



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

## CPS3233 Assignment Report

Manwel Bugeja

September 7, 2021

### Contents

<b>1</b>	<b>Elevator System Specification</b>	<b>2</b>
1.1	Finite State Automata . . . . .	2
1.2	Regular Expressions . . . . .	3
1.3	Timed Automata . . . . .	3
1.4	Duration Calculus . . . . .	5
<b>2</b>	<b>Runtime Verification</b>	<b>6</b>
2.1	Testing . . . . .	7
2.2	Larva script improvements . . . . .	8
<b>3</b>	<b>Model-Based Testing</b>	<b>9</b>
<b>4</b>	<b>Runtime Verification and Testing</b>	<b>9</b>

# 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 ,
  {}
}
```

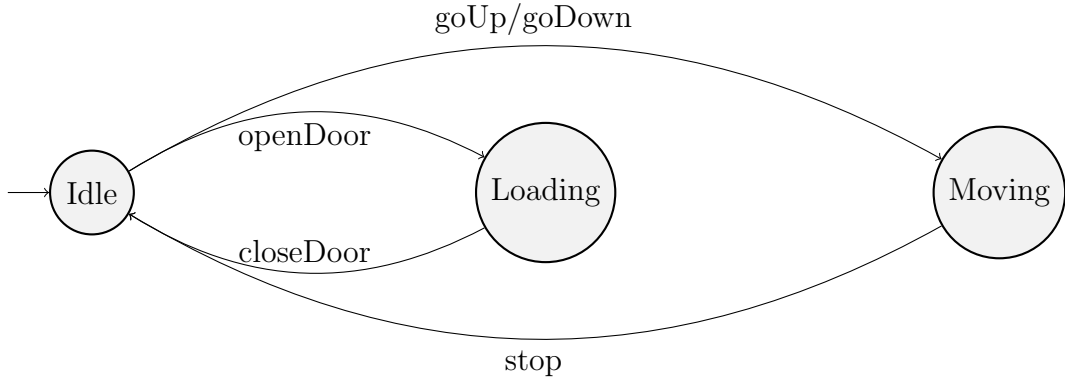


Figure 1: Basic FSA

## 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 verifies 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 (figure ??, 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 ]
}
}

```

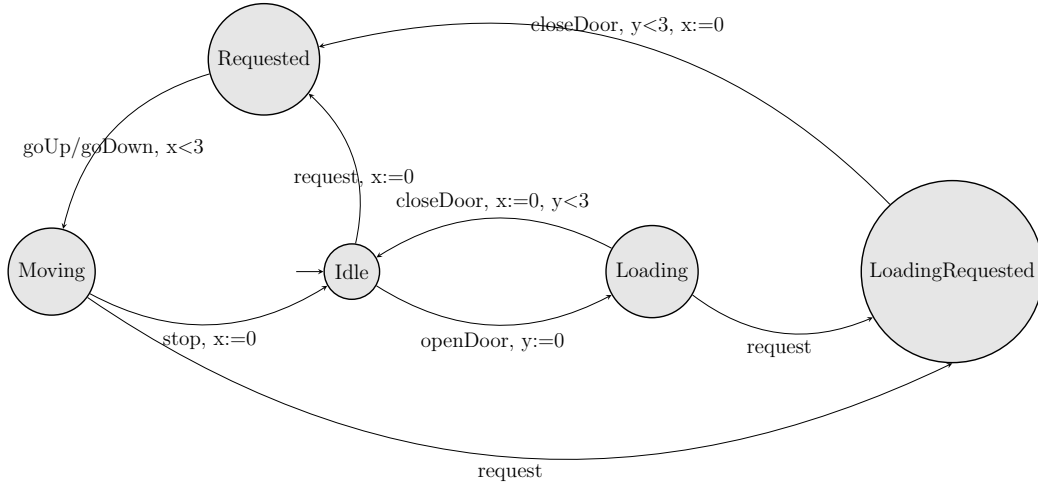


Figure 2: FSA figure

## 1.4 Duration Calculus

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

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

The square means that it is checking for all subintervals. Then the pattern the formula is looking for is *Idle*, followed by *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:

$$\Box[\text{doorOpen}]; [\neg\text{doorOpen}] \Rightarrow \int \text{doorOpen} < 3$$

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

To add support for multiple lift, the **FOREACH** Larva keyword was used to iterate over lift objects. All the event methods resided in the class *Lift.java*. This meant that adding multiple lift support was done by binding the Lift object in the event declarations and then assigning it to the **FOREACH** object using the **WHERE** keyword.

Listing 4: Multiple lift support for LiftOpenProperty

---

<sup>1</sup>\*isMoving parameter must be true

<pre> setMoving(isMoving) = {Lift l.setMoving(boolean     isMoving)} WHERE {lift = l;} closeDoors() = {Lift l.closeDoors()} WHERE {lift = l;} openDoors() = {Lift l.openDoors()} WHERE {lift = l;} setFloor() = {Lift l.setFloor(*)} WHERE {lift = l;} </pre>
---

The second property is called **StartMovingTimeProperty** and verifies that the lift start moving in less than 3 seconds after the door closes following a request. A new event was added for the summon request action. The method that corresponds to this event is **callLiftToFloor**.

This method provided some difficulty when switching to adding support for multiple lift verification. The reason being that this method does not reside in the *Lift.java* class. It resided in the *JavaController.java* class. When **callLiftToFloor** is called, it performs some calculations and then proceeds to execute **moveLift** with the chosen lift. To find out which lift was called via Larva, the source needed to be edited. The change done was very minimal and does not impact the design of the lift system at all. The **callLiftToFloor** method signature was changed to return a lift object instead of void. This is then used by Larva by making use of the **uponReturning** keyword when declaring the event. The whole event declaration is as follows:

```
callLiftToFloor(Lift l) = *callLiftToFloor(*)uponReturning(l) where
lift = l;, where 'lift' is the iterated object declared in the FOREACH
loop.
```

## 2.1 Testing

In the tests shown, one property at a time was tested to make the outputs more legible. Figure 3 shows the property **LiftOpenTimeProperty** running as should. several buttons were pressed and the lift always moved as specified.

For the second test shown in figure 4, time taken to close the door automatically was changed from the java source code in file *Shaft.java*. The figure shows that verification failed because the door was closing after 3 seconds.

Referirng to figure 5 the third test shown, displays the verification being conducted on several (three) lifts. Identifiers are used to differentiate the multiple lifts being verified.

Figure 6 shows the fourth documented test. This test shows the verifi-

```

1 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
2 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
3 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE:> Moving
4 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
5 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
6 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE:> Idle
7 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
8 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
9 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
10 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.gui.Shaft.openDoors() TO STATE:> Loading
11 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Loading
12 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.gui.Shaft.closeDoors() TO STATE:> Idle
13 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
14 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
15 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
16 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE:> Moving
17 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
18 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
19 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE:> Idle
20 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
21 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
22 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
23 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
24 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.gui.Shaft.openDoors() TO STATE:> Loading
25 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Loading
26 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.gui.Shaft.closeDoors() TO STATE:> Idle
27 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
28 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
29 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Idle
30 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE:> Moving
31 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
32 [LiftOpenTimeProperty]AUTOMATON:> LiftOpenTimeProperty() STATE:>Moving
33 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE:> Idle

```

Figure 3: LiftOpenTimeProperty verified

cation of the `StartMovingTimeProperty` working on more than one lift. To make the output more clear, the `LiftOpenTimeProperty` was removed from the script to focus only on the property discussed.

A failure was also induced for this property. The `animateLift` method in `Shaft.java` was edited to wait three seconds before starting to move the lift. The added code can be seen in figure 7. As expected, this caused the verification to fail. The output can be seen in figure 8

The final test showcased in this section can be seen in figure 9. This shows all the properties being run at once for several lifts.

## 2.2 Larva script improvements

The script developed is not perfect. One flaw with the automaton designed is viewable in the `StartMovingTimeProperty` property. Taking a look at the automaton which describes this property (figure ??). The automaton does not consider a summon request when the lift is moving. The reason behind this decision is the question 'what state would the flow go to?'. If the flow would go to the `Requested` or `LoadingRequested`

Another flaw in the design is



```

1 JTOMATON::> LiftOpenTimeProperty() STATE::>Idle
2 JTOMATON::> LiftOpenTimeProperty() STATE::>Idle
3 JVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE::> Moving
4 JTOMATON::> LiftOpenTimeProperty() STATE::>Moving
5 JTOMATON::> LiftOpenTimeProperty() STATE::>Moving
6 JVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE::> Idle
7 JTOMATON::> LiftOpenTimeProperty() STATE::>Idle
8 JTOMATON::> LiftOpenTimeProperty() STATE::>Idle
9 JTOMATON::> LiftOpenTimeProperty() STATE::>Idle
10 JVED ON METHODCALL: void com.liftmania.gui.Shaft.openDoors() TO STATE::> Loading
11 JTOMATON::> LiftOpenTimeProperty() STATE::>Loading
12 JVED ON METHODCALL: void com.liftmania.gui.Shaft.closeDoors() TO STATE::> !!!SYSTEM REACHED BAD STATE!!! OpenTooLong
13 before$aspects__asp_lift0$6$ed143934(_asp_lift0.aj:58)
14. animateLift(Shaft.java:182)
15.run(Shaft.java:288)
16.ead.java:748)
17 JTOMATON::> LiftOpenTimeProperty() STATE::>OpenTooLong
18

```

Figure 4: LiftOpenTimeProperty failed

### 3 Model-Based Testing

### 4 Runtime Verification and Testing

### References



```

153 public void animateLift(int toFloor) {
154
155     //Wait three seconds after call before moving
156     // (should cause verification)
157     try {
158         Thread.sleep(3000);
159     } catch (Exception e) {}
160
161     //Update lift state
162     lift.setMoving(true);
163
164     int fromFloor = lift.getFloor();
165     setLiftFloor(fromFloor);
166     lift.setMoving(true);
167
168     if (toFloor > fromFloor) {
169         for (int i = fromFloor; i < toFloor; i++) {
170             animateUp(i);
171             lift.setFloor(i);
172         }
173     }

```

Figure 7: Added code to stop for three seconds before starting to move lift in *Shaft.java*

```

1ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>Idle
2ingTimeProperty]MOVED ON METHODCALL: Lift com.liftmania.LiftController.callLiftToFloor(int) TO STATE::> Requested
3ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>Requested
4lift took too long to start moving
5ingTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE::> !!!SYSTEM REACHED BAD STATE!!! BadMoving
6asp_lift1.ajc$before$aspects__asp_lift1$2$74a464b4(_asp_lift1.aj:25)
7ania.gui.Shaft.animateLift(Shaft.java:162)
8ania.gui.Shaft.run(Shaft.java:295)
9.Thread.run(Thread.java:748)
10ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
11ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
12ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
13ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
14ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
15ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
16ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
17ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
18ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
19ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
20ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
21ingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@4c4251d4 ) STATE::>BadMoving
22

```

Figure 8: StartMovingTimeProperty failure

```

1 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
2 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
3 [StartMovingTimeProperty]MOVED ON METHODCALL: Lift com.liftmania.LiftController.callLiftToFloor(int) TO STATE::> Requested
4 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
5 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE::> Moving
6 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Requested
7 [StartMovingTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setMoving(boolean) TO STATE::> Moving
8 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Moving
9 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Moving
10 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Moving
11 [LiftOpenTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE::> Idle
12 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
13 [StartMovingTimeProperty]MOVED ON METHODCALL: void com.liftmania.Lift.setFloor(int) TO STATE::> Idle
14 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
15 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
16 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
17 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
18 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
19 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
20 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
21 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
22 [LiftOpenTimeProperty]AUTOMATON::> LiftOpenTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle
23 [StartMovingTimeProperty]AUTOMATON::> StartMovingTimeProperty(com.liftmania.Lift@e3a71a4 ) STATE::>Idle

```

Figure 9: All properties