COMP 408: Verifying Simple Protocols in SMV

Due in my mailbox in Duncan (room 3141) by 5pm, Friday, January 31. You should turn in your solutions in accordance with the instructions for documenting assignments in SMV on the course assignments page. However, **you do not need to provide error track analyses for this assignment** (with one exception as noted below).

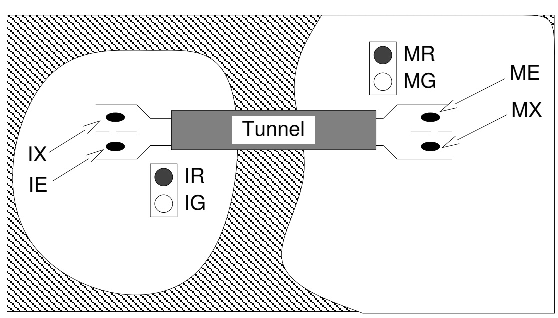
Instead, for this assignment, I want you to provide "sanity checks" for the properties that you verify. SPECs sometimes analyze to true because they represent vacuous statements. For example, a SPEC of the form $\mathsf{AG}(a \rightarrow b)$ may be true because a is never true. Sanity checks are additional SPECs that you write to make sure that a SPEC didn't verify by virtue of being vacuous. For $\mathsf{AG}(a \rightarrow b)$, the sanity check would be $\mathsf{EF}a$. For each SPEC that you verify in this assignment that could be vacuously true, also write down and verify a sanity check for it.

Exercise 1 (**Clayton Tunnel**) In `comp408/Assignments/2`, you will find *clayton.smv*, an SMV description of the Clayton Tunnel Protocol that we discussed in class. Copy that file and do the following:

1. Specify the desired property that no two trains should be in the tunnel on the same track at the same time. Check whether *clayton.smv* has that error. If it does, provide a summary of the relevant error track. If it does not, explain why this model doesn't have the problem discussed in class and provide a set of appropriate sanity checks.
2. Change the original protocol so that if multiple trains do get into the tunnel, future trains are prohibited from entering until the tunnel is cleared. Explain in English (either in a long comment or on a separate sheet of paper) how you changed the protocol. Also explain any assumptions that your protocol makes (about timing, the behavior of the trains, etc). How general is your solution (i.e., does it only work up to a certain number of trains getting stuck in the tunnel)?

Exercise 2 (**Island Tunnel Controller**)

Assume that we want to design a controller for the traffic lights at a one lane tunnel connecting the mainland to a small island as pictured below. There is a traffic light at each end of the tunnel; there are also four sensors for detecting the presence of vehicles: one at tunnel entrance on the island side (IE), one at tunnel exit on the island side (IX), one at tunnel entrance on the mainland side (ME), and one at tunnel exit on the mainland side (MX).



Although this looks quite similar to the one-lane bridge problem that we discussed in the first class, there are differences between the two systems. This tunnel controller does not limit the number of cars that it lets through while the light is green. However, the island is small, so a maximum of 16 cars is allowed on the island at any time. The controller has been designed under the assumption that once a car is waiting to use the tunnel, it keeps waiting until it gets to use the tunnel.

The desired properties of the controller are as follows:

1. The two lights are never green simultaneously.
2. A light does not turn green until the tunnel is empty.
3. If a car is waiting to use the tunnel, the light on that side of the tunnel eventually turns green.
4. There are never more than 16 cars on the island.

File `/comp408/Assignments/2/itlc.smv` contains an SMV description of the Island Traffic Light Controller. Copy the file and do the following:

1. Write an environment model for the controller. Your model should generate the signals IE, IX, ME, and MX (the sensors). Make sure what you generate makes sense; for example, it shouldn't be possible for IX to be true before ME was true. The controller interprets each sequence of 1s on these signals as corresponding to a single vehicle. You should not change the file I have given you other than to add your model and a main module that ties your model into the controller design. Turn in a copy of the SMV file with your model in it. Please provide a comment explaining how your model works.
2. Verify the controller (use the above properties) under your environment model.
3. Does your model limit the number of cars that exist? If so, what limit does it impose and why?
4. Can you verify the following property using your model: "If a car is waiting to use the tunnel, it eventually gets into the tunnel"? If you can verify this, provide the SPEC and the sanity checks. If you cannot, give an explanation in English as to why. (Note: you do not need to change your model to satisfy this property. Answer this question based on whatever model you eventually used to verify the controller. I'm interested in how you evaluate your particular choice of model for this question, not in whether your model satisfies the property.)