CS 6110, Spring 2022, Assignment 6

Given 3/4/22 – Due 3/15/22 by 11:59 pm via your Github

NAME: Jack Wilburn UNID: u0999308

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 Asg6/ 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! *But that does not mean copy solutions, but discuss the question plus surrounding issues.*

1. (40 points) I've checked in Logic[1-5].als. There are two parts to this assignment:
   (a) (20 points)
      i. (10 points) Go through what I've provided in the above models, and for each file, produce a gist of the encodings I've done and a two-line comment saying how the encoding works. (Save Logic4.als for the next part.)
      ii. (10 points) Get into Logic4.als and make sure that `f2` indeed is encoding a general two-ary function. To "make sure," you must demonstrate a sufficient number of alloy tests/checks/runs and/or model-examination. Have at least 3-4 convincing demos of checks you came up with.
   (b) (20 points)
      i. (10 points) Lecture 16, Slide 4, Image of question 13(b): Encode this assertion similar to the encoding in Logic5.als and check for validity.
      ii. (10 points) Lecture 16, Slide 4, Image of question 13(c): Encode this assertion similar to the encoding in Logic5.als and check for validity.

Logic 1:
This creates 2 zero-ary relationships and declares that there must be at least one of each of those relationships.
Logic 2:
This creates a 1-ary and 2-ary pred and then checks a FOL statement on the 2-ary.
Logic 3:
This encodes a 2-ary predicate with a 1-ary function. The given FOL statement holds given the relationship and the predicate.
Logic 5:
This encodes 13a from the slides. It appears to hold
Two-ary:
I’m unsure how to show this.
13(b)
See 13b.als
13(c)
See 13c.als

2. (40 points)
   (a) (20 points)
      i. (10 points) Go through Mike-Gordon-Slides.pdf and also read Mike Gordon’s book, making notes about the proof rules of Hoare Logic presented there. Include assignment, if, for, while, precondition strengthening, and postcondition weakening.
      ii. (10 points) Get Dafny and Verifast installed. Please report issues you faced. Say which platform. Put notes in the tools GDoc to help each other.
   (b) (20 points) Run the first two Dafny exercises in Lec16.pdf and ask questions on Piazza. All details are given to you.

Notes:

- notation called a partial correctness specification for specifying what a program does. E.g. {P} C {Q}
- C is a command, P and Q are conditions on the program variables used in C
- {P} C {Q} is true if whenever C is executed in a state satisfying P, and if the execution of C terminates, then the state in which C terminates satisfies Q
- Formal verification uses formal proof. A proof consists of a sequence of lines. Each line is an instance of an axiom or follows from previous lines by a rule of inference
- Computers can do formal verification. Formal verification by hand generally not feasible
- An expression P C Q is called a partial correctness specification. P is called its precondition, Q its postcondition. It is only required that if the execution terminates, then Q holds
- {X = 1} WHILE T DO X := X {Y = 2} – this specification is true!
- Total correctness requires that the program finishes. Above equation doesn't hold.
- Total correctness = Termination + Partial correctness. Usually easier to show partial correctness and termination separately
- Program states are specified with first-order logic (FOL)
- If S is a statement, S means S has a proof
- Q[E/V] is the result of replacing all occurrences of V in Q by E
- The Assignment Axiom: Q[E/V] V :=E Q
- Most intuitive to me was P V :=E P [E/V], which is wrong
- Precondition strengthening means if p $\implies$ p' and {p} C Q then {p} C Q
- You can do similar with the postcondition to weaken it: {p} C q' and q' $\implies$ q then {p} C q

The conditional rule

$$\frac{\vdash\ \{P \wedge S\}\ C_1\ \{Q\}, \qquad \vdash\ \{P \wedge \neg S\}\ C_2\ \{Q\}}{\vdash\ \{P\}\ \texttt{IF}\ S\ \texttt{THEN}\ C_1\ \texttt{ELSE}\ C_2\ \{Q\}}$$

- if rule:

The WHILE-rule

$$\frac{\vdash\ \{P \wedge S\}\ C\ \{P\}}{\vdash\ \{P\}\ \texttt{WHILE}\ S\ \texttt{DO}\ C\ \{P \wedge \neg S\}}$$

- while rule:

The FOR-axiom

$$\vdash\ \{P \wedge (E_2 < E_1)\}\ \texttt{FOR}\ V\texttt{:=}E_1\ \texttt{UNTIL}\ E_2\ \texttt{DO}\ C\ \{P\}$$

- for rule:

Installed Dafny and verifast very easily through the arch linux user repository. I ran Dafny with the 2 examples. The gcdm seems to throw errors even with the version you had as "validated". Maybe it's a mismatch in dafny versions causing an issue.

3. (20 points) Write a summary of your project along these lines, occupying two pages. Note: I'm adding topics to the GDoc `bit.ly/CS6110-S22-Project-Suggestions` and please add details there (this is a writeable GDoc also) or even new topics are welcome to be recorded there. In any case, please profit from the ideas there. Push it into your Github which can then be used to drive your project also.
   - A project name (tentative names are OK), a topic, and about 10 lines (12pt font) on why it matters to someone studying SW verification.
   - A description of the verification technologies that you'll learn by doing this project. (It could be a topic not yet covered in class such as static analysis.)
   - A few diagrams and other details you like to add to give me a better idea of the work.
   - If you'd like to have a project partner, plz note that.
   - Assuming you have 70% of the CS 6110 time available for your project, a brief timeline of how you'll deliver your working project by the first day of the Spring exam week.

Project Name:
Network Protocol Check-a-roo
Topic:
You (Ganesh) gave me a bunch of network protocol examples with murphi. I plan to look through them, understand them, and write up a longer description for the protocol and the checker code. I think for now, writing that up for 2 of the models should be sufficient to show my background understanding. After that, I'll chose a protocol that we don't have a model for and make a model for it. I'll then make a longer write up for it as I did with the chosen 2 from before.
Why it matters:
It matters, because we rely on web protocols to be bulletproof. If any protocols have flaws, it will be important to understand the flaws. I'm sure all these protocols have been checked formally, but if not, it could highlight undiscovered vulnerabilities.
Verification tech learned:
I'll be using murphi for the project so that I can utilize the given models. Once I have a model built for another protocol, I might try another language to verify my results.
Diagrams (if applicable):
There aren't any diagrams for now. I'll have them in my final write up with the protocols they represent.
Partner: N/A
Timeline:
By April 1: Looked through given models and chosen models to use
By April 14: Written up 2 models from the given code. Chosen another protocol to model
By April 28: Modeled the final protocol and written up the analysis for turn in