گزارش پروژه پایانی

کامیار رفعتی ۸٤۲۰۳٦۹٤
rafati@ce.sharif.edu
دانشگاه صنعتی شریف

نعیم اصفهانی ۸٤۲۰۱۰۰۳
esfahani@ce.sharif.edu
دانشگاه صنعتی شریف

درس درستی‌یابی سیستم‌های واکنشی
مدرس: دکتر موقر

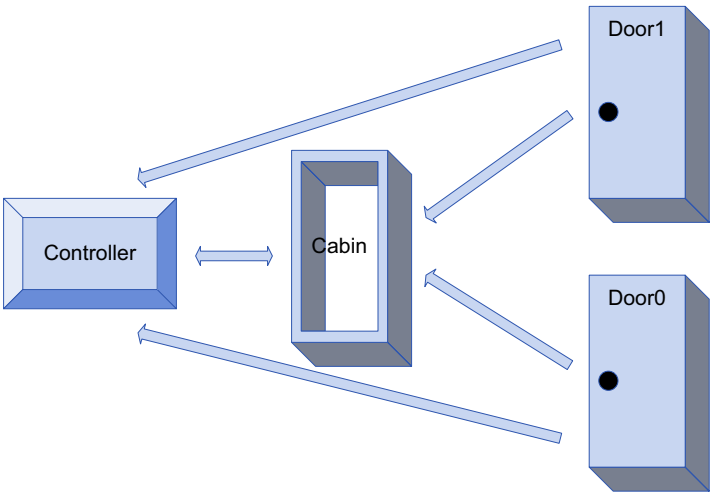
مقدمه:

در این پروژه سیستم کنترل آسانسور مدل شده است. در بخش اول پروژه یک آسانسور در یک ساختمان دو طبقه مدل شده است و در بخش دوم دو آسانسور در یک ساختمان چهار طبقه مدل شده اند که با یکدیگر همزمان هستند و با همکاری یکدیگر به درخواست‌ها پاسخ می‌دهند.

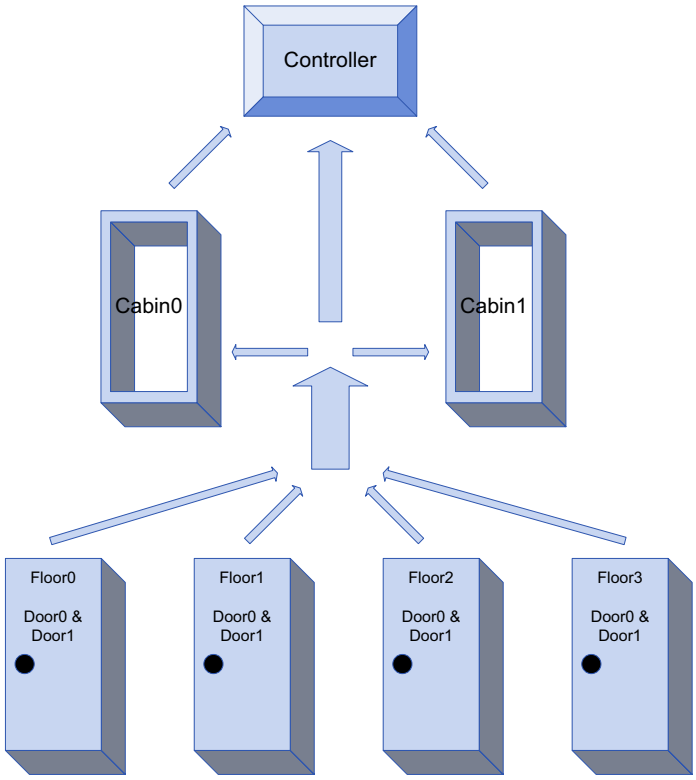
برای مدلسازی از زبان SMV استفاده شده است. همچنین از ابزار NuSMV برای درستی‌یابی استفاده شده است. ابزار SMV ، یک ابزار درستی‌یابی است که برای بررسی سیستم‌های با تعداد حالت محدود استفاده می‌شود. این ابزار از الگوریتم درستی‌یابی نمادین بر مبنای OBDD استفاده می‌کند.

شرح مدل:

مدل ارائه شده برای قسمت اول در شکل زیر آمده است:



مدل ارائه شده برای قسمت دوم نیز به صورت زیر است:

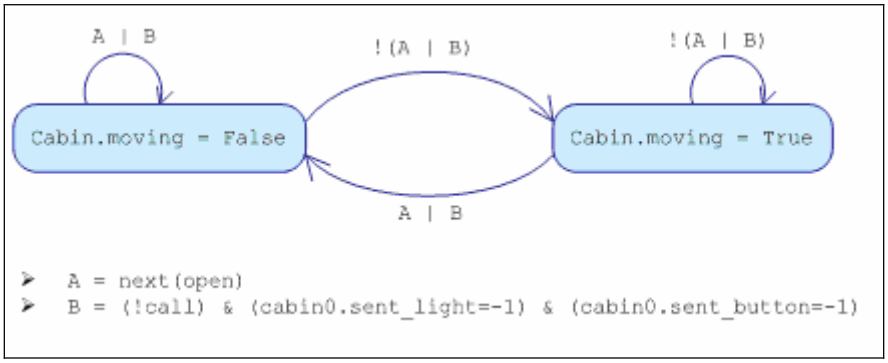


کد هر دو قسمت در پایان این گزارش آمده است.

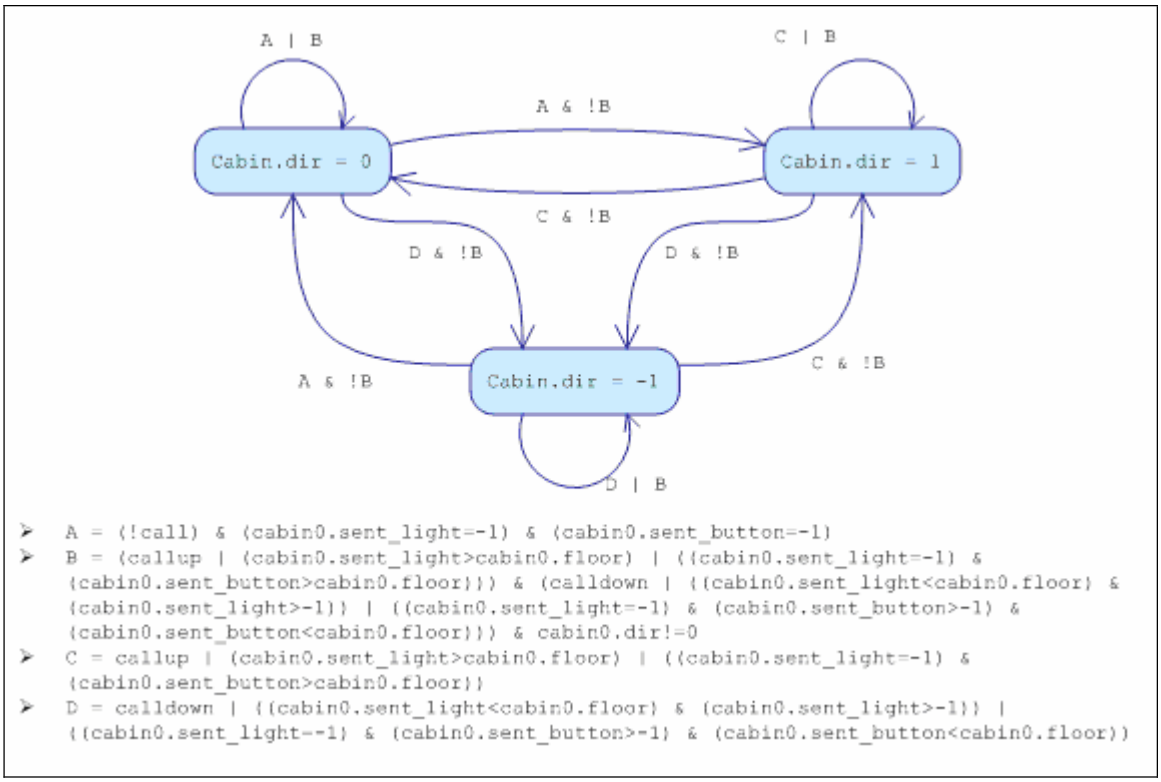
ماشین حالات کنترل کننده‌ها(Controller):

در این قسمت ماشین حالات هر یک از کنترل کننده‌ها را می‌بینیم. توجه کنید که به منظور جلوگیری از بزرگ شدن آنها ، برای هر یک از متغیرهای کنترلی جداگانه رسم شده است و در حقیقت ماشین اصلی از ضرب این نمودارهای حالت به دست می‌آید.

۱. بخش اول :



نمودار فوق مربوط به متغیر `cabin.moving` است که بیانگر این است که آیا کابین حرکت می‌کند یا خیر.

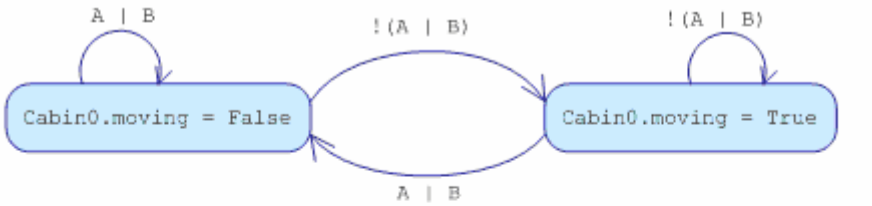


- A = (!call) & (cabin0.sent_light=-1) & (cabin0.sent_button=-1)
- B = (callup | (cabin0.sent_light>cabin0.floor) | ((cabin0.sent_light=-1) & (cabin0.sent_button>cabin0.floor))) & (calldown | ((cabin0.sent_light<cabin0.floor) & (cabin0.sent_light>-1)) | ((cabin0.sent_light=-1) & (cabin0.sent_button>-1) & (cabin0.sent_button<cabin0.floor))) & cabin0.dir!=0
- C = callup | (cabin0.sent_light>cabin0.floor) | ((cabin0.sent_light=-1) & (cabin0.sent_button>cabin0.floor))
- D = calldown | ((cabin0.sent_light<cabin0.floor) & (cabin0.sent_light>-1)) | ((cabin0.sent_light=-1) & (cabin0.sent_button>-1) & (cabin0.sent_button<cabin0.floor))

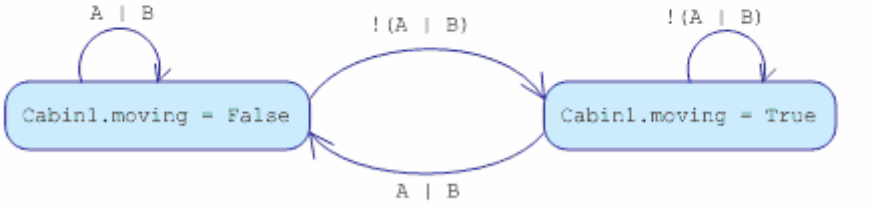
نمودار فوق مربوط به متغیر `cabin.dir` که جهت حرکت آسانسور را مشخص می‌کند.

همان طور که در ابتدا گفته شد نمودار کلی از ضرب این دو نمودار حاصل می‌شود.

۲. بخش دوم :

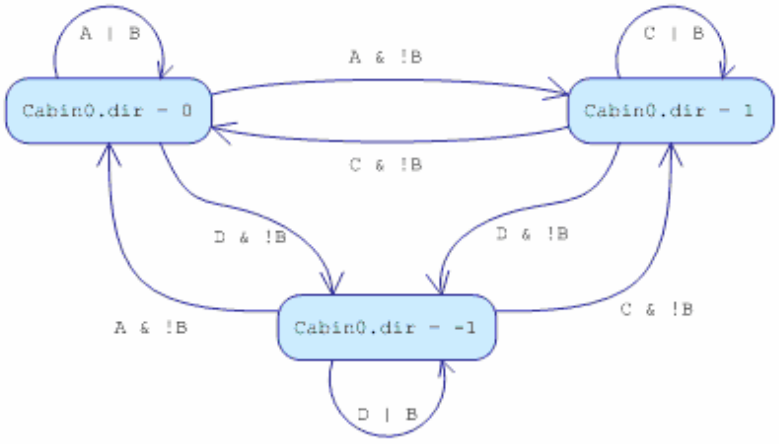


- A = next(open0)
- B = (!call0) & (cabin0.sent_light=-1) & (cabin0.sent_button=-1)

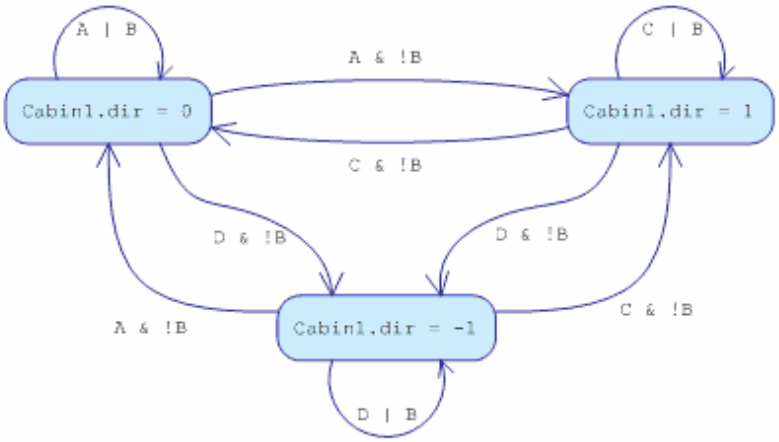


- A = next(open1)
- B = (!call1) & (cabin1.sent_light=-1) & (cabin1.sent_button=-1)

دو نمودار فوق مربوط به متغیرهای cabin0.moving و cabin1.moving است که بیان می‌کند که وضعیت هر کابین در حال حرکت است و یا متوقف است.

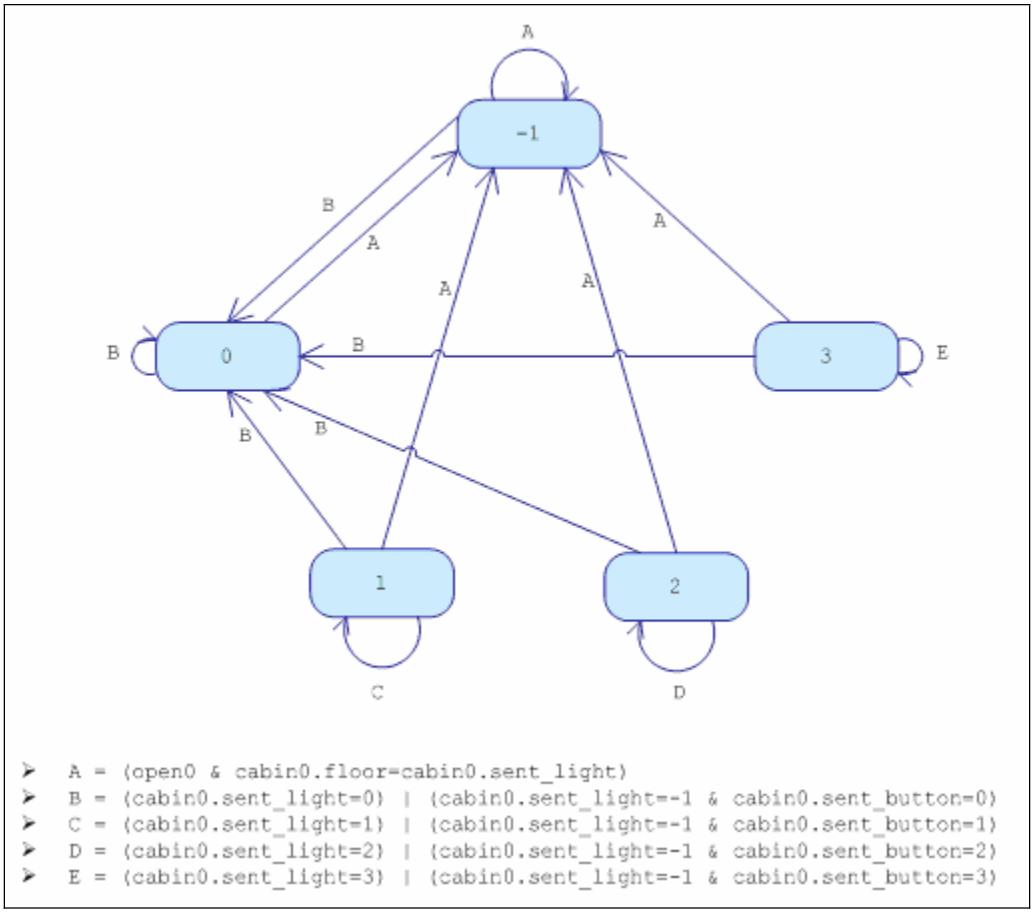


- A = (!call0) & (cabin0.sent_light=-1) & (cabin0.sent_button=-1)
- B = (callup0 | (cabin0.sent_light>cabin0.floor) | ((cabin0.sent_light=-1) & (cabin0.sent_button>cabin0.floor))) & (calldown0 | ((cabin0.sent_light<cabin0.floor) & (cabin0.sent_light>-1)) | ((cabin0.sent_light=-1) & (cabin0.sent_button>-1) & (cabin0.sent_button<cabin0.floor))) & cabin0.dir!=0
- C = callup0 | (cabin0.sent_light>cabin0.floor) | ((cabin0.sent_light=-1) & (cabin0.sent_button>cabin0.floor))
- D = calldown0 | ((cabin0.sent_light<cabin0.floor) & (cabin0.sent_light>-1)) | ((cabin0.sent_light=-1) & (cabin0.sent_button>-1) & (cabin0.sent_button<cabin0.floor))



- A = (!call1) & (cabin1.sent_light=-1) & (cabin1.sent_button=-1)
- B = (callup1 | (cabin1.sent_light>cabin1.floor) | ((cabin1.sent_light=-1) & (cabin1.sent_button>cabin1.floor))) & (calldown1 | ((cabin1.sent_light<cabin1.floor) & (cabin1.sent_light>-1)) | ((cabin1.sent_light=-1) & (cabin1.sent_button>-1) & (cabin1.sent_button<cabin1.floor))) & cabin1.dir!=0
- C = callup1 | (cabin1.sent_light>cabin1.floor) | ((cabin1.sent_light=-1) & (cabin1.sent_button>cabin1.floor))
- D = calldown1 | ((cabin1.sent_light<cabin1.floor) & (cabin1.sent_light>-1)) | ((cabin1.sent_light=-1) & (cabin1.sent_button>-1) & (cabin1.sent_button<cabin1.floor))

نمودارهای بالا مربوط به جهت حرکت هر یک از کابین‌ها می‌باشد.




```
A = (open0 & cabin0.floor=cabin0.sent_light)
B = (cabin0.sent_light=0) | (cabin0.sent_light=-1 & cabin0.sent_button=0)
C = (cabin0.sent_light=1) | (cabin0.sent_light=-1 & cabin0.sent_button=1)
D = (cabin0.sent_light=2) | (cabin0.sent_light=-1 & cabin0.sent_button=2)
E = (cabin0.sent_light=3) | (cabin0.sent_light=-1 & cabin0.sent_button=3)
```

در نهایت این نمودار مربوط به متغیر `cabin0.sent_light` است که در بیانگر مقصد بعدی آسانسور است. توجه کنید که برای ساده شدن شکل قسمت‌هایی از آن رسم نشده است. در حقیقت این یک گراف کامل است یعنی به گره ۱ یال‌هایی با برچسب C از تمام رئوس وارد شود. همچنین برای گره ۲ با برچسب D و گره ۳ با برچسب E نیز به همین ترتیب.

نموداری نظیر این برای کابین ۱ نیز وجود دارد که در آن به جای متغیر `cabin0` از متغیر `cabin1` استفاده شده است که به منظور اختصار از رسم آن صرف نظر کردیم. در انتها نمودار کلی ضرب ۶ نمودار فوق است.

خواص Fairness :

در این خواصی را که به عنوان خواص توازن در هر بخش مورد استفاده قرار گرفته است به هم می‌بینیم.

در بخش اول شرط توازن را این در نظر گرفته ایم که کابین آسانسور حرکت کند. در نتیجه آسانسور باید حرکت کند و نباید در یک نقطه ساکن باشد. در زیر کد آن را مشاهده می‌کنید :

```
FAIRNESS
  cabin0.moving
```

در بخش دوم شرط توازن این است که هر یک از دو آسانسور بی‌نهایت بار حرکت کنند در غیر این صورت ممکن است که یکی از دو کابین همیشه ثابت باشد که مطلوب نیست. لذا کد آن به صورت زیر است :

```
FAIRNESS
  cabin0.moving
FAIRNESS
  cabin1.moving
```

بیان Specifications :

در این بخش هر یک از خواصی که باید بررسی شود را به صورت مختصر توضیح داده و نوع آن را بیان می‌کنیم.

۱. بخش اول :

- ✓ خاصیت اول :

```
-- All the requests from particular floor are eventually serviced
SPEC
  AG ( door0.call -> AF door0.open )                    -- true
SPEC
  AG ( door1.call -> AF door1.open )                    -- true
```

این خاصیت از نوع liveness می‌باشد زیرا شرایطی را بیان می‌کند که باید در آینده برقرار باشد. این خاصیت در تمام حالات و گزارها بررسی شد و صادق بود.

- ✓ خاصیت دوم :

```
-- All the requests to particular floor are eventually serviced
SPEC
  AG ( (cabin0.sent_light=0) -> AF door0.open )         -- true
SPEC
  AG ( (cabin0.sent_light=1) -> AF door1.open )         -- true
```

این خاصیت نیز از نوع liveness می‌باشد و با موفقیت بررسی شد و در تمام حالات برقرار بود.

- ✓ خاصیت سوم :

```
-- The elevator never moves with its doors open
SPEC
  AG (cabin0.moving -> !mycontroller.open)              -- true
```

این خاصیت از نوع safety می‌باشد زیرا شرایطی را معرفی می‌کند که در تمام حالات برقرار باشد. این خاصیت در تمام حالات بررسی شد و همیشه صادق است.

۲. بخش دوم :

✓ خاصیت ۱ :

```
-- All the requests from particular floor are eventually serviced
SPEC
  AG ( door0.call -> AF door0.open )
SPEC
  AG ( door1.call -> AF door1.open )
SPEC
  AG ( door2.call -> AF door2.open )
SPEC
  AG ( door3.call -> AF door3.open )
```

این خاصیت از نوع liveness می‌باشد زیرا شرایطی را بیان می‌کند که باید در آینده برقرار باشد. این خاصیت در تمام حالات و گزار ها بررسی شد و صادق بود.

✓ خاصیت ۲ :

```
-- All the requests to particular floor are eventually serviced
SPEC
  AG ( (cabin0.sent_light=0) -> AF door0.open0 )
SPEC
  AG ( (cabin0.sent_light=1) -> AF door1.open0 )
SPEC
  AG ( (cabin0.sent_light=2) -> AF door2.open0 )
SPEC
  AG ( (cabin0.sent_light=3) -> AF door3.open0 )
SPEC
  AG ( (cabin1.sent_light=0) -> AF door0.open1 )
SPEC
  AG ( (cabin1.sent_light=1) -> AF door1.open1 )
SPEC
  AG ( (cabin1.sent_light=2) -> AF door2.open1 )
SPEC
  AG ( (cabin1.sent_light=3) -> AF door3.open1 )
```

این خاصیت نیز از نوع liveness می‌باشد و با موفقیت بررسی شد و در تمام حالات برقرار بود.

✓ خاصیت ۳ :

```
-- The elevator never moves with its doors open
SPEC
  AG(cabin0.moving -> !mycontroller.open0)
SPEC
  AG(cabin1.moving -> !mycontroller.open1)
```

این خاصیت از نوع safety می‌باشد زیرا شرایطی را معرفی می‌کند که در تمام حالات برقرار باشد. این خاصیت در تمام حالات بررسی شد و همیشه صادق است.

✓ خاصیت ۴ :

```
-- Two cabins are not scheduled to service request from one floor
SPEC
  AG (!((cabin0.floor=0 & cabin0.sent_button=0) |
      (cabin0.floor!=0 & (cabin0.sent_light=0 | cabin0.sent_button=0)) |
      (cabin1.floor=0 & cabin1.sent_button=0) |
      (cabin1.floor!=0 & (cabin1.sent_light=0 | cabin1.sent_button=0)) |
      (door0.open0 & (cabin1.sent_button=0 | cabin1.sent_light=0)) |
      (door0.open1 & (cabin0.sent_button=0 | cabin0.sent_light=0)) )
             -> !EX(door0.open0 & door0.open1))
SPEC
  AG (!((cabin0.floor=1 & cabin0.sent_button=1) |
      (cabin0.floor!=1 & (cabin0.sent_light=1 | cabin0.sent_button=1)) |
      (cabin1.floor=1 & cabin1.sent_button=1) |
      (cabin1.floor!=1 & (cabin1.sent_light=1 | cabin1.sent_button=1)) |
      (door1.open0 & (cabin1.sent_button=1 | cabin1.sent_light=1)) |
      (door1.open1 & (cabin0.sent_button=1 | cabin0.sent_light=1)) )
             -> !EX(door1.open0 & door1.open1))

SPEC
  AG (!((cabin0.floor=2 & cabin0.sent_button=2) |
      (cabin0.floor!=2 & (cabin0.sent_light=2 | cabin0.sent_button=2)) |
      (cabin1.floor=2 & cabin1.sent_button=2) |
      (cabin1.floor!=2 & (cabin1.sent_light=2 | cabin1.sent_button=2)) |
      (door2.open0 & (cabin1.sent_button=2 | cabin1.sent_light=2)) |
      (door2.open1 & (cabin0.sent_button=2 | cabin0.sent_light=2)) )
             -> !EX(door2.open0 & door2.open1))
SPEC
  AG (!((cabin0.floor=3 & cabin0.sent_button=3) |
      (cabin0.floor!=3 & (cabin0.sent_light=3 | cabin0.sent_button=3)) |
      (cabin1.floor=3 & cabin1.sent_button=3) |
      (cabin1.floor!=3 & (cabin1.sent_light=3 | cabin1.sent_button=3)) |
      (door3.open0 & (cabin1.sent_button=3 | cabin1.sent_light=3)) |
      (door3.open1 & (cabin0.sent_button=3 | cabin0.sent_light=3)) )
             -> !EX(door3.open0 & door3.open1))
```

این خاصیت به این علت که شرطی را بیان می‌کند که باید همیشه و در تمام حالات صادق باشد از Safety نوع می-باشد. این شرط نیز در تمام حالات بررسی شد و همیشه برقرار است.

✓ خاصیت ۵ :

```
-- The door will close if someone enters elevator
SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=0 &
       EX (door0.open0 & cabin0.passenger_present))
         -> (cabin0.door_open | door0.call | cabin0.sent_light=0))
SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=0 &
       EX (door0.open1 & cabin1.passenger_present))
         -> (cabin1.door_open | door0.call | cabin1.sent_light=0))
SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=1 &
       EX (door1.open0 & cabin0.passenger_present))
```

```
        -> (cabin0.door_open | door1.call | cabin0.sent_light=1))
SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=1 &
       EX (door1.open1 & cabin1.passenger_present))
        -> (cabin1.door_open | door1.call | cabin1.sent_light=1))
SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=2 &
       EX (door2.open0 & cabin0.passenger_present))
        -> (cabin0.door_open | door2.call | cabin0.sent_light=2))
SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=2 &
       EX (door2.open1 & cabin1.passenger_present))
        -> (cabin1.door_open | door2.call | cabin1.sent_light=2))
SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=3 &
       EX (door3.open0 & cabin0.passenger_present))
        -> (cabin0.door_open | door3.call | cabin0.sent_light=3))
SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=3 &
       EX (door3.open1 & cabin1.passenger_present))
        -> (cabin1.door_open | door3.call | cabin1.sent_light=3))
```

این خاصیت نیز از نوع Safety است و طبق بررسی انجام شده همیشه صادق است.

- ✓ خاصیت ۶:

```
-- The elevator won't react to door_open if there's another request
SPEC
  AG ( (cabin0.door_open & cabin0.floor=0 & !EX (door0.open0) )
       -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door_open & cabin1.floor=0 & !EX (door0.open1) )
       -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door_open & cabin0.floor=1 & !EX (door1.open0) )
       -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door_open & cabin1.floor=1 & !EX (door1.open1) )
       -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door_open & cabin0.floor=2 & !EX (door2.open0) )
       -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door_open & cabin1.floor=2 & !EX (door2.open1) )
       -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door_open & cabin0.floor=3 & !EX (door3.open0) )
       -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )
```

```
SPEC
  AG ( (cabin1.door_open & cabin1.floor=3 & !EX (door3.open1) )
       -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )
```

این خاصیت در تمام حالات بررسی شد و صادق است و از گونه Safety می‌باشد.

- ✓ خاصیت ۷ :

همان طور که در تمرین خواسته شده باید خواص جدیدی مطرح شود. این خاصیت بررسی می‌کند که در هر کدام از دو آسانسور در بیش از یک طبقه باز نباشد. این خاصیت نیز از نوع Safety می‌باشد. طبق بررسی در تمام حالات بر قرار است.

```
-- EXTRA CTL PROPS: Not two doors of a cabin open in the same time
SPEC
  AG (! two_door_open0 )
SPEC
  AG (! two_door_open1 )
```

نتایج درستی‌یابی مدل :

در این قسمت نتایج حاصل از درستی‌یابی آورده شده است. جدول زیر تعداد حالات ایجاد شده برای هر بخش را بیان می‌کند:

نمونه	تعداد کل حالات	تعداد حالات قابل دسترسی	قطر سیستم
بخش اول	۶۹۱۲	۵۷۶	۷
بخش دوم	2.593e+010	9.468e+006	۱۳

جدول زیر اطلاعات مربوط به زمان و نتیجه بررسی هر یک از ویژگی‌ها را در بر دارد.

	شماره خاصیت	نوع خاصیت	زمان بررسی (ثانیه)	نتیجه بررسی
بخش اول	۱	Liveness	۰	✓
	۲	Liveness	۰	✓
	۳	Safety	۰	✓
	تمام خواص همزمان		۰	✓
بخش دوم	۱	Liveness	۸۳٫۸	✓
	۲	Liveness	۲۶۲٫۵	✓
	۳	Safety	۱۸٫۴	✓
	۴	Safety	۱۸٫۵	✓
	۵	Safety	۱۸٫۸	✓
	۶	Safety	۱۸٫۹	✓
	۷	Safety	۱۸٫۵	✓
	تمام خواص همزمان		۳۳۷٫۷	✓

```
-- ----------------------------------------------------------------------
-- ----------------------------------------------------------------------
--                          2-Floor Elevator
--
--
--                       K. Rafati and N.Esfahani
-- ----------------------------------------------------------------------
-- ----------------------------------------------------------------------


------------------------------------------------------------------------------
MODULE door(thefloor,mycabin)
 -- Each "door" knows its (fix) floor and assigned cabin

VAR
  call light: boolean;          -- whether the lift has been called or not
  call_button: boolean;         -- whether someone is pushing on the call
       button
  open: boolean;                -- whether the door is open or not

DEFINE
  call := call light | call_button;
  floor := thefloor;

INIT
  !call & !open;

TRANS
  next(open) = case
   -- The door will be open at the next step if the lift has been
   -- called or sent at this floor, and the cabin is to arrive.
     (call | (mycabin.sent light=floor) | (mycabin.sent button=floor))
                  & mycabin.moving & (floor = mycabin.floor+mycabin.dir) :
                  TRUE;
   -- Another possibility is when the lift is "waiting"
   -- at this floor and someone calls (or sends it ?) here.
     (call | (mycabin.sent_light=floor) | (mycabin.sent_button=floor))
                         & (!mycabin.moving) & (floor = mycabin.floor) : TRUE;
   -- Otherwise, no need to open the door...
     1 : FALSE;
  esac
  &
  next(call_light) = case
     open : FALSE;
     call light|call_button : TRUE;
     1 : FALSE;
  esac;
------------------------------------------------------------------------------



------------------------------------------------------------------------------
MODULE controller(cabin0, door0, door1)
 -- The controller knows the states of all the cabins and doors

DEFINE
 -- if the cabin has been called at an upper floor
  callup :=   (door1.call & cabin0.floor<1);
 -- if the cabin has been called downstairs
  calldown := (door0.call & cabin0.floor>0);
 -- if the cabin has been called at all
  call := door0.call | door1.call;
 -- if a door is open somewhere
  open := door0.open | door1.open;

TRANS
  next(cabin0.moving) = case
   -- won't move if a door will open
    next(open) : FALSE;
   -- won't move if not been asked to
    (!call) & (cabin0.sent_light=-1) & (cabin0.sent_button=-1) : FALSE;
```

```
   -- otherwise, let's go !
    1 : TRUE;
  esac
  &
  next(cabin0.dir) = case
   -- if we don't know where to go...
    (!call) & (cabin0.sent light=-1) & (cabin0.sent button=-1) : 0;
   -- if we're busy: have to go both up- and downstairs
    (callup | (cabin0.sent light>cabin0.floor) | ((cabin0.sent_light=-1) &
      (cabin0.sent button>cabin0.floor))) & (calldown |
      ((cabin0.sent light<cabin0.floor) &
      (cabin0.sent light>-1)) | ((cabin0.sent_light=-1) &
      (cabin0.sent_button>-1) &
      (cabin0.sent button<cabin0.floor))) & cabin0.dir!=0 : cabin0.dir;
   -- if we just have to go upstairs
    callup | (cabin0.sent light>cabin0.floor) | ((cabin0.sent_light=-1) &
      (cabin0.sent button>cabin0.floor)) : 1;
   -- or downstairs
    calldown | ((cabin0.sent light<cabin0.floor) & (cabin0.sent_light>-1)) |
      ((cabin0.sent light=-1) & (cabin0.sent button>-1) &
      (cabin0.sent button<cabin0.floor)) : -1;
   -- don't know if something else if possible, but I would not know what to
   do...
    1 : 0;
  esac;
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
MODULE cabin(mycontroller)
-- the cabin knows nobody

VAR
  floor: 0..1;                  -- where the cabin is
  sent light: -1..1;            -- if the lift has been sent at some floor
  sent button: -1..1;           -- if a "send" button is being pushed on
  moving: boolean;              -- if the cabin is moving
  dir: -1..1;                   -- direction (down, "here" or up) of the cabin

INIT
  floor=0 & !moving & dir=0 & sent_light=-1 & sent_button=-1;


TRANS
  next(floor) = case
    moving : floor + dir;
    1 : floor;
  esac
  &
  next(sent light) = case
    mycontroller.open & (floor=sent_light) : -1;
    (sent light!=-1) : sent_light;
    1 : sent_button;
  esac;
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
MODULE main

VAR
 -- one door at each floor
  door0: door(0, cabin0);
  door1: door(1, cabin0);
 -- only one cabin here
  cabin0: cabin(mycontroller);
 -- the controller
  mycontroller: controller(cabin0, door0, door1);
```

```
FAIRNESS
  cabin0.moving
--------------------------------------------------------------------------------



--------------------------------------------------------------------------------
-- All the requests from particular floor are eventually serviced
SPEC
  AG ( door0.call -> AF door0.open )                    -- true
SPEC
  AG ( door1.call -> AF door1.open )                    -- true

-- All the requests to particular floor are eventually serviced
SPEC
  AG ( (cabin0.sent_light=0) -> AF door0.open )         -- true
SPEC
  AG ( (cabin0.sent_light=1) -> AF door1.open )         -- true

-- The elevator never moves with its doors open
SPEC
  AG(cabin0.moving -> !mycontroller.open)               -- true

-- TEST
--SPEC
--  AG(cabin0.dir=0 -> !cabin0.moving)                  -- true
--SPEC
--  AG(!cabin0.moving -> cabin0.dir=0)                  -- true
--------------------------------------------------------------------------------
```

```
-- ---------------------------------------------------------------------
-- ---------------------------------------------------------------------
--                       4-Floor Better Elevator
--
--
--                       K. Rafati and N.Esfahani
-- ---------------------------------------------------------------------
-- ---------------------------------------------------------------------


-----------------------------------------------------------------------------
MODULE door(thefloor,cabin0,cabin1)
 -- Each "door" knows its (fix) floor and assigned cabin

VAR
  call light: boolean;          -- whether the lift has been called or not
  call_button: boolean;         -- whether someone is pushing on the call
       button
  open0: boolean;               -- whether the door0 is open or not
  open1: boolean;               -- whether the door1 is open or not

DEFINE
  call := call light | call_button;
  floor := thefloor;
  open := open0 | open1;

INIT
  !call & !open0 & !open1;

TRANS
  next(open0) = case
   -- The door will be open at the next step if the lift has been
   -- called or sent at this floor, and the cabin is to arrive.
     ((call & !next(open1)) | (cabin0.sent light=floor) |
        (cabin0.sent button=floor & cabin0.sent light=-1)) &
        cabin0.moving & (floor = cabin0.floor+cabin0.dir) : TRUE;
   -- Another possibility is when the lift is "waiting"
   -- at this floor and someone calls (or sends it ?) here.
     ((call & !next(open1)) | (cabin0.sent light=floor) |
        (cabin0.sent button=floor & cabin0.sent_light=-1)) &
        !cabin0.moving & (floor = cabin0.floor) : TRUE;
   -- Otherwise, no need to open the door...
     1 : FALSE;
  esac
  &
  next(open1) = case
   -- The door will be open at the next step if the lift has been
   -- called or sent at this floor, and the cabin is to arrive.
     ((call & !next(open0)) | (cabin1.sent_light=floor) |
     (cabin1.sent button=floor & cabin1.sent light=-1)) &
     cabin1.moving & (floor = cabin1.floor+cabin1.dir) : TRUE;
   -- Another possibility is when the lift is "waiting"
   -- at this floor and someone calls (or sends it ?) here.
     ((call & !next(open0)) | (cabin1.sent light=floor) |
        (cabin1.sent button=floor & cabin1.sent_light=-1)) &
        !cabin1.moving & (floor = cabin1.floor) : TRUE;
   -- Otherwise, no need to open the door...
     1 : FALSE;
  esac
  &
  next(call light) = case
     next(open) : FALSE;
     call_light|call_button : TRUE;
     1 : FALSE;
  esac;
-----------------------------------------------------------------------------



-----------------------------------------------------------------------------
MODULE controller(cabin0, cabin1, door0, door1, door2, door3)
```

```
 -- The controller knows the states of all the cabins and doors

DEFINE
  -- All the situations in which cabin0 is down the calling floor
  cabin0_down := (door3.call & cabin0.floor<3) |
                 (door2.call & cabin0.floor<2) |
                 (door1.call & cabin0.floor<1) ;
  down0 going := cabin0 down & cabin0.dir=-1;
  down0 coming := cabin0 down & cabin0.dir=1;
  down0_stay := cabin0_down & cabin0.dir=0;

  -- All the situations in which cabin1 is down the calling floor
  cabin1_down := (door3.call & cabin1.floor<3) |
                 (door2.call & cabin1.floor<2) |
                 (door1.call & cabin1.floor<1) ;
  down1 going := cabin1 down & cabin1.dir=-1;
  down1 coming := cabin1 down & cabin1.dir=1;
  down1_stay := cabin1_down & cabin1.dir=0;

  -- All the situations in which cabin0 is up the calling floor
  cabin0_up := (door2.call & cabin0.floor>2) |
               (door1.call & cabin0.floor>1) |
               (door0.call & cabin0.floor>0) ;
  up0_going := cabin0_up & cabin0.dir=1;
  up0 coming := cabin0 up & cabin0.dir=-1;
  up0_stay := cabin0_up & cabin0.dir=0;

  -- All the situations in which cabin1 is up the calling floor
  cabin1_up := (door2.call & cabin1.floor>2) |
               (door1.call & cabin1.floor>1) |
               (door0.call & cabin1.floor>0) ;
  up1 going := cabin1 up & cabin1.dir=1;
  up1 coming := cabin1 up & cabin1.dir=-1;
  up1_stay := cabin1_up & cabin1.dir=0;

 -- if the cabin has been called up
  callup0 := (down0 coming & !(down1 coming & cabin1.floor>cabin0.floor)) |
             (down0_coming & (down1_stay | up1_stay)) |
             (down0 stay & (up1 going | down1_going));
  callup1 := (down1 coming & !(down0 coming & cabin0.floor>=cabin1.floor)) |
             (down1 coming & (down0 stay | up0 stay)) |
             (down1_stay & (up0_going | down0_going));

 -- if the cabin has been called down
  calldown0 := (up0 coming & !(up1 coming & cabin1.floor<cabin0.floor)) |
               (up0 coming & (down1 stay | up1 stay)) |
               (up0 stay & (up1 going | down1 going));
  calldown1 := (up1 coming & !(up0 coming & cabin0.floor<=cabin1.floor)) |
               (up1_coming & (down0_stay | up0_stay)) |
               (up1_stay & (up0_going | down0_going));

 -- if the cabin has been called at all
  call0 := callup0 | calldown0;
  call1 := callup1 | calldown1;
 -- if a door is open somewhere
  open := door0.open | door1.open | door2.open | door3.open;
  open0 := door0.open0 | door1.open0 | door2.open0 | door3.open0;
  open1 := door0.open1 | door1.open1 | door2.open1 | door3.open1;

TRANS
  next(cabin0.moving) = case
   -- won't move if a door will open
    next(open0) : FALSE;
   -- won't move if not been asked to
    (!call0) & (cabin0.sent_light=-1) & (cabin0.sent_button=-1) : FALSE;
   -- otherwise, let's go !
    1 : TRUE;
  esac
  &
  next(cabin0.dir) = case
   -- if we don't know where to go...
```

```
    (!call0) & (cabin0.sent light=-1) & (cabin0.sent button=-1) : 0;
   -- if we're busy: have to go both up- and downstairs
    (callup0 | (cabin0.sent light>cabin0.floor) | ((cabin0.sent_light=-1) &
      (cabin0.sent button>cabin0.floor))) & (calldown0 |
      ((cabin0.sent light<cabin0.floor) &
      (cabin0.sent_light>-1)) | ((cabin0.sent_light=-1) &
      (cabin0.sent button>-1) &
      (cabin0.sent button<cabin0.floor))) & cabin0.dir!=0 : cabin0.dir;
   -- if we just have to go upstairs
    callup0 | (cabin0.sent light>cabin0.floor) | ((cabin0.sent_light=-1) &
      (cabin0.sent button>cabin0.floor)) : 1;
   -- or downstairs
    calldown0 | ((cabin0.sent_light<cabin0.floor) & (cabin0.sent_light>-1)) |
      ((cabin0.sent light=-1) & (cabin0.sent button>-1) &
      (cabin0.sent button<cabin0.floor)) : -1;
   -- don't know if something else if possible, but I would not know what to
    do...
     1 : 0;
  esac
  &
  next(cabin1.moving) = case
   -- won't move if a door will open
    next(open1) : FALSE;
   -- won't move if not been asked to
    (!call1) & (cabin1.sent_light=-1) & (cabin1.sent_button=-1) : FALSE;
   -- otherwise, let's go !
    1 : TRUE;
  esac
  &
  next(cabin1.dir) = case
   -- if we don't know where to go...
    (!call1) & (cabin1.sent light=-1) & (cabin1.sent button=-1) : 0;
   -- if we're busy: have to go both up- and downstairs
    (callup1 | (cabin1.sent light>cabin1.floor) | ((cabin1.sent_light=-1) &
      (cabin1.sent button>cabin1.floor))) & (calldown1 |
      ((cabin1.sent light<cabin1.floor) &
      (cabin1.sent light>-1)) | ((cabin1.sent_light=-1) &
      (cabin1.sent_button>-1) &
      (cabin1.sent button<cabin1.floor))) & cabin1.dir!=0 : cabin1.dir;
   -- if we just have to go upstairs
    callup1 | (cabin1.sent light>cabin1.floor) | ((cabin1.sent_light=-1) &
      (cabin1.sent button>cabin1.floor)) : 1;
   -- or downstairs
    calldown1 | ((cabin1.sent_light<cabin1.floor) & (cabin1.sent_light>-1)) |
      ((cabin1.sent light=-1) & (cabin1.sent button>-1) &
      (cabin1.sent button<cabin1.floor)) : -1;
   -- don't know if something else if possible, but I would not know what to
    do...
     1 : 0;
  esac
  &
  next(cabin0.sent light) = case
    (open0 & cabin0.floor=cabin0.sent light) : -1;
    (cabin0.sent light!=-1) : cabin0.sent_light;
    1 : cabin0.sent_button;
  esac
  &
  next(cabin1.sent light) = case
    (open1 & cabin1.floor=cabin1.sent light) : -1;
    (cabin1.sent light!=-1) : cabin1.sent_light;
    1 : cabin1.sent_button;
  esac;

--------------------------------------------------------------------------



--------------------------------------------------------------------------
MODULE cabin(mycontroller)
-- the cabin knows nobody
```

```
VAR
  floor: 0..3;                  -- where the cabin is
  sent light: -1..3;            -- if the lift has been sent at some floor
  sent button: -1..3;           -- if a "send" button is being pushed on
  moving: boolean;              -- if the cabin is moving
  dir: -1..1;                   -- direction (down, "here" or up) of the cabin

DEFINE
  -- Door open button can be constructed using existing variables
  door open := (sent button=floor);
  -- Same for passenger present
  passenger_present := (sent_button!=-1 | sent_light!=-1);

INIT
  floor=0 & !moving & dir=0 & sent_light=-1 & sent_button=-1;


TRANS
  next(floor) = case
    moving : floor + dir;
    1 : floor;
  esac;
-------------------------------------------------------------------------



-------------------------------------------------------------------------
MODULE main

VAR
 -- two door at each floor
  door0: door(0, cabin0, cabin1);
  door1: door(1, cabin0, cabin1);
  door2: door(2, cabin0, cabin1);
  door3: door(3, cabin0, cabin1);
 -- two cabins here
  cabin0: cabin(mycontroller);
  cabin1: cabin(mycontroller);
 -- the controller
  mycontroller: controller(cabin0, cabin1, door0, door1, door2, door3);

DEFINE
  two_door_open0 := (door0.open0 & (door1.open0 | door2.open0 | door3.open0))
  |
                        (door1.open0 & (door2.open0 | door3.open0)) |
                        (door2.open0 & door3.open0);
  two_door_open1 := (door0.open1 & (door1.open1 | door2.open1 | door3.open1))
  |
                        (door1.open1 & (door2.open1 | door3.open1)) |
                        (door2.open1 & door3.open1);

FAIRNESS
  cabin0.moving
FAIRNESS
  cabin1.moving
-------------------------------------------------------------------------



-------------------------------------------------------------------------
-- All the requests from particular floor are eventually serviced
SPEC
  AG ( door0.call -> AF door0.open )
SPEC
  AG ( door1.call -> AF door1.open )
SPEC
  AG ( door2.call -> AF door2.open )
SPEC
  AG ( door3.call -> AF door3.open )

-- All the requests to particular floor are eventually serviced
```

```
SPEC
  AG ( (cabin0.sent_light=0) -> AF door0.open0 )
SPEC
  AG ( (cabin0.sent_light=1) -> AF door1.open0 )
SPEC
  AG ( (cabin0.sent_light=2) -> AF door2.open0 )
SPEC
  AG ( (cabin0.sent_light=3) -> AF door3.open0 )
SPEC
  AG ( (cabin1.sent_light=0) -> AF door0.open1 )
SPEC
  AG ( (cabin1.sent_light=1) -> AF door1.open1 )
SPEC
  AG ( (cabin1.sent_light=2) -> AF door2.open1 )
SPEC
  AG ( (cabin1.sent_light=3) -> AF door3.open1 )

-- The elevator never moves with its doors open
SPEC
  AG(cabin0.moving -> !mycontroller.open0)
SPEC
  AG(cabin1.moving -> !mycontroller.open1)

-- Two cabins are not scheduled to service request from one floor
SPEC
  AG (! ( (cabin0.floor=0 & cabin0.sent button=0) |
          (cabin0.floor!=0 & (cabin0.sent light=0 | cabin0.sent_button=0)) |
          (cabin1.floor=0 & cabin1.sent button=0) |
          (cabin1.floor!=0 & (cabin1.sent light=0 | cabin1.sent button=0)) |
          (door0.open0 & (cabin1.sent button=0 | cabin1.sent light=0)) |
          (door0.open1 & (cabin0.sent_button=0 | cabin0.sent_light=0)) )
                -> !EX(door0.open0 & door0.open1))
SPEC
  AG (! ( (cabin0.floor=1 & cabin0.sent button=1) |
          (cabin0.floor!=1 & (cabin0.sent light=1 | cabin0.sent_button=1)) |
          (cabin1.floor=1 & cabin1.sent button=1) |
          (cabin1.floor!=1 & (cabin1.sent light=1 | cabin1.sent button=1)) |
          (door1.open0 & (cabin1.sent_button=1 | cabin1.sent_light=1)) |
          (door1.open1 & (cabin0.sent button=1 | cabin0.sent_light=1)) )
                -> !EX(door1.open0 & door1.open1))

SPEC
  AG (! ( (cabin0.floor=2 & cabin0.sent button=2) |
          (cabin0.floor!=2 & (cabin0.sent light=2 | cabin0.sent_button=2)) |
          (cabin1.floor=2 & cabin1.sent button=2) |
          (cabin1.floor!=2 & (cabin1.sent light=2 | cabin1.sent button=2)) |
          (door2.open0 & (cabin1.sent button=2 | cabin1.sent light=2)) |
          (door2.open1 & (cabin0.sent button=2 | cabin0.sent_light=2)) )
                -> !EX(door2.open0 & door2.open1))
SPEC
  AG (! ( (cabin0.floor=3 & cabin0.sent button=3) |
          (cabin0.floor!=3 & (cabin0.sent light=3 | cabin0.sent_button=3)) |
          (cabin1.floor=3 & cabin1.sent button=3) |
          (cabin1.floor!=3 & (cabin1.sent light=3 | cabin1.sent button=3)) |
          (door3.open0 & (cabin1.sent button=3 | cabin1.sent light=3)) |
          (door3.open1 & (cabin0.sent_button=3 | cabin0.sent_light=3)) )
                -> !EX(door3.open0 & door3.open1))

-- The door will close if someone enters elevator
SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=0 &
        EX(door0.open0 & cabin0.passenger present))
          -> (cabin0.door_open | door0.call | cabin0.sent_light=0))

SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=0 &
        EX(door0.open1 & cabin1.passenger present))
          -> (cabin1.door_open | door0.call | cabin1.sent_light=0))

SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=1 &
```

```
        EX(door1.open0 & cabin0.passenger present))
          -> (cabin0.door_open | door1.call | cabin0.sent_light=1))

SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=1 &
        EX(door1.open1 & cabin1.passenger_present))
          -> (cabin1.door_open | door1.call | cabin1.sent_light=1))

SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=2 &
        EX(door2.open0 & cabin0.passenger present))
          -> (cabin0.door_open | door2.call | cabin0.sent_light=2))

SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=2 &
        EX(door2.open1 & cabin1.passenger present))
          -> (cabin1.door_open | door2.call | cabin1.sent_light=2))

SPEC
  AG ( (cabin0.passenger_present & cabin0.floor=3 &
        EX(door3.open0 & cabin0.passenger present))
          -> (cabin0.door_open | door3.call | cabin0.sent_light=3))

SPEC
  AG ( (cabin1.passenger_present & cabin1.floor=3 &
        EX(door3.open1 & cabin1.passenger present))
          -> (cabin1.door_open | door3.call | cabin1.sent_light=3))

-- The elevator won't react to door_open if there's another request
SPEC
  AG ( (cabin0.door_open & cabin0.floor=0 & !EX(door0.open0) )
        -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door open & cabin1.floor=0 & !EX(door0.open1) )
        -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door open & cabin0.floor=1 & !EX(door1.open0) )
        -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door open & cabin1.floor=1 & !EX(door1.open1) )
        -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door open & cabin0.floor=2 & !EX(door2.open0) )
        -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door open & cabin1.floor=2 & !EX(door2.open1) )
        -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

SPEC
  AG ( (cabin0.door open & cabin0.floor=3 & !EX(door3.open0) )
        -> (mycontroller.call0 | cabin0.sent_light!=-1 | cabin0.moving) )

SPEC
  AG ( (cabin1.door open & cabin1.floor=3 & !EX(door3.open1) )
        -> (mycontroller.call1 | cabin1.sent_light!=-1 | cabin1.moving) )

-- EXTRA CTL PROPS: Not two doors of a cabin open in the same time
SPEC
  AG (! two_door_open0 )
SPEC
  AG (! two_door_open1 )

--------------------------------------------------------------------------
```