```
1  open util/ordering[Floor]
2  open util/boolean
3  open util/integer
4
5  sig Floor {}
6  abstract sig Direction {}
7  one sig Up, Down extends Direction {}
8
9  abstract sig Door{}
10 one sig Open, Close extends Door {}
11
12 conc state ElevatorSystem {
13
14 env request: Floor -> lone Direction
15
16 conc state Controller {
17
18   trans sendUpReq {
19
20     when some request
21     do {
22       one e1: evelPID | one req: request |
23
24       ((Floo.req = Up and Elevator[e1]/direction = Up and
25       lt(Elevator[e1]/current, req.direction)
26       and
27       (no e2: evelPID | e2/direction = Up
28       and gt(Elevator[e1]/current, Elevator[e2]/current)
29       and lt(Elevator[e2]/current, req.direction))
30       =>
31       (Elevator[e1]/called' = Elevator[e1]/called + req)
32     }
33   }
34
35   trans sendDownReq{
36       do {
37       one e1: evelPID | one req: request |
38
39       ((Floo.req = Down and Elevator[e1]/direction = Down
      and
40       gt(Elevator[e1]/current, req.direction)
41       and
42       (no e2: evelPID | e2/direction = Up
43       and lt(Elevator[e1]/current, Elevator[e2]/current)
44       and gt(Elevator[e2]/current, req.direction))
45       =>
46       (Elevator[e1]/called' = Elevator[e1]/called + req)
47     }
48   }
49 }
```

```
50
51
52  conc state [p: elevPID] Elevator {
53      direction: one Direction
54      door: one Door
55      called: Floor -> Direction
56      current: set Floor
57
58      action NotArriving[ (current' -> direction) not in
        called' ] {}
59      action WaitingToArrive[{called - (current' ->
        direction)} in called'] {}
60      action OpenDoor [(current' -> direction) in called
        implies door' = Open] {}
61
62      init {
63          no called
64          direction = Down
65          current = max[Floor]
66          door = Close
67      }
68
69      state MovingUp {
70        //Move to the next floor
71        trans nextFloor {
72          when {
73            some called
74            door = Close
75            direction = Up
76            some nexts[current] & called.direction
77            !((current' -> direction) in called)
78          }
79          do {
80            current' = min[(nexts[current]
81            & called.direction)]
82            NotArriving
83          }
84        }
85
86        //Change state to moving down
87        trans ChangeDirToDown {
88            when {
89                some called
90                direction = Up
91                door = Close
92                no nexts[current] & called.direction
93            }
94            do {
95                direction' = Down
96                NotArriving
```

```
97            }
98            goto MovingDown
99        }
100
101       //Change state to destination
102       trans StopMoving {
103         when (current' -> direction) in called
104         goto OnDestinationFloor
105       }
106     }
107
108     state MovingDown {
109
110       //Move to the next floor
111       trans nextFloor {
112         when {
113           some called
114           door = Close
115           direction = Down
116           some nexts[current] & called.direction
117           !((current' -> direction) in called)
118         }
119         do {
120           current' = min[(nexts[current]
121           & called.direction)]
122           NotArriving
123         }
124       }
125
126       //Change state to moving down
127       trans ChangeDirToUp {
128           when {
129               some called
130               direction = Down
131               door = Close
132               no nexts[current] & called.direction
133           }
134           do {
135               direction' = Up
136               NotArriving
137           }
138           goto MovingUp
139       }
140
141       //Change state to destination
142       trans StopMoving {
143         when (current' -> direction) in called
144         goto OnDestinationFloor
145       }
146     }
```

```
147
148       state OnDestinationFloor {
149         //Open the door for passengers
150         //Remove current floor from called list
151         trans OpenDoor {
152           when door = Close
153           do {
154             WaitingToArrive
155             door' = Open
156           }
157           goto ContinueMoving
158         }
159
160         //Close the door and
161         //move to next up request
162         trans ContinueMovingUp {
163           when {
164             door = Open
165             direction = Up
166           }
167           do door' = Close
168           goto MoveUp
169         }
170
171          //Close the door and
172         //move to next request
173         trans ContinueMovingDown {
174           when {
175             door = Open
176             direction = Down
177           }
178           do door' = Close
179           goto MoveDown
180         }
181
182         //Go to idle if no more calls
183         trans GotoIdle
184         {
185           when no called
186           goto Idle
187         }
188       }
189
190       state Idle{
191         trans defaultToGround{
192           when {
193             no called
194             min[Floor] not in current
195           }
196           do{
```

```
197             current' = min[Floor]
198             direction' = Down
199           }
200         }
201
202         trans RemainIdle{
203           when {
204             no called
205             min[Floor] in current
206           }
207           do {
208             direction' = direction
209           }
210         }
211
212         trans StartMoving {
213           when some called
214           goto MoveUp
215         }
216       }
217
218   }
219   }
220
221
222   //PROPERTIES TO CHECK
223
224   //Door cannot be in an Open state if an elector is not at
          a requested floor
225
226   assert safeDoor{
227     ctl_mc[ag[all e: elevPid | !(Elevator[p]/current in
        Elevator[p]/called) => Elevator[p]/door = Closed]]
228   }
229
230   //Every elevator must eventually reach a requested floor
231
232   assert completeRequest{
233     ctl_mc[af[all e: elevPid | some Elevator[p]/called => (
        Elevator[p]/current in Elevator[p]/called)]]
234   }
235
236   //Some elevators must eventually have responded to every
        request in its list
237
238   assert emptyList{
239     ctl_mc[af[some e: elevPid | no Elevator[p]/called]]
240   }
241
242   //All elevators will have one current floor
```

```
243
244  assert emptyList{
245    ag(all e: elevPid | one Elevator[p]/current)
246  }
```