Formal Methods in Software Engineering

Assignment 2 (Spin and Model-Checking)

(Due on Wed 16th Feb 2022)

1. Consider the pairs of LTL assertions below. For each pair either give an informal proof that they are equivalent, or a counter-example model (an infinite sequence of propositional valuations) that shows they are not equivalent. Your counter-example should be in the form of an ulitmately periodic word of the form $u \cdot (v^\omega)$, where u and v are finite sequences of propsitional valuations over the propositions $\{p, q, r\}$.

   (a) $G(Fp)$ and $F(Gp)$.

   (b) $\neg(pUq)$ and $(\neg q)U(\neg p)$.

2. Consider the Promela model below.

```
byte x = 1;

proctype counter() {
  do
  :: x <= 1 -> x = x + 1;
  :: x >= 1 -> x = x - 1;
  od
}

init {
  run counter();
}
```

   (a) Describe the transition system that Spin compiles this model into. You may choose to ignore the intermediate states Spin uses to move to a satisfied guard. Show only the reachable states.

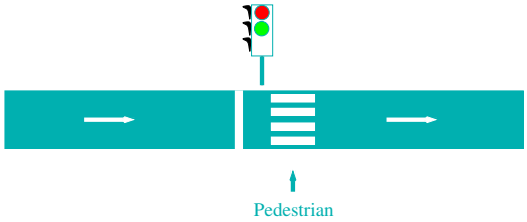
   (b) Does this model satisfy the property "$(x = 1) \implies F(x = 2)$"? Justify your answer.

   (c) Construct a state-based Büchi automaton (by hand if you like) for the *negation* of this formula, over the propositions p representing $x = 1$, and q representing $x = 2$.

   (d) Construct the product of the transition system of the model and the Büchi automaton for the formula above. Describe the counterexample path if any.

3. Construct the formula automaton for the LTL formula $true\,Up$, using the algorithmic construction described in class. You could treat *true* as $p \vee \neg p$.

4. Imagine that you have been asked to implement a traffic light for a pedestrian crossing as shown in the figure below. The aim of this exercise is to use Spin to (a) capture your design, and (b) debug and eventually verify it, before you move on to implementing/fabricating it.



There are three lights for the vehicles as usual (green, amber, red), and also two lights (green, red) for pedestrians who wish to cross. The pedestrians press a button to indicate that they wish to cross. Here are some of the requirements given to you:

(a) The vehicle light stays green for at least 3 time units, and is never red for more than 3 time units at a stretch.

(b) (Safety) Whenever the pedestrian light is green, the vehicle light should be red.

(c) (Ped-Liveness) Whenever the pedestrian presses the cross button, they should be able to cross within 5 time units.

(d) (Veh-Liveness) Whenever the vehicle light is red, it eventually becomes green.

(e) (Veh-green) Unless there are cross button presses, the vehicle light should remain green. More precisely, if we never have cross button presses after some point in time, then eventually the vehicle light should remain green forever.

Build a Promela model of your design, taking into account these requirements, with three active proctypes, "trafficlight", "timer", and "pedestrian" as shown below. Specify the last four requirements above as LTL properties in your Promela model.

```
mtype = { GREEN, AMBER, RED };
mtype = { GO, GSCHANGE, SGCHANGE, STOP };

bool tick = false;
bool cbutton = false;
bool ctr = 0;
vlight = GREEN;
plight = RED;
...

active proctype trafficlight() {
  do
    ...
  od;
}

active proctype timer() {
  do
  :: tick = false;
  :: tick = true;
  od;
```

```
}

active proctype pedestrian() {
  do
  :: pbutton = false;
  :: pbutton = true;
  od;
}
```

Your answer should contain a series of Promela models named `tlight-v1.pml`, `tlight-v2.pml`, ..., accompanied with a description of the issues you found (if any) while using Spin to verify the four properties, and how you plan to fix it in the next model.