```
1
2  /*
3  MIT License
4
5  Copyright (c) [2021] [Tamjid Hossain]
6
7  Permission is hereby granted, free of charge, to any
       person obtaining a copy
8  of this software and associated documentation files (the
       "Software"), to deal
9  in the Software without restriction, including without
       limitation the rights
10 to use, copy, modify, merge, publish, distribute,
       sublicense, and/or sell
11 copies of the Software, and to permit persons to whom the
        Software is
12 furnished to do so, subject to the following conditions:
13
14 The above copyright notice and this permission notice
       shall be included in all
15 copies or substantial portions of the Software.
16
17 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY
        KIND, EXPRESS OR
18 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
       MERCHANTABILITY,
19 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN
       NO EVENT SHALL THE
20 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
       DAMAGES OR OTHER
21 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
       OTHERWISE, ARISING FROM,
22 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
       OTHER DEALINGS IN THE
23 SOFTWARE. */
24
25 open util/ordering[Floor]
26 open util/boolean
27 open util/integer
28
29 sig Floor {}
30 abstract sig Direction {}
31 one sig Up, Down extends Direction {}
32
33 abstract sig Door{}
34 one sig Open, Close extends Door {}
35
36 conc state ElevatorSystem {
37
38 env request: Floor -> lone Direction
```

```
39
40  conc state Controller {
41
42    trans sendUpReq {
43
44      when some request
45      do {
46        one e1: evelPID | one req: request |
47
48        ((Floo.req = Up and Elevator[e1]/direction = Up and
49        lt(Elevator[e1]/current, req.direction)
50        and
51        (no e2: evelPID | e2/direction = Up
52        and gt(Elevator[e1]/current, Elevator[e2]/current)
53        and lt(Elevator[e2]/current, req.direction))
54        =>
55        (Elevator[e1]/called' = Elevator[e1]/called + req)
56      }
57    }
58
59    trans sendDownReq{
60        do {
61        one e1: evelPID | one req: request |
62
63        ((Floo.req = Down and Elevator[e1]/direction = Down
       and
64        gt(Elevator[e1]/current, req.direction)
65        and
66        (no e2: evelPID | e2/direction = Up
67        and lt(Elevator[e1]/current, Elevator[e2]/current)
68        and gt(Elevator[e2]/current, req.direction))
69        =>
70        (Elevator[e1]/called' = Elevator[e1]/called + req)
71      }
72    }
73  }
74
75
76  conc state [p: elevPID] Elevator {
77      direction: one Direction
78      door: one Door
79      called: Floor -> Direction
80      current: set Floor
81
82      action NotArriving[ (current' -> direction) not in
       called' ] {}
83      action WaitingToArrive[{called - (current' ->
       direction)} in called'] {}
84      action OpenDoor [(current' -> direction) in called
       implies door' = Open] {}
```

```
85
86      init {
87          no called
88          direction = Down
89          current = max[Floor]
90          door = Close
91      }
92
93      state MovingUp {
94        //Move to the next floor
95        trans nextFloor {
96          when {
97            some called
98            door = Close
99            direction = Up
100           some nexts[current] & called.direction
101           !((current' -> direction) in called)
102         }
103         do {
104           current' = min[(nexts[current]
105           & called.direction)]
106           NotArriving
107         }
108       }
109
110       //Change state to moving down
111       trans ChangeDirToDown {
112           when {
113               some called
114               direction = Up
115               door = Close
116               no nexts[current] & called.direction
117           }
118           do {
119               direction' = Down
120               NotArriving
121           }
122           goto MovingDown
123       }
124
125       //Change state to destination
126       trans StopMoving {
127         when (current' -> direction) in called
128         goto OnDestinationFloor
129       }
130     }
131
132     state MovingDown {
133
134       //Move to the next floor
```

```
135         trans nextFloor {
136           when {
137             some called
138             door = Close
139             direction = Down
140             some nexts[current] & called.direction
141             !((current' -> direction) in called)
142           }
143           do {
144             current' = min[(nexts[current]
145             & called.direction)]
146             NotArriving
147           }
148         }
149
150         //Change state to moving down
151         trans ChangeDirToUp {
152             when {
153                 some called
154                 direction = Down
155                 door = Close
156                 no nexts[current] & called.direction
157             }
158             do {
159                 direction' = Up
160                 NotArriving
161             }
162             goto MovingUp
163         }
164
165         //Change state to destination
166         trans StopMoving {
167           when (current' -> direction) in called
168           goto OnDestinationFloor
169         }
170       }
171
172       state OnDestinationFloor {
173         //Open the door for passengers
174         //Remove current floor from called list
175         trans OpenDoor {
176           when door = Close
177           do {
178             WaitingToArrive
179             door' = Open
180           }
181           goto ContinueMoving
182         }
183
184         //Close the door and
```

```
185         //move to next up request
186         trans ContinueMovingUp {
187           when {
188             door = Open
189             direction = Up
190           }
191           do door' = Close
192           goto MoveUp
193         }
194
195          //Close the door and
196         //move to next request
197         trans ContinueMovingDown {
198           when {
199             door = Open
200             direction = Down
201           }
202           do door' = Close
203           goto MoveDown
204         }
205
206         //Go to idle if no more calls
207         trans GotoIdle
208         {
209           when no called
210           goto Idle
211         }
212       }
213
214       state Idle{
215         trans defaultToGround{
216           when {
217             no called
218             min[Floor] not in current
219           }
220           do{
221             current' = min[Floor]
222             direction' = Down
223           }
224         }
225
226         trans RemainIdle{
227           when {
228             no called
229             min[Floor] in current
230           }
231           do {
232             direction' = direction
233           }
234         }
```

```
235
236         trans StartMoving {
237           when some called
238           goto MoveUp
239         }
240       }
241
242  }
243  }
244
245
246  //PROPERTIES TO CHECK
247
248  //Door cannot be in an Open state if an elector is not at
         a requested floor
249
250  assert safeDoor{
251    ctl_mc[ag[all e: elevPid | !(Elevator[p]/current in
       Elevator[p]/called) => Elevator[p]/door = Closed]]
252  }
253
254  //Every elevator must eventually reach a requested floor
255
256  assert completeRequest{
257    ctl_mc[af[all e: elevPid | some Elevator[p]/called => (
       Elevator[p]/current in Elevator[p]/called)]]
258  }
259
260  //Some elevators must eventually have responded to every
       request in its list
261
262  assert emptyList{
263    ctl_mc[af[some e: elevPid | no Elevator[p]/called]]
264  }
265
266  //All elevators will have one current floor
267
268  assert emptyList{
269    ag(all e: elevPid | one Elevator[p]/current)
270  }
```