NUSMV Exercises

Alessandra Giordani
agiordani@disi.unitn.it
http://disi.unitn.it/~agiordani

Formal Methods Lab Class, May 13, 2014

UNIVERSITÀ DEGLI STUDI DI TRENTO

*These slides are derived from those by Stefano Tonetta, Alberto Griggio, Silvia Tomasi, Thi Thieu Hoa Le for FM lab 2011/13

An Elevator Controller

The SMV program on the next slides describes the skeleton of an elevator system for a 4-floors building. The skeleton includes modules both for the physical system (reservation buttons, cabin, door), and for the controller. It also shows the connection between modules.

The objective of the exercise is to complete the program, and check that the given requirements are satisfied. You will have to formalize the transition relation for the existing variables, possibly introduce additional variables and definitions, formalize the requirements in temporal logic, and make sure that the design satisfies them.

Elevator – Button

For each floor there is a button to request service, that can be pressed. A pressed button stays pressed unless reset by the controller. A button that is not pressed can become pressed nondeterministically.

REQ : The controller must not reset a button that is not pressed.

Elevator – Cabin

The cabin can be at any floor between 1 and 4. It is equipped with an engine that has a direction of motion, that can be either standing, up or down. The engine can receive one of the following commands: nop, in which case it does not change status; stop, in which case it becomes standing; up (down), in which case it goes up (down).

- REQ : The controller can issue a stop command only if the direction is up or down.
- REQ : The controller can issue a move command only if the direction is standing.
- REQ : The cabin can move up only if the floor is not 4.
- REQ : The cabin can move down only if the floor is not 1.

Elevator – Door

The cabin is also equipped with a door (kept in a separate module in the SMV program), that can be either open or closed. The door can receive either open, close or nop commands from the controller, and it responds opening, closing, or preserving the current state.

- REQ : The controller can issue an open command only if the door is closed.
- REQ : The controller can issue a close command only if the door is open.

Elevator – Controller

The controller takes in input (as sensory signals) the floor and the direction of motion of the cabin, the status of the door, and the status of the four buttons. It decides the controls to the engine, to the door and to the buttons. The controller must also satisfy the following requirements.

- REQ : no button can reach a state where it remains pressed forever.
- REQ : no pressed button can be reset until the cabin stops at the corresponding floor and opens the door.
- REQ : a button must be reset as soon as the cabin stops at the corresponding floor with the door open.
- REQ : the cabin can move only when the door is closed.
- REQ : if no button is pressed, the controller must issue no commands and the cabin must be standing.

Elevator – SMV skeleton

```
MODULE Button(reset)
VAR
  pressed : boolean;
```

```
MODULE Door(door_cmd)
VAR
  status : { open, closed };
```

```
MODULE Cabin(move_cmd)
VAR
  floor     : { 1,2,3,4 };
  direction : { standing,
                moving_up,
                moving_down };
```

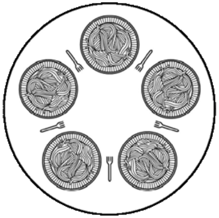
```
MODULE CTRL(floor, dir, door, pressed_1,
            pressed_2, pressed_3, pressed_4)
VAR
  move_cmd : {move_up, move_down, stop, nop};
  door_cmd : {open, close, nop};
  reset_1, reset_2, reset_3, reset_4 : boolean;
```

Elevator – main module

```
MODULE main
VAR
  cabin : Cabin(ctrl.move_cmd);
  door : Door(ctrl.door_cmd);
  button_1 : Button(ctrl.reset_1);
  button_2 : Button(ctrl.reset_2);
  button_3 : Button(ctrl.reset_3);
  button_4 : Button(ctrl.reset_4);
  ctrl : CTRL(cabin.floor, cabin.direction, door.status,
              button_1.pressed, button_2.pressed,
              button_3.pressed, button_4.pressed);
```

Philosophers Problem - Description

Five philosophers sit around a circular table and spend their life alternatively thinking and eating. Each philosopher has a large plate of noodles and a fork on either side of the plate. The right fork of each philosopher is the left fork of his neighbor. Noodles are so slippery that **a philosopher needs two forks to eat it**. When a philosopher gets hungry, he tries to **pick up his left and right fork, one at a time**. If successful in acquiring two forks, he **eats for a while** (preventing both of his neighbors from eating), then **puts down the forks, and continues to think**.



Philosophers Problem - Exercise

1. Implement in SMV a system that encodes the philosophers problem. Assume that when a philosopher gets hungry, he tries to pick up his left fork first and then the right one.
   Hint: you might consider an altruist philosopher.

2. Verify the correctness of the system, by specifiying and checking the following properties:
   - Never two neighboring philosophers eat at the same time.
   - No more than two philosophers can eat at the same time.
   - Somebody eats infinitely often.
   - If every philosopher holds his left fork, sooner or later somebody will get the opportunity to eat.