Joseph Muffoletto
ASE 389
Prof. Topcu

Homework 3

Question 1 (Q1.smv)

Part1:

The controller defines the following information for each state:

- **Floor:** current floor {0,1,2,3}
- **Doors:** current array of open doors, initially [FALSE, FALSE, FALSE, FALSE]
- **Buttons:** current array of buttons, controlled by the environment
- **Req1:** currently served request, takes on the values -1..3. "-1" means no floor request
- **Req2:** cached request, takes on the values -1..3
- **Timer:** time since floor 0 was requested, takes on the values 0..2

We satisfy the first spec, "**A requested floor will be served some time**", by serving req1. To "serve req1" is to go to its requested floor and open the door (floor=req1 & doors[req1]=TRUE). Once this occurs, req1 is set to the cached request (req2) if it has a requested floor (i.e. req2!=-1), otherwise, it is set to any button that is TRUE, or back to -1.

- Corresponding LTL spec: `LTLSPEC G(((req1=0 | req2=0) -> F(doors[0] = TRUE)) & ((req1=1 | req2=1) -> F(doors[1] = TRUE)) & ((req1=2 | req2=2) -> F(doors[2] = TRUE)) & ((req1=3 | req2=3) -> F(doors[3] = TRUE)))`
- The mechanism for moving to a floor is straightforward: move in the direction of the requested floor (req1) until floor=req1. Then, open the door.

We satisfy the second spec, "**Again and again the elevator returns to floor 0**", using the timer variable. The timer variable is the number of consecutive steps where req1!=0 & req2!=0, and caps at a value of 2. Once the timer reaches 2, req2 will be set to 0 at the next available step. This ensures that we will always return to floor 0. This does mean that, if no buttons are pressed, the door will infinitely stay open on the bottom floor, which is probably an odd behavior in the real world.

- Corresponding LTL spec: `LTLSPEC G (F floor=0)`

I included an additional spec, "A door cannot be open unless the elevator is at its floor", as a sanity check to ensure our doors are "safe".

- Corresponding LTL spec (1 out of 4): `LTLSPEC G (doors[0]=TRUE -> (floor=0 & doors[1]=FALSE & doors[2]=FALSE & doors[3]=FALSE))`

Part 2:

To satisfy the third spec, "**When the top floor is requested, the elevator serves it immediately and does not stop on the way there**", I forced all third-floor requests to be in req1 (i.e., G(req2!=3)). While this does mean that we cannot cache a top floor request, it is a restriction that my controller needed to be able to guarantee we would always serve it first. In essence, this restricts how often the top floor can be requested since it requires (req1=-1 & req2=-1 & buttons[3]=TRUE) for the top floor to be requested.

To partially alleviate this restriction, the controller prioritizes requests to the top floor: if buttons[3]=TRUE and req1 is empty, then regardless of which other buttons were pressed, req1 will select floor 3 to serve.

- Corresponding LTL spec: `LTLSPEC G ((req1=3 | req2=3) -> ((floor=0 & doors[0]=FALSE) -> X floor=1) & ((floor=1 & doors[1]=FALSE) -> X floor=2) & ((floor=2 & doors[2]=FALSE) -> X floor=3) & (floor=3 -> ((X doors[3]=TRUE) | doors[3]=TRUE)))`
- Essentially, if the top floor is requested, we must move up until we are at floor 3 and then we must open the door.

Results: satisfies all specs

Q2: (Q2.smv)

Below is the NuSMV output trace that reaches the goal state. We use the specification "AG (goal=FALSE)" to "trick" NuSMV into generating a path that ends in the goal state.

I used the goal state in the textbook example.

-- specification AG goal = FALSE is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
 -> State: 1.1 <-
  move = u
  h[0] = 1
  h[1] = 2
  h[2] = 3
  h[3] = 1
  h[4] = 2
  h[5] = 3
  h[6] = 1
  h[7] = 2
  h[8] = 3
  v[0] = 3
  v[1] = 3
  v[2] = 3
  v[3] = 2
  v[4] = 2
  v[5] = 2
  v[6] = 1
  v[7] = 1
  v[8] = 1

K = 3
N = 3
goal = FALSE
-> State: 1.2 <-
move = d
-> State: 1.3 <-
v[0] = 2
v[3] = 3
-> State: 1.4 <-
move = r
v[0] = 1
v[6] = 2
-> State: 1.5 <-
h[0] = 2
h[7] = 1
-> State: 1.6 <-
move = u
h[0] = 3
h[8] = 2
-> State: 1.7 <-
v[0] = 2
v[5] = 1
-> State: 1.8 <-
move = l
v[0] = 3
v[2] = 2
-> State: 1.9 <-
h[0] = 2
h[1] = 3
-> State: 1.10 <-
move = d
h[0] = 1
h[3] = 2
-> State: 1.11 <-
v[0] = 2
v[6] = 3
-> State: 1.12 <-
move = r
v[0] = 1
v[7] = 2
-> State: 1.13 <-
h[0] = 2
h[8] = 1
-> State: 1.14 <-

move = u
h[0] = 3
h[5] = 2
-> State: 1.15 <-
v[0] = 2
v[2] = 1
-> State: 1.16 <-
move = l
v[0] = 3
v[1] = 2
-> State: 1.17 <-
h[0] = 2
h[3] = 3
-> State: 1.18 <-
move = d
h[0] = 1
h[6] = 2
-> State: 1.19 <-
v[0] = 2
v[7] = 3
-> State: 1.20 <-
move = r
v[0] = 1
v[8] = 2
-> State: 1.21 <-
h[0] = 2
h[5] = 1
-> State: 1.22 <-
move = u
h[0] = 3
h[2] = 2
-> State: 1.23 <-
v[0] = 2
v[1] = 1
-> State: 1.24 <-
move = l
v[0] = 3
v[3] = 2
-> State: 1.25 <-
h[0] = 2
h[6] = 3
-> State: 1.26 <-
move = d
h[0] = 1

```
  h[7] = 2
-> State: 1.27 <-
  v[0] = 2
  v[8] = 3
-> State: 1.28 <-
  move = r
  v[0] = 1
  v[5] = 2
-> State: 1.29 <-
  h[0] = 2
  h[2] = 1
-> State: 1.30 <-
  move = u
  h[0] = 3
  h[1] = 2
  goal = TRUE
```