COMP 408: Modeling and Verifying Simple Designs in SMV
due: Feb 1, 2001; 6pm (electronically)

These exercises are intended to give you some practice modeling designs, specifying properties, and using SMV. Turn in your solutions in accordance with the instructions for documenting SMV assignments on the course assignments page.

Exercise 1 (**Arithmetic Shifter**) An *arithmetic shifter* stores a bit-vector of some specified length. It has two control signals: shift-left and shift-right. When shift-left is asserted, the bits in the vector are shifted one bit to the left, with a 0 shifted into the rightmost bit. When shift-right is asserted, the bits in the vector are shifted one bit to the right, with a 0 shifted into the leftmost bit. For example:

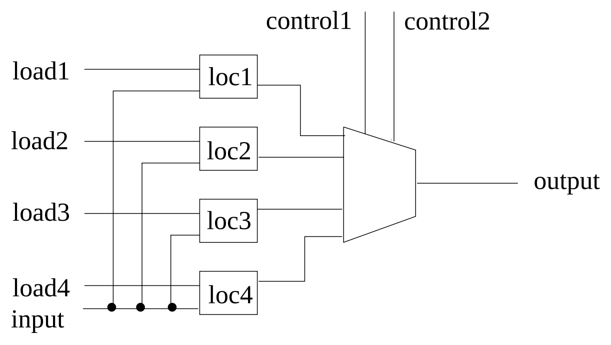
Contents of vector	Operation	New contents of vector
1 1 0 1	shift-right	0 1 1 0
1 1 0 1	shift-left	1 0 1 0

1. Write an SMV specification of a 4-bit arithmetic shifter; call the file *shifter.smv*. Use a style similar to that used for the counter in the SMV manual (*i.e.*, make a cell module, where each cell corresponds to a single bit of the shifter).
2. Determine the properties you need to test in order to verify the shifter. Argue why the set of properties you propose is sufficient.
3. Use SMV to verify the properties given in response to part 2.

Exercise 2 (**Simple Memory**) Suppose we have a simple memory with a single input bus and four storage locations. Each storage location has its own load signal; when the load signal is asserted, the value currently on the input bus is stored in that memory location. There is a single output bus for the memory. The value on the bus is controlled (via a multiplexer, if you're familiar with that term) by a pair of output control signals. The values on these signals indicates which location's contents should be placed on the bus, according to the following table:

Control1	Control2	Location
0	0	1
1	0	2
0	1	3
1	1	4

The layout of this memory appears as follows:



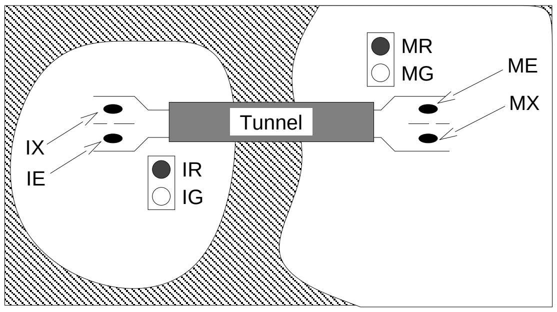
The desired properties of the memory are:

- Values are properly stored in the memory based on the load signals.

- Values are routed from the locations to the output bus as specified by the above table.
- Once a value has been loaded into a location, the location retains the value until the next load to that location occurs.

Download the file *simplemem.smv* from the assignments page and use SMV to verify the above properties against that design. Turn in your modified *simplemem.smv* file.

Exercise 3 (**Island Tunnel Controller**) Assume that we want to verify 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).



The 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 *itlc.smv* on the assignments page contains an SMV description of the Island Traffic Light Controller. Download 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: do not 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.)

Exercise 4 (**Clayton Tunnel**) In class, we discussed the Clayton Tunnel accident and the (faulty) protocol that gave rise to the accident. Do the following exercises using the protocol description handed out in class:

1. Develop an SMV model of the faulty protocol. Put it in a file called *clayton-bad.smv*.
2. Specify the desired property that no two trains should be in the tunnel on the same track at the same time. Check the property against your protocol model (it should fail). Summarize the error trace in a comment.
3. 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. Create a file *clayton-good.smv* containing a model of your corrected protocol, and check that it satisfies the new constraint.

   Explain in English (in a long comment) 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)?