**CS 5103 – PROJECT 2**

**Elevator Control System**

**MAY 8, 2017**

**HAIWEN SHI(HMU981)**

**FEIOU ZHANG(GTD813)**

**INTURI SAI HARSHINI(PPG583)**

Contents

[1. Design of the system 3](#_1fob9te)

[1.1 Variables of the elevator system 3](#_3znysh7)

[1.1.1 floor button 3](#_2et92p0)

[1.1.2 up and down button 4](#_tyjcwt)

[1.1.3 door button 4](#_3dy6vkm)

[1.1.4 elevator 4](#_1t3h5sf)

[1.2 Assignment of the elevator system 4](#_4d34og8)

[1.2.1 door 5](#_2s8eyo1)

[1.2.2 Elevator 5](#_17dp8vu)

[1.2.3 Modification 6](#_3rdcrjn)

[2. Verification Results 7](#_26in1rg)

[2.1 Basic model checking 7](#_lnxbz9)

[2.1.1 Requests to be delivered to a particular floor are eventually serviced 7](#_35nkun2)

[2.1.2 The elevator never moves with its doors open 8](#_1ksv4uv)

[2.2 model checking for other variables 8](#_44sinio)

[2.2.1 Direction follows the move command 8](#_2jxsxqh)

[2.2.2 Up and down button will send the elevator to that floor 9](#_z337ya)

[2.2.3 The elevator can still move when no floor button pressed 9](#_3j2qqm3)

[3. Experience Gained 10](#_1y810tw)

[3.1 Advantage 10](#_4i7ojhp)

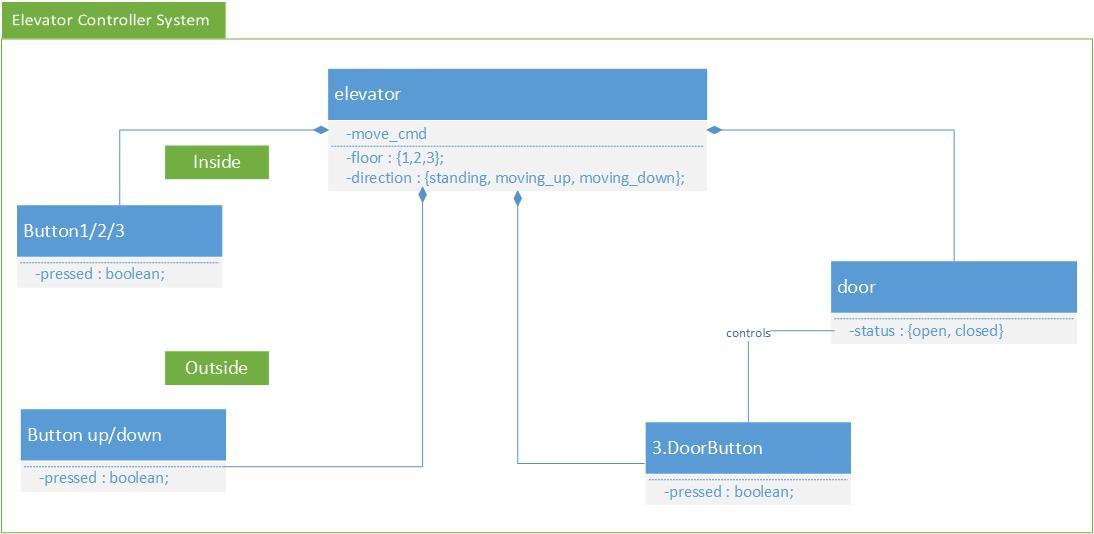
[3.2 Disdvantage 10](#_2xcytpi)

# **1. Design of the system**

## 1.1 Variables of the elevator system

The basic function of the elevator is to move between bottom floor and top floor. There are floor buttons which send the floor request, and up and down buttons which tell the moving direction of the elevator when users are on each floor.

It is assumed that button1/2/3, button up/down, door button are three main variables of the elevator. Cabin has floor and direction variables, which can be changed with the move command. Door has open and closed status, which can be changed by the door command.



### 1.1.1 floor button

There are buttons inside the elevator which send floor request to elevator. Their possible state is boolean which means its status can be pressed or not pressed.

*VAR*

*button1: boolean;*

*button2: boolean;*

*button3: boolean;*



### 1.1.2 up and down button

There are request buttons for each floor outside elevator on floor 1 to floor 3. Their possible state can be pressed or not pressed.

*VAR*

*upF1: boolean;*

*upF2: boolean;*

*downF2: boolean;*

*downF3: boolean;*



### 1.1.3 door button

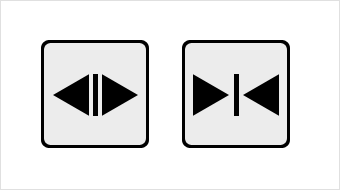
There are request buttons to open or close the door. The possible state can be pressed or not pressed.

To make the project simple , we define door button can be pressed by users either from inside or outside.

*VAR*

*buttonOpen: boolean;*

*buttonClose: boolean;*



### 1.1.4 elevator

The elevator cabin can reach from floor 1 to 3, the first variable should be floor.

Each floor has initial direction of standing and other two possible directions of moving up and down, which is also the second variable.

The elevator receives command from elevator controller to stop, go up and go down. The move command should be another variable of elevator.

The door variable has two status of open and close.

*VAR*

*move\_cmd : {stop, move\_up, move\_down, nop};*

*floor : {1,2,3};*

*direction : {standing, moving\_up, moving\_down};*

*door : {open, closed};*

## 1.2 Assignment of the elevator system

The controller inside the elevator sends move command and door open/close command to elevator cabin. It is necessary to define variable of them, and how the status change by using assignment.

### 1.2.1 door

The door status of the elevator cabin can be open or closed. After it receives the door command, it will change its status by following rules:

Never send any command during the period when the cabin is moving.

If the cabin is stopped and the door is closed, when open door button is pressed, change the door status to open. Or if the door is open and close door button is pressed, change the door status to closed.

If the move request is up or down and the door is open, change the door status to closed to ensure security during the movement.

Otherwise do nothing.

*VAR*

*doorstatus : {open, closed};*

*ASSIGN*

*next(door) := case*

*direction != standing : closed;*

*move\_cmd = stop & door = closed & buttonOpen : open;*

*move\_cmd = stop & door = open & buttonClose : closed;*

*(move\_cmd = move\_up | move\_cmd = move\_down) & door = open : closed;*

*TRUE:closed;*

*esac;*

### 1.2.2 Elevator

The direction of elevator is related to move\_cmd. After the elevator receives command from elevator controller to stop, go up and go down, the next direction will change.

*ASSIGN*

*next(direction) := case*

*move\_cmd = stop : standing;*

*move\_cmd = move\_up : moving\_up;*

*move\_cmd = move\_down : moving\_down;*

*move\_cmd = nop : direction;*

*esac;*

Two kinds of buttons change the value of floor. If button1 is pressed inside the cabin or any user press the upF1 button on the first floor, the floor will eventually go to floor1.

If button2 is pressed inside the cabin or any user press the either upF2 or downF2 button on the second floor, the floor will eventually go to floor2.

If button3 is pressed inside the cabin or any user press the downF3 button on the third floor, the floor will eventually go to floor3.

*ASSIGN*

*next(floor) := case*

*button1 | upF1 : 1;*

*button2 | upF2 | downF2 : 2;*

*button3 | downF3 : 3;*

*TRUE : floor;*

*esac;*

It is the most complex command generated from controller. The move command is determined by which floor the user is on, floor and up/down buttons.

We assume a user is on any floor from 1 to 3. If the corresponding floor button is pressed, for instance someone presses button1 on floor 1, the move command should be stop at the current floor.

If the user is on floor 1, and button2 or button3 is pressed which means he will go up, or any of upF2 | downF2 | downF3 is pressed which means outside user sends go up command to elevator, the move\_cmd will change to move\_up. If on floor2, any of button3 | downF3 is pressed, move\_cmd will come to move\_up. On floor 3, never move up.

If the user is on floor 3, and button1 or button2 is pressed which means he will go down, or any of upF2 | downF2 | upF1 is pressed which means outside user sends go down command to elevator, the move\_cmd will change to move\_down. If on floor2, any of button1 | downF1 is pressed, move\_cmd will come to move\_down. On floor 1, never move down.

*ASSIGN*

*next(move\_cmd) := case*

*door = open : nop;*

*(floor = 1 & button1) | (floor = 2 & button2) | (floor = 3 & button3) : stop;*

*(floor = 1 & ( button2 | button3 | upF2 | downF2 | downF3)) | (floor = 2 & (button3 | downF3)) : move\_up;*

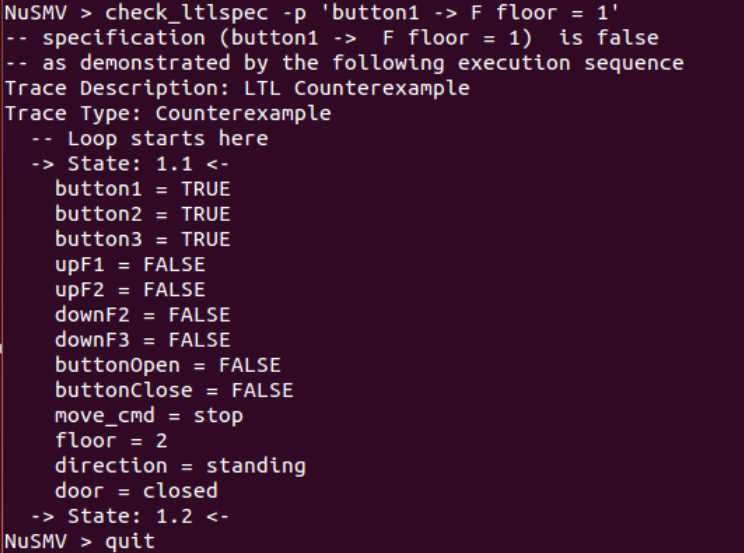
*(floor = 3 & ( button1 | button2 | upF2 | downF2 | upF1)) | (floor = 2 & (button1 | upF1)) : move\_down;*

*TRUE : nop;*

*esac;*

### 1.2.3 Modification

Whenever we test the G(button1 -> F floor = 1) logic specification, it will give us counterexample as follows. It means when button1 and button2 and button3 are pressed at the same time, the elevator never move since the floor command is unclear. So we define limitation to those three buttons that only one of them can be pressed.



*ASSIGN*

*next(button1):= case*

*button1 = TRUE & button2 = FALSE & button3 = FALSE : TRUE;*

*TRUE : FALSE;*

*esac;*

*next(button2):= case*

*button1 = FALSE & button2 = TRUE & button3 = FALSE : TRUE;*

*TRUE : FALSE;*

*esac;*

*next(button3):= case*

*button1 = FALSE & button2 = FALSE & button3 = TRUE : TRUE;*

*TRUE : FALSE;*

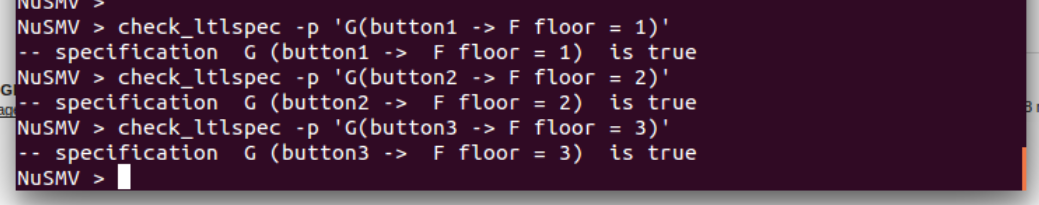
*esac;*

# **2. Verification Results**

## 2.1 Basic model checking

### 2.1.1 Requests to be delivered to a particular floor are eventually serviced

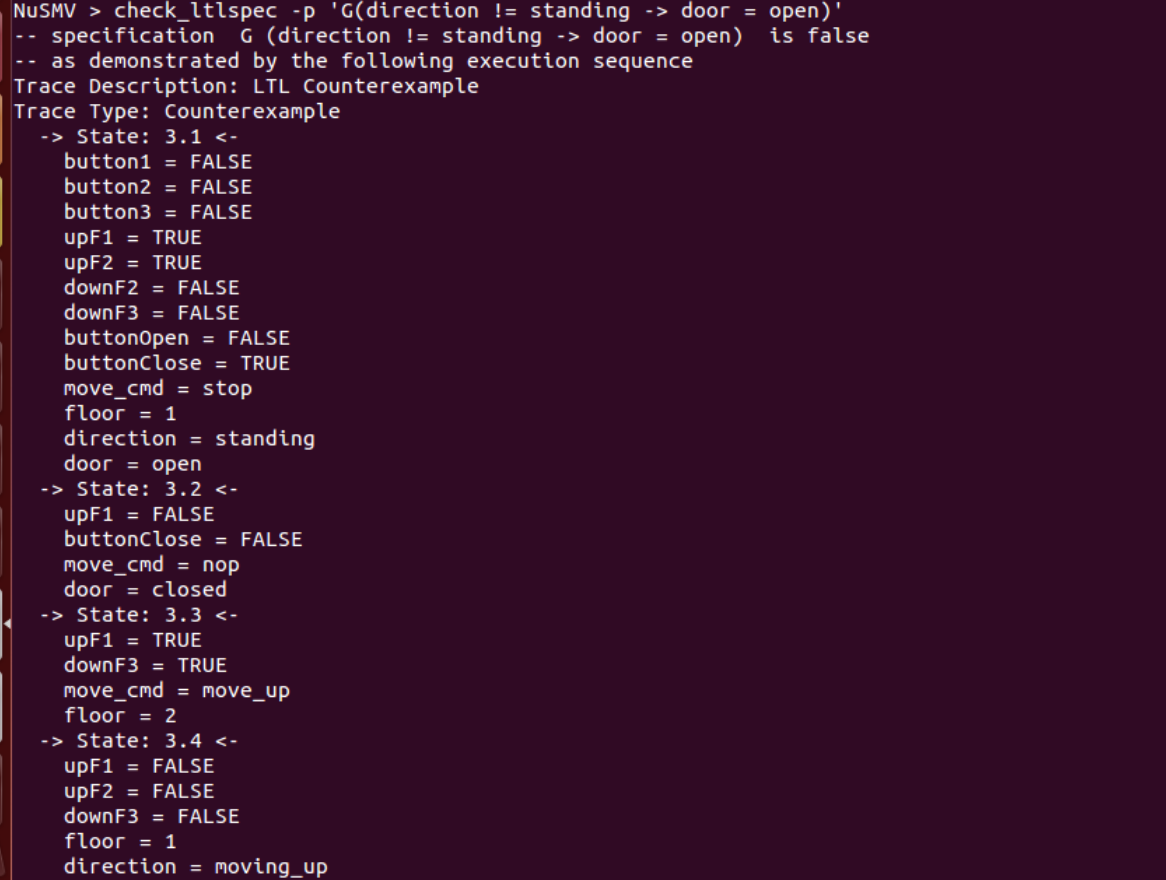
Whenever the user press floor button, the future status could be the corresponding floor. That means the request of the user has been responded to.

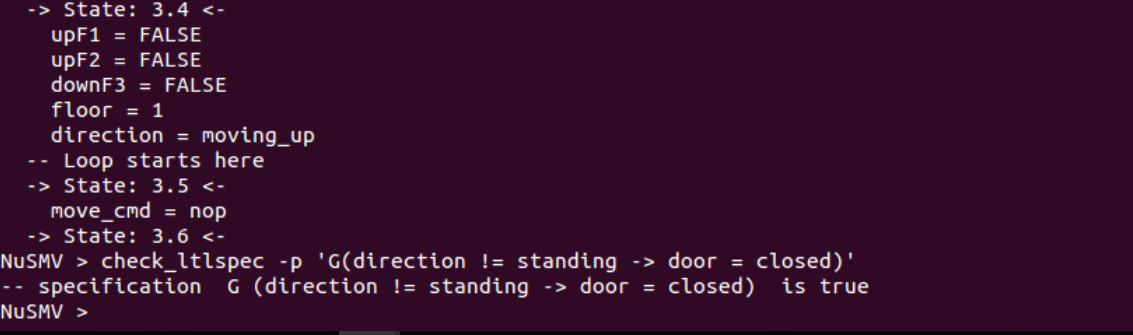


### 2.1.2 The elevator never moves with its doors open

Check the logic specification ‘G (direction != standing -> door = open)’, the result is false. It gives us several counterexample which illustrate the door can’t be open when the cabin is moving.

Check the logic specification ‘G (direction != standing -> door = closed)’, the result is true.



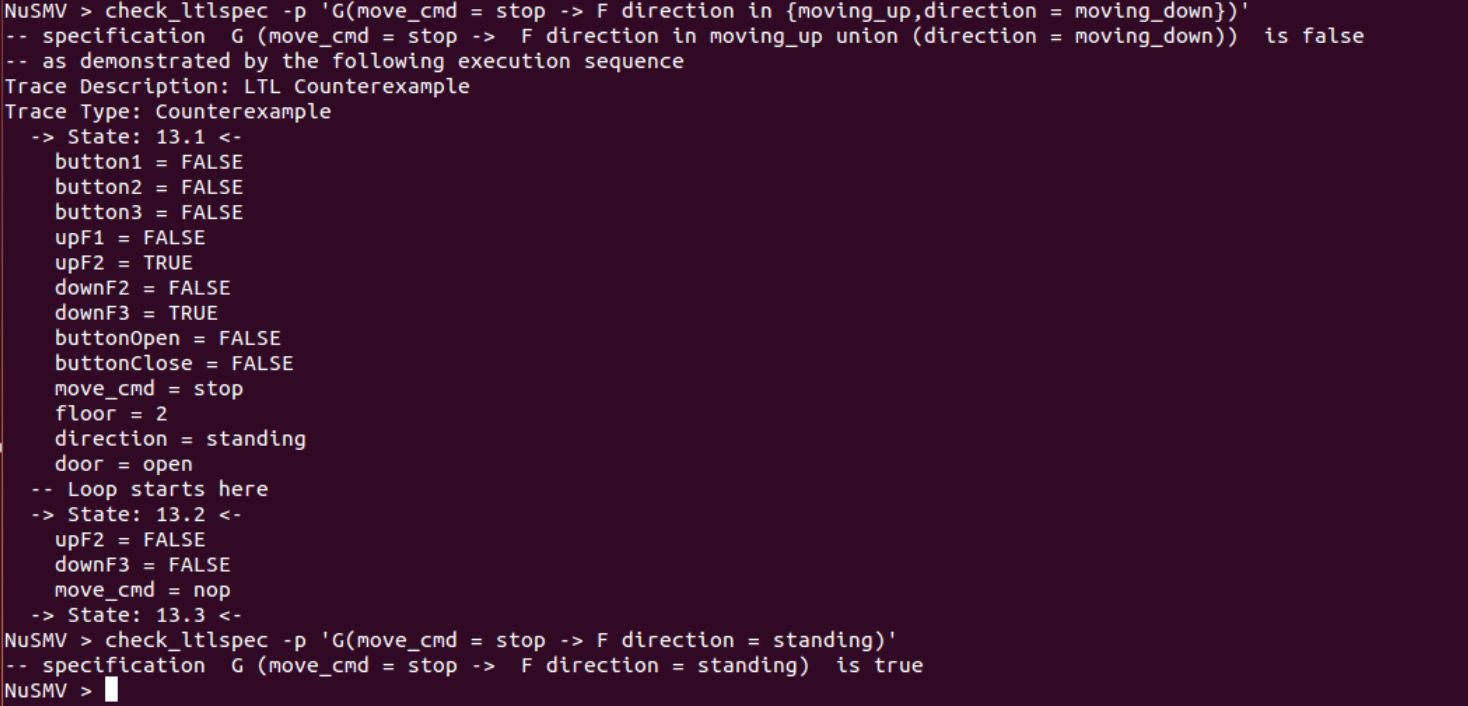


## 2.2 model checking for other variables

### 2.2.1 Direction follows the move command

Check the logic specification ‘G (move\_cmd = stop -> F direction in {moving\_up, moving\_down})’, the result is false, since there is possibility that the cabin does not move.

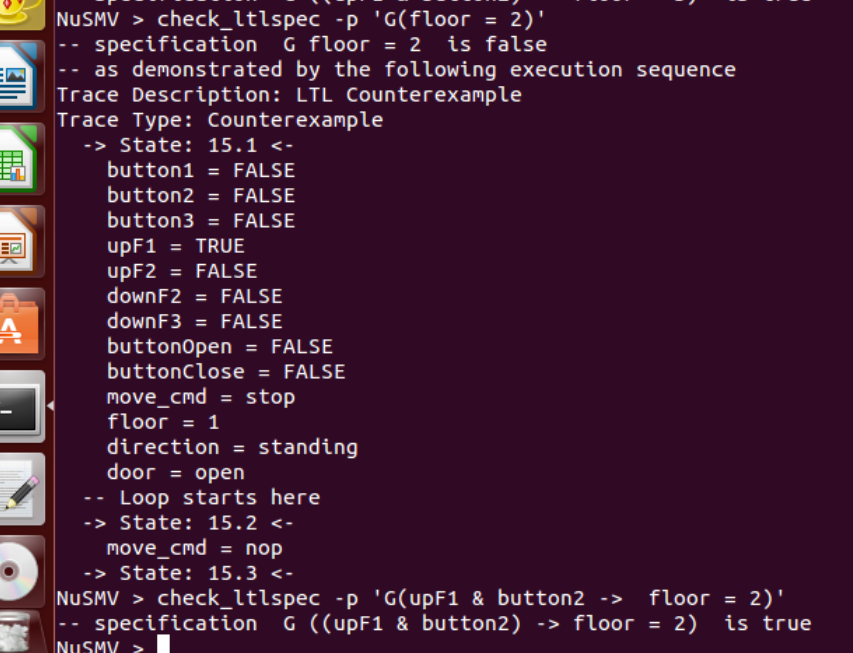
Check the logic specification ‘G (move\_cmd = stop -> F direction = standing)’, the result is true. After the cabin received the stop command, its status will change to standing.



### 2.2.2 Up and down button will send the elevator to that floor

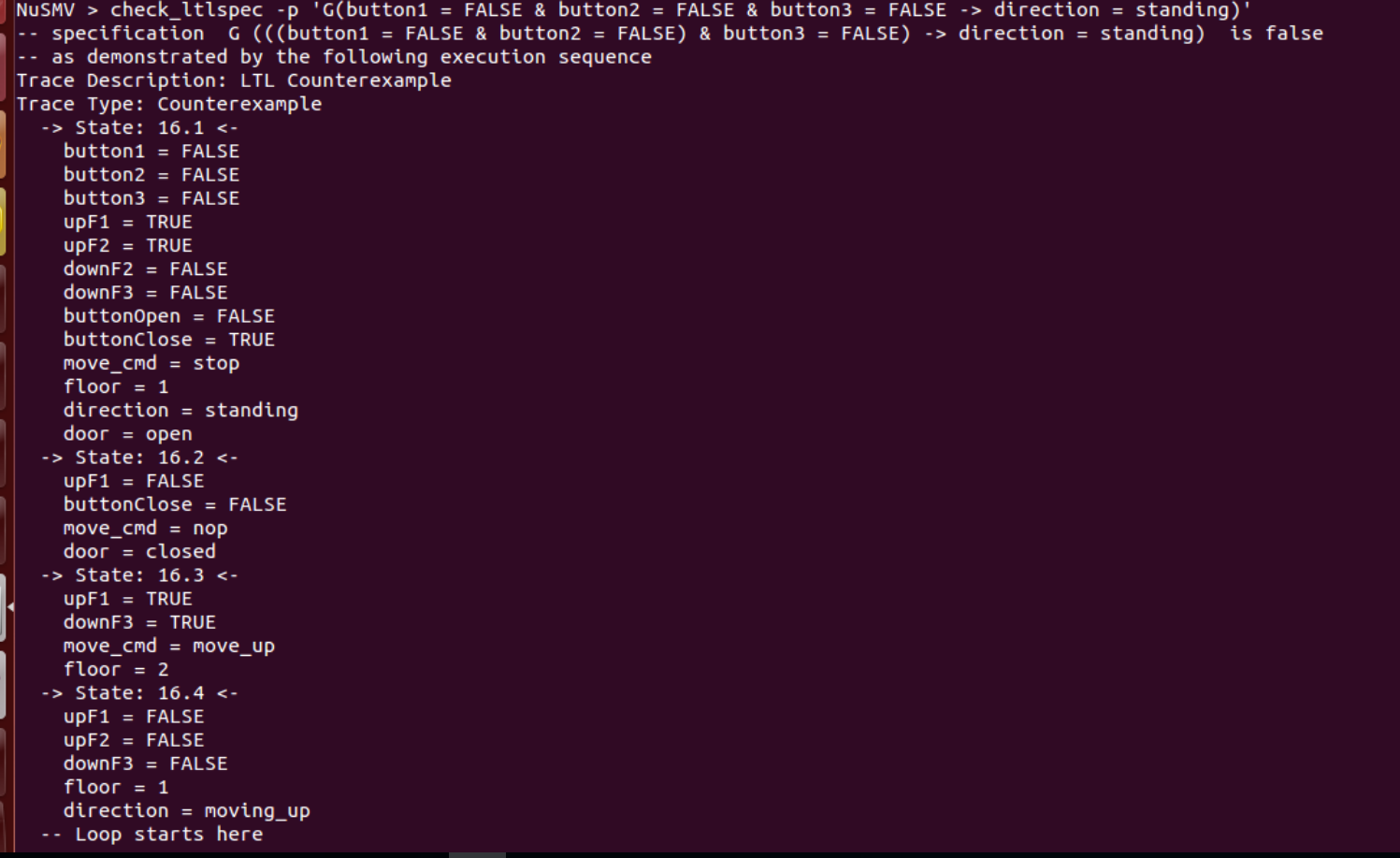
Check ‘G(floor = 2)’ will get false result, since under some condition the elevator will not go to floor 2. But when the upF2 is pressed, the future result of floor = 2 can be true.

Suppose a use enter the elevator on floor 1 and someone press upF2 on second floor, the elevator will eventually go to floor 2.



### 2.2.3 The elevator can still move when no floor button pressed

Even if neither of button1, button2, button3 is impressed, the elevator can still move since someone has pressed the up and down button to send move command to the elevator.



# **3. Experience Gained**

## 3.1 Advantage

This project of elevator control system proved to be a good learning experience in NuSMV. During the project, we are able to know the abilities of NuSMV. It is used to create the model for a software system and do model checking by writing temporal logic specification statements.

The key point of using NuSMV is to define variables and assignments which define how status of variable changes. For variables, like for elevator control system, we need to draw a diagram to make sure which properties are going to change and should be refined as a variable. Then we should judge transition relation, also transition of status of each variable. We can see clearly the result of the logic specification from NuSMV module. If it is not true, we can make modification of our code by checking counterexample listed below.

## 3.2 Disdvantage

When there’s too many variables, it is difficult to tell the initial status. Also the simulate command will not work.

Although NuSMV gives us many possible states, it is hard to tell the whole logic map especially when the model is complex.