# A

# Software Tools

The first section of this Appendix describes how to install SPIN. The following sections present an overview of the SPIN-based tools that I have developed. The tools are written in JAVA and distributed both as source code and as compiled `jar` files that can be run directly. For more detail on each tool, consult the documentation that is included in the distributions. All this software can be freely downloaded from the websites whose addresses are listed in Appendix B.

## A.1 SPIN

SPIN can be downloaded as an executable file for WINDOWS and LINUX. You can also download the source code and build it for any system that has an ANSI C compiler.

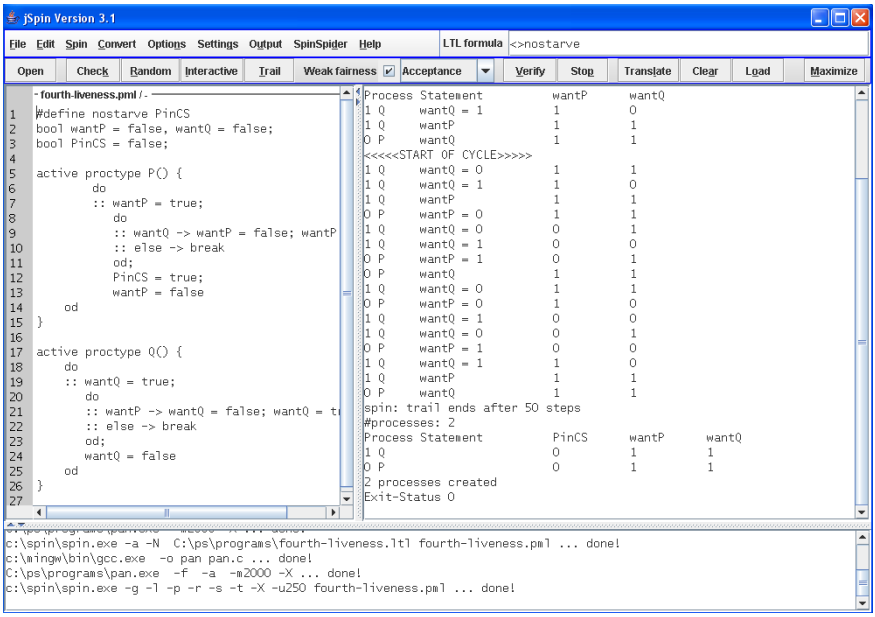
A C compiler is needed to compile the verifier programs that SPIN generates. In WINDOWS, `gcc` can be installed as part of CYGWIN, a LINUX-like environment. If you do not wish to install the CYGWIN environment, `gcc` can be obtained as part of the MINGW (Minimalist GNU for Windows) toolset. Download and open the following archives in the order given:

```
binutils-N.tar.gz
gcc-core-N.tar.gz
mingw-runtime-N.tar.gz
w32api-N.tar.gz
```

where `N` is the version and build number. It is acceptable if some files are overwritten when the archives are opened.

The environment variable `Path` should be set to include the directories containing the SPIN executable file and the C compiler (for MINGW, this

**Fig. A.1.** The JSPIN integrated development environment




defaults to `c:\mingw\bin`). The variable can be changed from the window obtained by selecting:

```
Start/Control Panel/System/Advanced/Environment Variables/PATH
```

You are now ready to use SPIN as a command-line tool.

## A.2 JSPIN

JSPIN is an integrated development environment for SPIN with a graphical user interface (Figure A.1). The user interface consists of a single window with menus, a toolbar and three adjustable panes where text is displayed. All menu and toolbar commands have keyboard mnemonics or accelerators. The left pane is used to display PROMELA source files. The lower pane is used to display messages from both SPIN and JSPIN. The right pane is used to display the output from **`printf`** statements, messages from SPIN (in particular those concerning verification), and displays of data from random, interactive, and guided simulations.

Most of the arguments used by SPIN are supplied automatically by JSPIN, so that you only have to select a button to execute SPIN in one of its modes.

You can explicitly add to or modify the arguments by selecting from the `Options` menu.

During simulation runs the SPIN output is filtered and appears in the right pane in tabular form, one state per line, as described in Sections 3.1.1 and 2.2.2.

JSPIN contains commands for translating formulas in linear temporal logic into never claims and incorporating them into verification runs (Section 5.3.3).

JSPIN is implemented using the SWING library of JAVA for building graphical user interfaces. SPIN, the C compiler, and the verifiers are run by forking subprocesses to execute commands that are built with the proper arguments. The textual output from the subprocesses is piped back to JSPIN for filtering and display.

The SPINSPIDER tool described in the next section is integrated into JSPIN, although it can also be run independently from the command line.

## A.3 SPINSPIDER

SPINSPIDER is a software tool for automatically generating the state transition diagram of a PROMELA program (Figure 4.1). When SPIN performs a verification, it searches the full state space and sufficient information is available on the search to enable the construction of the state diagram. SPINSPIDER works with four input files (Figure A.2):

- the PROMELA source file;
- the debug file obtained by running a verification of the program with the `-DCHECK` option and with a never claim that prints out the program counters and variable values;
- the statement file obtained by running a verification with the `-d` option;
- the trail file of a computation.

The debug file traces the search state by state. It contains data written by a special never claim in the source file; this claim is constructed automatically from information (the number of processes and the variable names) supplied interactively when running SPINSPIDER. The source file and the statement file are used to translate codes in the debug file so that the location counters can be displayed with a line number and the source code. These data are used to create a description of the state diagram in `dot` graphics format. Then, the DOT program is called to layout the diagram and to convert it to a displayable graphics format such as PNG. Optionally, the trail file of a computation can be used to emphasize a path within in the state diagram or to display a diagram consisting only of a single path.

**Fig. A.2.** The structure of SPINSPIDER

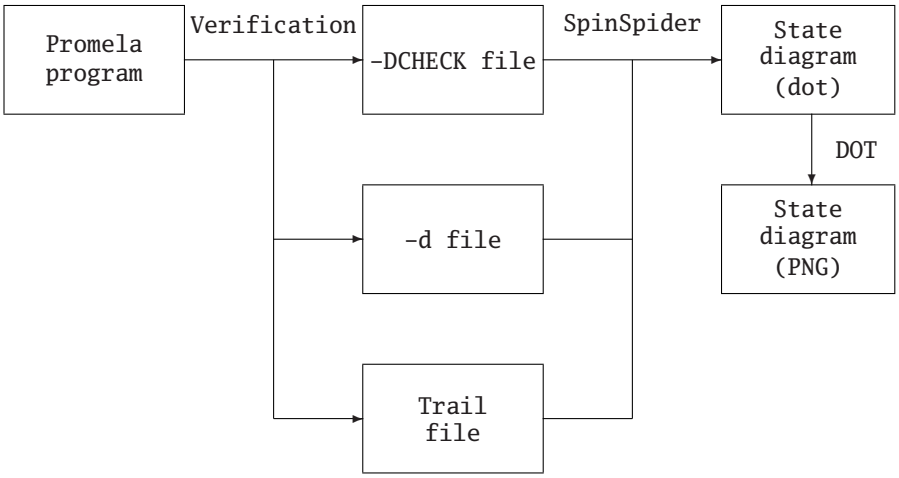



Figure A.3 shows the state diagram that was automatically generated for the program in Listing 5.2 described in Section 5.5. Recall that that program terminates but only for weakly fair computations. Attempting to verify the LTL formula <>`flag` results in an error and the trail is used to emphasize the cycle in the diagram corresponding to an infinite path.

## A.4 VN: Visualizing nondeterminism

The use of the VN software tool for visualizing nondeterminism was described in Section 8.2.

The structure of VN is shown in Figure A.4. An NDFA is first constructed interactively using the JFLAP tool and saved in the XML format that JFLAP defines. VN reads this file and displays the NDFA. A PROMELA program similar to that in Listing 8.1 is generated from the NDFA and an input string. Depending on the mode selected: `Random`, `Create`, `Find` or `Next`, the appropriate commands are built and subprocesses forked. Output from the subprocesses is piped to VN and used to display computations.

VN writes instructions for drawing the NDFA and the paths in the `dot` graphical format, and then calls the DOT tool to lay out the graphs and to convert them into PNG format for display.

**Fig. A.3.** State diagram generated by SPINSPIDER

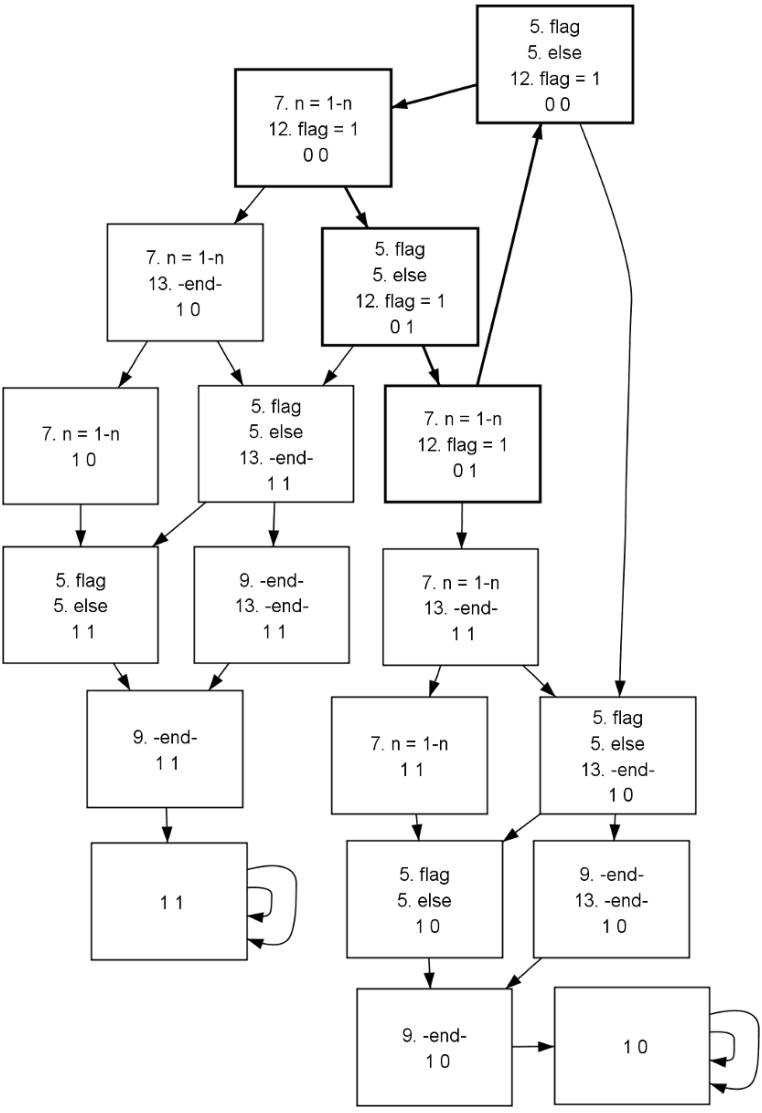
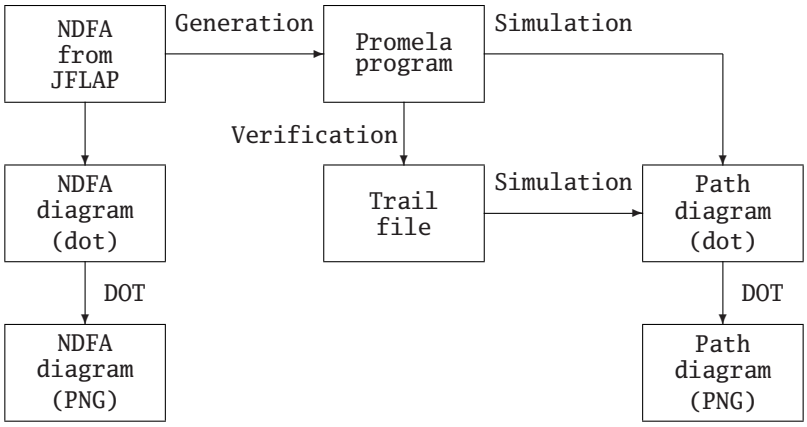


**Fig. A.4.** The structure of VN



# B

# Links

| **Websites for software** | |
|---|---|
| CYGWIN | cygwin.com |
| DTSPIN | www.win.tue.nl/~dragan/DTSpin |
| GRAPHVIZ (DOT) | graphviz.org |
| JFLAP | jflap.org |
| JSPIN, SPINSPIDER | sourceforge.net/projects/pcdp, stwww.weizmann.ac.il/g-cs/benari/jspin |
| MINGW | mingw.org |
| SPARK | www.sparkada.com |
| SPIN | spinroot.com |
| Temporal logic patterns | patterns.projects.cis.ksu.edu |
| UPPAAL | uppaal.com |
| VN | sourceforge.net/projects/pcdp, stwww.weizmann.ac.il/g-cs/benari/vn |

| **Websites for books** | |
|---|---|
| *Principles of the Spin Model Checker* | www.springer.com/978-1-84628-769-5 |
| *Mathematical Logic for Computer Science* | stwww.weizmann.ac.il/g-cs/benari/books |
| *Principles of Concurrent and Distributed Programming* | www.pearsoned.co.uk/ben-ari |
| *The Spin Model Checker* | spinroot.com/spin/Doc/Book_extras |

# References


1. Krzysztof R. Apt and Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer, Berlin, 1991.
2. Michal Armoni and Judith Gal-Ezer. Introducing non-determinism. *Journal of Computers in Mathematics and Science Teaching*, 25(4):325–359, 2006.
3. John Barnes. *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley, Harlow, UK, 2003.
4. M. Ben-Ari. *Principles of Concurrent and Distributed Programming (Second Edition)*. Addison-Wesley, Harlow, UK, 2006.
5. Mordechai Ben-Ari. *Mathematical Logic for Computer Science (Second Edition)*. Springer, London, 2001.
6. Mordechi Ben-Ari and Alan Burns. Extreme interleavings. *IEEE Concurrency*, 6(3):90, 1998.
7. K.M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.
8. Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.
9. Matthew B. Dwyer, George S. Avrunin, and James C. Corbett. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering*, pages 411–420, 1999.
10. Robert W. Floyd. Nondeterministic algorithms. *Journal of the ACM*, 14(4):636–644, 1967.
11. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, Hemel Hempstead, UK, 1985/2004. `http://www.usingcsp.com/cspbook.pdf`.
12. Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, Reading, MA, 2004.
13. Mike Jones. What really happened on Mars Rover Pathfinder. *The Risks Digest*, 19(49), 1997. `http://catless.ncl.ac.uk/Risks/19.49.html`.
14. Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, 2000.
15. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufman, San Francisco, CA, 1996.

16. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, New York, 1992.
17. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995.
18. Susan H. Rodger and Thomas W. Finley. *JFLAP: An Interactive Formal Languages and Automata Package*. Jones & Bartlett, Sudbury, MA, 2006.
19. Theo C. Ruys. *Towards Effective Model Checking*. PhD thesis, University of Twente, 2001. `http://wwwhome.cs.utwente.nl/~ruys/ruys-phd-thesis.pdf`.

# Index