xSpin. You can still use xSpin with the newer versions of Spin, but you should know that xSpin is no longer supported.

iSpin has several tabs in its upper part, as we can see in Figure 1, which shows the *Edit/View* tab. This panel allows us to *Open* (in the left side mid panel) or *Save* a Promela [10] model (whose default file extension is .pml). *Syntax Check*, *Redundancy Check*, and *Symbol Table* options can be used to produce the corresponding output in the black log window at the bottom of the panel. *Find* allows us to locate a search string in the Promela model text.

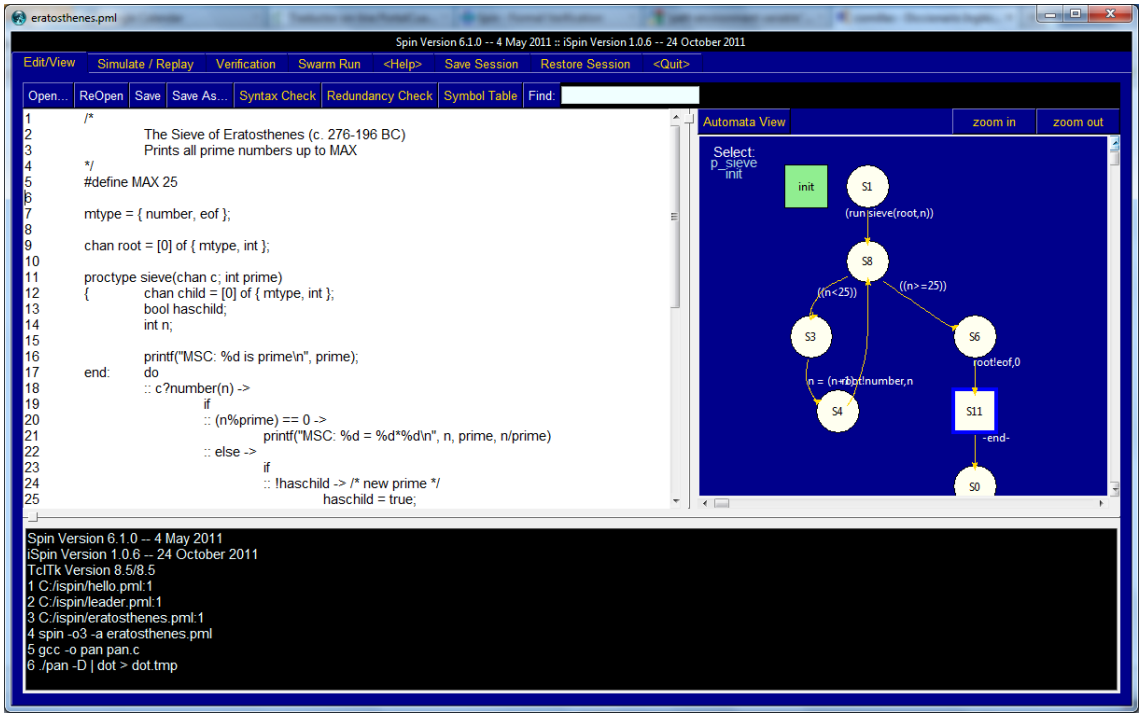

Figure 1. Options in the Edit/View tab

Since each command issued by iSpin is actually performed by standard Spin running in the background, without Spin (or with a previous version to 6.0) not much of interest can happen.

By clicking the *Automata View* button (in the right side mid panel of Figure 1) the names of proctypes [11] and never claims [12] appears in the blue canvas. It does so by first generating and compiling the model checking code, so if there are syntax errors that prevent compilation, we will be warned of those errors first. Once these names are visible, we must click on a name to generate the control-flow graph of the corresponding state transition system. Currently, the text in the graphs does not scale when we zoom in or out, so it is limited in some sense.

Figure 2 shows the *Simulate/Replay* tab, which contains all the options that are relevant for random, interactive or guided simulations of the model. Remember that a guided simulation uses an error-trail file produced in a *Verification* or *Swarm Run* (further bellow we will explain these tabs) to guide the execution. *(Re)Run*, *Stop* and *Rewind* buttons respectively (re)starts a simulation run, stops it, and rewinds a completed run to the start. *Step Forward/Backward* moves one step forward/backward through an earlier run. The box at the top right exhibits the *Background command executed* by Spin to generate the output. Bellow that box, the Message Sequence Chart (MSC) corresponding to the simulation run displays the messages sent and received by each of the processes through the channels.

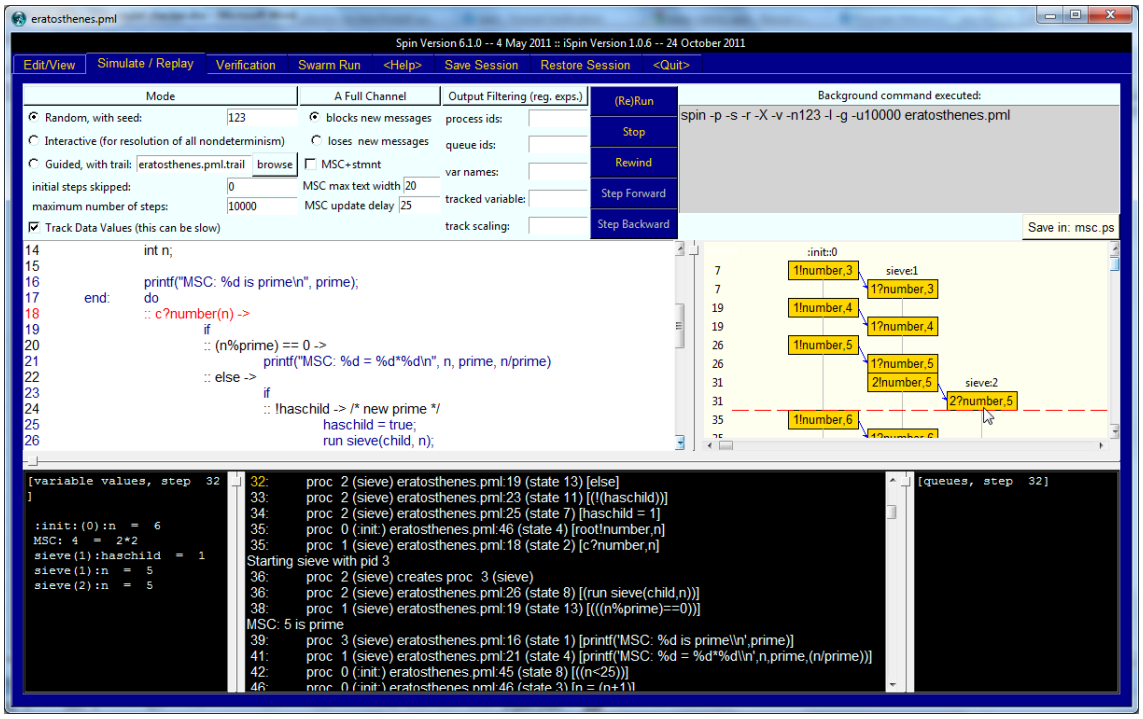

Figure 2. Options in the Simulate/Replay tab

Clicking on either any line number or filename:line_number reference in the simulation output panel (which is in the central part at the bottom) moves to the corresponding line in the Promela model (shown in red in the left middle panel) and updates variable values (in the left bottom panel) and queue contents values (in the right bottom panel) to that point in

the execution. We can also click on the boxes in the MSC display to achieve the same effect.

The entry box for *process ids* (pids) allows us to define a regular expression of pids that will be used to restrict the output to only processes with matching pids; for instance, we can use 1|3 to display output for only processes 1 and 3 or use [^1-3] to suppress output for processes 1, 2, and 3. The entry box for *queue ids* similarly allows the definition of a regular expression filter for operations on channels. The entry box for *var names* allows us to restrict the output in the data values (left bottom) panel to only variable names matching the regular expression given. The entry box for *tracked variable* is an experimental option to display a bar in the MSC panel indicating the size of the variable specified. The size of the bar can be scaled with the value given in the *track scaling* box (e.g. 10, 1 or 0.01).

As there are many available options in the *Verification* tab (see Figure 3), we are not going to explain all of them, since the purpose of most is clear from their labels. A good practice is to go through the options from left to right, keeping the default values if we do not have good reasons to change them: Firstly, choosing the type of verification (*Safety* or *Liveness*) to be done. Secondly, selecting a *Storage Mode* (the options other than *exhaustive* are there just to help us reduce memory) and *Never Claims* if we wish (e.g. in the entry box for *claim name* we can write the name of any of the LTL properties defined in the model: `p0`, `p1`, `p2` or `p3` in the case shown in Figure 3, but only one at each time). Thirdly, deciding the specific *Search Mode* to be performed. Fourthly, indicating what types of error trails we want to capture. Using the *Help* button (on the far right, in the middle) gives more detailed information on methods to reduce verification complexity. Finally, the panel at the far right allows us to provide more detailed parameters, each of them comes with a short explanation that we can see by pressing the *explain* button next to the corresponding parameter.

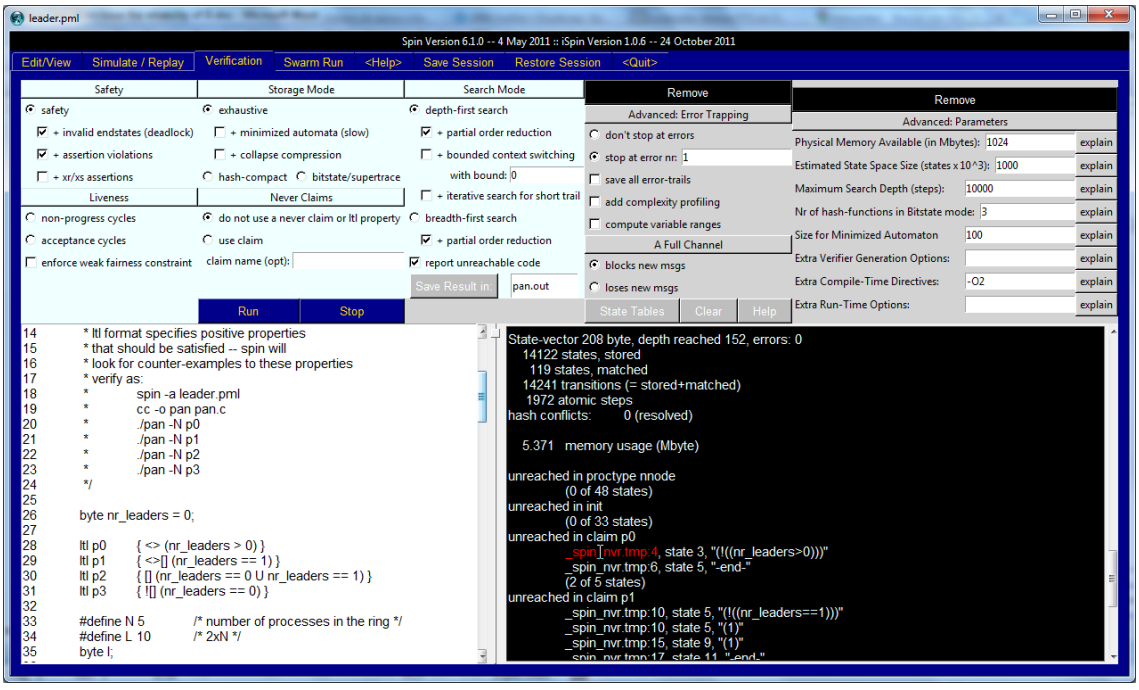

Figure 3. Default options in the Verification tab

Once selected the wished options, clicking the *Run* button generates and compiles the specific model checker and will execute it (if no errors prevent the compilation), and the *Stop* button interrupts a long running verification run. Like in the simulation panel, clicking on a filename:line_number reference (which is highlighted in red when the pointer is over it) in the verification results panel (at the bottom right corner in Figure 3) leads to the corresponding line in such file (shown in the left bottom panel).

Keeping the default options, a *safety* verification is performed (as shown in Figure 3, with no errors found). To check one of the LTL properties included in the model, we must select the *acceptance cycles* and *use claim* options and provide the name of the claim to be verified (`p0` in the case shown in Figure 4). The leader.pml file (which come as an example model with Spin) contains the model of an algorithm for leader election in a unidirectional ring with 5 nodes (see line 33). Initially, there is no leader (since the variable that counts the number of leaders is initialize to 0 in line 26). The property `p0` (line 28) specifies that eventually (at some state in the future) a leader will be elected. Note that this property is equivalent to the LTL formula `p3`. As a result of the verification process, we get that the model fulfils the property `p0` (no errors found).

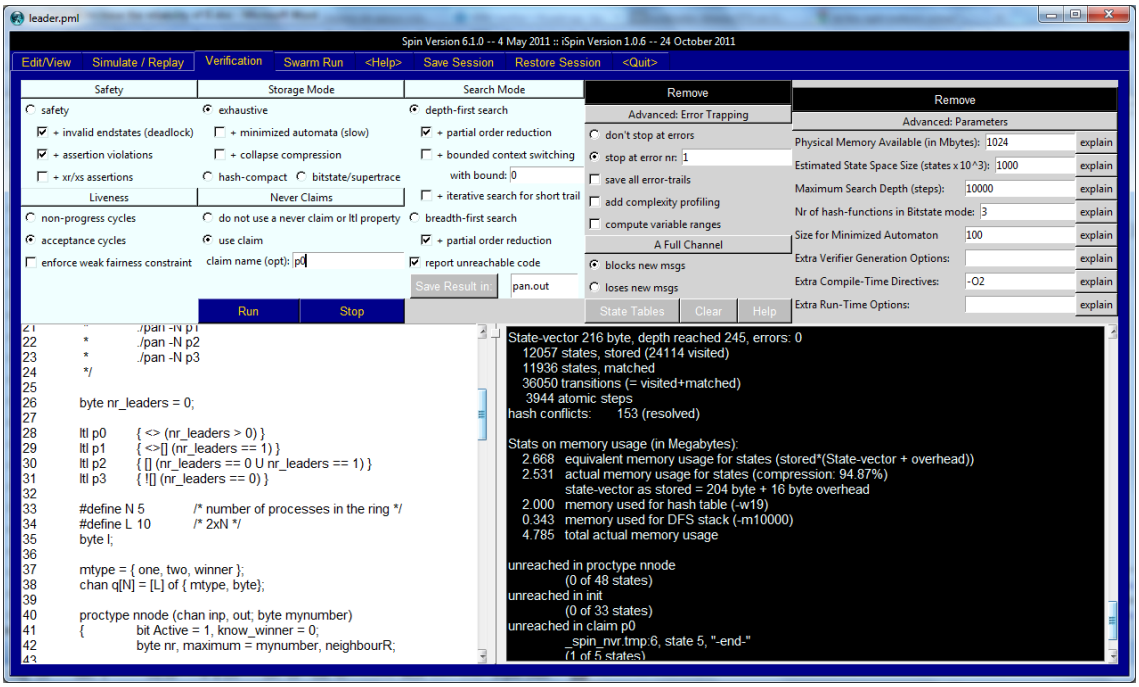

Figure 4. Verification of a correct LTL property

To explain how to operate when a property is false, we have added the property `p4` (see line 32 in Figure 5) to the Promela model. This new property specifies that the leader is never elected. This time, the verification run detects one error and therefore generates a counterexample. To visualize it, we must replay the error trail from the *Simulate/Replay* tab (this is noticed at the bottom of the black panel).

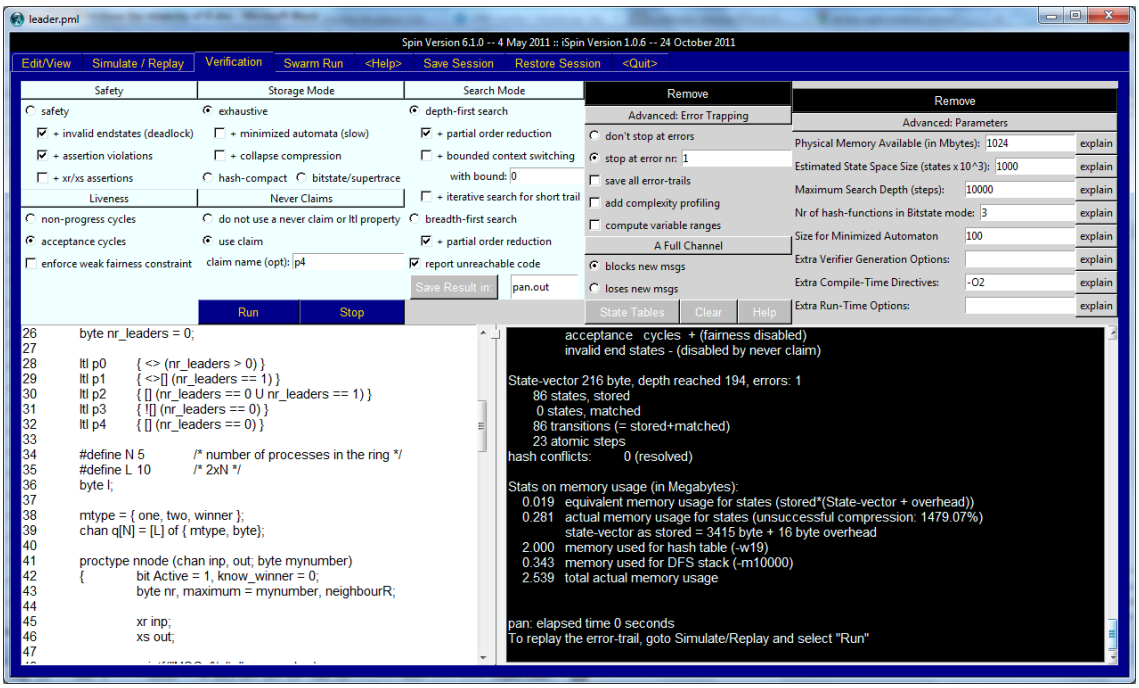

Figure 5. Verification of an incorrect LTL property

Clicking on the simulation tab we notice that the *Guided* mode and the name of the trail file is selected (see Figure 6). Remember that a guided simulation can only be done for an error sequence that was produced earlier by the model checker. Performing the guided simulation replays the error sequence found. In our example case, this indicates that one leader is elected (see last line in the bottom left panel), which constitutes a violation of the property checked.

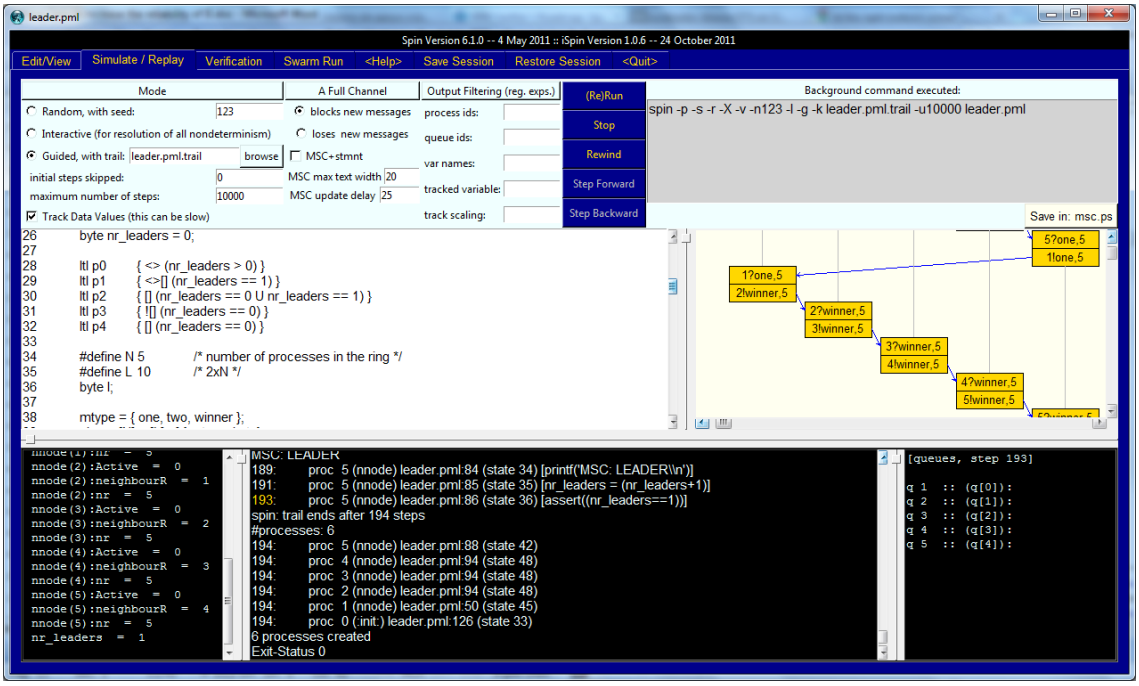

Figure 6. Guided simulation to visualize a counterexample of the property violated

Figure 7 shows the available options in the *Swarm Run* tab, which allows us to configure a swarm verification run that can be quite effective for large models, since it can make

optimal use of large numbers of available computing cores to leverage parallelism and search diversification techniques, which increases the chance of locating defects in very large verification models. We can define how many processing cores are available, how much memory can be used, and how much time is maximally available, among a range of other optional parameter settings. Based on this information, this verification mode generates a script that runs as many independent verification jobs as possible in parallel (using local CPU cores or remote machines in a grid network), without exceeding any of the user-defined constraints. To use this verification mode and the options included in this panel, we must have installed the swarm pre-processor, which can be downloaded from [13].

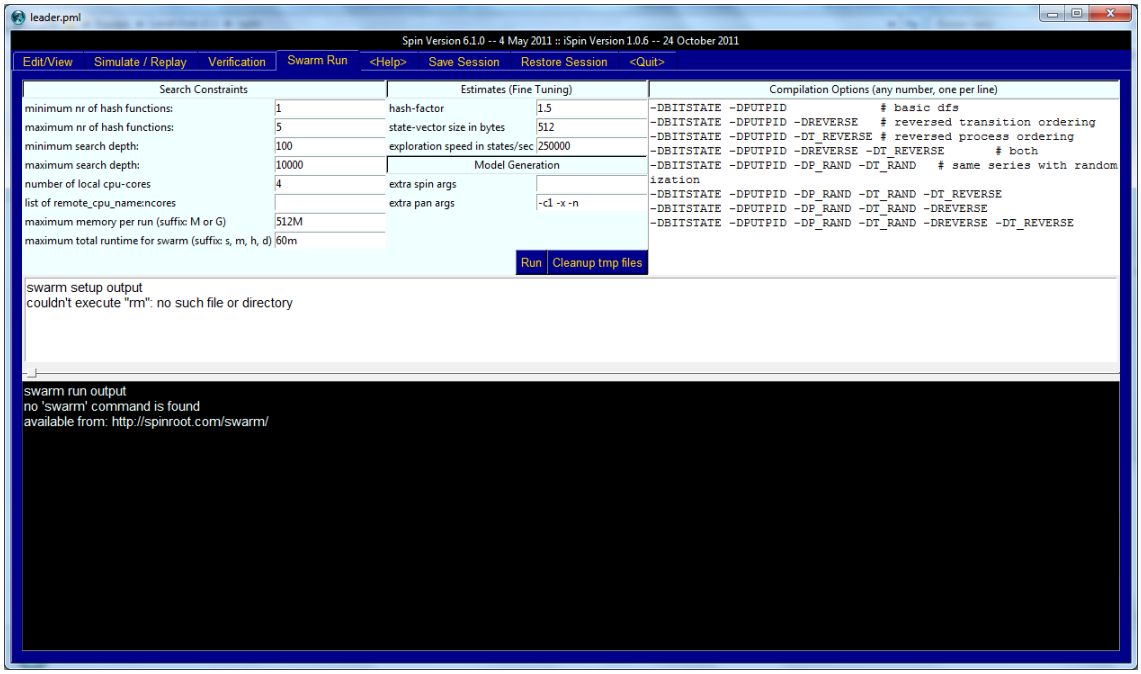

Figure 7. Options in the Swarm Run tab

Clicking on the <*Help*> tab displays a window that gives us some general information about Spin and iSpin, as well as a brief explanation on the main functionalities and options included in each tab of this user interface.

The *Save Session* tab allows us to save the state and contents of all panels and selections made, as well as all textual outputs displayed. These data are recorded in a session snapshot file with file extension .isf.

The *Restore Session* tab restores the iSpin displays and selections to a previously saved state (making use of the precedent tab).

The last tab, namely <*Quit*>, performs an orderly exit from iSpin, cleaning up temporary files, and giving us a warning if we forgot to save a modified model before to exit.

References

[1] G. J. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, Boston, 2003.

[2] M. Ben-Ari, *Principles of the Spin Model Checker*, Springer Verlag, London, 2008.

[3] *Spin Readme*, http://spinroot.com/spin/Man/README.html.

[4] *Cygwin*, http://www.cygwin.com/.

[5] *MinGW*, http://sourceforge.net/projects/mingw/files/MinGW/.

[6] *Spin sources*, http://spinroot.com/spin/Src/index.html.

[7] *Spin Verifier's Roadmap: Using iSpin*, http://spinroot.com/spin/Man/GettingStarted.html.

[8] *Tcl/Tk website*, http://www.tcl.tk/.

[9] *Graphviz - Graph Visualization Software*, http://www.graphviz.org/.

[10] *Promela Manual Pages*, http://spinroot.com/spin/Man/promela.html.

[11] *Proctype*, http://spinroot.com/spin/Man/proctype.html.

[12] *Never claim*, http://spinroot.com/spin/Man/never.html.

[13] *Swarm*, http://spinroot.com/swarm.