**问题描述：**

用NuSMV 对高楼电梯运行建模。假设十层楼。一台电梯，每层有上下按

钮（最低楼层没有下，最高楼层没有上）。称楼层是向上激活的，如果楼层向上

按钮被激活。电梯可能停留在某一层，也可能处于上升或者下降状态。如处于

上升状态（下降状态类似），电梯下一个楼层是：

• 高于当前楼层中如有向上激活楼层，则到其中最低的楼层；

• 高于当前楼层中如没有向上激活楼层，但有向下激活楼层，则到其中最

高的楼层；

• 高于当前楼层中如没有向上、向下激活楼层，则到比当前楼层高的最低

的楼层。

细化上述模型以能够合理模拟电梯内人员去目标楼层。用不确定性更新

模拟每一层可能随时发生的向上、向下的请求。

进一步，对两辆电梯场景进行建模。

• 场景一：假设电梯间没有通讯。

• 场景二：假设一台电梯只停奇数高楼层，另外一台只停偶数高楼层。

• 场景三：假设电梯间有通讯，激活楼层会调度空闲更近的电梯移动到

本楼层。

•（\*）场景四：假设有地下一、二层，并且一台电梯只能到地下一层，另外

一台地下两层都能到达。

四个文件：

single\_elevator.smv:单电梯普通建模

double\_elevator\_nocom.smv:场景一建模

double\_elevator\_odd\_even.smv:场景二建模

double\_elevator\_schedule.smv:场景三建模

**对单辆电梯进行建模：**

整个过程拆开看有：电梯厢，每一层的按钮，电梯控制器，电梯门 一共四个对象，每个对象对应一个模块。

每层的人员要去到目标楼层的时候就会按下该层的按钮，按钮被按下以后，之后是可以被控制器重置成初始状态的。

--Button:每层一个按钮，按钮被按下去以后，控制器可以重置它。

MODULE Button(reset)

VAR

pressed:boolean;

ASSIGN

init(pressed) := FALSE;

next(pressed) :=

case

pressed & reset :FALSE;

pressed & !reset :TRUE;

!pressed :{FALSE,TRUE};

esac;

--只有在按钮被按下之后，控制器才可以重置按钮状态。

INVARSPEC(reset->pressed)

电梯厢根据控制器发过来的指令决定是运行方向

--CABIN:电梯厢可以停留在1~4层，电梯厢可以向上，向下，或者准备状态。

MODULE Cabin(move\_cmd)

VAR

floor:{1,2,3,4};

direction:{standing,moving\_up,moving\_down};

ASSIGN

init(direction):=standing;

next(direction):=

case

move\_cmd=stop:standing;

move\_cmd=move\_up:moving\_up;

move\_cmd=move\_down:moving\_down;

move\_cmd=nop:direction;

esac;

next(floor):=

case

next(direction)=standing:floor;

next(direction)=moving\_up:

case

floor=4:4;

TRUE:floor + 1;

esac;

next(direction)=moving\_down:

case

floor=1:1;

TRUE:floor - 1;

esac;

esac;

--只有在电梯厢运行的时候，控制器才可以发送stop命令

INVARSPEC(move\_cmd=stop -> direction in {moving\_up,moving\_down});

--电梯厢准备运行的时候，控制器才可以发送move命令

INVARSPEC(move\_cmd in {move\_up,move\_down} -> direction = standing);

--最高楼层没有上，最低楼层没有下

SPEC AG(floor=4 -> AX(direction != moving\_up))

SPEC AG(floor=1 -> AX(direction != moving\_down))

每层楼的电梯门根据控制器的指令决定是开还是关

--每一层楼有一个门，门的状态可以是开、关。

MODULE Door(door\_cmd)

VAR

status:{open,closed};

ASSIGN

next(status):=

case

door\_cmd=open:open;

door\_cmd=close:closed;

door\_cmd=nop:status;

esac;

--只有门的状态是关的时候，控制器才可以发open命令

INVARSPEC(door\_cmd=open -> status=closed)

--只有门的状态是开的时候，控制器才可以发closed命令

INVARSPEC(door\_cmd=closed -> status=open)

--控制器接收的状态包括：楼层号，电梯厢移动的方向,门的状态，四个楼层按钮的状态。

MODULE CTRL(floor,direction,door,pressed\_1,pressed\_2,pressed\_3,pressed\_4)

VAR

move\_cmd:{move\_up,move\_down,stop,nop};

door\_cmd:{open,close,nop};

reset\_1:boolean;

reset\_2:boolean;

reset\_3:boolean;

reset\_4:boolean;

--当电梯处于第N层，并且电梯门打开，被按下的按钮可以弹回

ASSIGN

reset\_1:=(pressed\_1 & floor=1 & door=open);

reset\_2:=(pressed\_2 & floor=2 & door=open);

reset\_3:=(pressed\_3 & floor=3 & door=open);

reset\_4:=(pressed\_4 & floor=4 & door=open);

--检查是否在当前楼层、高楼层、低楼层有人按了电梯按钮,说明出现了等待请求

DEFINE

pending\_here:=(floor=1 & pressed\_1) | (floor=2 & pressed\_2) | (floor=3 & pressed\_3)

| (floor=4 & pressed\_4);

pending\_up:=(floor=1 &(pressed\_2 | pressed\_3 | pressed\_4))

| (floor=2 &(pressed\_3 | pressed\_4))

| (floor=3 &(pressed\_4));

pending\_down:=(floor=4 &(pressed\_3 | pressed\_2 | pressed\_1))

| (floor=3 &(pressed\_2 | pressed\_1))

| (floor=2 &(pressed\_1));

--如果电梯正在移动，不要给门发送命令

--如果当前楼层出现了等待请求，门是关闭的，打开门

--如果在不同的楼层出现了等待请求，门是打开的，关闭它

--其它原因不给门发送命令

ASSIGN

door\_cmd:=

case

direction!=standing : nop;

pending\_here&door=closed:open;

pending\_up&door=open:close;

pending\_down&door=open:close;

TRUE:nop;

esac;

--last\_dir记录电梯厢最后移动的方向

VAR

last\_dir:{moving\_up,moving\_down};

ASSIGN

next(last\_dir):=

case

direction=standing:last\_dir;

TRUE : direction;

esac;

--如果门打开了，不要给电梯发送命令

--如果当前楼层有一个等待请求，电梯正在移动，停止电梯

--如果在高以及低楼层同时有等待请求，保持last\_dir的移动

--如果高楼层有等待请求，往上移动，低楼层类似

--其它情况，不要发送命令给电梯厢

ASSIGN

move\_cmd:=

case

door=open:nop;

pending\_here:case

direction!=standing:stop;

TRUE:nop;

esac;

pending\_up&pending\_down:case

direction!=standing:nop;

last\_dir=moving\_up:move\_up;

last\_dir=moving\_down:move\_down;

esac;

pending\_up:case

direction!=standing:nop;

TRUE:move\_up;

esac;

pending\_down:case

direction!=standing:nop;

TRUE:move\_down;

esac;

TRUE:nop;

esac;

MODULE main

VAR cabin:Cabin(ctrl.move\_cmd);

door:Door(ctrl.door\_cmd);

button\_1:Button(ctrl.reset\_1);

button\_2:Button(ctrl.reset\_2);

button\_3:Button(ctrl.reset\_3);

button\_4:Button(ctrl.reset\_4);

ctrl:CTRL(cabin.floor,cabin.direction,door.status,

button\_1.pressed,button\_2.pressed,

button\_3.pressed,button\_4.pressed);

--Specification

--任何一个按钮都不能永远被按下去

SPEC AG AF ! button\_1.pressed

SPEC AG AF ! button\_2.pressed

SPEC AG AF ! button\_3.pressed

SPEC AG AF ! button\_4.pressed

--电梯厢停在相应的楼层并且电梯门打开的时候，被按下的按钮才能被reset

SPEC AG (button\_1.pressed ->

A [button\_1.pressed U (cabin.floor =1 & door.status=open)])

SPEC AG (button\_2.pressed ->

A [button\_2.pressed U (cabin.floor =2 & door.status=open)])

SPEC AG (button\_3.pressed ->

A [button\_3.pressed U (cabin.floor =3 & door.status=open)])

SPEC AG (button\_4.pressed ->

A [button\_4.pressed U (cabin.floor =4 & door.status=open)])

--一旦电梯停在相应的楼层并且开了门，按钮就要被reset

SPEC AG ((button\_1.pressed & cabin.floor=1 & door.status=open)

-> AX !button\_1.pressed)

SPEC AG ((button\_2.pressed & cabin.floor=2 & door.status=open)

-> AX !button\_2.pressed)

SPEC AG ((button\_3.pressed & cabin.floor=3 & door.status=open)

-> AX !button\_3.pressed)

SPEC AG ((button\_4.pressed & cabin.floor=4 & door.status=open)

-> AX !button\_4.pressed)

--只有门关闭的时候电梯才可以运行

INVARSPEC (door.status=open -> cabin.direction=standing)

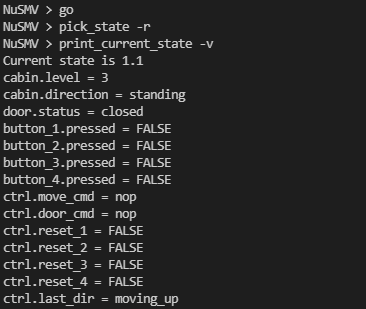
--如果按钮没有被按下，则控制器不能发送命令并且电梯厢一定处于standing状态

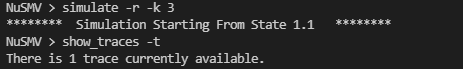
INVARSPEC(((!button\_1.pressed)&(!button\_2.pressed)

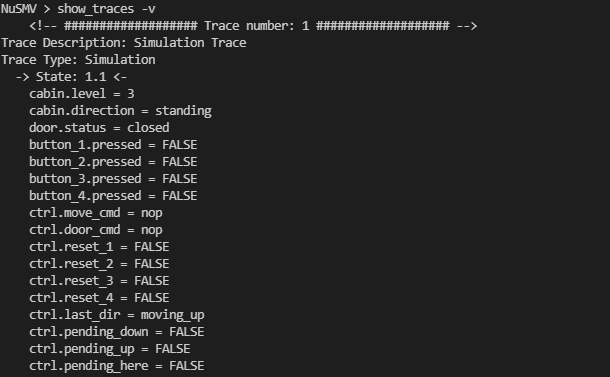
&(!button\_3.pressed)&(!button\_4.pressed))

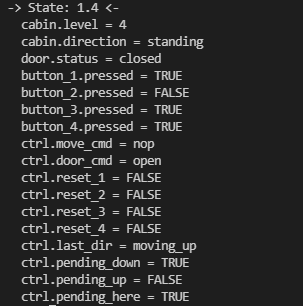
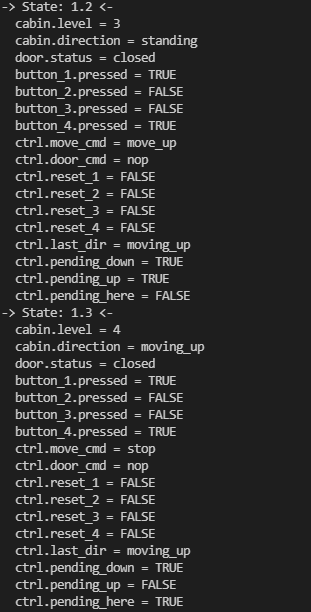
->(ctrl.door\_cmd=nop & ctrl.move\_cmd=nop))

**程序运行结果截图：**



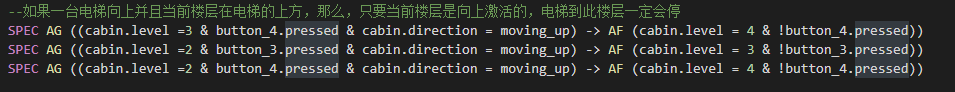






**验证SPEC：如果电梯向上并且当前楼层在电梯上方，那么，只要当**

**前楼层是向上激活的，电梯到此楼层一定会停。**





**对无通讯之间的两辆电梯进行建模：**

两辆电梯可以看作是两个独立的迁移系统

MODULE main

VAR

--第一辆电梯

cabin1:Cabin(ctrl1.move\_cmd);

door1:Door(ctrl1.door\_cmd);

button\_1\_1:Button(ctrl1.reset\_1);

button\_2\_1:Button(ctrl1.reset\_2);

button\_3\_1:Button(ctrl1.reset\_3);

button\_4\_1:Button(ctrl1.reset\_4);

ctrl1:CTRL(cabin1.level,cabin1.direction,door1.status,

button\_1\_1.pressed,button\_2\_1.pressed,

button\_3\_1.pressed,button\_4\_1.pressed);

--第二辆电梯

cabin2:Cabin(ctrl2.move\_cmd);

door2:Door(ctrl2.door\_cmd);

button\_1\_2:Button(ctrl2.reset\_1);

button\_2\_2:Button(ctrl2.reset\_2);

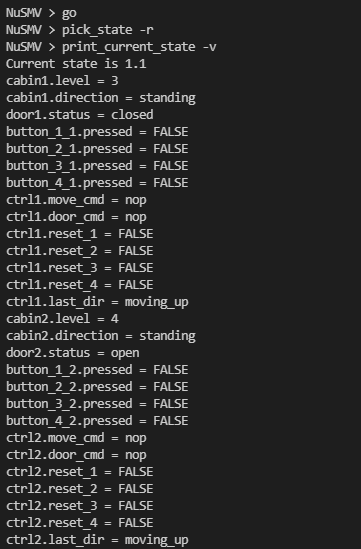
button\_3\_2:Button(ctrl2.reset\_3);

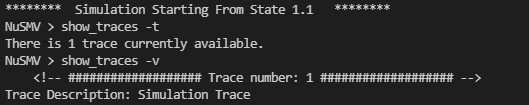
button\_4\_2:Button(ctrl2.reset\_4);

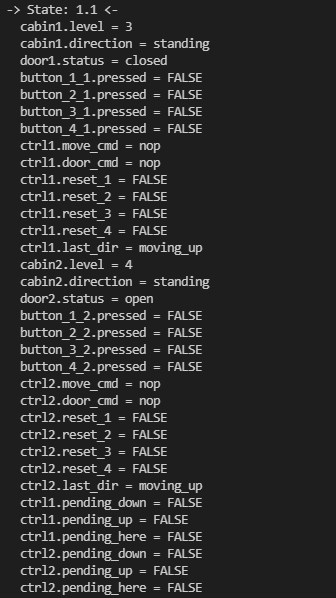
ctrl2:CTRL(cabin2.level,cabin2.direction,door2.status,

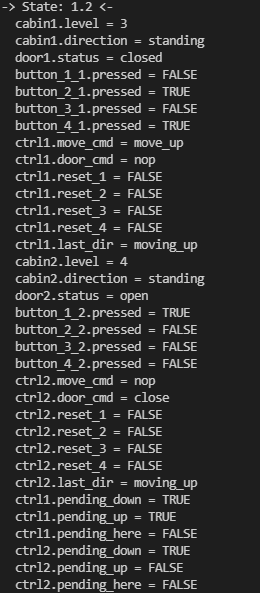
button\_1\_2.pressed,button\_2\_2.pressed,

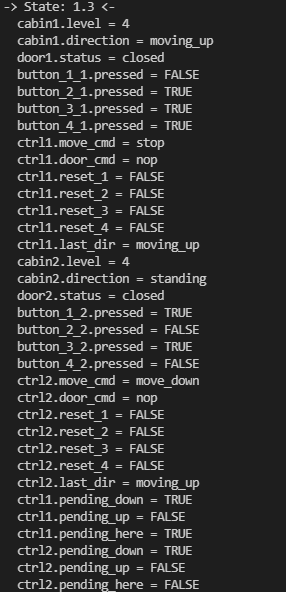
button\_3\_2.pressed,button\_4\_2.pressed);

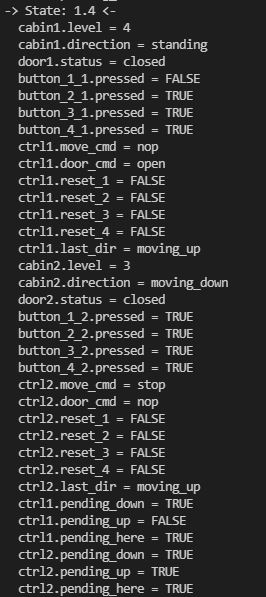




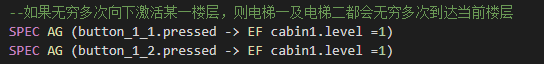








**验证SPEC：如果无穷多次向下激活某一楼层，则电梯一及电梯二都会无穷多次到达当前楼层**





**对两辆电梯进行建模，其中一辆电梯只停奇数楼层，另一辆只停偶数楼层：**

只需要将相应电梯厢的楼层状态改为奇偶，然后更新的时候+2。

MODULE main

VAR cabin\_odd:Cabin\_odd(ctrl\_odd.move\_cmd);

door\_odd:Door(ctrl\_odd.door\_cmd);

button\_1:Button(ctrl\_odd.reset\_1);

button\_3:Button(ctrl\_odd.reset\_3);

ctrl\_odd:CTRL(cabin\_odd.level,cabin\_odd.direction,door\_odd.status,

button\_1.pressed,FALSE,

button\_3.pressed,FALSE);

cabin\_even:Cabin\_even(ctrl\_even.move\_cmd);

door\_even:Door(ctrl\_even.door\_cmd);

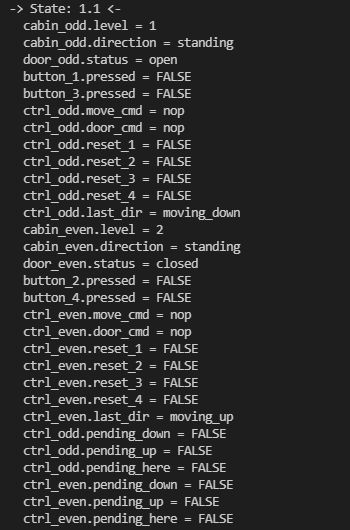
button\_2:Button(ctrl\_even.reset\_2);

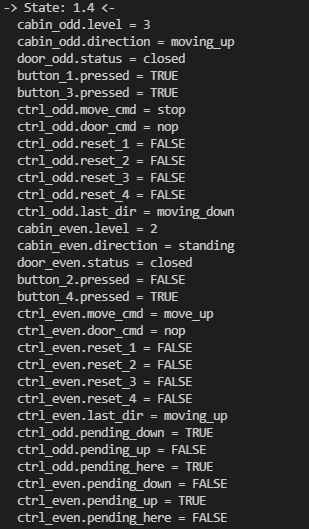
button\_4:Button(ctrl\_even.reset\_4);

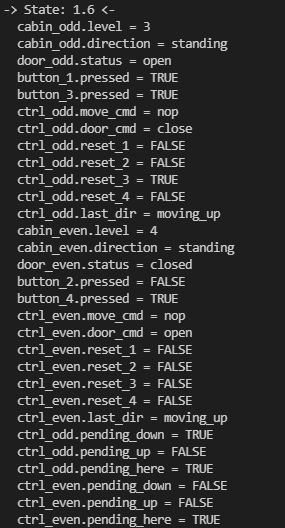
ctrl\_even:CTRL(cabin\_even.level,cabin\_even.direction,door\_even.status,

FALSE,button\_2.pressed,

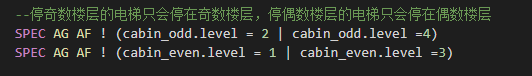
FALSE,button\_4.pressed);







**验证SPEC：两台电梯场景：一台电梯只会停靠在奇数楼层**





**对电梯间有通讯，激活楼层会调度空闲更近的电梯移动到对应楼层建模：**

代码有bug



