Homework #2

Code, execution runs and comments thereon should be submitted electronically to `mitchell@cs.ubc.ca` as plain ASCII text. Diagrams and discussion should be submitted in hardcopy. Keep your answers brief, and **get started early!**

1. **Bubbles and Arrows.** To my knowledge, British Columbia is the only driving jurisdiction that has *flashing green* traffic lights to denote traffic lights which will change only if a pedestrian (or in some cases a cyclist) presses a button. This behaviour might be confusing to visitors, so we will try to describe it to them using our finite automata tools.

   (a) Sketch a bubble and arrow model of how a British Columbia flashing green traffic light works from a pedestrian's perspective. Include all the different lights: flashing green, amber, red, don't walk (usually a red hand), and walk (usually a white stick figure).

   (b) Is the model deterministic? Can the model deadlock? Briefly explain your answers.

   (c) Is your *model* Markovian? If so, briefly explain how the real system can violate the Markovian behaviour of the model. If not, briefly explain why not.

2. **Finite Automata.** Consider a system modeled with the following asynchronous guarded commands. Symbols p, d, b and a represent the occurance of an external event. Symbols o and s are internal Boolean state variables, and $f \in \{0, 1, 2\}$ is an internal integer subrange.

$$\begin{aligned} o \wedge \neg s \wedge d &\rightarrow s = True, f = 0 \\ o \wedge s \wedge d &\rightarrow f = \max(f - 1, 0) \\ o \wedge \neg s \wedge a &\rightarrow s = True \\ o \wedge s \wedge a &\rightarrow f = \min(f + 1, 2) \\ o \wedge b &\rightarrow s = False \\ o \wedge p &\rightarrow s = False, o = False, f = 0 \\ \neg o \wedge p &\rightarrow o = True \end{aligned}$$

   The initial state of the system is $\neg o$, $\neg s$ and $f = 0$. Assume that a timestep only occurs when one of the inputs p, d, b or a occurs. Only a single input may occur at any timestep.

   (a) Sketch a bubble and arrow diagram of the system behavior. States should be labelled with o, s, $\neg o$ and/or $\neg s$ as well as a value for f. Arcs should be labelled with input symbols. Your diagram need not include states that cannot be reached from the initial state.

   (b) Is the system deterministic? Briefly explain your answer.

   (c) Is the system blocking? Briefly explain your answer.

3. **Experiments with Murφ.** Consider an intersection between two roads: one runs north-south and the other east-west. The east-west road has stop signs, so traffic traveling in those directions must stop and wait until it is clear to proceed across the intersection. At the course web site you will find a Murφ source file `stopsign.m` describing this model and instructions for compiling it.

   (a) Sketch a bubble and arrow diagram (an informal Kripke structure) showing the behavior of the model for a single car. To get a feeling for the model, you may find it useful to generate some simulation traces with `stopsign -s -p`.

   (b) Are there any redundant states in the model? If so, how might you remove them?

(c) Demonstrate by model checking that the protocol in `stopsign` may permit a collision. Generate a trace which violates the invariant. Briefly describe the source of the failure in plain English.

(d) Modify the protocol, and demonstrate by model checking that your new protocol satisfies the invariant. Briefly explain how your modification works.

(e) Write an invariant that specifies that north-south traffic is never forced to wait. Demonstrate by model checking whether this invariant is satisfied by either the old or your new protocol, or both.

Because it allows only a finite number of cars and each car may pass through the intersection only once, the model will eventually deadlock. If necessary, you may disable deadlock detection with `-ndl` in order to detect the invariant failures.

Please include your sketch with your paper submission, but you may email your modified code (with comments identifying and explaining all modifications) and your executions (results and traces) to the professor.

4. **Write your own Murφ.** Create a Murφ model `trafficlight.m` for an intersection controlled by a traffic light. The rules for a traffic light are:

   - When the light is green in their direction, cars may enter and exit the intersection.
   - When the light is yellow in their direction, cars may exit but not enter the intersection.
   - When the light is red in their direction, cars may not enter or exit the intersection.
   - Either the north-south light or the east-west light must be red at any given time.

   You will probably find it easiest to start with `stopsign.m` and work from there. You will need to add variables and rules for the traffic light.

   (a) Demonstrate that your model is free of collisions.

   (b) Demonstrate that all cars eventually pass through the intersection. Is this the same as demonstrating that no car will wait forever? Briefly explain your answer.

   (c) Briefly discuss significant ways in which this model is unrealistic, and whether Murφ would ever be capable of modeling and/or checking these features.

5. **Bonus: Taming State Space Explosion with Symmetry.** Try validating your model from the previous example with larger and larger values of `carCount`. At some point Murφ will fail because there are too many states in the queue. This state space explosion occurs because Murφ is trying to check all possible arrival patterns of all possible cars, even though we know that it does not matter whether car 1 arrives eastbound and car 2 northbound or car 1 northbound and car 2 eastbound. To specify to Murφ that the cars can be treated symmetrically, we use the `scalarset` construct for `carID` instead of a regular integer range.

   (a) Modify your code to use `scalarset` for `carID`. Demonstrate that you can prove correctness for more cars after this modification.

   (b) Using `scalarset` places some restrictions on how a `carID` variable may appear in the program, so you may have had to modify some of the program's rules. If you did modify rules, specify your modifications and describe how the set of behaviors for the modified protocol compares to the set of behaviors for the original protocol (eg: superset, subset, equal to, etc). What can we conclude about the ability of the original protocol to handle the number of cars which were proved in the modified protocol?