Tutorial 2: Promela/Spin

CISC422/853

Scott Grant

Overview

- Installing Spin
- Starting Spin
- Running Spin
- General Usage and Tips
- Advice for Assignment 2

Installing Spin

- Spin has a few distributions
  - **Xspin** is the main one, **jSpin** is a Java GUI
  - If you're on a Linux machine at home, you should be fine with Xspin (gcc, tcl)
  - Installing Xspin requires you to jump through a few hoops on a Windows or Mac machine
  - It's possible though, so this presentation will explain how to get set up with Xspin and the MinGW compilers on a Windows machine
    - (sorry, Mac users, I'm not going there on mine)

Installing jSpin

- Windows and Mac users might prefer this
  - http://stwww.weizmann.ac.il/g-cs/benari/jspin/
    - (or use Google to search for "jSpin")
- Download the following files:
  - mingw.exe (Windows GNU compilers)
  - jspin-4-6.exe (jSpin installation)

Installing Xspin

- First, install the Spin model checker
  - http://spinroot.com/
  - Click "Downloading and installation"
    - http://spinroot.com/spin/Man/README.html
  - Skim through the document until Section 2
    - Installing Spin is broken up into sections based on your operating system
    - I'm going to cover the Windows install in detail here
    - If you try installing at home on Linux or OSX, let me know how it goes!

Installing Xspin

- Download the Spin executable from Section 2b. Installing Spin on a Windows PC
  - http://spinroot.com/spin/Bin/index.html
  - As of today, the file is *spin517.exe*
- Save that file somewhere convenient, and to make things easy, rename it to spin.exe
- Don't forget where you saved that file, jeez!

Installing Xspin

- Next, download the Xspin GUI
  - In Section 2b, the first two links are to the binary and source distributions of Spin
    - You went to the first link initially in order to get your spin.exe file
    - This time, visit the second link: http://spinroot.com/spin/Src/index.html
    - Download the most recent version of the xspin GUI (xspin only), which is currently called *xspin510.tcl*
    - Save this somewhere, maybe even with the spin.exe file from before

Installing Xspin

- Next, download a Tcl/Tk interpreter
  - wtf is tcl.
  - The Xspin GUI is written in the Tcl (tickle, I kid you not) scripting language
  - In order to run it on your machine, download an interpreter like ActiveTcl
    - http://www.activestate.com/activetcl/
    - or just Google activetcl
  - This will allow you to run tcl scripts

Installing Xspin

- Next, install MinGW
  - The GCC compilers have been ported to Windows, and since Spin generates and compiles C code for full verification, you'll need a C compiler
    - http://www.mingw.org/
    - Remember where it installs to!
      - (Probably c:\mingw)
  - Other compilers are fine, if you've got something else installed already

Installing Xspin

- Finally, modify the xspin510.tcl script
  - Remember, Xspin is just a script, so you can open it in your favourite text-editor
  - Xspin makes certain assumptions about program locations, and you'll need to update the script with your local settings
  - You'll need to update the location to the gcc compiler, and the location to the spin executable

Installing Xspin

- Update CC, CC0, CPP, and SPIN with your relative paths

```
41  #    => with Tcl/Tk 7.5/4.1 or later, this happens automatically
42
43  # set CC   "cc -w -Wl -woff,84" ;# ANSI-C compiler, suppress warnings
44  # set CC   "cl -w -nologo"  ;# Visual Studio C/C++ compiler, prefered c
45    set CC   "c:/mingw/bin/gcc -w"    ;# standard gcc compiler - no warn:
46    set CC0  "c:/mingw/bin/gcc"
47
48  # set CPP  "cpp"    ;# the normal default C preprocessor
49    set CPP  "c:/mingw/bin/gcc -E -x c" ;# c preprocessor, assuming we ha
50
51    set SPIN "C:/Docume~1/scott/Desktop/spin" ;# use a full path-name if
52    set DOT  "dot"  ;# optional, graph layout interface
```

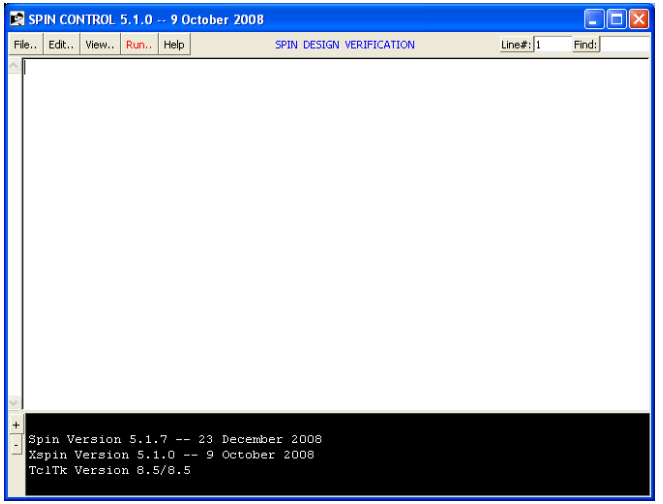
Installing Spin

- Is there a difference?
  - Probably not. If you find one easier than the other, feel free to use that one.
  - I find jSpin slightly easier to use thanks to the simpler GUI (one window, one-click buttons)
  - jSpin's distribution uses an old version of Spin (4.3.0, current version is 5.1.7), but again, there's probably no difference
  - I'll almost certainly be verifying assignment code in Xspin to be safe, but if that doesn't work for an assignment, I'll try verifying in jSpin

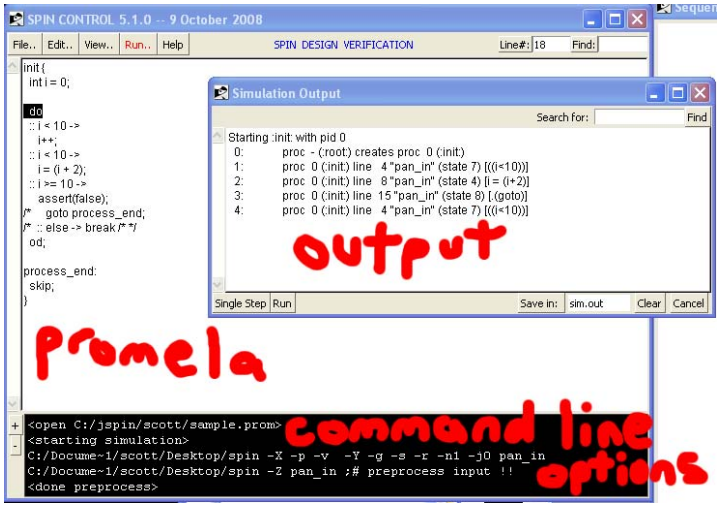
Starting Spin

- Unless you're hardcore, you'll be using a GUI to interact with the Spin engine
- Why on earth is this detail important?
  - Xspin is not Spin, and neither is jSpin!
  - The GUI facilitates your use of a command-line model checker
  - If you're curious why Spin is giving you results in a certain format, you can see exactly what the buttons in your GUI are telling it what to do

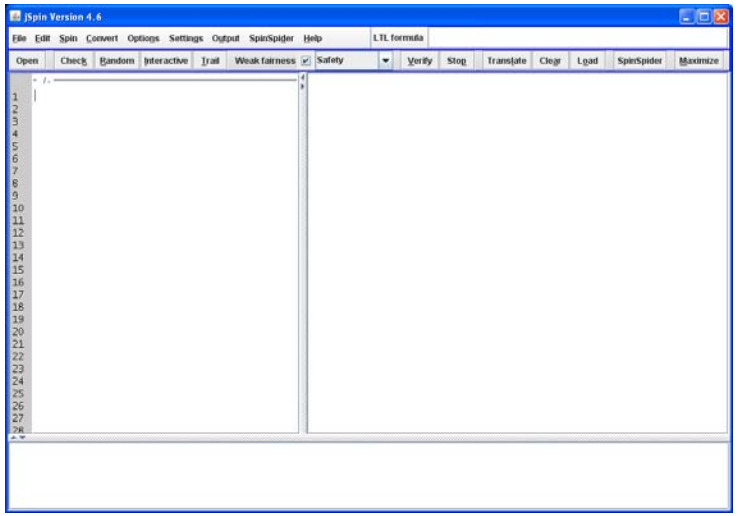
Starting Xspin



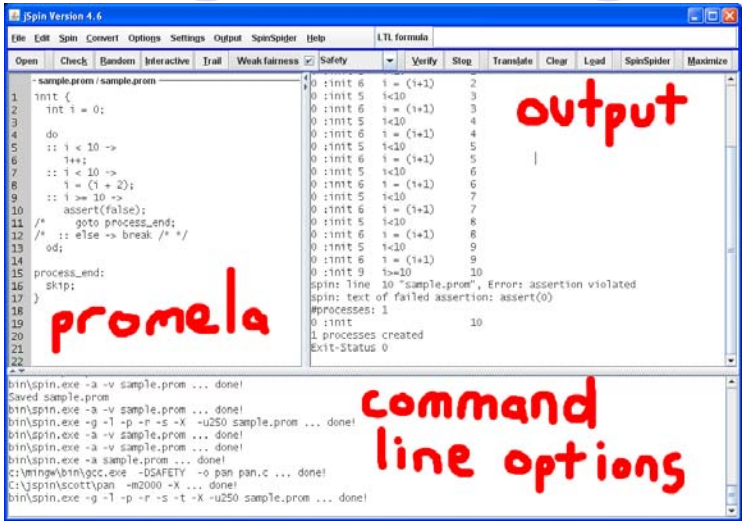
Starting Xspin



Starting jSpin

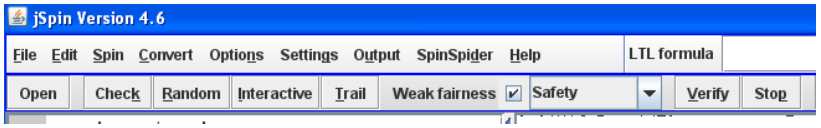


Starting jSpin



Running Spin

- How do you actually run Spin?
  - *Check*: generate a verifier for your specification
  - *Random*: view the path of a random walk
  - *Interactive*: manually select each decision that the model checker can make
  - *Trail*: view the results of an error trace
  - Verify: perform a guided verification of the model (ie, find errors, if they exist)



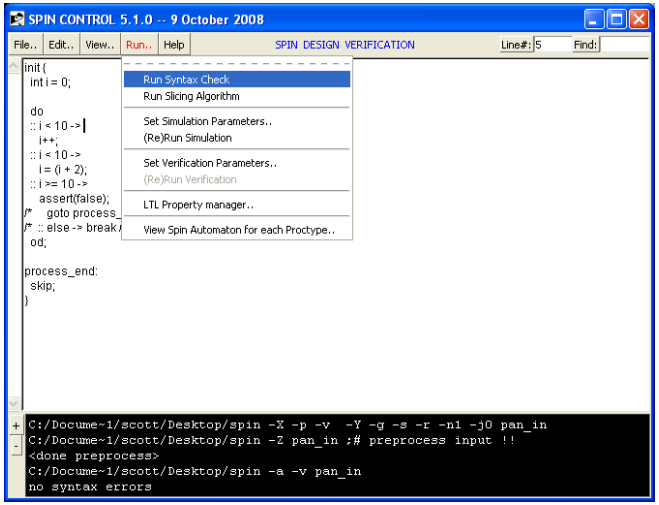
Running Spin

- Check
  - Spin generates C code from your Promela source, which is then compiled and analysed
    - (This is why you need a compiler like gcc or MinGW)
  - The Check button tells Spin to create this code, and acts primarily as a syntax checker
    - ie, Is my code properly formed?

bin\spin.exe -a -v sample.prom ... done!

-a Generate a verifier (model checker) for the specification.
-v Verbose mode, adds some more detail, and generates more hints and warnings about the model.

Running Spin



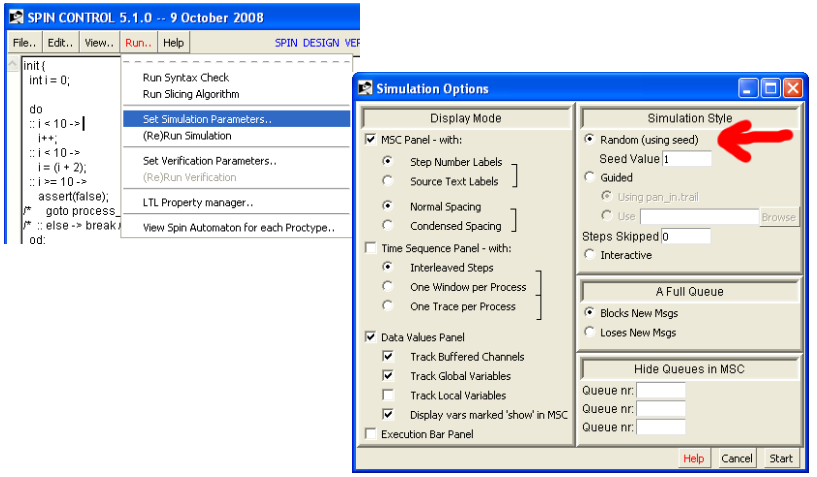
Running Spin

- Random
  - Runs a random simulation on your model
  - Not guaranteed to find errors, and certainly not an exhaustive search
  - However, this is fast, and can help you track down problems in your model

bin\spin.exe -g -l -p -r -s -X -u250 sample.prom ... done!

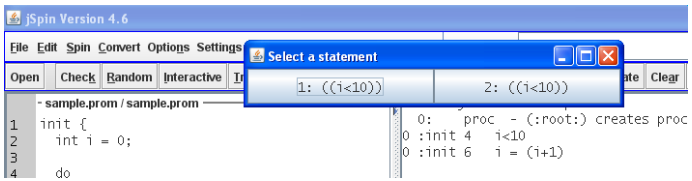
-p Shows at each simulation step which process changed state, and what source statement was executed.
-uN Stop a random or guided simulation after the first N steps.

Running Spin

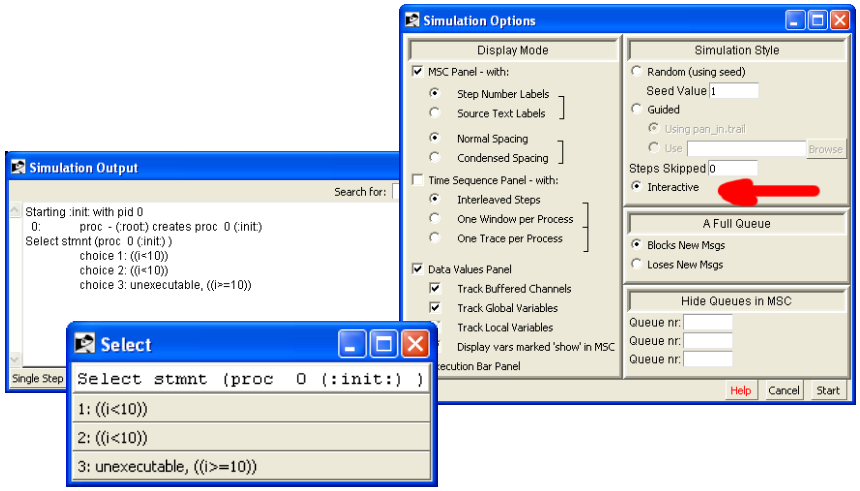


Running Spin

- Interactive
  - Runs an interactive simulation on your model
  - At each decision point in the model where multiple paths can be taken, you will be given the choice to decide which one to follow
    - If you want to test edge cases where you believe something will break, this is extremely helpful!



Running Spin

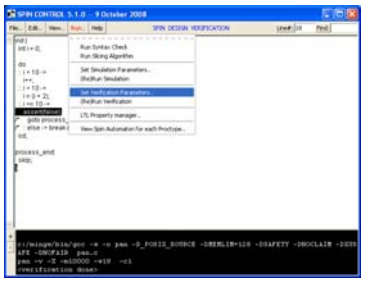


Running Spin

- Trail
  - Runs a guided simulation using the trail file created by the execution of the analyzer
  - Xspin generates a graphical representation of the trail in the form of a Message Sequence Chart, and jSpin gives you a wall of text
  - Both are useful, but make sure you read what's going on very carefully!
    - One example of the need for caution is the fact that the number of columns in jSpin's (read: Spin's) trail output changes based on how many variables it's tracking, so please read carefully

Running Spin

- Verify
  - Runs a full verification of your model
  - If errors are found, you'll see a message like the following:
    - pan: assertion violated 0 (at depth 10)



Running Spin

```
pan: assertion violated 0 (at depth 10)
pan: wrote sample.prom.trail
(Spin Version 4.3.0 -- 22 June 2007)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim         - (none specified)
        assertion violations    +
        cycle checks        - (disabled by -DSAFETY)
        invalid end states +
State-vector 16 byte, depth reached 10, ••• errors: 1 •••
     11 states, stored
      0 states, matched
     11 transitions (= stored+matched)
      0 atomic steps
hash conflicts: 0 (resolved)
2.302     memory usage (Mbyte)
```

Advice for Assignment 2

- Part 1
  - A single paragraph answer should be enough to explain what is going on.
    - Be brief, and don't write a full page answer, but make sure you say enough to show you understand what the code is doing.

Advice for Assignment 2

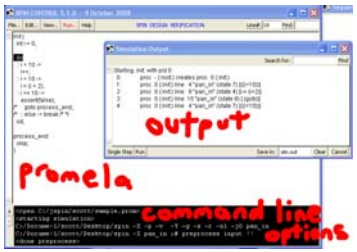
- Part 1
  - The assignment states that you cannot use model checking to determine what is going on, but you can definitely use Spin to help guide you to a solution.
    - Substitute different values into the str[] array and see what's going on.
  - To help reduce the overall complexity, you can also try reducing the size of the MAX constant.

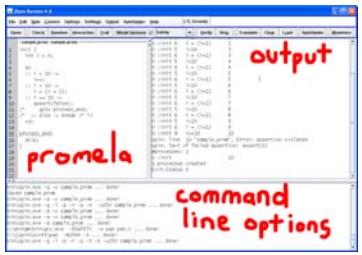
Advice for Assignment 2

- Part 2, Q1
  - Make sure you understand what your code is doing!
    - It is not enough to have code that looks like it's doing the right thing; it is important to understand how the model is being verified
  - As in Part 1, your answer should be concise, while covering each of the points listed
    - Obviously a sentence is too short, but don't worry, this is not an essay
    - Just describe how your model satisfies the requirements

Advice for Assignment 2

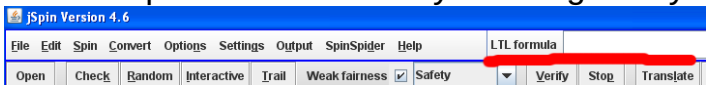
- Part 2, Q2, Q3
  - Show the command line parameters, and the output generated by Spin
    - If you're using Xspin or jSpin, the earlier slides show where to find this output

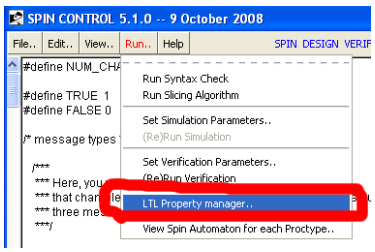


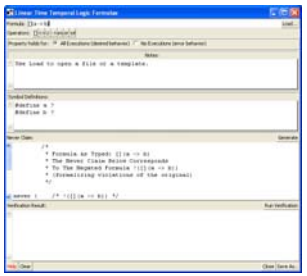


Advice for Assignment 2

- Part 2, Q4, Q5, Q6
  - You can enter your LTL statements in jSpin or Xspin in order to verify them against your model







Advice for Assignment 2

- Part 2, General Advice
  - The assignment gives the following tip:
    - After an analysis, Spin tells you which parts of your Promela code were "unreached"; use that to avoid unreached code in your solution as much as possible.
  - Please heed this advice!
    - A model should not contain lots of unused code, and if things are too messy, this will impact the "quality of your code" marking metric

Advice for Assignment 2

- Part 2, General Advice
  - How exactly do you see the unused sections?
    - After you do a full verification of your model, you'll see a list of the unreached states
    - 0 unreached states = all code is used in the model

```
unreached in proctype chameleon
        line 72, state 1, "gone[id] = 1"
        line 73, state 2, "-end-"
        (2 of 2 states)
unreached in proctype frog
        line 83, state 3, "(1)"
        line 89, state 6, "-end-"
        (2 of 6 states)
unreached in proctype :init:
        (0 of 16 states)
```

```
unreached in proctype chameleon
        (0 of x states)
unreached in proctype frog
        (0 of y states)
unreached in proctype :init:
        (0 of z states)
```

Advice for Assignment 2

- Part 2, General Advice
  - Be aware of the following:
    - "Using the provided lock mechanism to guarantee that the frog does not see unstable intermediate states associated with two chameleons changing color"
  - This can be very tricky, so please be cautious!
    - One of the chameleons will change colour before the other does, so you need to make sure the frog doesn't intercept during this transition state

Advice for Assignment 2

- Part 2, General Advice
  - Using the *atomic* and *timeout* keywords are not acceptable ways to solve the assignment
  - The only *atomic* block in your assignment should be the one that's already given in the assignment code

References for Assignment 2

- Some references for Assignment 2:
  - http://spinroot.com/spin/Doc/SpinTutorial.pdf
    - A fairly comprehensive description of the Promela language, and how Spin goes about verifying models
    - The first half is relevant, so don't worry about the detailed memory analysis or anything beyond the scope of the language syntax
  - http://spinroot.com/spin/Man/Spin.html
    - If you look closely at the command-line parameters, you can understand what the Xspin and jSpin GUIs are specifically telling the Spin engine to do