CS 6110, Spring 2021, Assignment 1
Given 1/20/21 – Due 1/27/21 by 11:59 pm (and no Quiz this first week)

Submission: Please submit a file `UNID_Lastname_Asg1.pdf` with your solution on Canvas. While discussions are allowed to learn Promela and SPIN and its flags and features, please be original in your actual solution.

In the file name, please start your UNID with a `digit`. **Thus, u1122333 must be entered 01122333.**

I'm not providing you the sources which you may kindly type in; that is educational.

1. (30 pts) Run the program in Book 1, Exercises 21.4 (Page 396) on "Bubble sorting," and see how the bug is discovered (the "sortedness assertion" fails). Fully explain all uses of nondeterminism in this Promela model. How does it help attain full state-space coverage of the finite-state model?

2. (30 pts) Change this Bubblesort to a working version; verify that it works now, removing its present bug (plus others you might notice). Enter it as a Promela program, run it under SPIN, and show that the "sortedness assertion" now passes.

3. (40 pts) Change the example in 21.2 as follows:

   (a) (20 pts) Instead of asking the right fork "are you free" and then taking an "ack" or a "nack," try to grab the left fork atomically (via rendezvous) and then the right fork. A piece of plausible Promela is given below; finish its missing parts. Type this in, and verify that it deadlocks. Steps to run SPIN in the shell are below. Interactive runs will be demo'ed in class.

   (b) (30 pts – 10 pts each for deadlock, communal progress, indiv progress)
   A way to avoid deadlock is to have one philosopher grab the right first, then the left (all others grab left first then right). Make this change. Does this prevent deadlocks? Does it allow communal progress (at any point, some philosopher is able to eventually eat). Does it allow individual progress by each philosopher? (For this, capitalize on the problem symmetry and check one member of a symmetric set of processes.) Note: After checking for deadlocks, put suitable **never** automaton checks for these "progress" assertions and verify. (To keep solutions uniform, remove **progress** labels and rely on **never** automata.)

```
// Each phil's algorithm is below
proctype phil(chan lfp, lfv, rfp, rfv)
 { do
   :: obtain left semaphore -> obtain right semaphore
      -> printf("Eating") ->
      release left semaphore -> release right semaphore
   od
 }

// This is a mutex with the P and V operations
// https://en.wikipedia.org/wiki/Semaphore_(programming)
//
proctype fork(chan p, v)
```

```
  { do
    :: p?0     // Fork taken!
       -> v?0  // Fork returned!
    od
  }

init {
  chan p0 = [0] of { bit };
  chan v0 = [0] of { bit };

  chan p1 = [0] of { bit };
  chan v1 = [0] of { bit };

  chan p2 = [0] of { bit };
  chan v2 = [0] of { bit };

  atomic {
  // Connect and run Phil P0 to grab
  // fork 0 on left, then
  // fork 2 on right

  // Connect and run Phil P1 to grab
  // fork 1 on left
  // fork 0 on right

  // Connect and run Phil P2 to grab
  // fork 2 on left
  // fork 1 on right
  }
}
```

Diagramming, and Running SPIN on a terminal

You must attempt to draw message-sequence charts on a notepad or ipad to debug effectively. Please submit these with the assignment. The URL `http://spinroot.com/spin/Man/Spin.html` gives you info on SPIN's flags. There is more documentation on the **spinroot** page.

```
spin -a atomicphil.pml
gcc -DMEMLIM=1024 -O2 -DXUSAFE -DSAFETY -DNOCLAIM -w -o pan pan.c
./pan -m10000
Pid: 66796

GET pan HELP AS FOLLOWS:

[ganesh@thinmac Examples]$ ./pan --help
saw option --
Spin Version 6.4.5 -- 1 January 2016
Valid Options are:
-a,-l,-f  -> are disabled by -DSAFETY
-A  ignore assert() violations
-b  consider it an error to exceed the depth-limit
-cN stop at Nth error (defaults to -c1)
-D  print state tables in dot-format and stop
-d  print state tables and stop
-e  create trails for all errors
-E  ignore invalid end states
-hN use different hash-seed N:0..499 (defaults to -h0)
-hash generate a random hash-polynomial for -h0 (see also -rhash)
      using a seed set with -RSn (default 12345)
-i  search for shortest path to error
-I  like -i, but approximate and faster
-J  reverse eval order of nested unlesses
-mN max depth N steps (default=10k)
-n  no listing of unreached states
-QN set time-limit on execution of N minutes
-q  require empty chans in valid end states
-r  read and execute trail - can add -v,-n,-PN,-g,-C
-r trailfilename  read and execute trail in file
-rN read and execute N-th error trail
-C  read and execute trail - columnated output (can add -v,-n)
-r -PN read and execute trail - restrict trail output to proc N
-g  read and execute trail + msc gui support
-S  silent replay: only user defined printfs show
-RSn use randomization seed n
-rhash use random hash-polynomial and randomly choose -p_rotateN, -p_permute, or p_reverse
-T  create trail files in read-only mode
-t_reverse  reverse order in which transitions are explored
-tsuf replace .trail with .suf on trailfiles
-V  print SPIN version number
-v  verbose -- filenames in unreached state listing
-wN hashtable of 2^N entries (defaults to -w24)
-x  do not overwrite an existing trail file

options -r, -C, -PN, -g, and -S can optionally be followed by
a filename argument, as in '-r filename', naming the trailfile
        [ganesh@thinmac Examples]$


== One tries to catch bugs at the most shallow depth ==
== SPIN also has a BFS mode and also a depth minimization mode ==

[ganesh@thinmac Examples]$ ./pan -m10000  <== search depth

=== LOOK AT HOW ERRORS ARE SUMMARIZED AND REPORTED ===
=== You MUST read these bugs and warnings carefully! ===
=== Also read the statistics carefully! ===
```

```
pan:1: invalid end state (at depth 20)
pan: wrote atomicphil.pml.trail

(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
+ Partial Order Reduction

Full statespace search for:
never claim          - (not selected)
assertion violations +
cycle checks        - (disabled by -DSAFETY)
invalid end states +

State-vector 108 byte, depth reached 23, errors: 1
       12 states, stored
        2 states, matched
       14 transitions (= stored+matched)
        5 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.002 equivalent memory usage for states (stored*(State-vector + overhead))
    0.291 actual memory usage for states
  128.000 memory used for hash table (-w24)
    0.534 memory used for DFS stack (-m10000)
  128.730 total actual memory usage

== THIS IS ERROR-TRAIL SIMULATON BELOW - understand all these flags! ==

[ganesh@thinmac Examples]$ spin -p -r -s -c  -t atomicphil.pml
proc 0 = :init:
using statement merging
Starting phil with pid 1
proc 1 = phil
  1: proc  0 (:init::1) atomicphil.pml:28 (state 1) [(run phil(p0,v0,p2,v2))]
Starting phil with pid 2
proc 2 = phil
  2: proc  0 (:init::1) atomicphil.pml:32 (state 2) [(run phil(p1,v1,p0,v0))]
Starting phil with pid 3
proc 3 = phil
  3: proc  0 (:init::1) atomicphil.pml:36 (state 3) [(run phil(p2,v2,p1,v1))]
Starting fork with pid 4
proc 4 = fork
  4: proc  0 (:init::1) atomicphil.pml:39 (state 4) [(run fork(p0,v0))]
Starting fork with pid 5
proc 5 = fork
  5: proc  0 (:init::1) atomicphil.pml:41 (state 5) [(run fork(p1,v1))]
Starting fork with pid 6
proc 6 = fork
  6: proc  0 (:init::1) atomicphil.pml:43 (state 6) [(run fork(p2,v2))]
q\p   0   1   2   3   4   5   6
  5   .   .   .   lfp!0
  7: proc  3 (phil:1) atomicphil.pml:4 (state 1) [lfp!0]
  5   .   .   .   .   .   .   p?0
  8: proc  6 (fork:1) atomicphil.pml:10 (state 1) [p?0]
  3   .   .   .   rfp!0
  9: proc  3 (phil:1) atomicphil.pml:4 (state 2) [rfp!0]
  3   .   .   .   .   .   p?0
 10: proc  5 (fork:1) atomicphil.pml:10 (state 1) [p?0]
                 Eating 11: proc  3 (phil:1) atomicphil.pml:4 (state 3) [printf('Eating')]
  6   .   .   .   lfv!0
 12: proc  3 (phil:1) atomicphil.pml:4 (state 4) [lfv!0]
  6   .   .   .   .   .   .   v?0
 13: proc  6 (fork:1) atomicphil.pml:11 (state 2) [v?0]
  1   .   lfp!0
 14: proc  1 (phil:1) atomicphil.pml:4 (state 1) [lfp!0]
  1   .   .   .   .   p?0
```

```
 15: proc  4 (fork:1) atomicphil.pml:10 (state 1) [p?0]
  4   .   .   .   rfv!0
 16: proc  3 (phil:1) atomicphil.pml:4 (state 5) [rfv!0]
  4   .   .   .   .   .   v?0
 17: proc  5 (fork:1) atomicphil.pml:11 (state 2) [v?0]
  5   .   .   .   lfp!0
 18: proc  3 (phil:1) atomicphil.pml:4 (state 1) [lfp!0]
  5   .   .   .   .   .   .   p?0
 19: proc  6 (fork:1) atomicphil.pml:10 (state 1) [p?0]
  3   .   .   lfp!0
 20: proc  2 (phil:1) atomicphil.pml:4 (state 1) [lfp!0]
  3   .   .   .   .   .   p?0
 21: proc  5 (fork:1) atomicphil.pml:10 (state 1) [p?0]
spin: trail ends after 21 steps
-------------
final state:
-------------
#processes: 7
 21: proc  6 (fork:1) atomicphil.pml:11 (state 2)
 21: proc  5 (fork:1) atomicphil.pml:11 (state 2)
 21: proc  4 (fork:1) atomicphil.pml:11 (state 2)
 21: proc  3 (phil:1) atomicphil.pml:4 (state 2)
 21: proc  2 (phil:1) atomicphil.pml:4 (state 2)
 21: proc  1 (phil:1) atomicphil.pml:4 (state 2)
 21: proc  0 (:init::1) atomicphil.pml:46 (state 8) <valid end state>
7 processes created

=== NOW FIND OUT WHY THE ABOVE IS DESCRIBING A DEADLOCK ===
=== Drawing a diagram will help                         ===
```