CS 6110, Spring 2022, Assignment 2
Given 1/25/22 – Due 2/1/22 by 11:59 pm via your Github

**NAME:** **UNID:**

**CHANGES: Please look for lines beginning with underlined words.**
**Answering, Submission:** Please provide a PDF with these questions answered in the spaces indicated (if you don't retain the frameboxes, at least please begin each answer on a new page with the question numbers/parts indicated). Also have your work ready on your Github for me to pull/test. Note that Asg2 will be gone over in class and even partially worked out; you'll be finishing the unfinished parts and submitting the full solution. Orientation videos and further help will be available (drop a note anytime on Piazza for help)—**start early**.

1. (20 points) Read about LTL from CEATL's Chapter 22. (You may also look at Ben-Ari's Chapter 5.) By way of practice, in the directory `Lec5` within the class Git `https://github.com/ganeshutah/cs6110s22.git`, you have been given four Promela files `p1.pml` through `p4.pml`. Run these for your own understanding and repeat the results at the end of these files (if any). In particular, in `p4.pml`, you are checking whether *Justice* implies *Compassion*. These terms are defined in the paper "All you need is Compassion" `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.3239` (more properly `https://dl.acm.org/doi/10.5555/1787526.1787547`). Does Justice imply Compassion? Does Compassion imply Justice? Based on your observations (please provide terminal sessions showing the error trace(s) that substantiate your conclusions), do you agree that Pnueli's paper's title is true? Also answer these:

   (a) A solicitor walks around the neighborhood knocking on one door, then the other, taking a "round-robin" walk, and the home-owners are found to be infinitely-often opening and closing their doors in response. Are the home-owners being just of compassionate? Why? (justice is also known as *weak fairness*, while compassionate is also known as *strong fairness*).

   (b) A solicitor walks to one door, leans on the bell and rings the bell continuously (till it almost melts). The home-owner infinitely-often opens the door. Now are the home-owners being just of compassionate? Why?

   (c) Repeat the above answers assuming that the solicitor is a network packet and home-owner doors are ports of a switch.

   **This is an invitation** for you to think about such a router—or if you have seen a more realistic situation like this, plz write in your answer.

   Observations from Pnueli's Paper :-

   (a) Weak Fairness (Justice )
      i. It is associated with a assertion (First Order state formula) J
      ii. Computation $\sigma$ is just with respect to requirement $J$, if transition $\sigma$ contains infinitely many occurrences of states that satisfy J.
      iii. If the transition is continuously enabled, it must be taken infinitely many times. Equivalently there must be infinitely many states at which $\sigma$ is disabled or taken. Asserting that every concurrent process will be eventually progress

iv. Justice: $(<> []p- > [] <> q)$ eventually-henceforth $p$ then infinitely-often $q$

(b) Strong fairness (Compassion)

i. It is associated with a pair of assertions $< p, q >$

ii. Computation $\sigma$ is compassionate with respect to the requirement $< p, q >$ if either $\sigma$ contains only finitely many p-positions or $\sigma$ contains infinitely many q-positions .

iii. If the transition is infinitely often enabled, it must be taken infinitely many times. Asserting that competition on a shared resource is arbitrated in a fair way.

iv. Compassion: $([] <> p- > [] <> q)$ infinitely-often p then infinitely-often $q$

Therefore, Justice doesn't imply Compassion but Compassion imply Justice.Below is the trace for the P4.pml source code.

```
ajantha@knight:~/Desktop/repos/cs6110s22/Lec5$ ./a.out -a
pan:1: acceptance cycle (at depth 14)
pan: wrote p4.pml.trail
(Spin Version 6.4.9 -- 17 December 2018)
Warning: Search not completed
+ Partial Order Reduction
Full statespace search for:
never claim          + (ltl_0)
assertion violations + (if within scope of claim)
acceptance   cycles  + (fairness disabled)
invalid end states - (disabled by never claim)
State-vector 28 byte, depth reached 25, errors: 1
       24 states, stored
       45 states, matched
       69 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)
Stats on memory usage (in Megabytes):
    0.001 equivalent memory usage for states (stored*(State-vector + overhead))
    0.272 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

pan: elapsed time 0 seconds
```

(a) In this case it is compassion because, the home-owners are infinitely often opening and closing the door. $(<> []s- > [] <> h)$ eventually-henceforth s then infinitely-often $h$

(b) Here, it is Justice as the solicitor rings the bell continuously and the home-owners are infinitely often opens the door. $([] <> s- > [] <> h)$ infinitely-often s then infinitely-often h

(c) Assuming the solicitor as Network packet and home -owners door as switch,for the above cases i) Switch is accepting the packets from the Network and ideally there wont be either any congestion or packet loss. ii) Packets are being sent to the switch continuously which might cause congestion or packet loss.

2. (20 points) Now turn to Page 419 of CEATL and look at questions 22.3 and 22.5. In 22.5, a Promela model and `never` automata that distinguish 22.3(a) and 22.3(b) are given.

   (a) Remove the `never` and replace it with an equivalent LTL. For instance, in the code we have `never !(foo)` and here, the LTL being checked is `foo`. Make sure that my claimed checks work (the first formula is true of `sb` but not `sa`, while the second formula is true of both structures; notice that each diagram that warrants being a Kripke structure has been given the name `sa` through `sh`).

   (b) In the earlier question, a method to check the validity of LTL formulas using the most general Kripke Structure was introduced. Using that method, answer these questions, with evidence furnished.

   **We will call this "LTL of sa":**

   ```
   !(<>([](a && b))) -> ((<>([]a)) || (<> (!a && !b)))
   ```

   This can be read as "[Not eventually-henceforth (a and b)] IMPLIES [either (eventually-henceforth a) or eventually (!a and !b)]"

   We call this "LTL of sa" because on Page 420, we have this (in a never automation, the LTL is given one more negation):

   ```
   /* sb satisfies this, and not sa */
   /*Type 'spin -f "formula"' and cut&paste the resulting never aut. */
   /*-------------------------------------------------------------------*/
   never {/* !( !(<>([](a && b))) -> ((<>([]a)) || (<> (!a && !b)))) */
   ```

   **We will call this "LTL of sb":**

   ```
   !(<>([](a && b))) -> ( (<>([]a)) || (<> (b)) )
   ```

   This can be read as "[Not eventually-henceforth (a and b)] IMPLIES [either (eventually-henceforth a) or eventually b]"

   This is because on Page 421, we have this (in a never automation, the LTL is given one more negation):

   ```
   /*--- in contrast, both sa and sb satisfy this --->
    never {  /* !( !(<>([](a && b))) -> ( (<>([]a)) || (<> (b)) ) ) * /}
    <---*/
   ```

   i. Does the LTL of `sa` implies that of `sb`? Provide reasons after observing the trace from your experiment.

      **This is a validity-checking** situation, so you must invoke the most general Kripke-structure here.

   ii. Does the LTL of `sb` implies that of `sa`? Provide reasons after observing the trace from your experiment.

      **This is a validity-checking** situation, so you must invoke the most general Kripke-structure here.

(a) For 22.3 (a) and (b) ,LTL formula could be deduced from the Kirpke Structure and the Promela code with Never Automata distinguishing both are provided.As per question 2,

- Sa = !(<>([](a  b))) -> ((<>([]a)) || (<> (!a  !b)))

- Sb = !(<>([](a  b))) -> ( (<>([]a)) || (<> (b)) )

From the code provided in pg 415 of CEATL book, replaced Never Automata with LTL for Sa and Sb structure respectively.

i) Sa does not imply SbDuring validity checking situation, it satisfied the LTL formula ( !(<>([](a  b))) -> ((<>([]a)) || (<> (!a  !b)))) But received error for LTL

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./a.out
warning: never claim + accept labels requires -a flag to fully verify

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     - (not selected)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 15, errors: 0
       24 states, stored
       19 states, matched
       43 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001       equivalent memory usage for states (stored*(State-vector + overhead))
    0.284       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


unreached in proctype generic
        ks_sa.pml:21, state 53, "D_STEP21"
        ks_sa.pml:22, state 61, "D_STEP22"
        ks_sa.pml:23, state 69, "D_STEP23"
        ks_sa.pml:24, state 77, "D_STEP24"
        ks_sa.pml:25, state 85, "D_STEP25"
        ks_sa.pml:26, state 93, "D_STEP26"
        ks_sa.pml:27, state 101, "D_STEP27"
        ks_sa.pml:21, state 102, "D_STEP21"
        ks_sa.pml:21, state 102, "D_STEP22"
        ks_sa.pml:21, state 102, "D_STEP23"
        ks_sa.pml:21, state 102, "D_STEP24"
        ks_sa.pml:21, state 102, "D_STEP25"
        ks_sa.pml:21, state 102, "D_STEP26"
        ks_sa.pml:21, state 102, "D_STEP27"
        ks_sa.pml:32, state 107, "-end-"
        (9 of 107 states)
unreached in init
        (0 of 2 states)
unreached in claim ltl_0
        _spin_nvr.tmp:18, state 24, "-end-"
        (1 of 24 states)

pan: elapsed time 0 seconds
```

formula ( !(<>([](a  b))) -> ( (<>([]a)) || (<> (b)) ) )
ii) Sb -> Sa During validity checking situation, it satisfied both the LTL formula ( !(<>([](a  b))) -> ((<>([]a)) || (<> (!a  !b)))) and ( !(<>([](a  b))) -> ( (<>([]a)) || (<> (b)) ) )

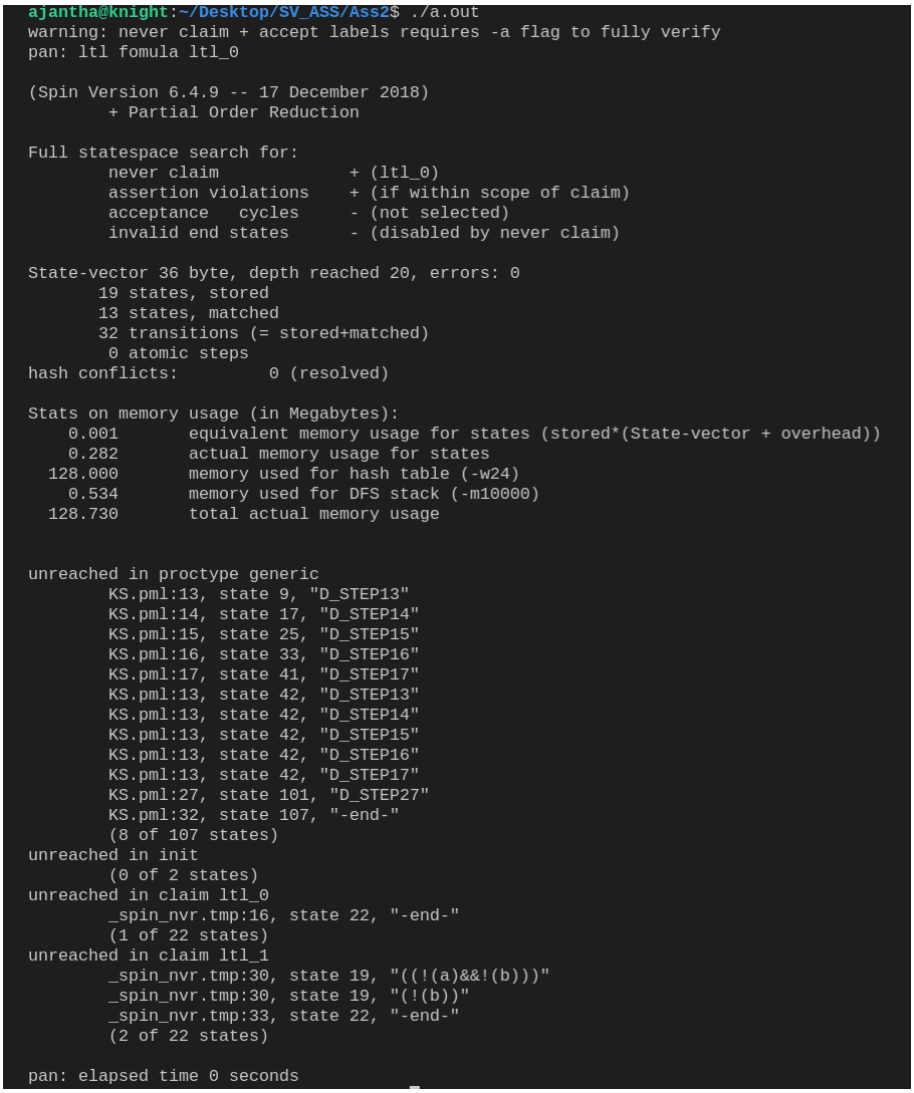
```
33  init
34  { run generic(sa) }
35  //ltl {!( !(<>([](a && b))) -> ((<>([]a)) || (<> (!a && !b))))} //Sa -> Sb
36  ltl { !( !(<>([](a && b))) -> ( (<>([]a)) || (<> (b)) ) )} //sb->Sa
37
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./a.out
warning: never claim + accept labels requires -a flag to fully verify
pan:1: assertion violated  !(b) (at depth 0)
pan: wrote ks_sa.pml.trail

(Spin Version 6.4.9 -- 17 December 2018)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     - (not selected)
        invalid end states      - (disabled by never claim)

State-vector 28 byte, depth reached 0, errors: 1
        1 states, stored
        0 states, matched
        1 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.285       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



pan: elapsed time 0 seconds
```

3. (20 points) Fix the broken Bubble-sort, either through a mild repair or a wholesale rewrite. Verify (upto the limits of finite-state model checking) that it works. Submit evidence that you ran exhaustively (possible for this simple a model). For fun, increase the array size from its current value in steps of 2 (or 1) and do a plot of the number of states generated, revisited, and matched. This tells you how many states are typically generated and how these grow. In symbolic (SAT/SMT-based model-checking), the states are not in a hash-table and the representation grows differently. In fact, a BDD-based hash-table has a size of 1 when it is empty as well as when it is full! These are K-layer DFAs basically. See `https://spinroot.com/gerard/pdf/sttt98.pdf`.

   (a) The buggy version of bubble sort was fixed. Before the assertion block t was initialised to 1 makes it keeps looking everywhere, "until the end of time".

```
        #define Size 5
#define aMinIndx 1
#define aMaxIndx (Size-1)
active proctype bubsort()
{
  byte j, t;   /* Init to 0 by SPIN     */
  bit a[Size]; /* Use 1-bit abstraction */
  do ::break ::a[1]=1 ::a[2]=1 ::a[3]=1 ::a[4]=1 od;
```




```
t=a[aMinIndx]; j=aMinIndx+1;

      do /* First “repeat” iteration */
      :: (j >(aMaxIndx)) -> break /*-- For-loop exits --*/
      :: (j<=(aMaxIndx)) ->
          if
          :: (a[j-1] > a[j]) -> t=a[j-1]; a[j-1]=a[j]; a[j]=t
          :: (a[j-1] <= a[j])
          fi;
      j++
      od;

 do /* Subsequent “repeat” iterations */
      :: t=a[1] ->
      t=a[aMinIndx]; j=aMinIndx+1;
```

```
        do
        :: (j > (aMaxIndx)) -> break /*-- For-loop exits --*/
        :: (j<=((aMaxIndx))) ->
        if
        :: (a[j-1] > a[j]) -> t=a[j-1]; a[j-1]=a[j]; a[j]=t
    :: (a[j-1] <= a[j])
        fi;
        j++ /*-- for-index increments --*/
        od  /*-- end of for-loop --*/
  od;
  do
    :: t < aMaxIndx-1 -> t++
    :: t > aMinIndx   -> t--
    :: a[t] > a[t+1]  -> assert(0) /*- announce there is a bug! -*/
  od
}
```

Tested the same code for array size of 6 and 10. Also, initialised different values for the array to verify the code.

```
38
39      do
40      :: t < aMaxIndx-1 -> t++
41      :: t > aMinIndx   -> t--
42      :: a[t] > a[t+1]  -> assert(0) /*- announce there is a bug! -*/
43      od
44    }
45
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ spin -a bs-fix.pml
ajantha@knight:~/Desktop/SV_ASS/Ass2$ gcc -DMEMLIM=1024 -O2 -w -o pan pan.c
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./pan

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 20 byte, depth reached 35, errors: 0
      192 states, stored
       65 states, matched
      257 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         1 (resolved)

Stats on memory usage (in Megabytes):
    0.009       equivalent memory usage for states (stored*(State-vector + overhead))
    0.287       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


unreached in proctype bubsort
        bs-fix.pml:40, state 51, "((t<((5-1)-1)))"
        bs-fix.pml:40, state 51, "((t>1))"
        bs-fix.pml:40, state 51, "((a[t]>a[(t+1)]))"
        bs-fix.pml:44, state 54, "-end-"
        (2 of 54 states)

pan: elapsed time 0 seconds
```

```
38
39    do
40    :: t < aMaxIndx-1 -> t++
41    :: t > aMinIndx   -> t--
42    :: a[t] > a[t+1]  -> assert(0) /*- announce there is a bug! -*/
43    od
44  }
45
```

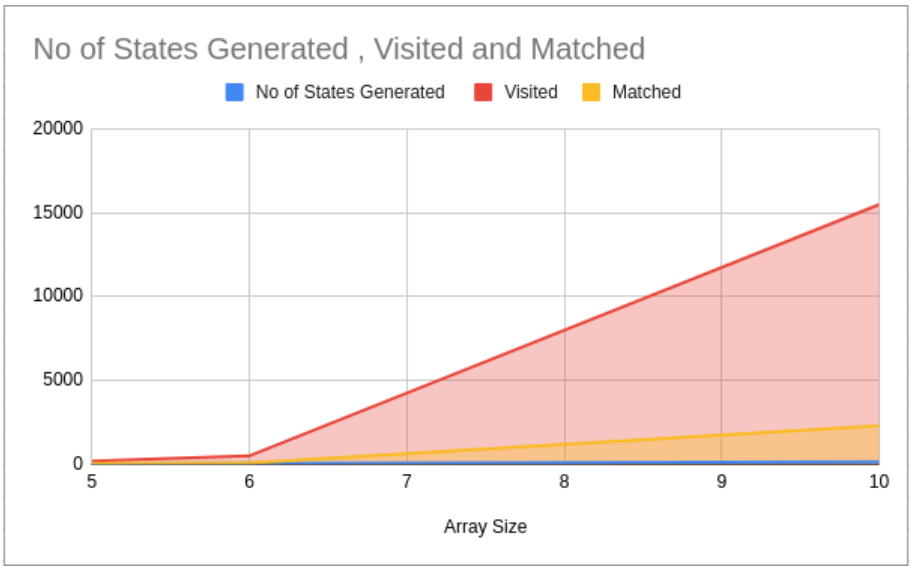
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ spin -a bs_6.pml
ajantha@knight:~/Desktop/SV_ASS/Ass2$ gcc -DMEMLIM=1024 -O2 -w -o pan pan.c
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./pan

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 20 byte, depth reached 54, errors: 0
      505 states, stored
       81 states, matched
      586 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         1 (resolved)

Stats on memory usage (in Megabytes):
    0.023       equivalent memory usage for states (stored*(State-vector + overhead))
    0.287       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


unreached in proctype bubsort
        bs_6.pml:40, state 52, "((t<((6-1)-1)))"
        bs_6.pml:40, state 52, "((t>1))"
        bs_6.pml:40, state 52, "((a[t]>a[(t+1)]))"
        bs_6.pml:44, state 55, "-end-"
        (2 of 55 states)

pan: elapsed time 0 seconds
ajantha@knight:~/Desktop/SV_ASS/Ass2$
```

```
#define Size 10
#define aMinIndx 1
#define aMaxIndx (Size-1)


active proctype bubsort()
{ byte j, t;   /* Init to 0 by SPIN     */
  bit a[Size]; /* Use 1-bit abstraction */


  do ::break ::a[1]=1 ::a[2]=11 ::a[3]=12 ::a[4]=9 ::a[5]=0  ::a[6]=1 ::a[7]=6 ::a[8]=1 ::a[9]=5  od;

  t=a[aMinIndx]; j=aMinIndx+1;

   do /* First ''repeat'' iteration */
   :: (j >(aMaxIndx)) -> break /*-- For-loop exits --*/
   :: (j<=(aMaxIndx)) ->
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ spin -a bs-size10.pml
ajantha@knight:~/Desktop/SV_ASS/Ass2$ gcc -DMEMLIM=1024 -O2 -w -o pan pan.c
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./pan

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 24 byte, depth reached 151, errors: 0
    15474 states, stored
     2305 states, matched
    17779 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         19 (resolved)

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


unreached in proctype bubsort
        bs-size10.pml:40, state 56, "((t<((10-1)-1)))"
        bs-size10.pml:40, state 56, "((t>1))"
        bs-size10.pml:40, state 56, "((a[t]>a[(t+1)]))"
        bs-size10.pml:44, state 59, "-end-"
        (2 of 59 states)

pan: elapsed time 0 seconds
```




| Array Size | No of States Generated | Visited | Matched |
|---|---|---|---|
| 5 | 35 | 192 | 65 |
| 6 | 54 | 505 | 81 |
| 10 | 151 | 15474 | 2305 |

4. (20 points) In the Philosophers example, the example suffered from "livelock" (all of them take the left, get nacked for the right, and repeat). One way to avoid such livelocks is to break symmetry (make one philosopher reach for the right fork first). Implement this solution. Now show that the system is lock-free but not wait-free (see `https://en.wikipedia.org/wiki/Non-blocking_algorithm` for these definitions). Lock-free is *communal progress* and wait-free is *individual progress*.
   - old instructions:
     - Use LTL assertions (not `never` automata).
     - Submit your work accompanied by traces and helpful observations.
   - new instructions:
     - Given the fix I posed on piazza (see `dp_contrarian.pml` pushed in)
     - Understand the fix (the LTL used). Try it to make sure it works. Make any suggestions on improvement

   In the Philosopher's example, to avoid deadlocks the forks were obtained based on two-phase locking. To sum up,

   - Acquire all the required resources in some sequential order
   - when some resource is found unavailable, instead of holding on to all the resources acquired thus far and waiting for the unavailable resource (which can cause deadlocks), release all the resources acquired thus far and restart the acquisition process.

   But it introduces livelocks in which neither anyone is waiting for resources nor executing the states.

   - Therefore, never automata is designed to check whether the progress bit is set to infinitely often.The never automaton can non deterministically loop in its initial do/od loop.
   - In fact, it can even stay in its initial state if progress is false. However, it may also choose to go to the final state labeled accept when progress is false. Here, it can continue to visit accept so long as progress is false
   - Hence, this never automaton accepts only an infinite sequence of progress being false—precisely when our desired liveness property is violated.
   - The code is modified to check for liveness property and LTL formula is $([] <> prop1[] <>!prop1[] <> prop2[] <>!prop2[] <> prop3[] <>!prop3(<> []!progress))$

   The code does not produce any errors when traced with claim in ispin or ran with -a for Never Automata.

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./a.out -a
warning: no accept labels are defined, so option -a has no effect (ignored)

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 144 byte, depth reached 22, errors: 0
       12 states, stored
        1 states, matched
       13 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))
    0.283       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


unreached in proctype phil
        phill2.pml:11, state 2, "prop1 = 1"
        phill2.pml:11, state 3, "prop1 = 1"
        phill2.pml:32, state 26, "lf!release"
        phill2.pml:37, state 36, "-end-"
        (4 of 36 states)
unreached in proctype altphil
        phill2.pml:44, state 2, "prop3 = 1"
        phill2.pml:44, state 3, "prop3 = 0"
        phill2.pml:49, state 10, "rf?yes"
        phill2.pml:49, state 10, "rf?no"
        phill2.pml:47, state 12, "rf!are_you_free"
        phill2.pml:57, state 17, "progress = 1"
        phill2.pml:58, state 18, "rf!release"
        phill2.pml:59, state 19, "lf!release"
        phill2.pml:60, state 20, "progress = 0"
        phill2.pml:64, state 23, "rf!release"
        phill2.pml:56, state 25, "lf?yes"
        phill2.pml:56, state 25, "lf?no"
        phill2.pml:54, state 27, "lf!are_you_free"
        phill2.pml:69, state 33, "-end-"
        (12 of 33 states)
unreached in proctype fork
        phill2.pml:76, state 4, "lp!no"
        phill2.pml:81, state 13, "rp!no"
        phill2.pml:85, state 22, "-end-"
        (3 of 22 states)
unreached in init
        (0 of 8 states)

pan: elapsed time 0 seconds
```

5. (20 points) Follow-along and finish the design of Dijkstra's distributed termination algorithm. You'll be asked to type these with me in class live, and finish. `http://people.cs.aau.dk/~adavid/teaching/MVP-10/17-Termination-lect14.pdf` and `https://www.cs.rochester.edu/u/sree/courses/csc-258/spring-2018/slides/22-td.pdf` do a good job of providing slides that I'll go thru in class. The original is `https://www.cs.utexas.edu/~EWD/ewd08xx/EWD840.PDF` from Dijkstra's collection `https://www.cs.utexas.edu/~EWD/indexBibTeX.html`. Look for files in the directory `Lec5`. Come Thu to type this fully and check it!

   **ADDENDUM:**

   (a) `DT.pml` is the initial template we began with

   (b) `DT_latest.pml` is the latest version we ended up with

   (c) Your task is:

      i. Begin with `DT_latest.pml` or your latest good solution

      ii. Attempt a random initialization of the A/P status (see code below). While the original Dijkstra implementation started with the root alone in "A," with the fix suggested below, all initializations are OK—which is a nice generalization!

      iii. **Fixed problem:** If you did not get any errors, then no fix needed! But, please plot the state-space growth for N=4 (if it finishes) and N=5 (if it finishes). Just give it about 15 min max and see if it finishes. That is all (no need to burn up your time and electricity).

      To tell you the back-story, I ran into a fix, then I thought the "upstream logic" was wrong. But now I can't reproduce the error. So perhaps this is after all a fixed protocol.

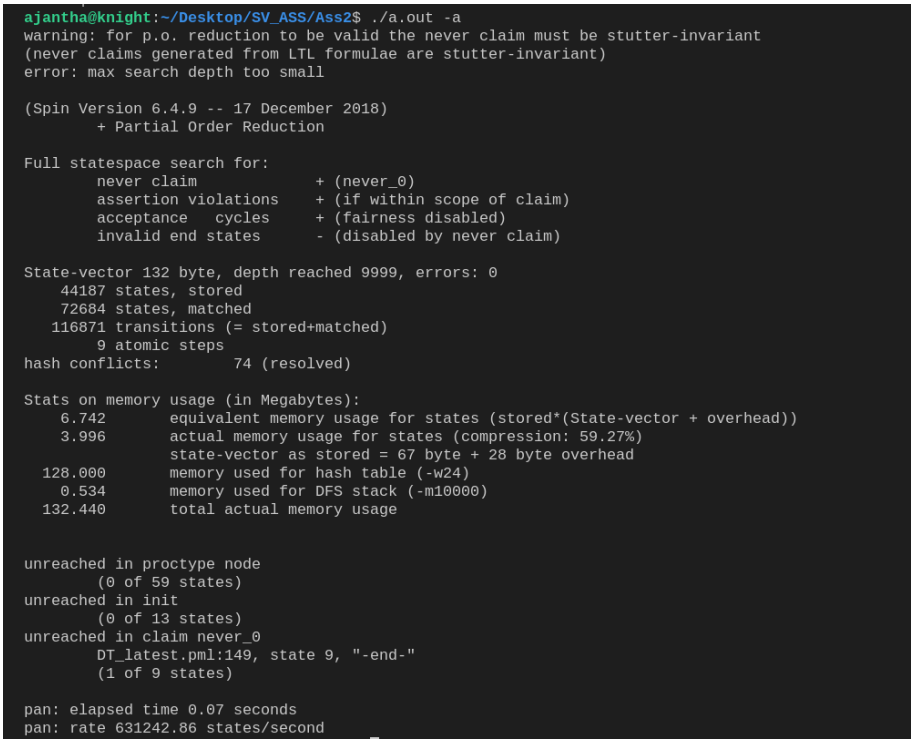
      iv. Fix enough about the protocol's implementation (see Piazza discussion). The hint is to think about how "upstream" must be implemented. **As if you want more hints, but as a private Piazza post!**

      v. Then run the protocol verification!

   **Detail of random initialization:**

```
init {
byte i = Ns-1;
        atomic {
        do
        :: i > 0 ->
           run node(tokqArray[i], tokqArray[i-1], workqArray[i], i);
   if //-- nondet initialization of initial state
   :: ns[i] = A
   :: ns[i] = P
   fi;
           i-- ;

        :: i == 0 ->
   run node(tokqArray[0], tokqArray[Ns-1], workqArray[i], i);
   if //-- nondet initialization of initial state
   :: ns[i] = A
   :: ns[i] = P
   fi;
   break
        od
        }
}
```

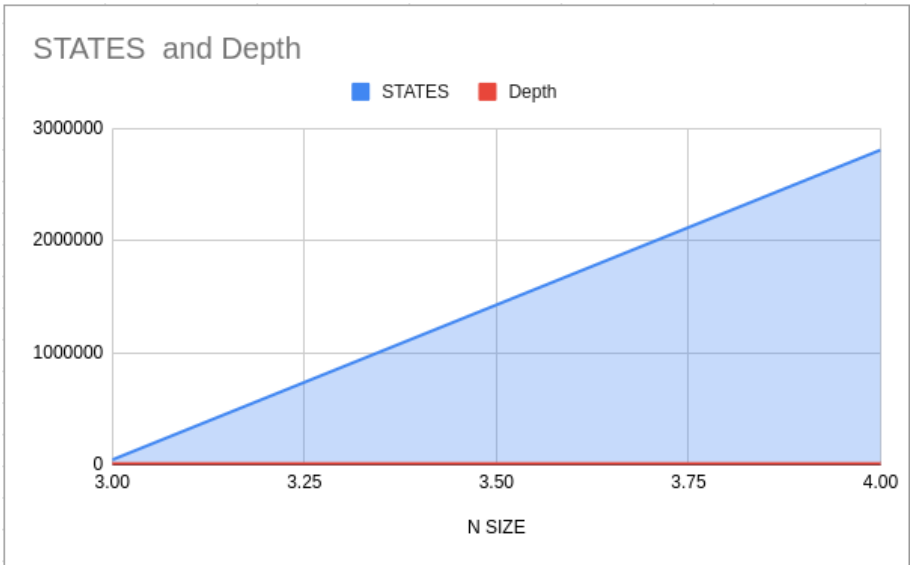
The modified code `DT_latest.pml` does not produce any errors.It ran for N = 3 and N=4 killed when N was 5.



```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./a.out -a
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
error: max search depth too small
Depth=    9999 States=    1e+06 Transitions= 3.04e+06 Memory=   227.948 t=     1.51 R=   7e+05
Depth=    9999 States=    2e+06 Transitions= 6.09e+06 Memory=   327.167 t=     3.08 R=   6e+05

(Spin Version 6.4.9 -- 17 December 2018)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (never_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 164 byte, depth reached 9999, errors: 0
  2808032 states, stored
  5859493 states, matched
  8667525 transitions (= stored+matched)
       12 atomic steps
hash conflicts:    236413 (resolved)

Stats on memory usage (in Megabytes):
  514.166       equivalent memory usage for states (stored*(State-vector + overhead))
  279.070       actual memory usage for states (compression: 54.28%)
                state-vector as stored = 76 byte + 28 byte overhead
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  407.343       total actual memory usage


unreached in proctype node
        (0 of 59 states)
unreached in init
        (0 of 13 states)
unreached in claim never_0
        DT_latest.pml:149, state 9, "-end-"
        (1 of 9 states)

pan: elapsed time 4.49 seconds
pan: rate 625396.88 states/second
ajantha@knight:~/Desktop/SV_ASS/Ass2$
```

```
ajantha@knight:~/Desktop/SV_ASS/Ass2$ ./a.out -a
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
error: max search depth too small
Depth=    9999 States=    1e+06 Transitions= 3.46e+06 Memory=   235.565 t=     1.65 R=   6e+05
Depth=    9999 States=    2e+06 Transitions= 6.82e+06 Memory=   342.401 t=      3.3 R=   6e+05
Depth=    9999 States=    3e+06 Transitions= 1.04e+07 Memory=   449.237 t=     5.06 R=   6e+05
Depth=    9999 States=    4e+06 Transitions= 1.39e+07 Memory=   556.073 t=     6.79 R=   6e+05
Depth=    9999 States=    5e+06 Transitions= 1.74e+07 Memory=   662.909 t=     8.54 R=   6e+05
Depth=    9999 States=    6e+06 Transitions= 2.07e+07 Memory=   769.745 t=     10.3 R=   6e+05
Depth=    9999 States=    7e+06 Transitions= 2.42e+07 Memory=   876.581 t=       12 R=   6e+05
Depth=    9999 States=    8e+06 Transitions= 2.76e+07 Memory=   983.417 t=     13.8 R=   6e+05
Depth=    9999 States=    9e+06 Transitions= 3.11e+07 Memory=  1090.253 t=     15.7 R=   6e+05
Depth=    9999 States=    1e+07 Transitions= 3.46e+07 Memory=  1197.089 t=     17.6 R=   6e+05
Depth=    9999 States=  1.1e+07 Transitions= 3.81e+07 Memory=  1304.022 t=     19.5 R=   6e+05
Depth=    9999 States=  1.2e+07 Transitions= 4.17e+07 Memory=  1410.858 t=     21.5 R=   6e+05
Depth=    9999 States=  1.3e+07 Transitions= 4.51e+07 Memory=  1517.694 t=     23.4 R=   6e+05
Depth=    9999 States=  1.4e+07 Transitions= 4.86e+07 Memory=  1624.530 t=     25.4 R=   6e+05
Depth=    9999 States=  1.5e+07 Transitions= 5.22e+07 Memory=  1731.366 t=     27.4 R=   5e+05
Depth=    9999 States=  1.6e+07 Transitions= 5.56e+07 Memory=  1838.202 t=     29.3 R=   5e+05
Depth=    9999 States=  1.7e+07 Transitions=  5.9e+07 Memory=  1945.038 t=     31.2 R=   5e+05
Depth=    9999 States=  1.8e+07 Transitions= 6.23e+07 Memory=  2051.874 t=     33.2 R=   5e+05
Depth=    9999 States=  1.9e+07 Transitions=  6.6e+07 Memory=  2158.710 t=     35.3 R=   5e+05
Depth=    9999 States=    2e+07 Transitions= 6.95e+07 Memory=  2265.546 t=     37.3 R=   5e+05
Depth=    9999 States=  2.1e+07 Transitions= 7.29e+07 Memory=  2372.382 t=     39.4 R=   5e+05
Depth=    9999 States=  2.2e+07 Transitions= 7.63e+07 Memory=  2479.315 t=     41.5 R=   5e+05
Depth=    9999 States=  2.3e+07 Transitions= 7.97e+07 Memory=  2586.151 t=     43.6 R=   5e+05
Depth=    9999 States=  2.4e+07 Transitions= 8.31e+07 Memory=  2692.987 t=     45.6 R=   5e+05
Depth=    9999 States=  2.5e+07 Transitions= 8.65e+07 Memory=  2799.823 t=     47.8 R=   5e+05
Depth=    9999 States=  2.6e+07 Transitions=    9e+07 Memory=  2906.659 t=     49.8 R=   5e+05
Depth=    9999 States=  2.7e+07 Transitions= 9.34e+07 Memory=  3013.495 t=       52 R=   5e+05
Depth=    9999 States=  2.8e+07 Transitions= 9.67e+07 Memory=  3120.331 t=     55.1 R=   5e+05
Depth=    9999 States=  2.9e+07 Transitions=    1e+08 Memory=  3227.167 t=     57.3 R=   5e+05
Depth=    9999 States=    3e+07 Transitions= 1.04e+08 Memory=  3334.003 t=     59.8 R=   5e+05
Killed
```

| N SIZE | STATES | Depth |
|---|---|---|
| 3 | 44187 | 9999 |
| 4 | 2808032 | 9999 |


STATES and Depth
STATES
Depth
3000000
2000000
1000000
0
3.00
3.25
3.50
3.75
4.00
N SIZE