Distributed Systems Programming F29NM1

SPIN: Simple Promela INterpreter

Andrew Ireland

School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Overview

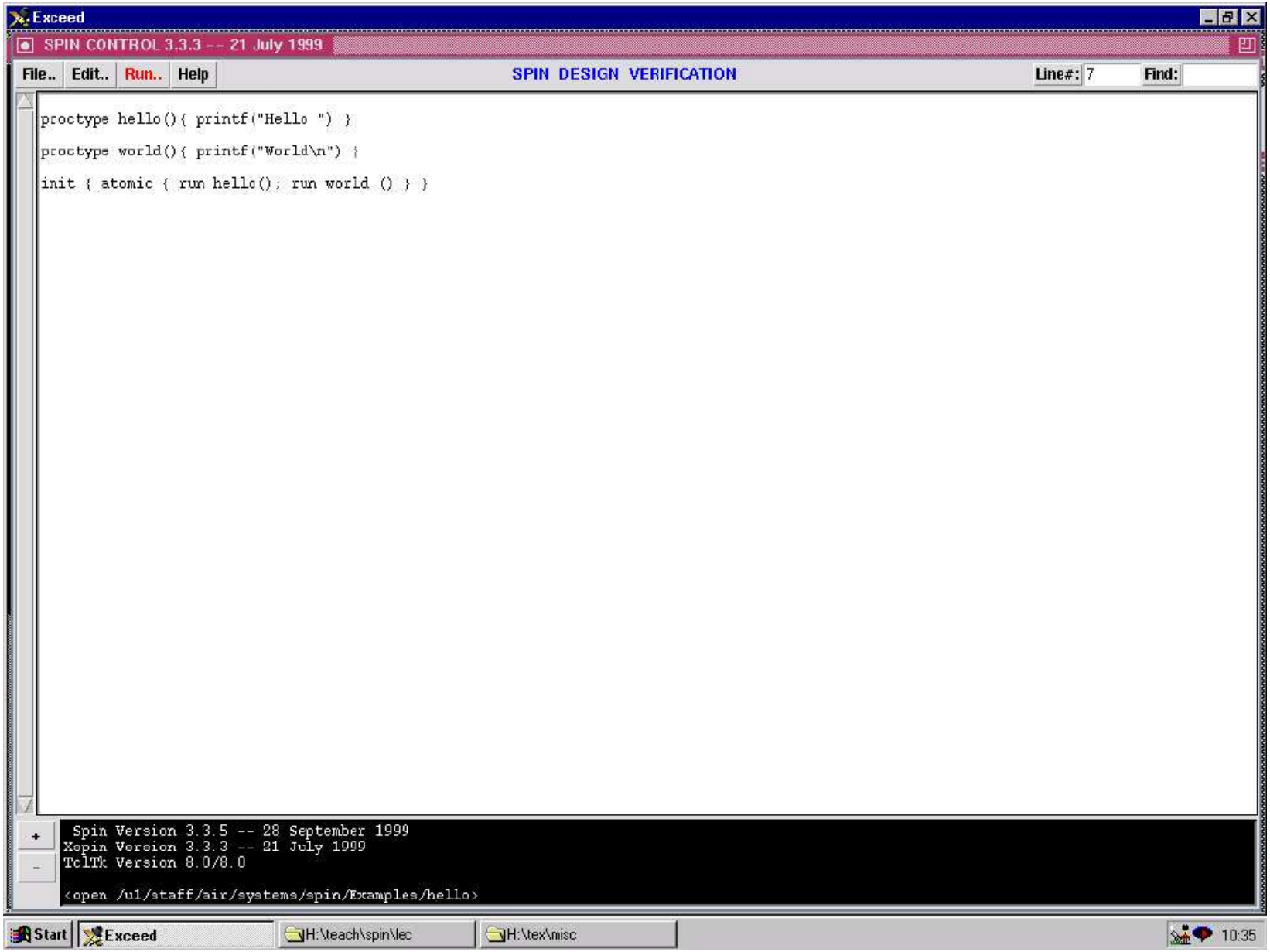
- Provide an introduction to SPIN via XSPIN.
- Focus upon XSPIN's simulation capabilities.
- Introduce the notion of assertion checking.

SPIN & Simulation

- SPIN provides a simulator that enables designers to gain early feedback on their system models. Such feedback plays an important role in the development of the designer's understanding of the design space before they invest in any formal analysis.
- While the SPIN simulator does not represent a formal analysis tool, it does provide a limited form support for verification in terms of **assertion checking**, *i.e.* the checking of local and global system assertions (or properties) during particular simulation runs.

Getting Started with SPIN

- Best way to get started with SPIN is via XSPIN.
- XSPIN is available on all the Linux PCs within Lab 2.50, simply type `xspin` at the prompt.
- The look-and-feel of XSPIN:
  - Main text window: basic edit & search capabilities.
  - File handling: basic browsing, loading, saving capabilities.
  - File editing: cut, copy & paste.
  - Tool support: syntax checking, simulation, verification, automaton portray.
  - Help facilities.
- XSPIN's main window showing the "Hello World" program is pictured on the next slide ...

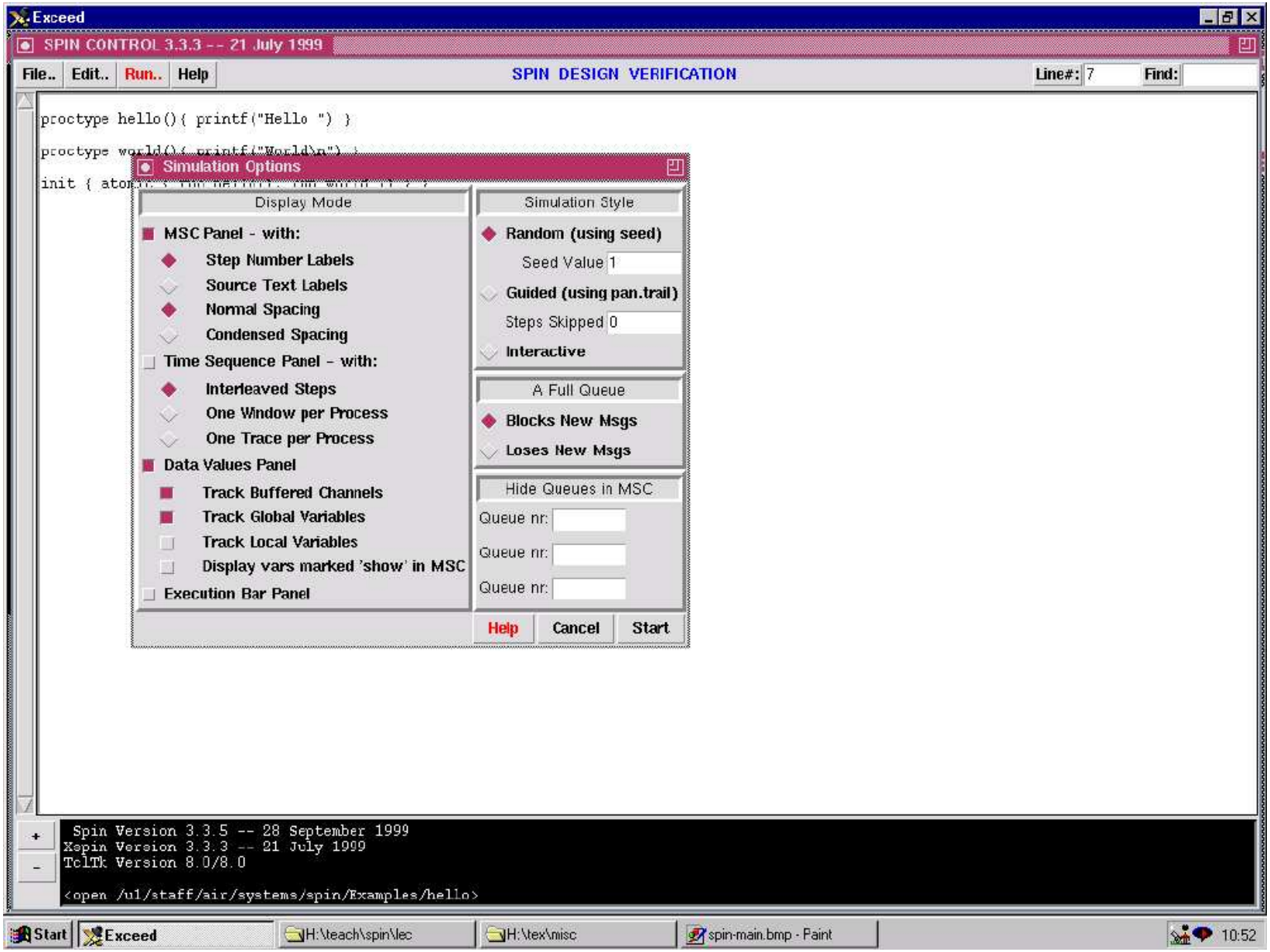

Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File..
Edit..
Run..
Help
SPIN DESIGN VERIFICATION
Line#: 7
Find:
proctype hello(){ printf("Hello ") }
proctype world(){ printf("World\n") }
init { atomic { run hello(); run world () } }
Spin Version 3.3.5 -- 28 September 1999
Xspin Version 3.3.3 -- 21 July 1999
TclTk Version 8.0/8.0
<open /u1/staff/air/systems/spin/Examples/hello>
Start
Exceed
H:\teach\spin\lec
H:\tex\misc
10:35

Simulation: Setting Display Mode Parameters

- Message Sequence Chart (MSC) Panel: provides a graphical presentation of inter-process communication over time. Control over the presentation of links between the MSC and Promela code within the main text window is supported via this panel.
- Time Sequence Panel: provides a graphical presentation of process execution over time. Multiple perspectives are supported, *i.e.* execution steps interleaved; one window per process; one execution trace per process.
- Data Values Panel: presents data values across time. Options include buffered channels; global and local variables.

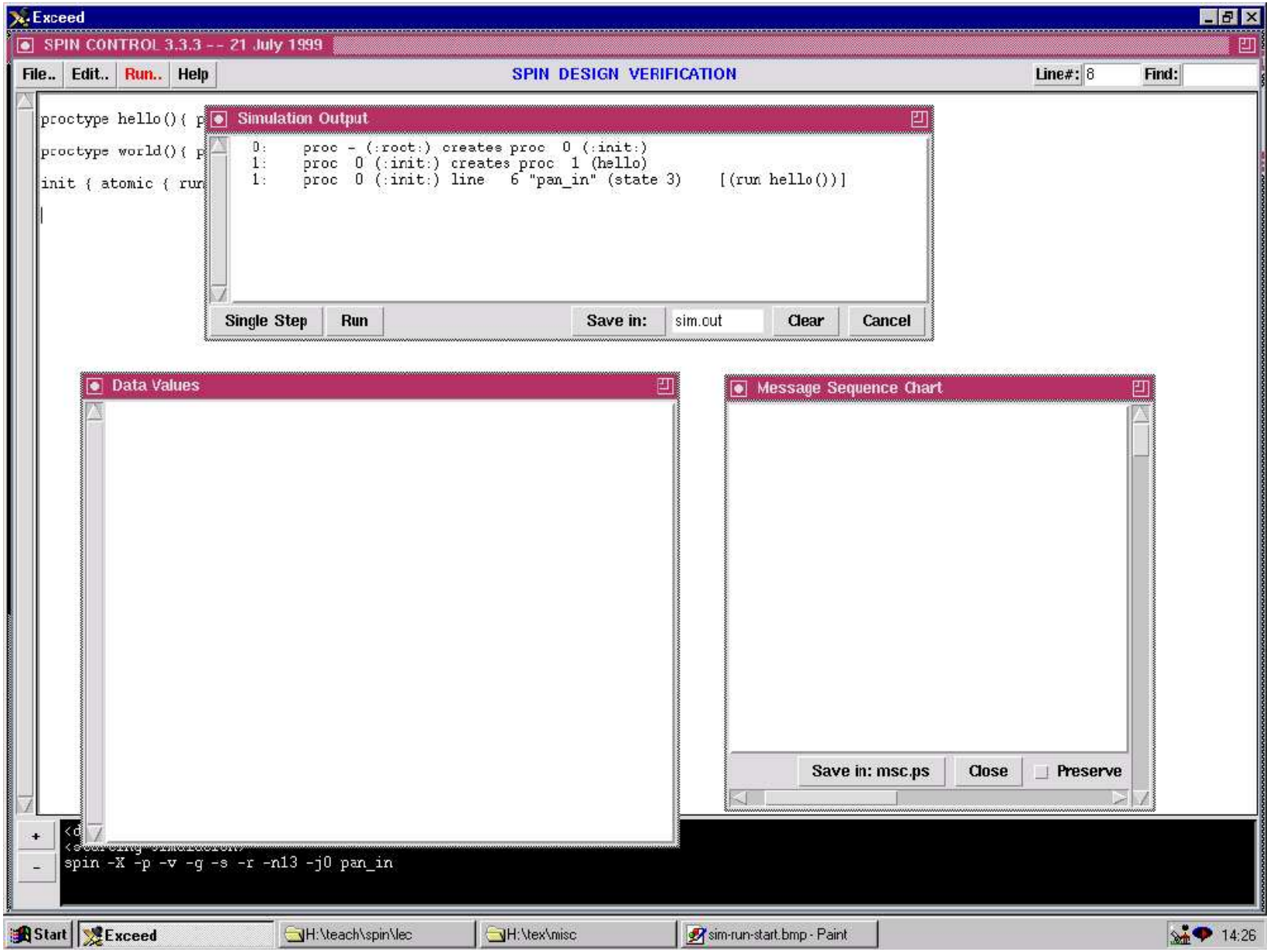
Simulation: Setting Style Parameters

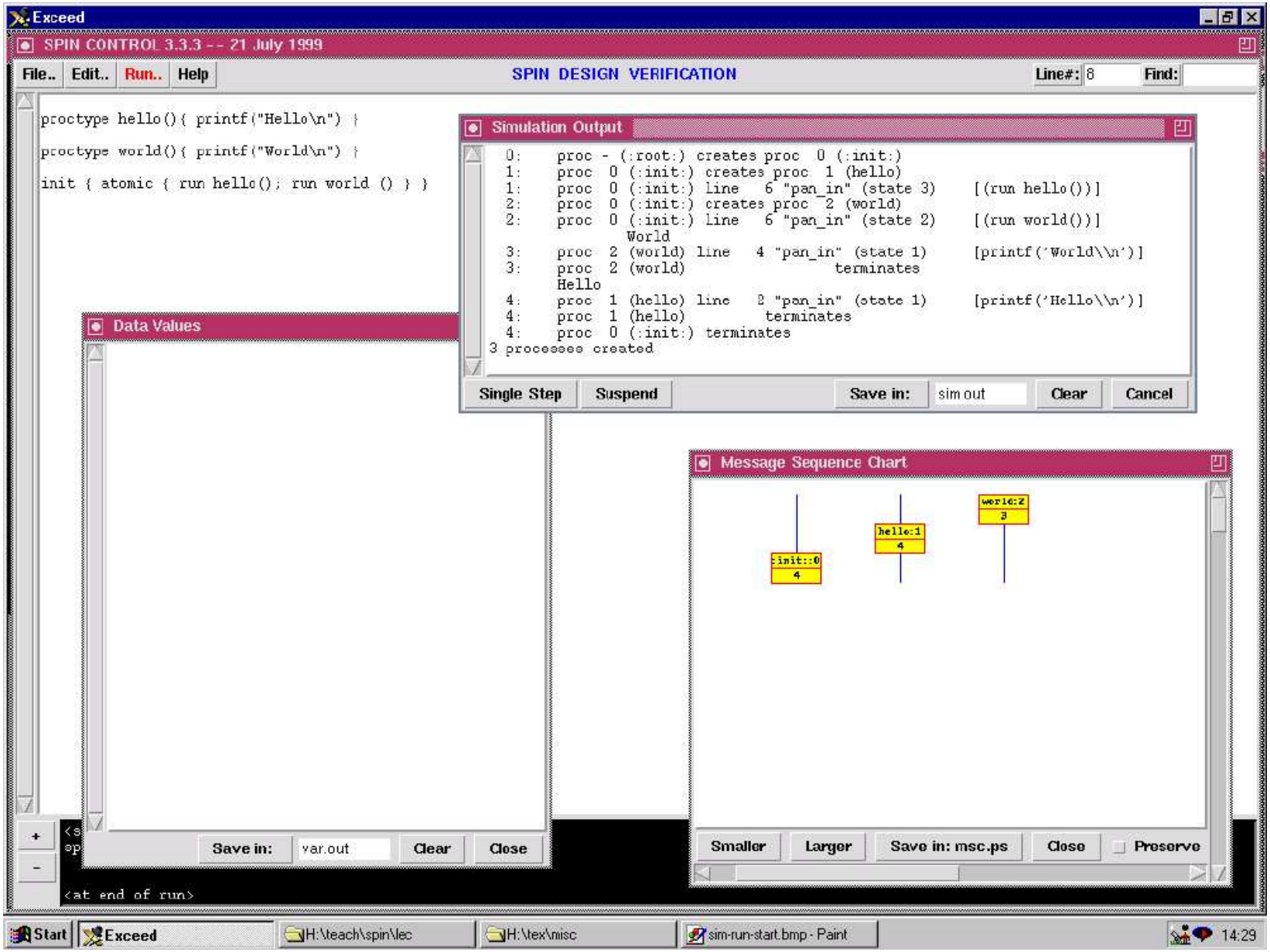
- Random: requires the user to provide a seed, obviously using the same seed will generate the same execution trace.
- Guided: requires a failure trail generated during a previous verification effort to be available, *i.e.* provides a mechanism for viewing a counter example generated by SPIN's verifier.
- Interactive: requires user interaction to resolve non-deterministic choice points within a simulation run.
- XSPIN's simulation options window is pictured on the next slide ...


Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File.. Edit.. Run.. Help
SPIN DESIGN VERIFICATION
Line#: 7 Find:
proctype hello(){ printf("Hello ") }
proctype world(){ printf("World\n") }
init { atomic { run hello(); run world() } }
Simulation Options
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:
Help Cancel Start
Spin Version 3.3.5 -- 28 September 1999
Xspin Version 3.3.3 -- 21 July 1999
TclTk Version 8.0/8.0
<open /u1/staff/air/systems/spin/Examples/hello>
Start Exceed H:\teach\spin\lec H:\tex\misc spin-main.bmp - Paint 10:52

Running Simulations

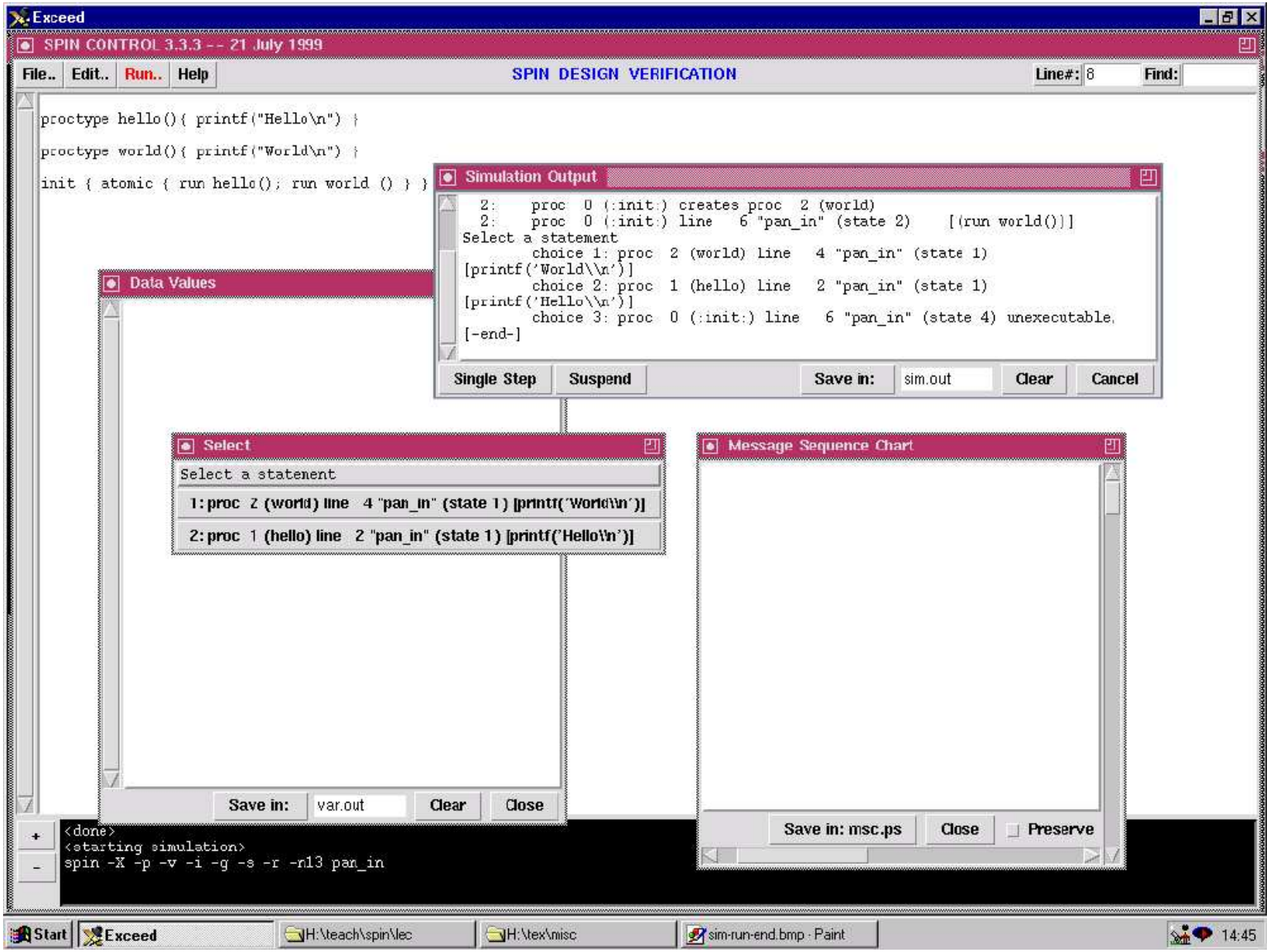
- Simulation parameters must be confirmed at least once before a simulation run can take place.
- The default setting for the simulation parameters will generate the following three windows:
  - Simulation Output: provides both single step and continuous modes, as well as a text output option.
  - Message Sequence Chart: zoom focus and postscript output option provided.
  - Data Values: text output option provided.
- XSPIN snap-shots, before and after a simulation run, provided on the next two slides ...

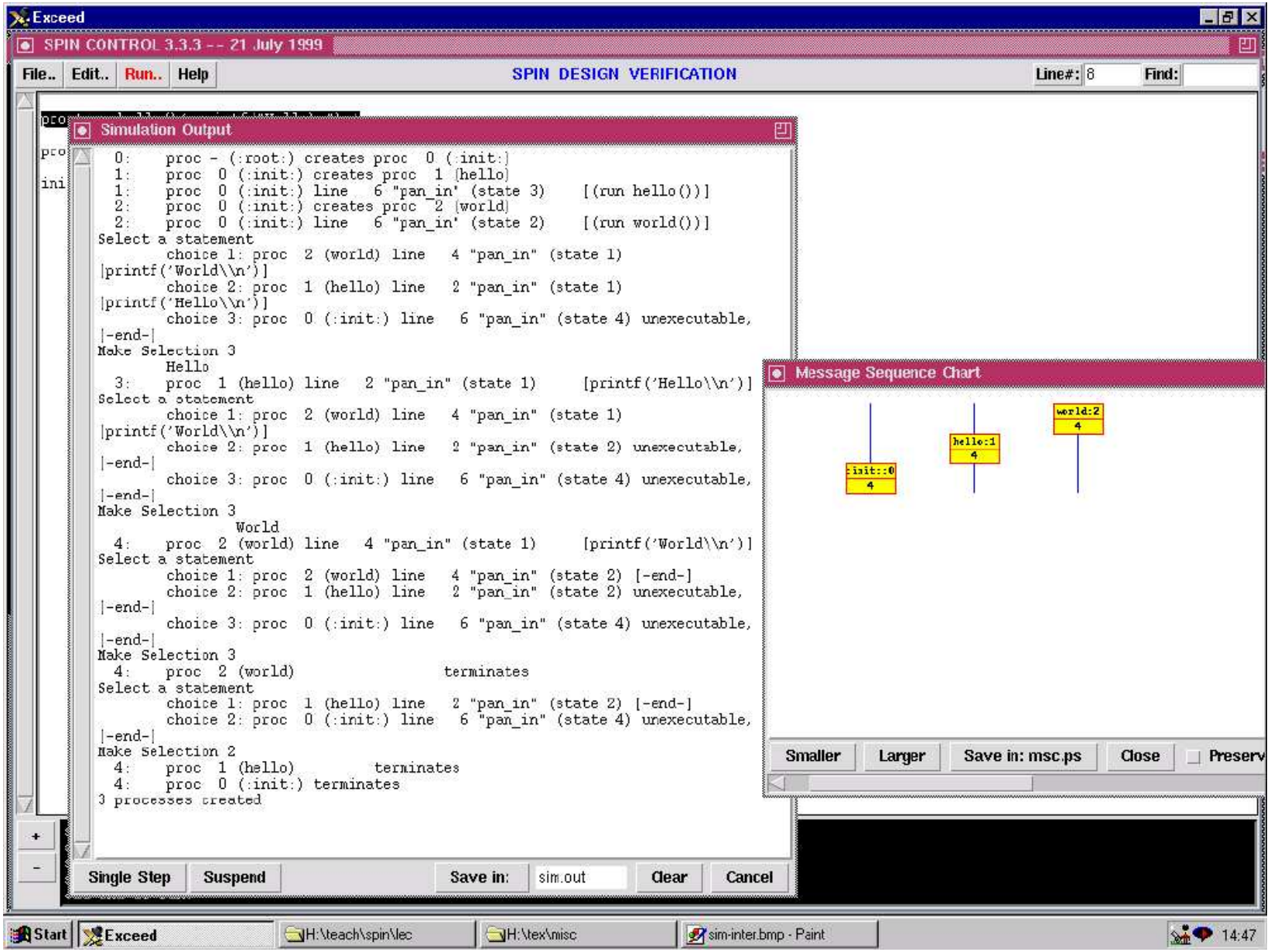

Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File..
Edit..
Run..
Help
SPIN DESIGN VERIFICATION
Line#: 8
Find:
proctype hello(){ p
proctype world(){ p
init { atomic { rur
Simulation Output
0: proc - (:root:) creates proc 0 (:init:)
1: proc 0 (:init:) creates proc 1 (hello)
1: proc 0 (:init:) line 6 "pan_in" (state 3) [(run hello())]
Single Step
Run
Save in:
sim.out
Clear
Cancel
Data Values
Message Sequence Chart
Save in: msc.ps
Close
Preserve
spin -X -p -v -g -s -r -n13 -j0 pan_in
Start
Exceed
H:\teach\spin\lec
H:\tex\misc
sim-run-start.bmp - Paint
14:26


Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File..
Edit..
Run..
Help
SPIN DESIGN VERIFICATION
Line#: 8
Find:
proctype hello(){ printf("Hello\n") }
proctype world(){ printf("World\n") }
init { atomic { run hello(); run world () } }
Simulation Output
0: proc - (:root:) creates proc 0 (:init:)
1: proc 0 (:init:) creates proc 1 (hello)
1: proc 0 (:init:) line 6 "pan_in" (state 3) [(run hello())]
2: proc 0 (:init:) creates proc 2 (world)
2: proc 0 (:init:) line 6 "pan_in" (state 2) [(run world())]
World
3: proc 2 (world) line 4 "pan_in" (state 1) [printf('World\\n')]
3: proc 2 (world) terminates
Hello
4: proc 1 (hello) line 2 "pan_in" (state 1) [printf('Hello\\n')]
4: proc 1 (hello) terminates
4: proc 0 (:init:) terminates
3 processes created
Single Step
Suspend
Save in:
sim.out
Clear
Cancel
Data Values
Save in:
var.out
Clear
Close
Message Sequence Chart
world:2
3
hello:1
4
:init::0
4
Smaller
Larger
Save in: msc.ps
Close
Preserve
<at end of run>
Start
Exceed
H:\teach\spin\lec
H:\tex\misc
sim-run-start.bmp - Paint
14:29

Interactive Simulations

- Interactive simulation is useful when a Promela model contains non-deterministic choice that needs to be debugged.
- During a simulation run, a selection window appears whenever a non-deterministic choice point is reached. The simulation run is interrupted until **you** select which statement should be executed next.
- This selection process is illustrated in the next two slides ...


Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File.. Edit.. Run.. Help
SPIN DESIGN VERIFICATION
Line#: 8 Find:
proctype hello(){ printf("Hello\n") }
proctype world(){ printf("World\n") }
init { atomic { run hello(); run world () } }
Simulation Output
2: proc 0 (:init:) creates proc 2 (world)
2: proc 0 (:init:) line 6 "pan_in" (state 2) [(run world())]
Select a statement
choice 1: proc 2 (world) line 4 "pan_in" (state 1) [printf('World\\n')]
choice 2: proc 1 (hello) line 2 "pan_in" (state 1) [printf('Hello\\n')]
choice 3: proc 0 (:init:) line 6 "pan_in" (state 4) unexecutable.
[-end-]
Single Step Suspend Save in: sim.out Clear Cancel
Data Values
Save in: var.out Clear Close
Select
Select a statement
1: proc 2 (world) line 4 "pan_in" (state 1) [printf('World\\n')]
2: proc 1 (hello) line 2 "pan_in" (state 1) [printf('Hello\n')]
Message Sequence Chart
Save in: msc.ps Close Preserve
<done>
<starting simulation>
spin -X -p -v -i -g -s -r -n13 pan_in
Start Exceed H:\teach\spin\lec H:\tex\misc sim-run-end.bmp - Paint 14:45


Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File.. Edit.. Run.. Help
SPIN DESIGN VERIFICATION
Line#: 8 Find:
Simulation Output
0: proc - (:root:) creates proc 0 (:init:)
1: proc 0 (:init:) creates proc 1 (hello)
1: proc 0 (:init:) line 6 "pan_in" (state 3) [(run hello())]
2: proc 0 (:init:) creates proc 2 (world)
2: proc 0 (:init:) line 6 "pan_in" (state 2) [(run world())]
Select a statement
choice 1: proc 2 (world) line 4 "pan_in" (state 1)
|printf('World\\n')]
choice 2: proc 1 (hello) line 2 "pan_in" (state 1)
|printf('Hello\\n')]
choice 3: proc 0 (:init:) line 6 "pan_in" (state 4) unexecutable,
|-end-|
Make Selection 3
Hello
3: proc 1 (hello) line 2 "pan_in" (state 1) [printf('Hello\\n')]
Select a statement
choice 1: proc 2 (world) line 4 "pan_in" (state 1)
|printf('World\\n')]
choice 2: proc 1 (hello) line 2 "pan_in" (state 2) unexecutable,
|-end-|
choice 3: proc 0 (:init:) line 6 "pan_in" (state 4) unexecutable,
|-end-|
Make Selection 3
World
4: proc 2 (world) line 4 "pan_in" (state 1) [printf('World\\n')]
Select a statement
choice 1: proc 2 (world) line 4 "pan_in" (state 2) [-end-]
choice 2: proc 1 (hello) line 2 "pan_in" (state 2) unexecutable,
|-end-|
choice 3: proc 0 (:init:) line 6 "pan_in" (state 4) unexecutable,
|-end-|
Make Selection 3
4: proc 2 (world) terminates
Select a statement
choice 1: proc 1 (hello) line 2 "pan_in" (state 2) [-end-]
choice 2: proc 0 (:init:) line 6 "pan_in" (state 4) unexecutable,
|-end-|
Make Selection 2
4: proc 1 (hello) terminates
4: proc 0 (:init:) terminates
3 processes created
Single Step Suspend Save in: sim.out Clear Cancel
Message Sequence Chart
:init::0
4
hello:1
4
world:2
4
Smaller Larger Save in: msc.ps Close Preserv
Start Exceed H:\teach\spin\lec H:\tex\misc sim-inter.bmp - Paint 14:47

Assertions

- An **assertion** is a statement which can be either true or false.
- Interleaving assertion evaluation with code execution provides a simple yet very useful mechanism for checking **desirable** as well as **erroneous** behaviour with respect to our models.
- The syntax for an assertion within Promela takes the form:

  ```
  assert( <logical-statement> )
  ```

  for example:

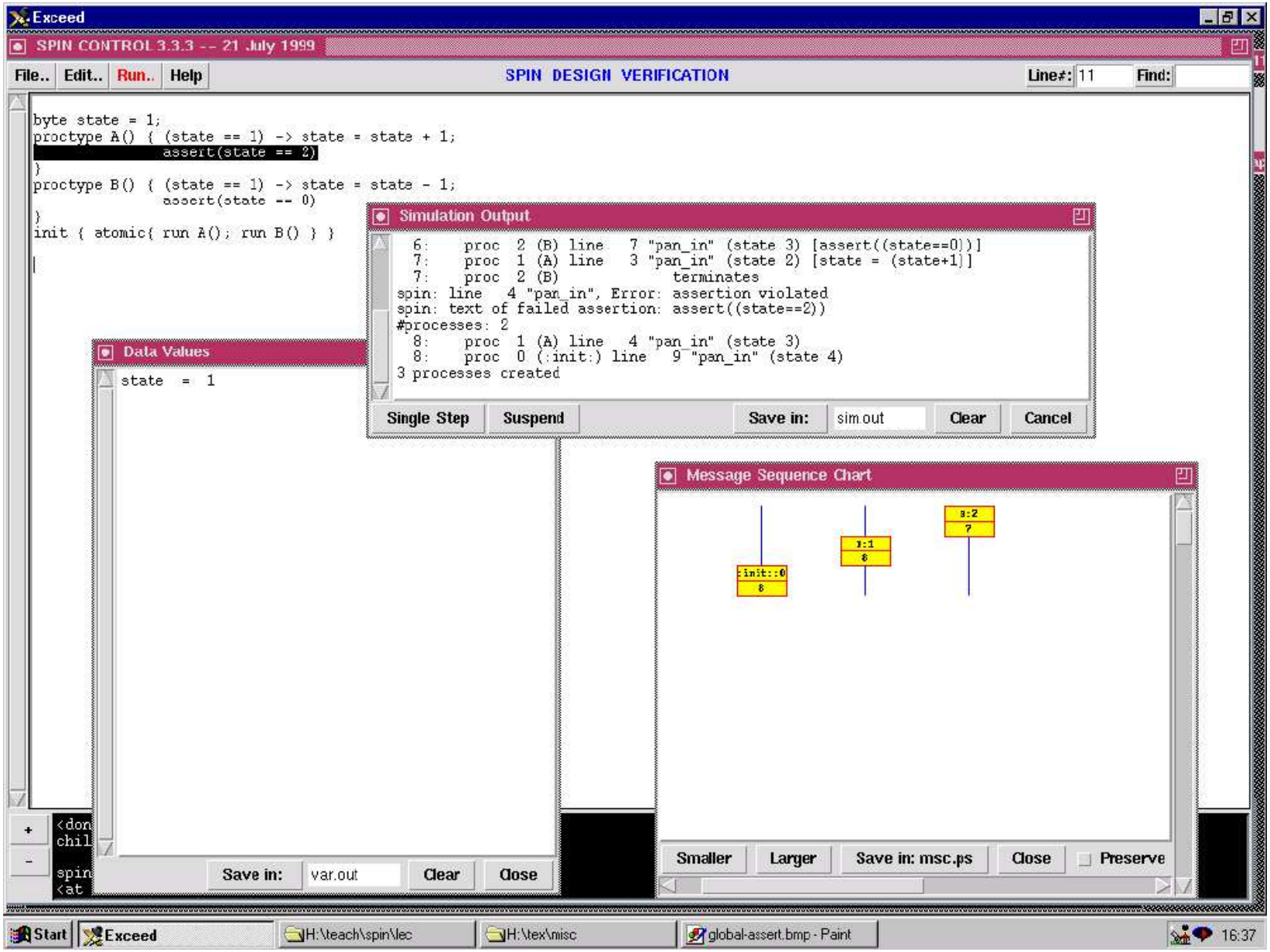
  ```
  assert( !(doors == open && lift == moving) )
  ```

- Within Promela we can express **local** assertions as well **global** system assertions.

Local Assertions

```
byte state = 1;
proctype A() { (state == 1) -> state = state + 1;
               assert(state == 2)
}
proctype B() { (state == 1) -> state = state - 1;
               assert(state == 0)
}
init { atomic{ run A(); run B() } }
```

Will the assertion checking succeed or fail?

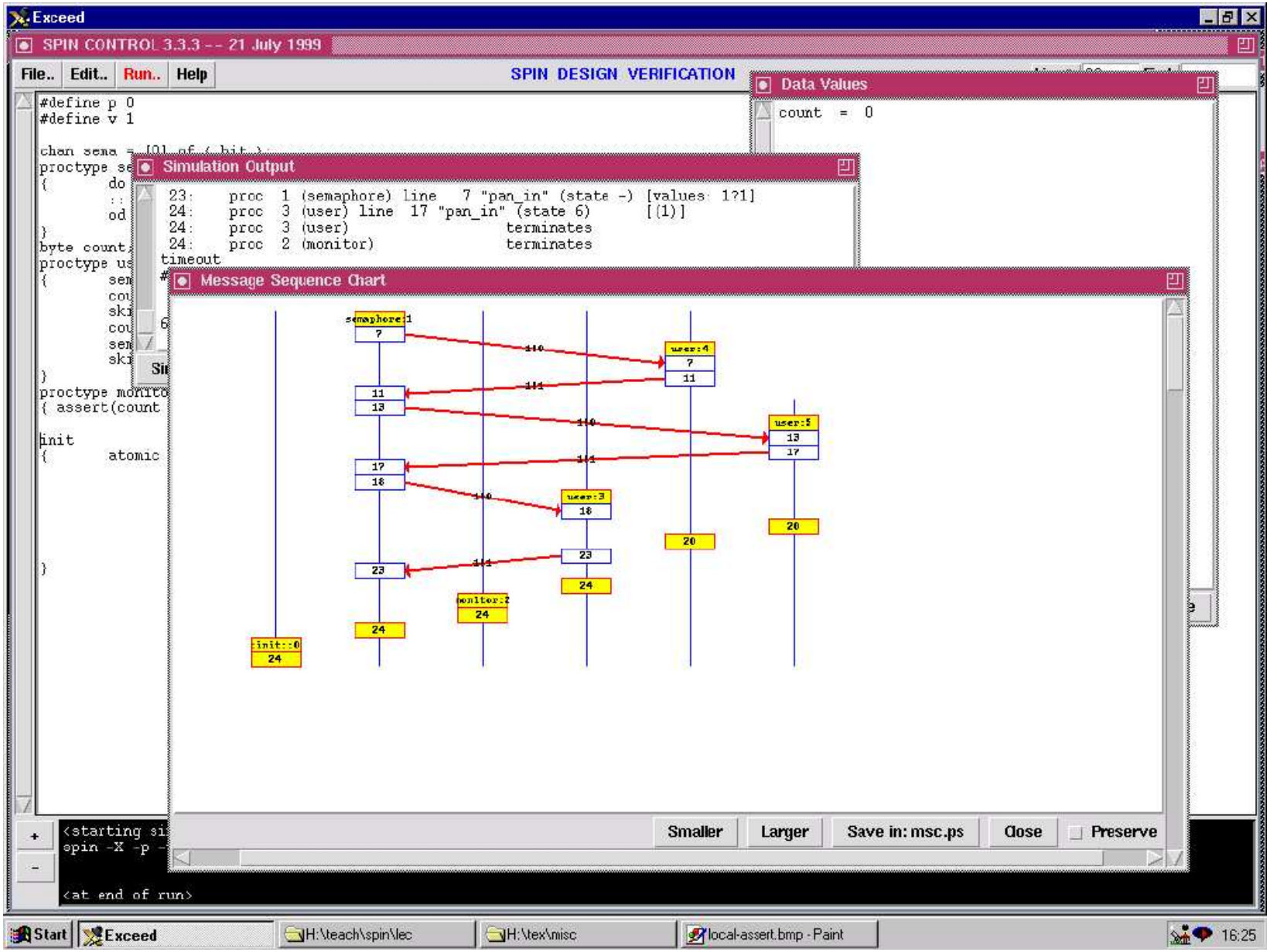

Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File.. Edit.. Run.. Help
SPIN DESIGN VERIFICATION
Line#: 11 Find:
byte state = 1;
proctype A() { (state == 1) -> state = state + 1;
assert(state == 2)
}
proctype B() { (state == 1) -> state = state - 1;
assert(state == 0)
}
init { atomic{ run A(); run B() } }
Simulation Output
6: proc 2 (B) line 7 "pan_in" (state 3) [assert((state==0))]
7: proc 1 (A) line 3 "pan_in" (state 2) [state = (state+1)]
7: proc 2 (B) terminates
spin: line 4 "pan_in", Error: assertion violated
spin: text of failed assertion: assert((state==2))
#processes: 2
8: proc 1 (A) line 4 "pan_in" (state 3)
8: proc 0 (:init:) line 9 "pan_in" (state 4)
3 processes created
Single Step Suspend Save in: sim.out Clear Cancel
Data Values
state = 1
Save in: var.out Clear Close
Message Sequence Chart
Smaller Larger Save in: msc.ps Close Preserve
Start Exceed H:\teach\spin\lec H:\tex\misc global-assert.bmp - Paint 16:37

Global Assertions

- A **global assertion** or **system invariant** is a property that is true in the initial system state and remains true in all possible execution paths.
- To express a system invariant within Promela one must define a monitor process that contains the desired system invariant.
- Running an instance of the monitor process along with the rest of the system model means that the global assertion can be checked at any point during the execution.
- Note that in the case of a simulation the checking is not exhaustive, this is achieved within verification mode.

Semaphores Revisited

```
#define p 0
#define v 1
chan sema = [0] of { bit };
proctype semaphore()
{       do :: sema!p -> sema?v od}
byte count;
proctype user()
{       sema?p;
        count = count + 1;
        skip;   /* critical section */
        count = count - 1;
        sema!v;
        skip    /* non-critical section */ }
proctype monitor(){
        do :: assert(count == 0 || count == 1) od}
init {  atomic {run monitor(); run semaphore();
                run user(); run user(); run user()}}
```


Exceed
SPIN CONTROL 3.3.3 -- 21 July 1999
File.. Edit.. Run.. Help
SPIN DESIGN VERIFICATION
Data Values
count = 0
Simulation Output
23: proc 1 (semaphore) line 7 "pan_in" (state -) [values: 1?1]
24: proc 3 (user) line 17 "pan_in" (state 6) [(1)]
24: proc 3 (user) terminates
24: proc 2 (monitor) terminates
timeout
Message Sequence Chart
semaphore:1
user:4
user:5
user:3
monitor:2
:init::0
Smaller
Larger
Save in: msc.ps
Close
Preserve
<at end of run>
Start
Exceed
H:\teach\spin\lec
H:\tex\misc
local-assert.bmp - Paint
16:25

Observations on MSCs

- Each process is associated with a vertical line within a MSC. The "start of time" corresponds to the top of the MSC, moving down the MSC corresponds to the passing of time.
- The temporal ordering of events, *i.e.* the passing of messages, is represented by the relative ordering of arrows between process execution lines.
- Note that the start of an arrow denotes the relative point in time when a process sends a message to a channel while the arrow head denotes the relative point in time when the message is removed from the channel by a process.
- Note that the vertical distance between the start of an arrow and the arrow head represents the relative time that the associated message was stored in the channel.

Summary

Learning outcomes:

- To be able to use XSPIN to consult and syntax check Promela programs.
- To be able to use the SPIN simulator (via XSPIN) to explore system models, *i.e.* the setting of simulation options and the running of the simulator in its various modes.
- Understand how to construct and use both local and global system assertions with in the context of the SPIN simulator.

Recommended reading:

- "Basic Spin Manual" — see course homepage