JSPIN - Java GUI for SPIN
User's Guide

Version 5.0

Mordechai (Moti) Ben-Ari
Department of Science Teaching
Weizmann Institute of Science
Rehovot 76100 Israel
http://stwww.weizmann.ac.il/g-cs/benari/

December 15, 2010

1 Introduction

1.1 JSPIN

JSPIN is a graphical user interface for the SPIN Model Checker that is used for verifying concurrent and distributed programs. It is an alternative to the XSPIN GUI and was developed primarily for pedagogical purposes. JSPIN is written in Java, because the Java platform is both portable and widely used in computer science education. The user interface of JSPIN is simple, consisting of a single window with menus, a toolbar and three adjustable text areas. SPIN option strings are automatically supplied and the SPIN output is filtered and formatted. All aspects of JSPIN are configurable: some at compile time, some at initialization through a configuration file and some at runtime. The filtering algorithm in JSPIN can also be run as a standalone program.

1.2 SPINSPIDER

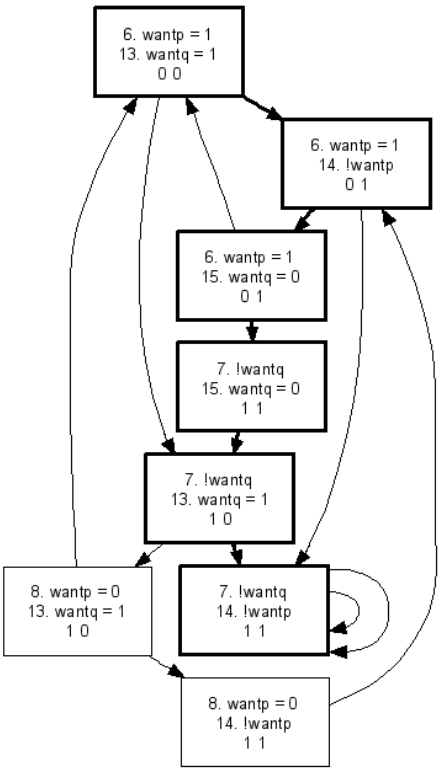
JSPIN includes the SPINSPIDER tool for creating a graphical representation for the state diagram of a PROMELA program by postprocessor the SPIN output. SPINSPIDER is integrated into JSPIN, but it can also be run as a standalone program.

SPINSPIDER generates the complete state diagram of a PROMELA program; the diagrams are useful for demonstrating properties of concurrent programming. Consider the *third attempt* at solving the critical section problem (line numbers added):

```
1.  bool wantp = false, wantq = false;
2.
3.  active proctype p() {
4.      do :: wantp = true;
5.            !wantq;
6.            wantp = false;
7.      od
8.  }
9.
10. active proctype q() {
11.     do :: wantq = true;
12.           !wantp;
13.           wantq = false;
14.     od
15. }
```

The processes p and q are in their critical sections if they are at lines 6 and 13, respectively. The figure on the next page shows the state diagram for this program. Each node contains the source code lines (and line numbers) for the two processes and the values of the variables wantp and wantq. The arrows show the possible transitions between states.

Since no state has the processes together at lines 6 and 13 mutual exclusion is achieved. It is easy to see there is a computation that leads to deadlock in the state with `p` at line 5 and `q` at line 15.



SPINSPIDER processes output of the `pan` verifier in order to create the full state diagram of a program. The diagram is produced in `dot` format and (optionally) the DOT program can be run to produce a graphics file. SPINSPIDER can also produce state diagrams in `fsm` format used by visualization tools developed at the Technische Universiteit Eindhoven (see the URLs below).

1.3 References

- M. Ben-Ari. *Principles of the Spin Model Checker*. Springer, 2008.
- M. Ben-Ari. *Principles of Concurrent and Distributed Programming (Second Edition)*. Addison-Wesley, 2006.
- Gerard J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley, 2004.

1.4 URLs

fsm	http://www.win.tue.nl/~fvham/fsm/
	http://www.mcrl2.org/
GRAPHVIZ (DOT)	http://www.graphviz.org/
JSPIN	http://stwww.weizmann.ac.il/g-cs/benari/jspin/
MINGW	http://mingw.org/
SPIN	http://spinroot.com/

1.5 Acknowledgement

I would like to thank Gerard J. Holzmann for his generous assistance throughout the development of this project.

2 Installation and execution

2.1 Requirements

- JSPIN requires that the Java SDK or JRE (at least version 1.5) be installed.[1]

2.2 Installation

This section describes installation on a Windows system; for custom installation and for other systems, see Appendix A.

- Download the JSPIN installation file called `jspin-N.exe`, where `N` is the version number. Execute the installation file.
- Create a directory `c:\mingw` and download the C compiler archive `mingw.exe` into that directory. Execute this self-extracting file.

2.3 Configuring and running JSPIN

- The installation will create the following subdirectories: `docs` for the documentation, `jspin` and `spinSpider` for the source files, `txts` for the text files (help, about and copyright), and `jspin-examples` and `spider-examples` for example programs.
- To run JSPIN, execute the command `javaw -jar jSpin.jar`. An optional argument names the PROMELA file to be opened initially. A batch file `run.bat` is supplied which contains this command.

[1] The default font for JSPIN is `Lucida Sans Typewriter`. This font may no longer be available in the JRE you use. If you have the fonts from a previous version you can copy them to the `lib/fonts` directory as explained in `http://java.sun.com/j2se/1.5.0/docs/guide/intl/font.html`. Alternatively, you can change the configuration file to use a monospaced font such as `Courier` that is installed by default.

- Configuration data (Appendix B) is in the file `config.cfg`. **When upgrading JSPIN, erase the configuration file before installing a new version, so that new configuration options will be recognized.**
- JSPIN searches for the configuration file in the current directory; if it is not found, JSPIN searches for it in the directory where the jar file is installed; if it is not found there, a new file with default values is written.

2.4 Running SPINSPIDER without JSPIN*

If you want to run SPINSPIDER without JSPIN, create a configuration file with entries for the `SPIN`, `C_COMPILER`, `PAN` and `DOT` commands. The prologues for the `dot` file can be changed in `Config.java` in the package `spinSpider`.

The `SpinSpider` class has a main method that can be invoked from a command line:

```
java -cp jSpin.jar spinSpider.SpinSpider arguments filename
```

`filename` is the source file name. The arguments are:

`-pn`	The number of processes `n`;
`-vname`	One parameter for each variable `name`;
`-dot`	`dot` format;
`-Txxx`	`dot` format followed by conversion to `xxx` format, the default is `png`;
`-fsm`	`fsm` format;
`-t1`	Emphasize trail;
`-t2`	Display the trail only;
`-a`	Display automata for PROMELA source;
`-small`	Small size for states; if absent, large size is used;
`-bold`	Bold for emphasis; if absent, color is used.
`-debug`	Write debug file;

The `-p` and `-v` arguments must be given (unless `-a` is selected).

2.5 Running FILTERSPIN without JSPIN*

To run FILTERSPIN from the command line, first create a file with the output of a (random or guided) simulation:

```
spin -p -l -g -r -s -X src.pml > src.raw
```

Next, run FILTERSPIN:

```
java -cp jSpin.jar filterSpin.FilterSpin src.raw
```

There are two optional arguments: `-v` to give a list of excluded variables and `-s` to give a list of excluded statements, as described in Section 4.1.2. The arguments may be file names or lists of identifiers separated with a separator string (by default `"#"`, but it can be changed in `Config.java`[2]). At least one separator must be given so that the argument is not interpreted as a file name):

```
java -cp jSpin.jar filterSpin.FilterSpin -v "temp#i" -s "i+1#" src.raw
```

The filtered output is sent to standard output and can be piped or redirected as needed. The configuration file must contain values for the properties listed in Appendix B under *filter settings*.

3 JSPIN Overview

The user interface includes a menu, a toolbar and three text areas. Menu and toolbar commands have keyboard mnemonics or accelerators. These can be easily configured by modifying the file `Config.java` and rebuilding.

The left text area is used to display PROMELA source files. The bottom text area is used to display messages from both SPIN and JSPIN. The right text area is used to display the output from SPIN: statements executed, values of variables and results of verifications. The text areas can be resized by dragging the dividers and an area can be hidden by clicking on the triangles in the dividers; the initial sizes can be set in the configuration file. The toolbar button Maximize (with its mnemonic Alt-M) toggles between a normal split pane and a maximized right text area that displays the SPIN output.

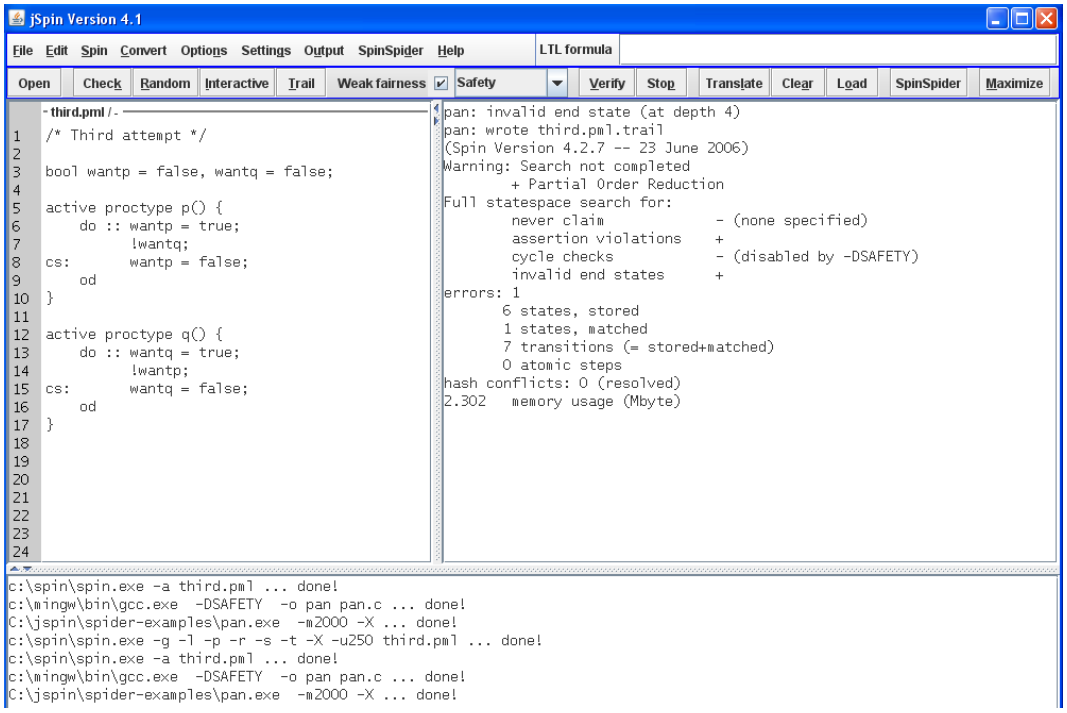
File This menu includes selections for New, Open, Save, Save As, and Exit. Switch file closes the current file and opens the last file that was edited, if any. This is useful for working with both a source file and an included file.

Edit This menu includes selections for Undo, Redo, Copy, Cut, Paste, Find and Find again.

Options This menu enables you to edit the option strings for each of the phases of execution of SPIN and the C compiler. Normally, you will not need to change the options except for those that are more conveniently changed in Options/Common or Settings (Section 4). Default restores the default values and Save saves the current set of option strings in the configuration file. In addition, the following configuration items are automatically saved:

- `SOURCE_DIRECTORY`: the last directory from which a source file was opened.
- The current values of the splitpane dividers `TB_DIVIDER` and `LR_DIVIDER`.
- The current value of the width `SELECT_BUTTON` for interactive simulation.

[2]Do not use a backslash or dollar sign.



The file can be saved to the current or install directory. **Changes in the SPIN options can cause the assumptions on the input to `filter` to become falsified. Be careful!**

Settings This menu sets parameters for the simulation and verification (Section 4).

Output This menu controls the display of the simulation output in the right text area. Output/Maximize toggles the text area to be maximized, hiding the editing area. Output/Save output saves the contents of the text area in a file with extension `.out`. To debug the filtering algorithm, select Output/Raw output to write a file with extension `.raw` that contains the raw SPIN output; Output/Display raw will display this file. The other items in this menu are discussed in Section 4.

SpinSpider This menu enables interactive execution of the SPINSPIDER tool for creating graphical representations for the state diagram of a PROMELA program. See the separate documentation for this tool.

Help Help displays a short help file and About displays copyright information about the software.

4 Running SPIN

In the Spin menu (and on the toolbar) are five selections for running SPIN. They all use the PROMELA source file that has been opened, and save the file before execution. During

simulation and verification, you can select Stop to terminate the SPIN process that has been forked.

Check Runs a *syntax check*.

Random Runs a *random simulation*.

Interactive Runs an *interactive simulation*.

Guilded Runs a *guided simulation* using the trail file created by the execution of the analyzer.

Verify Runs an *verification*.

If you terminate JSPIN while SPIN is running (for example by entering ctrl-C from the command line), make sure to terminate the SPIN process as well.
In Windows this is done by pressing ctrl-alt-del, followed by selecting Task List and Processes. Select spin.exe and End Process.

4.1 Simulation

Check should be run before any simulation run to ensure better display of errors.

Settings/Max steps changes the `-u` option to limit the number of simulation steps, and Settings/Max depth changes the `-m` option to limit the `pan` search depth.

A nonzero value for Settings/Seed will be used as the seed for generating random numbers, enabling a random simulation to be repeated.

4.1.1 Interactive simulation

During an interactive simulation, a dialog frame will pop up with a list of statements that can be executed. The list can be displayed either as a row of buttons, or—if there is not enough room—as a pulldown menu. The choice of the format is dynamically determined according to the value of the configuration option `SELECT_MENU`. By setting this value to an extreme, you can force the list to a certain format. There are also configuration options for setting the width and height of the buttons or menu items.

The dialog can be navigated using the mouse; closing the dialog frame will terminate interactive simulation. For keyboard navigation:

Buttons Tab and Shift-Tab move through the buttons and Space selects the highlighted button. Press Esc to terminate.

Menu Press down arrow to display the list and to highlight the item you want; press Return to select it. Press Esc to terminate.

4.1.2 Filtered output

The contents of the SPIN output can be changed by selecting Options/Common. This pops up a dialog with check boxes to select the SPIN output options: statements (`-p`), local variables (`-l`), global variables (`-g`), messages sent (`-s`) and messages received (`-r`).

Select Output/Exclude variables to create a list of strings defining variables to be *excluded* from the display. Any variable containing a string from the list is not displayed; for example, `:init:` will exclude all variables of the `init` process. If the variable name is prefixed by `+`, it will be included anyway. For exmple, if you have an array variable `test`, then the entries `test` and `+[1]` will exclude display of the elements of `test` except for `test[1]`. The list is saved on a file with extension `.exc`.

Similarly, a file of excluded statements can be created. The file extension is `.exs` and it may be edited by selecting Output/Exclude statements. Exclude statements should *not* be used with interactive simulation.

The output strings from `printf` statements is displayed normally. Optionally, you can request that only strings beginning with `MSC:` be displayed (`MSC=true`).

4.2 Verification

Selecting Spin/Verify or the Verify button on the tool bar performs the three steps required to verify a PROMELA program in SPIN:

- Run SPIN to create an analyzer in files `pan.*`.
- Run the C compiler to compile the analyzer `pan.c`.
- Run the analyzer `pan.exe`.

There are three modes of verification in SPIN, one of which must be selected from the combo box on the toolbar or the radio buttons in the menu Settings:

Safety Checks basic safety properties such as assertions.

Acceptance Checks for acceptance cycles.

Non-Progress Checks that every cycle contains a `progress` label.

See the SPIN reference manual for descriptions of these modes. Checking Weak fairness in the menu Settings or on the toolbar performs the verification only for executions that are weakly fair.

4.3 LTL formulas

A correctness claim specified by a *Linear Temporal Logic (LTL)* formula must be converted to an automaton written as a PROMELA `never` claim. LTL formulas can be contained within the PROMELA source code or they can be placed in external files.

4.4 Internal formulas

An LTL formula can be included in a PROMELA program:

```
ltl { []<>csp && []<>csq }
```

The formula is translated internally by SPIN into a `never` claim.

A number of *named* formulas can be included within a PROMELA program:

```
ltl p0 { [](gate <= 1) }
ltl p1 { [](count == 0) -> (gate == 0)) }
ltl p2 { [](((gate == 0) && notInTest) -> (count == 0)) }
```

To select which claim will be used in a verification, enter the *name* in the LTL formula text field and select LTL name.

The above examples show that internal LTL formulas can have expressions and not just names as atomic propositions.

SPIN automatically negates LTL formulas contained within a PROMELA program.

4.4.1 External formulas

Enter a formula (such as `[]p` or `<>p`) in the text field labelled LTL formula on the menu bar and select Translate from the toolbar or the Convert menu; the formula translated into a `never` claim which will be added to the next SPIN verification. The `never` claim is stored in a file with extension `.ltl`. The formula is not automatically translated, so whatever `never` claim exists, if any, will continue to be used until Translate is invoked again.

Before the formula is translated it is negated by JSPIN. This is done so that the LTL formula as displayed represents the correctness property being checked and not the negated formula which if satisfied represents a counterexample. Should you wish to enter negations yourself as in SPIN, you can change the configuration file option `NEGATE_LTL` to false; this can also be done by toggling Settings/Negate LTL.

LTL formulas are saved in *property files* with extension `.prp`. By default, when you open a PROMELA file, JSPIN opens and loads a `.prp` file of the same name. You can also load a different file by selecting Load from the toolbar or the Convert menu; the file name will be displayed on the toolbar. If the file does not exist, the text field is cleared and the new name is used when you Translate the formula you enter.

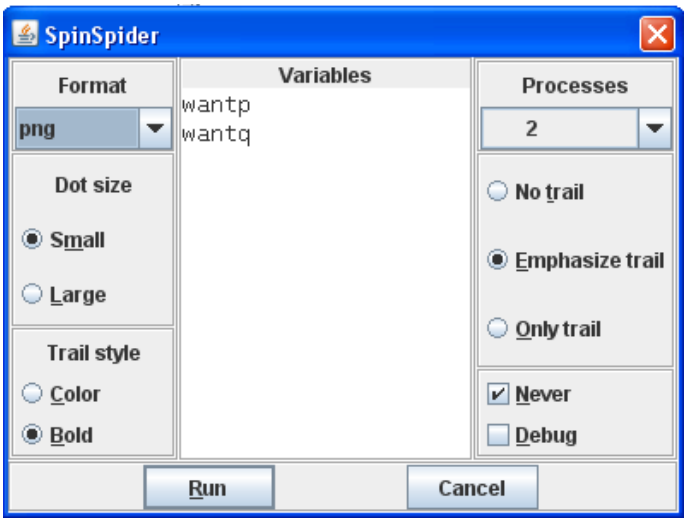
The property file is saved whenever the source file is and also before translation.

Convert/Clear not only to clears the text field, but ensures that no LTL formula will be used in the next verification. After selecting Convert/Clear an empty file will not be saved. Note: In earlier versions of JSPIN, a button Clear was available on the toolbar; this has now been removed and the menu selection must be used.

5 SPINSPIDER

Open a PROMELA program. If you wish to emphasize or draw the computation described by a `trail` file, execute SPIN in verification mode with appropriate assertions or LTL formulas to create the trail file for a counterexample.

Select SpinSpider/SpinSpider (ctrl-D) from the menu or SpinSpider from the toolbar and the following frame will appear:



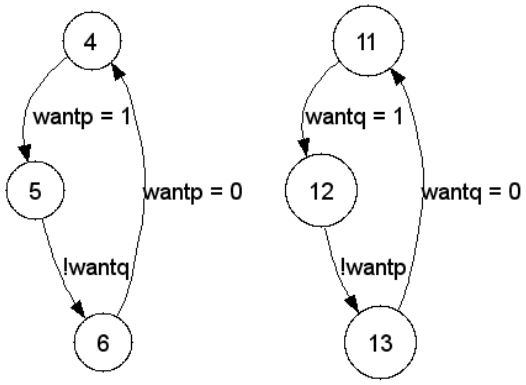
Format The output format, normally dot or one of the formats such as png that can be obtained by running DOT. The fsm format is experimental and the processing of the trail file is not done for this format. **If png is selected as the format, the state diagram will be displayed in a separate frame when SPINSPIDER has terminated.**

Processes The number of processes in the program (for generation of the `never` claim).

Variables A list of the variables in the program (for generation of the `never` claim).

No trail, Emphasize trail, Only trail If the first option is selected, the entire state diagram is created. The second option colors nodes and edges in the state diagram that are part of the trail. The third option creates a diagram containing only those states and nodes that are part of the trail.

Automata This displays the source of the PROMELA program as a set of automata, one for each `proctype`; for the *third attempt*, this is:



The states are labeled with the source code line numbers and the edges with the statements associated with each transition. `atomic` transitions are emphasized.

Dot size Chooses between two prologues for the `dot` file. The small option works better for display within JSPIN.

Trail style Chooses between two ways of emphasizing the trail in a computation: in color or bold. The latter is useful for including the diagram in a document or for users with difficulty seeing colors.

Debug Writes a file (extension `.dbg`) with the internal data structures as collected from the execution of SPIN: states, transitions, statements and the trail. (The file is automatically generated if Automata is selected.) This file can be displayed by selecting SpinSpider/Display debug from the JSPIN menu.

Run Saves the current parameters in a file with the same name as the source file and with extension `.spd`. Then SPINSPIDER is run.

Cancel Returns without saving the parameters or running SPINSPIDER.

5.1 PROMELA programs for SPINSPIDER

The directory spider-examples contains concurrent programs that demonstrate the use of SPINSPIDER.

To reduce the size of the diagrams, use *remote references* rather than ghost variables in the specification of correctness properties. For example, to prover that mutual exclusion holds in the third attempt (`third.pml`), declare labels `cs:` at the two critical sections and then define the symbol:

```
#define notmutex (p@cs && q@cs)
```

Mutual exclusion can shown by doing a verification in Safety mode with the LTL formula:

```
!<>notmutex
```

Recall that SPIN finds the *first* counterexample in a depth-first search, not the shortest one. The trail for the program for the fourth attempt (`fourth.pml`) was created by running `pan` with the `-i` parameter to do an iterative search for a short counterexample.

The line numbers will be accurate if the first alternative of a `do`- or `if`-statement is written on the same line as the keyword. Note that all alternatives are considered to belong to the line containing the keyword.

6 Troubleshooting

SPIN won't execute Check the configuration items for SPIN and the C compiler. It is recommended that they be in your path.

Exceptions in Filter Your own output might be interfering with the filtering algorithm. If configuration option `MSC` is selected, use `printf` only with strings prefixed by `MSC:` and terminated by a newline.

Verification doesn't seem to work Make sure that you have selected the appropriate mode from the Settings menu: Safety, Acceptance, Non-progress. If you are using an LTL formula, Translate it before verifying.

Your computer works slowly Forked processes for SPIN are still executing. Terminate them from the operating system.

Settings, options and the directory aren't remembered You must select Options/Save before exiting SPIN.

Preprocessing failed Simple preprocessing like the use of `#define` is done internally by SPIN. For more complicated processing, SPIN calls the C compiler's preprocessor. This error message can result if there is a problem with the installation; check also that the C compiler is in the path. To debug preprocessing, execute SPIN with the argument `-I`; this will perform preprocessing only on the PROMELA file and display the result.

7 Software structure*

The software is divided into three Java packages.

7.1 Package `jspin`

`jSpin` is the main class and contains declarations of the GUI components and instances of the classes `Editor` and `RunSpin`. Method `init` and the methods it calls initialize the GUI. Method `actionPerformed` is the event handler for all the menu items and buttons; it calls the appropriate methods for handling each event. The About and Help options are implemented by reading files `copyright.txt` and `help.txt`, respectively, and displaying them in a `JFrame` with a scrollable `JTextArea`.

`JSpinFileFilter` is used with a `JFileChooser` when opening and saving files: PROMELA source files, LTL property files and SPIN display output files.

`Options` is the dialog frame with check boxes for changing the SPIN display options.

`Exclude` is the dialog frame for editing the list of variable strings to be excluded from the display.

`Config` contains declarations of compile time configuration items. Method `init` calls `setDefaultProperties` to initialize the instance of `Properties` with the default values of the dynamic configuration items; it then attempts to load the configuration file, and if unsuccessful, the default properties are written to a new file.

`Editor` implements an editor using operations on a `JTextArea`. It implements the interface `DocumentListener` to flag when the source has been modified. The class is also responsible for the LTL formula `JTextField`. `jSpin` calls method `writeFile` to write `ltl` and `out` files, and method `readFile` to read the text files to be displayed.

`LineNumbers` extends a `JComponent` to create line numbers for the `RowHeaderView` of the editor `JScrollPane` (thanks to Niko Myller for this code).

`UndoRedo` was extracted from an example on the Java web site.

The event handler in `jSpin` calls `run` in class `RunSpin` to execute SPIN, the C compiler and the analyzer `pan`. `run` creates a thread of class `RunThread`, and uses `ProcessBuilder` to set up the command, directory, merge the output and error streams, and set up streams for I/O. The call `runAndWait` is used for short calls like Check and the creation and compilation of a model; this call does not return until the completion of the subprocess. The call `run` will return immediately after it has created the thread. In this case, the event handler in `jSpin` calls `isSpinRunning` to create a thread to poll for termination of SPIN; by creating a separate thread, the event handler is freed to accept a selection of Stop.

When `Select a statement` is received during an interactive simulation, method `select` is called. This method displays the choices in the bottom text area and pops up a dialog to enable the user to make a selection. A `JFrame` is created in a new thread of the inner class `SelectDialog` to wait for the selection. `select` polls `selectedValue` which is set

with the selected button value or zero if the frame is closed or Esc pressed. In that case, `q` is sent to SPIN to terminate the simulation.

JSPIN contains source code for interactively invoking SPINSPIDER. The class `SpiderOptions` pops up frame where parameters can be entered and SPINSPIDER invoked. Each PROMELA file can have its own set of parameters; these are stored in property files with extension `.spd` and managed by `SpiderFile`. If the diagram is created in PNG format, it is displayed interactively: `DisplayImage` creates a frame and the image is loaded into an instance of `ImagePanel`.

7.2 Package `filterSpin`

The output filtering is encapsulated in class `Filter`. For each line received from the output stream, `run` (of `RunThread`) calls `filter` which modifies the string; the modified string is displayed in the appropriate text area. A `TreeMap` is maintained for mapping variable names into value strings. Each new variable is checked against the list of excluded strings before it is entered in the map. For standalone filtering, the package contains a `Config` class and a class `FilterSpin` with a main method.

7.3 Package `spinSpider`

Most of the processing of the SPINSPIDER program is done in the class `SpinSpider`. Static data like strings and the prologue for the `dot` file are contained in the class `Config`.

Four classes are used for objects containing information extracted from the files:

- `State` contains the program counter values and the variables names and values that are extracted from the output of the `never` claim in the `.chk` file.
- `Transition` contains the transitions and is obtained by following the `New`, `Old`, `Up` and `Down` debugging trace in the `.chk` file. A stack is used to simulate the depth-first search of SPIN.
- `Statement` contains the states, line numbers and source statements from the `.d` file.
- `Trail` contains the `id` of an entry in the trail file.

The processing sequence is invoked from the method `runSpider`. `runProcess` is called several times to fork processes as described in Section 7.4. The `.d` file is analyzed in `createStatements`, and the `.chk` file is analyzed in `readCheckFile` of class `Read`. If requested, these data structures are written to a file in method `writeDebug` of class `Write`.

The class `SetTrail` traverses the state diagram according to the trail and marks states and transitions that are part of the trail.

The data structures are used to write the files in class `Write`. See the description of the `fsm` format to understand `writeFSMGraph`. In `writeDOTGraph`, the `states` array is traversed

sequentially writing out numbered nodes including labels for each node. The elements of this array store the program counters of each process in the state; these are used to search the `statements` array for *all* statements that start from the program counter. The line numbers and source code of the statements are appended to the label. The variable values are also obtained from the `states` array. The transitions are taken directly from the `transitions` array. Finally, DOT is called, if requested.

The class `DrawAutomata` uses the information in the `.d` file to write a `dot` file and then calls DOT.

7.4 How SPINSPIDER works

SPINSPIDER works with numerous files:

- The source file with extension `.pml`.
- A `never` claim file with extension `.nvr` as described below; this is normally generated automatically.
- The file with extension `.chk` obtained by running a verification of the program with the `-DCHECK` option and with the `never` claim that prints out the program counters and variable values.
- The statement file with extension `.d` obtained by running a verification with the `-d` option.
- A file with extension `.spd` that is used by JSPIN to save SPINSPIDER parameters.
- A debug file with extension `.dbg`.

SPINSPIDER runs the following commands in subprocesses (`prog.pml` is the source file):

```
spin -a prog.pml
gcc -DCHECK -DSAFETY -DPRINTF -o pan pan.c
pan > prog.chk
pan -d > prog.d
```

Then it runs its own algorithm to create a `dot` file. Finally, DOT is optionally run to create a display file in some format like `png`.

A PROMELA program must be modified by giving a special `never` claim; this is generated automatically and included using the `-N` argument to SPIN.

The `never` claim contains a single `printf` statement in a loop, which is executed with every step of the verifier. The statement prints the following tokens separated by spaces:

- The string `*spd*`;
- The number of processes, followed by the values of `pc_value` for each process;

- The number of variables, followed by their names and values.

For the algorithm shown in the introduction the `never` claim is:

```
never {
  do :: printf("*spd* 2 %d %d 2 wantp %d wantq %d \n",
        pc_value(0), pc_value(1), wantp, wantq)
  od
}
```

A Installation

A.1 SPIN version

Version 6.0.0 of SPIN changed its output format; starting with version 5.0, JSPIN expects this format. You can still run JSPIN with earlier versions of SPIN (for example, 4.30): set the configuration option `VERSION` to a number less than 6 before you run JSPIN and ensure that the option `SPIN` names the earlier version of the SPIN executable (or copy the file over the later version).

A.2 Custom installation of JSPIN

- Download the JSPIN distribution file called `jspin-N.zip`, where `N` is the version number. Extract the files into a clean directory.
- Install SPIN and DOT (DOT is only needed for SPINSPIDER).
- Install an ANSI C compiler.
- Modify the configuration file `config.cfg` to reflect the locations of the programs.

A.3 Building JSPIN

To rebuild JSPIN, execute `build.bat`, which will compile all the source files and create the file `jSpin.jar` with the manifest file.

A.4 Installing a C compiler

The `gcc` compiler is normally used with SPIN. On Windows, there are two distributions: Cygwin (`http://cygwin.com`) is a comprehensive Linux-like environment, and a smaller distribution called MinGW (`http://mingw.org/`). To install MinGW, download the following archives and open them in directory `c:\mingw` *in the following order*:

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

where V is the version and build number. It is OK if some files are overwritten when opening the archive.
Set the path in

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

to include `c:\mingw\bin`.

A.5 Installing JSPIN on MAC OS X

Thanks to Christiaan Ottow for providing this material.

Currently, there is no precompiled version of SPIN, so you will have to compile it from the source code. Instructions for doing this are given in Section 2c of the webpage:

```
http://spinroot.com/spin/Man/README.html
```

An alternative approach is change the location of the preprocessor specified in lines 75–88 of `main.c` from `/lib/cpp` to `/usr/bin/cpp`.

JSPIN is compiled with `-target 1.5` to conform with the JAVA version currently available on the MAC.

To install JSPIN:

- Unzip the file `jspin-V-V.zip`.
- Download GRAPHVIZ from

  ```
  http://www.pixelglow.com/graphviz/download
  ```

  Open the `dmg` file you downloaded and copy the GRAPHVIZ application to the Applications folder.
- Open the configuration file `config.cfg` in a text editor and change the values of the following properties:
  - `SPIN` to the directory with the compiled SPIN;
  - `SOURCE_DIRECTORY` to the subdirectory `jspin-examples` of the directory where you unpacked JSPIN;
  - `C_COMPILER` to directory containing `gcc`, usually `/usr/bin/gcc`. You can verify this by issuing the command `which gcc`;
  - `PAN` to `pan`;
  - `DOT` to `/Applications/Graphviz.app/Contents/MacOS/dot`.

JSPIN can now be run by using the `Terminal` to execute the following command in the `jspin` directory:

```
java jspin.jSpin
```

A.6 Installing JSPIN on LINUX

Thanks to Vsevolod Krishchenko for providing this material.

- Install gcc, C preprocessor, dot, java. For Debian/Ubuntu:

```
sudo apt-get install gcc cpp graphviz sun-java6-jre
```

- Install spin as described above.
- Make the following changes in `config.cfg`:

  ```
  C_COMPILER=gcc
  DOT=dot
  ```

  You many need to set `SINGLE_QUOTE` to `true`.

- Create a shell file with:

  ```
  #!/bin/sh
  java -jar jSpin.jar
  ```

B Configuration file

These tables give the properties in the configuration file and their default values.

Version of SPIN	
VERSION	6

Directories for source and compilers	
SOURCE_DIRECTORY	"jspin-examples"
C_COMPILER	"c:\\mingw\\bin\\gcc.exe"
SPIN	"bin\\spin.exe"
DOT	"bin\\dot.exe"
PAN	"pan"

Options for executing SPIN	
COMMON_OPTIONS	"-g -l -p -r -s"
CHECK_OPTIONS	"-a -v"
RANDOM_OPTIONS	"-X"
INTERACTIVE_OPTIONS	"-i -X"
VERIFY_OPTIONS	"-a"
C_COMPILER_OPTIONS	"-o pan pan.c"
PAN_OPTIONS	"-X"
TRAIL_OPTIONS	"-t -X"
TRANSLATE_OPTIONS	"-f"
MAX_STEPS (-u)	250
MAX_DEPTH (-m)	2000
SEED (-n)	0
FAIRNESS	true
RAW	false
SINGLE_QUOTE	false
NEGATE_LTL	true
VERIFY_MODE	"Safety"

Filter settings	
PROCESS_TITLE	Process
PROCESS_WIDTH	7
STATEMENT_TITLE	Statement
STATEMENT_WIDTH	18
VARIABLE_WIDTH	10
LINES_PER_TITLE	20
MSC	false

Text settings	
WRAP	true
TAB_SIZE	4
FONT_FAMILY	"Lucida Sans Typewriter"
FONT_STYLE	java.awt.Font.PLAIN
FONT_SIZE	14

Frame size	
WIDTH	1000
HEIGHT	700

Interactive dialog settings	
SELECT_BUTTON	120
SELECT_HEIGHT	70
SELECT_MENU	5

Location of dividers	
LR_DIVIDER	400
TB_DIVIDER	500
MIN_DIVIDER	50

Delay while waiting for user input	
POLLING_DELAY	200