CS 6110, Spring 2022, Assignment 4
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 Asg4/ 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. (40 points - 20 for pre and 20 for partial - Alloy) To learn Alloy, you can get a PDF copy of the **older** edition of the book by Daniel Jackson called "Software Abstractions." I've found a PDF by searching for the above. Since this is a very old edition, this is probably OK. Other tutorials are[1]

   `https://haslab.github.io/formal-software-design/overview/index.html` `https://www.cs.montana.edu/courses/se422/currentLectures/AlloyIntro.pdf` and `https://alloytools.org/tutorials/online/`. Start reading through this book and also Roger Costello's slides on the class Github.

   We have to understand mathematical relations properly before we can use Alloy. Read my chapter in CEATL on relations (the chapter featured in the tutorial I recorded). This is CS 2100 material—so, leaving it for your self-study.

   Here are some experiments I ran to study preorders and partial orders.

   TL;DR *The intersection of a preorder and its inverse is not an identity relation.*

   TL;DR *The intersection of a partial order and its inverse is an identity relation.*

   These are the conclusions to be drawn in the experiments to follow.

   Your task is to fill out the ellipsed portions (where I provide an English phrase for you to fill as `<text>`) and answer the questions below (questions begin with "Q:" and comments by "C:"):

```
-- C: We are defining "some old" relation 'pre'
-- C: nd slowly endowing it with the properties that make it a preorder
--
sig S { pre  :  set S } -- (1) C: This defines 'pre' as a binary relation over S
fact  { some pre }      -- (2) Q: Can you explain what this means?
fact  { <state here that pre is reflexive>  } -- (4) Q: answer the question below
fact  { <state here that pre is transitive> } -- (5) Q: answer the question below
assert preAndPreinvIden
```

[1] There is one more fantastic web-based one (I sent it to TM John once, but forgot where it is.)

```
 { <FALSELY Assert that the intersection of pre and its inverse is identity.> }
  -- (6) Q: answer the question below
check preAndPreinvIden for exactly 3 S
  -- (7) Q: Report on the result of this check
run {} for exactly 4 S
  -- (8) Q: If the check in (7) fails, comment (7) and run this to diagnose why
```

(1) `pre` is introduced as a “plain old” relation
(2) Explain what this assertion does for `pre`
(3) At this juncture, “run” for “`exactly 4 S`” and show the models generated.
(4) How did you endow ’pre’ with the property that it is reflexive? Explain.
(5) Explain how you endowed ’pre’ to be transitive
(6) How did you falsely assert that the intersection of pre and its inverse is identity?
(7) Did this check pass? You can use a pull-down menu and run this check. If the check in (7) failed, look at the counterexample and explain why it failed.
(8) Also run this “run” statement and see the instance generated. Does the instance generated provide another explanation as to why the above `check` failed?
(9) Now, define a *partial order* along the same lines as below. Instead of the assertion `preAndPreinvIden`, define `partAndPartinvIden`, where we changed “pre” to “part” (partial order). Did this check pass? Justify the answer.

2. (10 points, Store Buffer) Run the store-buffer example created in Promela. Argue that it simulates the situation of the writes being buffered before it goes to memory (as in TSO which is described at `https://en.wikipedia.org/wiki/Memory_ordering` and elsewhere). Observe the assert failure and explain the interleaving causing the bug. (The file in question is `Peterson_tso.prm`.)

Your Answer Here

3. (10 points, The Java example with volatiles) Read-up on Java volatiles. Run the example `VBad.java`. Insert volatiles selectively (just for req or just for ack). Does that correct the apparent hang? (I don’t know the answer but thought you’d like to try.) To get the apparent hang, first you must leave out the volatile totally and get the hangs on your machine. *Then* try to add one volatile and see if you get a hang. Explain your observations. (Bound your empirical testing to say an hour.)

Your Answer Here

4. (10 points, the MWGC game) Write a pseudo-code for a DFS model-checker by modifying the BFS model-checker’s pseudo-code here `https://www.cs.utah.edu/~kirby/Publications/Kirby-33.pdf`. Do you now understand why a model-checker does not “infinitely loop?” Now, run the murphi model I wrote today (you should be able to recreate it from memory or the recording) and run it in DFS. Does the error-trace (winning sequence) get longer? Can you go after longer error traces (by not stopping at the first error)? Try to produce a longer error trace than with DFS. Describe that “winnning sequence” (error trace for the negated invariant).

Your
Answer
Here

5. (20 points, DCL) Read Pugh's analysis of the Java Memory Model through the URL `https://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html` and describe what weak memory ordering issues are discussed there? (This is a classic website that sowed the whole area of understanding weak memory models; Java was supposed to be a "safe" language, but with weak memory, you could export an object reference before initializing the object—thus leaking a secret that you did not wipe out.)

   Your
   Answer
   Here