L-Università ta' Malta
Faculty of Information & Communication Technology

CPS3233 Assignment Report

Manwel Bugeja

September 3, 2021

Contents

1 Elevator System Specification

In this section, the specifications of the elevator system are expressed in several different formal notations. The notations are Finite State Automata (FSAs), Regular Expressions (RE), Timed Automata (TAs) and Duration Calculus (DC). From the ones listed, TAs and DC are the capable of expressing timed events.

1.1 Finite State Automata

A finite state automata was designed to describe the elevator system. The formal definition of the state machine can be seen in listing 1. The lift can be in one of three states: Idle, Loading or Moving. Idle is when the lift is not moving and has its door closed. Loading when the lift is not moving but has the doors open. Moving is when the lift is moving either up or down. Note that a 'bad' state could be added for when the lift is moving and has the doors open as the lift should never be in this state.

This FSA describes the flow of the lift lifetime in a very basic nature for example it does not consider elevator summoning or floor requests. This can be seen visually in figure 2. A more detailed automaton considering more actions and time constraints is discussed in section 1.3.

Listing 1: FSA definition

```
M = {
{Idle, Loading, Moving},
{openDoor, closeDoor, stop, goUp, goDown},
{
    Idle, openDoor -> Loading,
    Idle, goUp -> Moving,
    Idle, goDown -> Moving,
    Loading, closeDoor -> Idle,
    Moving, stop -> Idle
},
Idle,
{}
}
```

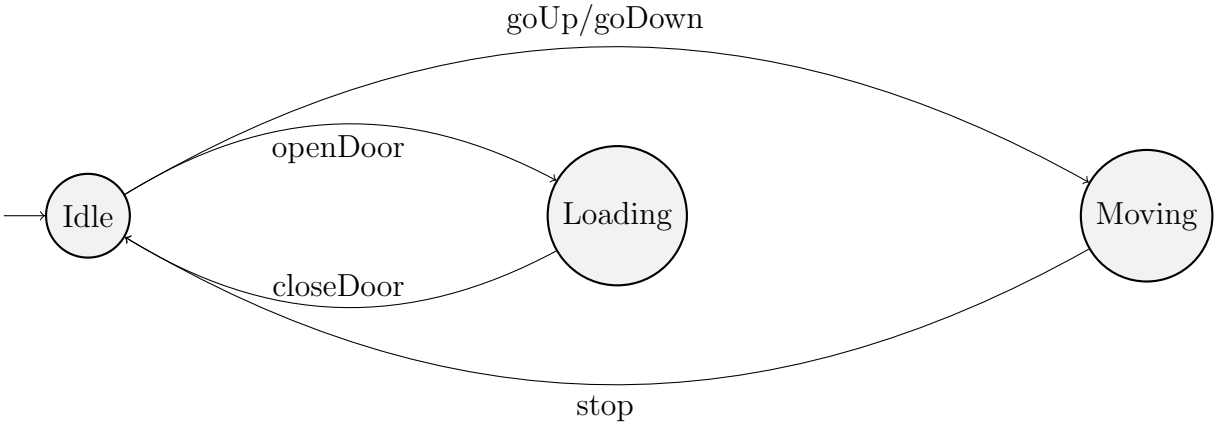

Figure 1: Basic FSA

1.2 Regular Expressions

The lift system was also expressed via regular expressions. This RE can be seen in listing 2. The RE starts of with a series of *openDoor* and *closeDoor* actions which must occur consecutively. This is done so that when the lift moves to the next stage in the RE, the door is always closed, since the lift cannot move with its door open.

Next, the RE verifiers that the lift goes an arbitrary number of steps up or down (depending on what the user decided) followed by a stop. The *stop* is there to make sure the lift is not moving when the progress move back to the open/close door part of the expression.

An example run of the elevator is: `openDoor > closeDoor > goUp > goUp > stop > openDoor > closeDoor > openDoor > closeDoor > goDown > stop`. This run through is correct according to the RE provided.

Listing 2: RE definition

```
((openDoor.closeDoor)* (goUp* + goDown*).stop )*
```

1.3 Timed Automata

For the timed automata, the same state machine above was used with some clocks and states added to verify time-constrained events. In the formal definition of the automaton, the transition functions where modified to accommodate such changes. The formal specification can be seen in listing

3. The transition functions follow the format `[from, to, letter, clocks reseted, condition]`.

The first time property that is handled says that "upon a request, after the door closes, the elevator starts moving in less than 3 seconds". When adapted to the provided automaton, this translates to "when the lift is *Idle* (because a floor button was pressed so the door closed), it goes either up or down in less than 3 seconds". This is verified by clock x.

The second time property expresses that "after the door has been open for 3 seconds, it closes automatically". This implies that when the lift has its doors opened i.e. its in the *Loading* state, it must go to the *Idle* in less than 3 seconds. This is because if a user presses the button in less than 3 seconds, the door will close and if he does not press any button, the door closes automatically. This is verified by clock y.

The two new states are `Requested` and `LoadingRequested`. A new event, `requested` is also added to the alphabet. These new additions concern the summon button present at each floor. The `Requested` state signifies when the lift a floor was requested while the lift was moving or idle. The state `LoadingRequested` is the same except that before the request, the lift was in the `Loading` state.

The state `Requested` was added for the time property that states that the lift must start moving in less than 3 seconds i.e., the time property handles by clock x. Therefore, every transition that leads to the `Requested` state resets clock x. Meanwhile, `LoadingRequested`) is needed for the close-door time property handled by clock y.

Listing 3: TA definition

```
M = {
{openDoor, closeDoor, stop, goUp, goDown},
{Idle, Loading, Moving},
{idle},
{x, y},
{
    [ Idle, Loading, openDoor, {y:=0}, true ]

    [ Idle, Requested, request, {x:=0}, true ]

    [ Loading, Idle, closeDoor, {x:=0}, y<3 ]
```

```
    [Loading, LoadingRequested, request, {}, true]

    [ Moving, Idle, stop, {x:=0}, true ]

    [ LoadingRequested, Requested, closeDoor, {x:=0}, y
      <3 ]

    [ Requested, Moving, goUp/goDown, {}, x<3 ]
}
}
```

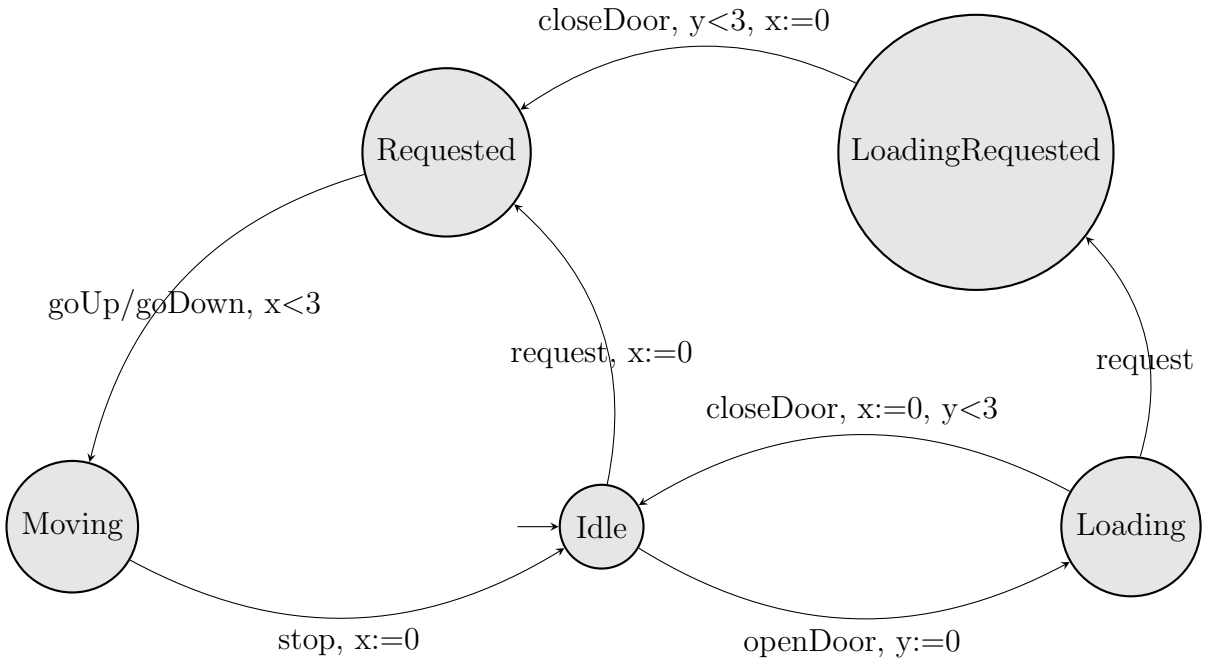

Figure 2: FSA figure

1.4 Duration Calculus

For the DC specifications, two formulas where defined, one for each time constrain. The first time constrain states that: "upon a request, after the door closes, the elevator starts moving in less than 3 seconds". This is expressed by the formula:

$$\square[Idle];[Loading];[Moving] \wedge [idle];[Loading];[Idle] \Rightarrow \int loading < 3$$

The square means that it is checking for all subintervals. Then the pattern the formula is looking for is *Idle*, followed my *Loading*, followed by *Moving*. Then, the expression verifies that the time of loading is less than 3.

The second time constrain says that: after the door has been open for 3 seconds, it closes automatically". The formula describing this is as follows:

$$\square[doorOpen];[\neg doorOpen] \Rightarrow \int doorOpen < 3$$

The formula looks for the pattern of an open door followed by a closed door and then verifies that the door was open for less than 3 seconds.

2 Runtime Verification

For runtime verification, the Larva tool was used. The automata applied where directly derived from the timed automaton discussed in section 1.3. However, it was not implemented as a single automaton in order to check for properties individually.

The first property investigated was the door close property. First off, to make the investigation simpler, a single lift was considered. This is because the automaton designed only considers one lift. The Java code was analysed to find the methods that correspond with the designed events. These methods were present in *Lift.java*. The methods and their corresponding events is shown in table 1.

Method	Event
Lift.setMoving(boolean isMoving)	goUp/goDown *
Lift.closeDoors()	closeDoor
Lift.openDoors()	openDoor
Lift.setFloor()	stop

Table 1: LiftOpenTimeProperty events table

[1]

[1]*isMoving parameter must be true