CS 6110, Spring 2022, Assignment 3
Given 2/3/22 – Due 2/10/22 by 11:59 pm via your Github

NAME: **UNID:**

CHANGES: Please look for lines beginning with underlined words when they are made. none yet.

Answering, Submission: Have these on your private Github: a folder Asg3/ containing your submission, which in detail comprises:

- A clear README.md describing your files.
- Files that you ran + documentation (can be integrated in one place).
- A high level summary of your cool findings + insights + learning – briefly reported in a nicely bulletted fashion in your PDF submission.

Start Early, Ask Often! Orientation videos and further help will be available (drop a note anytime on Piazza for help). *I encourage students constructing answers jointly!*

1. (8 points) This is on practicing the use of Murphi. At the end of this assignment, I include the N-Process Peterson protocol for mutual exclusion, described in `https://en.wikipedia.org/wiki/Peterson%27s_algorithm`. (This comes from the Murphi distribution; pasting it here for your convenience.) Your task is to set-up Murphi or Rumur, run this protocol for 3 processes, and see if you can get an estimate of the state-space of the protocol. I recommend that Murphi be obtained from `http://mclab.di.uniroma1.it/site/index.php/software/18-cmurphi` (latest version) or obtain Rumur from `https://github.com/Smattr/rumur`. Your luck in building these may vary—let's have a dialog next week on this. We will offer help if you get stuck, so please raise it on Piazza. Once you understand the coding of Murphi, kindly develop a bulletted summary of its constructs. I'll be adding more Murphi documentation in the interim. Here are some tips:

   - looks like we just need to run with the default -v instead of -s which seems to be the difference between verification and continuous simulation. [Correct. also -vdfs is for DFS verification.]
   - Please try for N=3 and N=5 (constants at the top). I don't mind if > 5 is not tried (although it ought to finish).
   - Please also try with symmetry reduction (the -sym flag ; it also takes a number like 1 or 2). With that, it will take more time per printout (per batch of rules) but will be very memory-efficient.
   - If it says "out of memory", you can give e.g. another flag "-m32" and then "-m64" etc. That is the megabytes allocated.
   - Finally, the printouts take time away from the model-checker. Making its printing rate go down can make it somewhat faster. (Already the printing rate is not too high, but every bit helps.)
   - At the bottom of the file there is a recording of former runs they finished using this tool (way back in 1998 or so). That is the kind of result I'm looking for.

Your Answer Here

2. (2 points) Read about the Photoshop bug (a light read) from the paper photoshop-bug.pdf in the repo. Summarize your thought in a few lines. This is to tell you how real-world bugs often fester, and interface-differences may trip up people.

3. (90 points) The main part of this assignment is to debug a locking protocol given in Promela. The protocol written by an expert (my former PhD student) is included with comments in locking-prot.tex and locking-prot.pdf. Set-up:

   (a) This protocol, as implemented, has a bug

   (b) Like almost all bugs, this occurs exactly on one line

   (c) Like almost all bugs, the fix is also a small change on the line

   Your task:

   (a) Discover this bug by a "`spin -a`" run

   (b) You can discover the bug in a depth of 90 or less by running "`pan -a -m90`"

   (c) You can simply look at the last state printed by "`spin -s -r -t locking-buggy.pml`" and spot the bug

   (d) Fix the bug and rerun. *The fix consists of making one of the guard conditions stronger.*

   (e) Can a node decide to toss requests to a random node rather than along the PO chain? (It might do this to avoid network congestion.) Justify this, and implement + verify in the fixed version of your protocol (make some nodes nondeterministically toss the request to a random PO than its actual PO).

   (f) Explain your insights!

   Your
   Answer
   Here

```
------------------------------------------------------------------------
-- Copyright (C) 1992 by the Board of Trustees of
-- Leland Stanford Junior University.
--
-- This description is provided to serve as an example of the use
-- of the Murphi description language and verifier, and as a benchmark
-- example for other verification efforts.
--
-- License to use, copy, modify, sell and/or distribute this description
-- and its documentation any purpose is hereby granted without royalty,
-- subject to the following terms and conditions, provided
--
-- 1.  The above copyright notice and this permission notice must
-- appear in all copies of this description.
--
-- 2.  The Murphi group at Stanford University must be acknowledged
-- in any publication describing work that makes use of this example.
--
-- Nobody vouches for the accuracy or usefulness of this description
-- for any purpose.
------------------------------------------------------------------------


------------------------------------------------------------------------
--
--
-- File:        muxn.m
--
-- Content:     Peterson's algorithm (mutual exclusion for n-processes)
--
-- Summary of result:
```

```
--          1)  No bug is discovered
--          2)  Details of result can be found at the end of this file.
--
-- References:
-- Peterson, G.L.,  Myths about the mutual exclusion problem,
-- Information processing letters, Vol 12, No 3, 1981.
--
-- Date created:         28 Oct 92
-- Last Modified:        17 Feb 93
--
--------------------------------------------------------------------------

Const
  N: 7;

Type
  --  The scalarset is used for symmetry, which is implemented in Murphi 1.5
  --  and not upgraded to Murphi 2.0 yet
  pid: scalarset (N);
  -- pid: 1..N;
  priority: 0..N;
  label_t: Enum{L0, -- : non critical section; j := 1; while j<n do
L1,  -- : Beginwhile Q[i] := j
L2,  -- : turn[j] := i
L3,  -- : wait until (forall k != i, Q[k] < j) or turn[j] != i ; j++; Endwhile
L4   -- : critical section; Q[i] := 0
  };
Var
  P: Array [ pid ] Of label_t;
  Q: Array [ pid ] Of priority;
  turn: Array [ priority ] Of pid;
  localj: Array [ pid ] Of priority;

Ruleset i: pid  Do

  Rule "execute inc j and while"
    P[i] = L0  ==>
  Begin
    localj[i] := 1;
    P[i] := L1;
  End;

  Rule "execute assign Qi j"
    P[i] = L1  ==>
  Begin
    Q[i] := localj[i];
    P[i] := L2;
  End;

  Rule "execute assign TURNj i"
    P[i] = L2  ==>
  Begin
    turn[localj[i]]  := i;
    P[i] := L3;
  End;

  Rule "execute wait until"
    P[i] = L3  ==>
  Begin
    If ( Forall k: pid Do
           ( k!=i ) -> ( Q[k]<localj[i] )
         End --forall
       | ( turn[localj[i]] != i ) )
    Then
      localj[i] := localj[i] + 1;
      If ( localj[i]<N )
      Then
        P[i] := L1;
      Else
        P[i] := L4;
      End; --If
    End; --If
  End;

  Rule "execute critical and assign Qi 0"
    P[i] = L4  ==>
  Begin
    Q[i] := 1;
    P[i] := L0;
  End;

End; --Ruleset

Startstate
Begin
  For i:pid Do
    P[i] := L0;
    Q[i] := 0;
  End; --For

  For i: priority Do
    Undefine turn[i];
  End; --For

  Clear localj;
End;

Invariant
  ! Exists i1: pid Do
    Exists i2: pid Do
      ( i1 != i2
      & P[i1] = L4 -- critical
      & P[i2] = L4 -- critical
      )
    End  --Exists
    End; --Exists

/******************

Summary of Result (using release 2.3):

1) 3 processes

   breath-first search
   29 bits (4 bytes) per state
   771 states with a max of about 54 states in queue
   2313 rules fired
   0.73s in sun sparc 2 station

2) 5 processes

   breath-first search
   63 bits (8 bytes) per state
   576,551 states with a max of about 22,000 states in queue
   2,882,755 rules fired
   1201.66s in sun sparc 2 station

2.73S

* 3 processes (sparc  2 station)
* The size of each state is 35 bits (rounded up to 5 bytes).

BFS -nosym
882 states, 2646 rules fired in 0.73s.

BFS -sym1
172 states, 516 rules fired in 0.36s.

* 5 processes (sparc 2 station)
* The size of each state is 63 bits (rounded up to 8 bytes).

BFS -sym1
6770 states, 33850 rules fired in 22.55s.
249 states max in the queue.

BFS -nosym
628868 states, 3144340 rules fired in 758.92s.
25458 states max in the queue.

gamma2.9S on theforce.stanford.edu

   5 proc
-O4 compile 119.7s 2.7Mbytes
    (24 bytes per states)
-sym2,3,4  6770 states, 33850 rules 14.35s

6 proc
-O4 compile 120.2s 2.7Mbytes
    (28 bytes per states)
        -sym2,3,4  35,159 states, 210954 rules 117.45s

Release 2.9S (Sparc 20, cabbage.stanford.edu)

   7 processes
         * The size of each state is 232 bits (rounded up to 32 bytes).
      -b * The size of each state is 101 bits (rounded up to 16 bytes).

         163298 states, 1143086 rules fired in 341.93s.
      -b 163298 states, 1143086 rules fired in 378.04s.
      -c 163298 states, 1143086 rules fired in 292.42s.

******************/
```