Elevator(UPPAAL)
2차발표

정혁준, 이명재

Contents

- Elevator System Specifications
- System Design
- Demo Play
- System Verification

- ***Elevator System Specifications***

• Elevator System

요구사항 정의

- 메인 모듈과 엘리베이터 내부 모듈은 정확한 위치를 동기화 한다.
- 엘리베이터 문은 특정한 위치에서만 열려야 한다.
- 엘리베이터 내부에서는 한 방향으로만 움직여야 한다.
- 엘리베이터 내부에서 누른 층으로 반드시 언젠가는 이동해야 한다.
- 특정 층에 서있을 때 엘리베이터 내부에서 열기 버튼을 누르면 엘리베이터 내부와 외부 문이 반드시 열려야 한다. 각각의 엘리베이터에는 엘리베이터를 구동하는 모듈이 있다.
- 전체 층과 전체 엘리베이터를 관리하는 메인 모듈이 있다.
- 엘리베이터는 4대로 구성되어 있다.
- 5층 건물이다.
- 이 외의 상세사항은 일반적인 엘리베이터의 특성을 따른다.

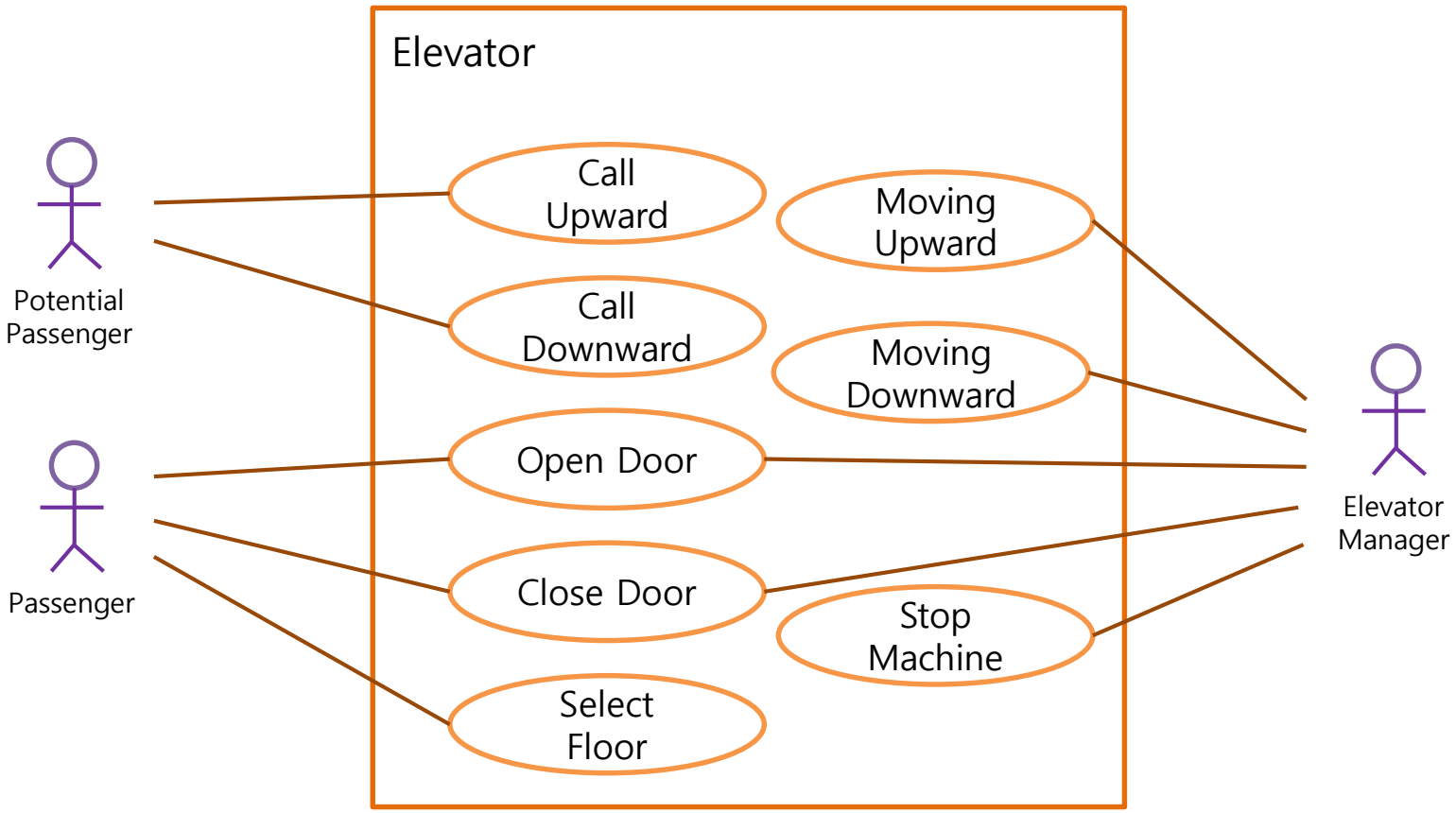
• Elevator System

요구사항 정의 – 층별 요구사항

- 특정 층에서 위 혹은 아래 버튼을 누르면 이동 방향을 고려해서 가장 가까운 엘리베이터가 와야 한다.
- 특정 층에서 위 혹은 아래 버튼을 누르면 언젠가는 반드시 엘리베이터가 와야 한다.
- 한 층에서 위 혹은 아래 버튼은 4대의 엘리베이터와 동기화 되어 버튼을 누르면 한 대의 엘리베이터만 와야 한다.
- 엘리베이터가 도착하지 않은 상태에서 각 층의 문은 절대 열리면 안 된다.
- 엘리베이터의 문과 각 층의 문은 항상 동시에 열려야 한다.

• Elevator System

Use Case



- ***System Design***

• System Design

Design Discussion (개선 중점사항)

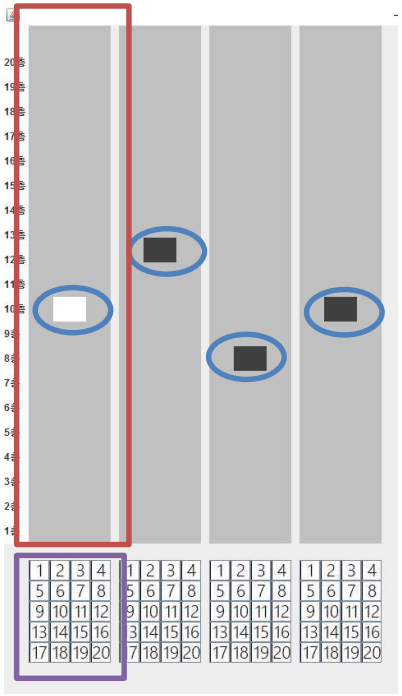
1. 확장성을 고려한 시스템 설계 및 코드 재사용
   → **함수의 제거**

2. 너무 많은 기능을 고려한 시스템 설계
   → **중요한 기능에 대해 만족하는 시스템을 설계**

3. 오토마타의 지나친 단순화
   → **재사용하지 않고 펼쳐진 형태의 오토마타 설계**

• System Design

Template 구성



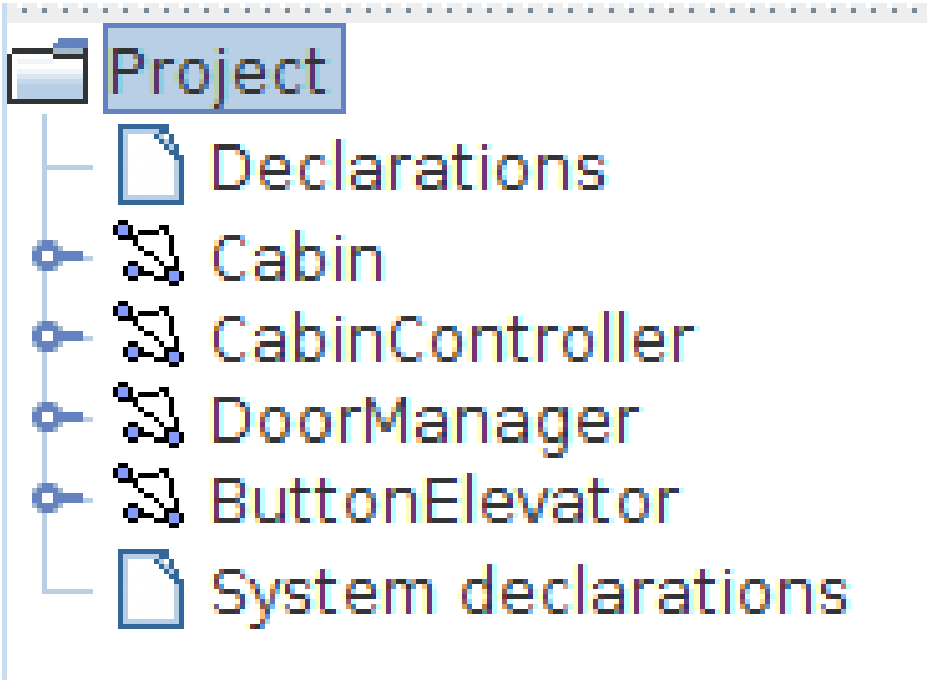
Cabin + CabinController

DoorManager

ButtonElevator

• System Design

Template 구성



- Declarations
  - 변수 정의
- Cabin
  - Cabin 오토마타 정의
- CabinController
  - Cabin을 제어하는 오토마타 정의
- DoorManager
  - 복도측 문을 제어하는 오토마타 정의
- ButtonElevator
  - 엘리베이터 내부의 버튼을 제어

- # System Design

declarations

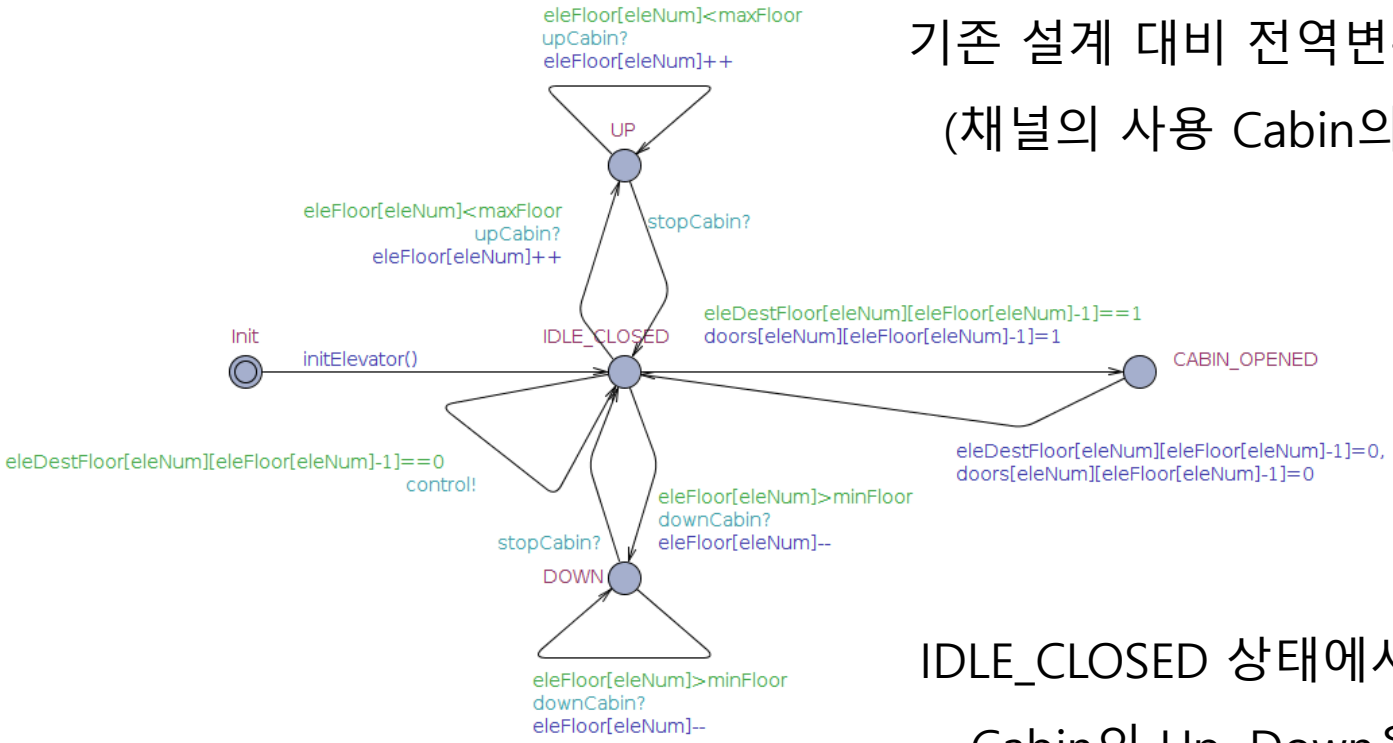
```
const int numberEle = 4;
const int maxFloor = 5;
const int minFloor = 1;
int[minFloor, maxFloor] eleFloor[numberEle];
int[0,1] eleDestFloor[numberEle][maxFloor];
int[0,1] doors[numberEle][maxFloor];
```

기존에 생성했던 다양한 **전역 변수** 및 **calcEle 함수 제거**

다양한 채널 추가 및 함수를 오토마타 형태로 변경

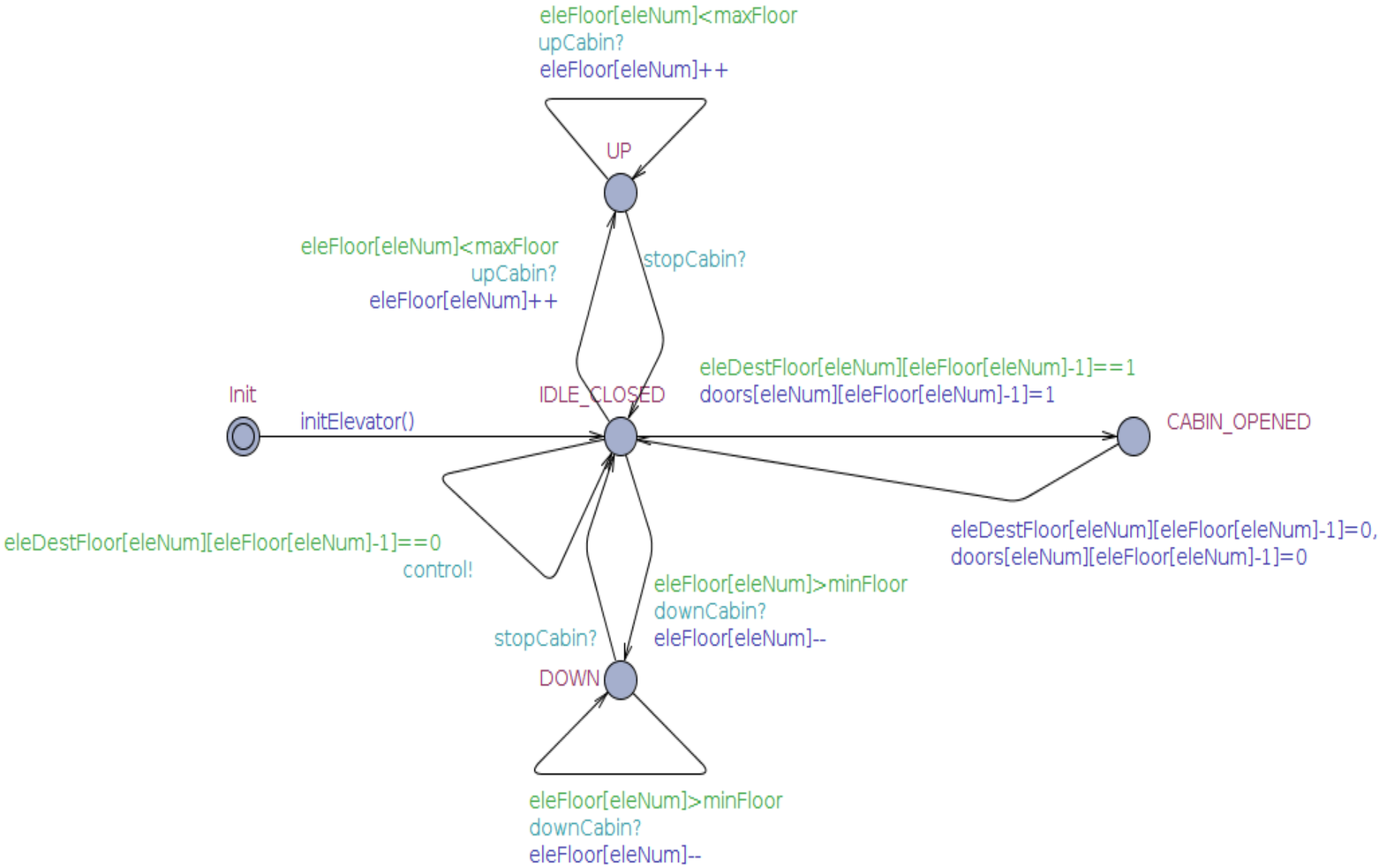
• System Design

Cabin int eleNum, chan &upCabin, chan &downCabin, chan &stopCabin, chan &control



기존 설계 대비 전역변수(배열) 사용을 줄임
(채널의 사용 Cabin의 up, down을 구현)

IDLE_CLOSED 상태에서 control 채널을 통해
Cabin의 Up, Down을 Control하도록 함
(기존의 sysCal 함수 제거)


eleFloor[eleNum]<maxFloor
upCabin?
eleFloor[eleNum]++
UP
eleFloor[eleNum]<maxFloor
upCabin?
eleFloor[eleNum]++
stopCabin?
eleDestFloor[eleNum][eleFloor[eleNum]-1]==1
doors[eleNum][eleFloor[eleNum]-1]=1
Init
initElevator()
IDLE_CLOSED
CABIN_OPENED
eleDestFloor[eleNum][eleFloor[eleNum]-1]==0
control!
eleDestFloor[eleNum][eleFloor[eleNum]-1]=0,
doors[eleNum][eleFloor[eleNum]-1]=0
eleFloor[eleNum]>minFloor
downCabin?
eleFloor[eleNum]--
stopCabin?
DOWN
eleFloor[eleNum]>minFloor
downCabin?
eleFloor[eleNum]--

- # System Design

CabinController(지난 주)

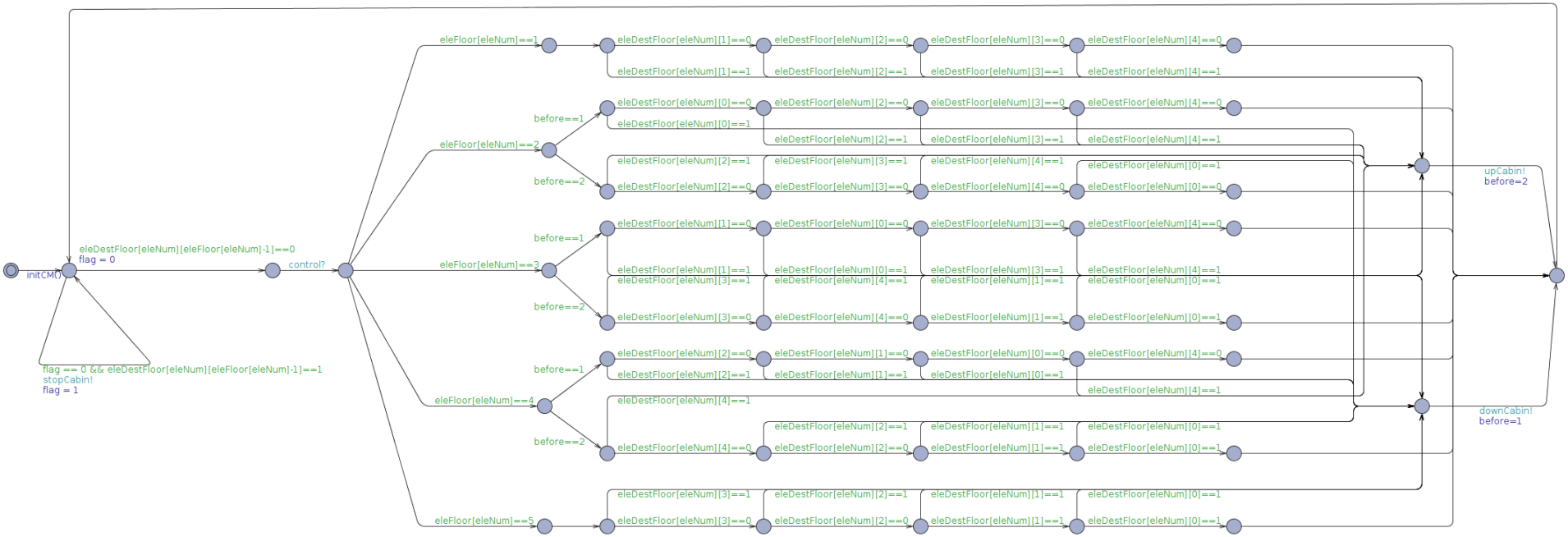
```
void calcEle(int[0,numberEle] num){
    int[-1,maxFloor+1] i=0;
    int[0,10] count1=0;
    int[0,10] count2=0;
    if(eleDestFloor[num][eleFloor[num]]==1){
        direction[num]=STANDBY;
        return;
    }
    for(i=eleFloor[num];i<=maxFloor;i++){
        if(eleDestFloor[num][i]==1){
            count1++;
        }
    }
    for(i=eleFloor[num];i>=minFloor;i--){
        if(eleDestFloor[num][i]==1){
            count2++;
        }
    }
    if(count1==0&&count2==0){
        direction[num]=STANDBY;
        return;
    }
    if(count1==0&&direction[num]==UPWARD){
        direction[num]=STANDBY;
    }
    if(count2==0&&direction[num]==DOWNWARD){
        direction[num]=STANDBY;
    }
    if(count1>0&&direction[num]!=DOWNWARD){
        direction[num]=UPWARD;
    }
    if(count2>0&&direction[num]!=UPWARD){
        direction[num]=DOWNWARD;
    }
    if(directionSave[num]==DOWNWARD && count2>0){
        direction[num]=DOWNWARD;
    }

    if(directionSave[num]==UPWARD && count1>0){
        direction[num]=UPWARD;
    }
}
```

- # System Design

CabinController

int eleNum, chan &upCabin, chan &downCabin, chan &stopCabin, chan &control



Cabin의 현재 층과 목적지에 따른 Cabin의 위, 아래 이동과

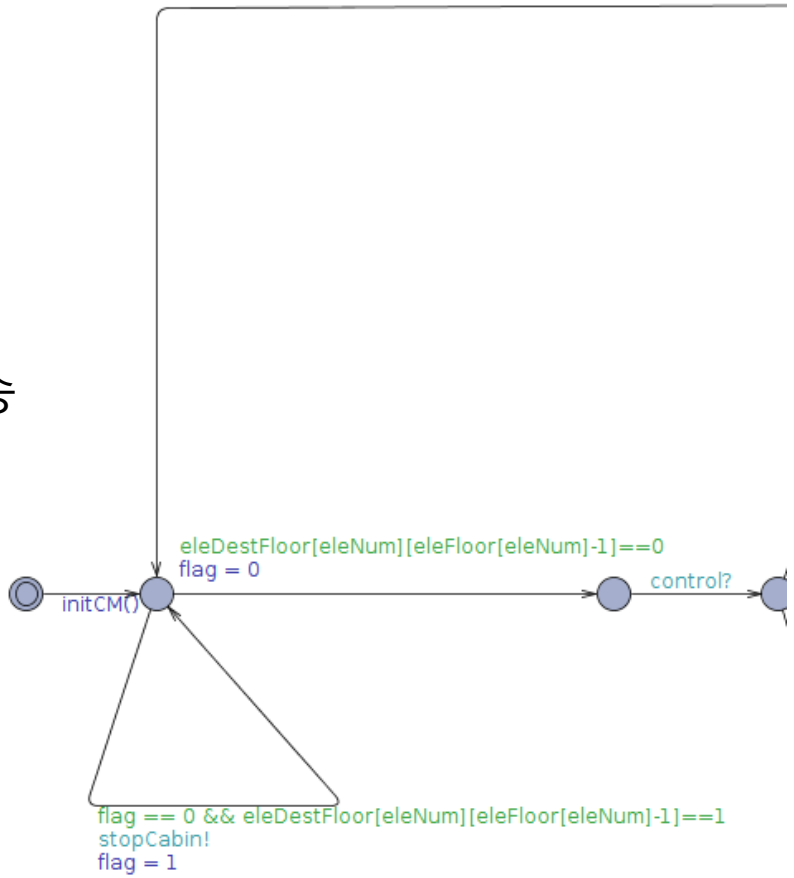
Cabin의 Stop을 채널을 통해 송신함

• System Design

CabinController

Cabin의 현재 층과, 목적지를 고려하여
목적지에 도달했을 때의 Stop 신호를 전송

계속 움직여야 하는 상태라면
계속해서 State를 거치면서
위, 아래 방향을 결정

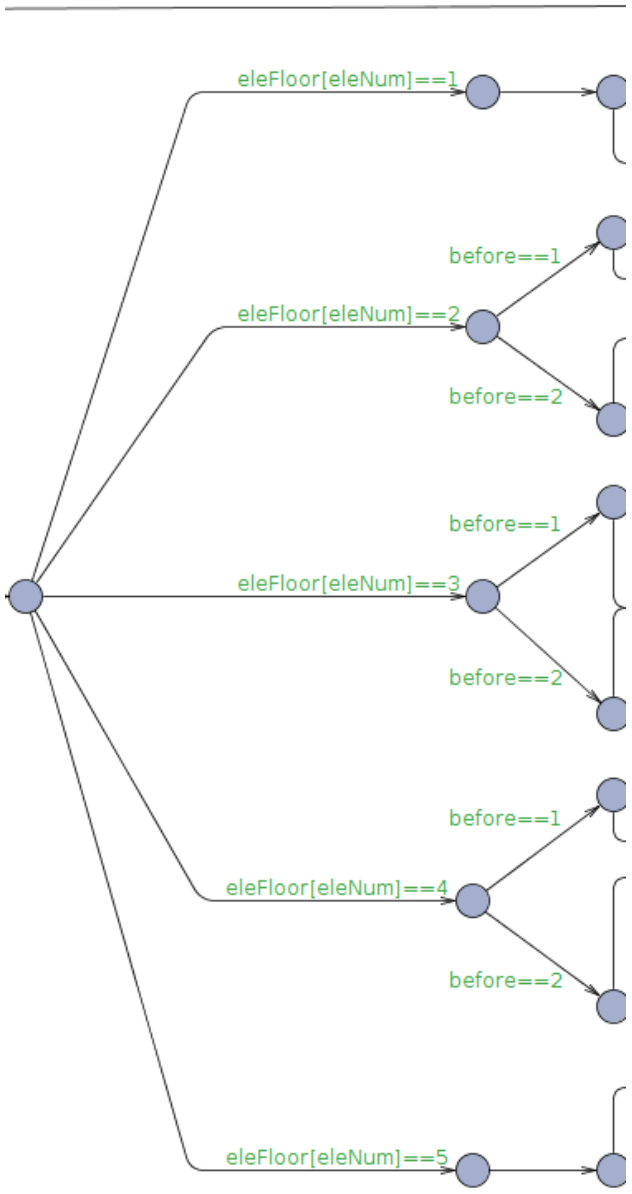


• System Design

CabinController

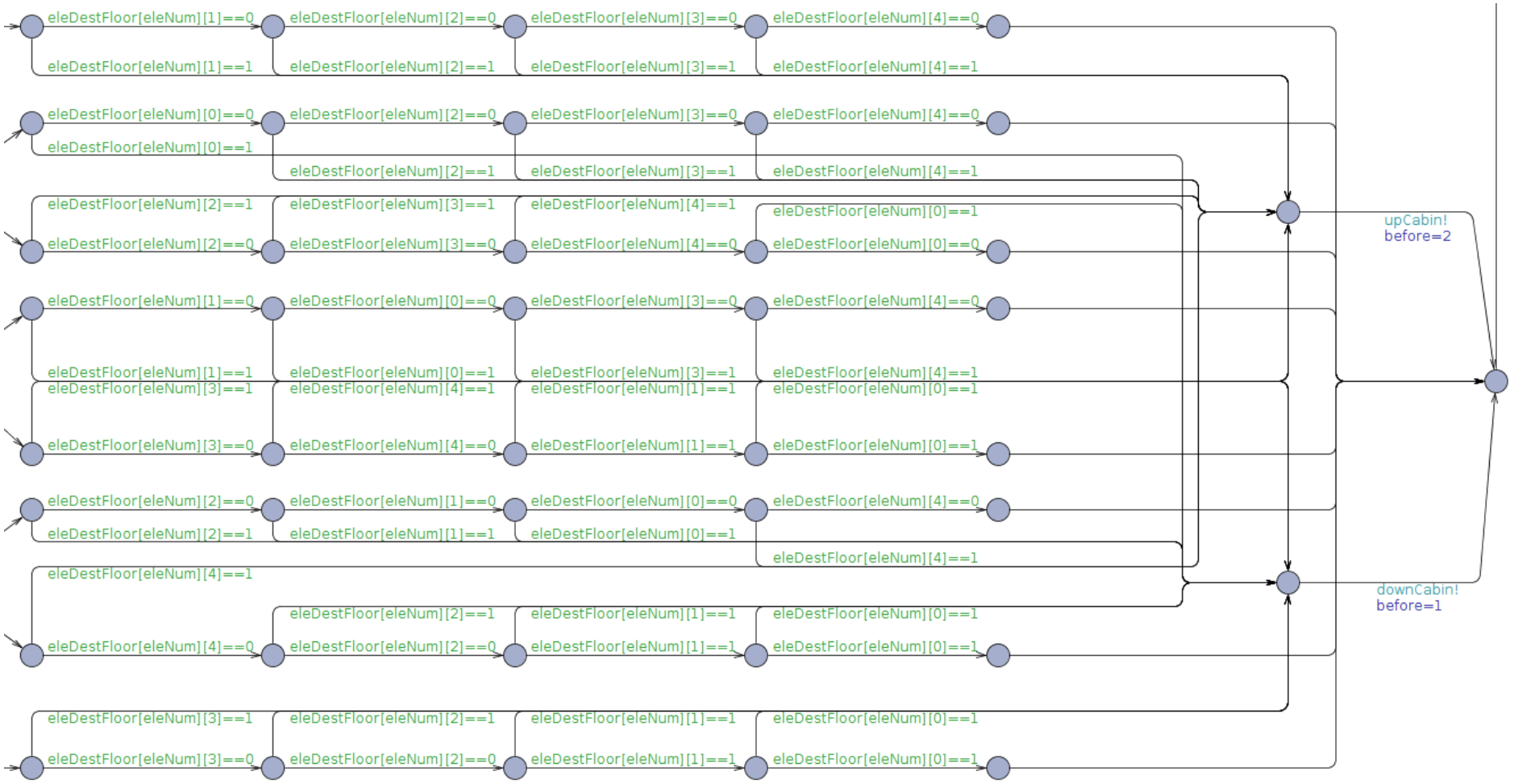
Cabin의 현재 층과, 기존의 이동경로를 고려하여 State를 분기함

기존의 이동경로는 로컬변수를 사용



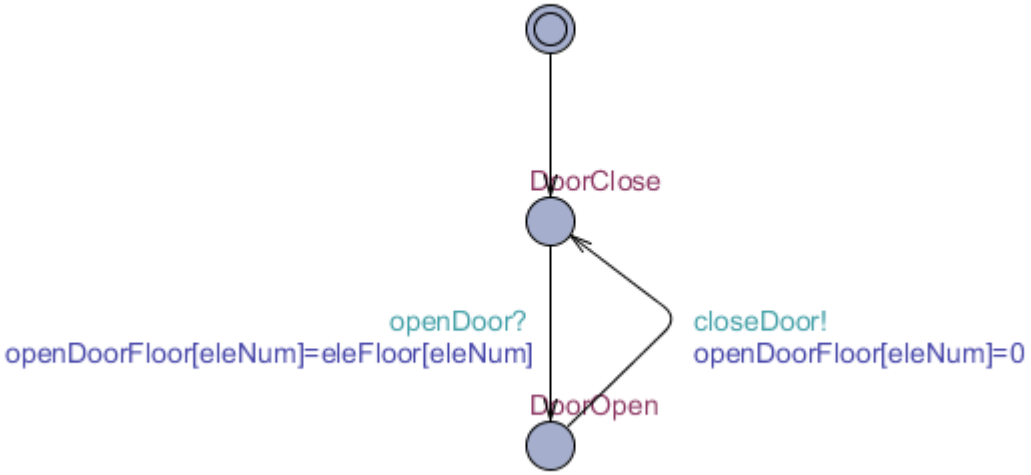
- # System Design

CabinController



- # System Design

DoorManager(지난 주)

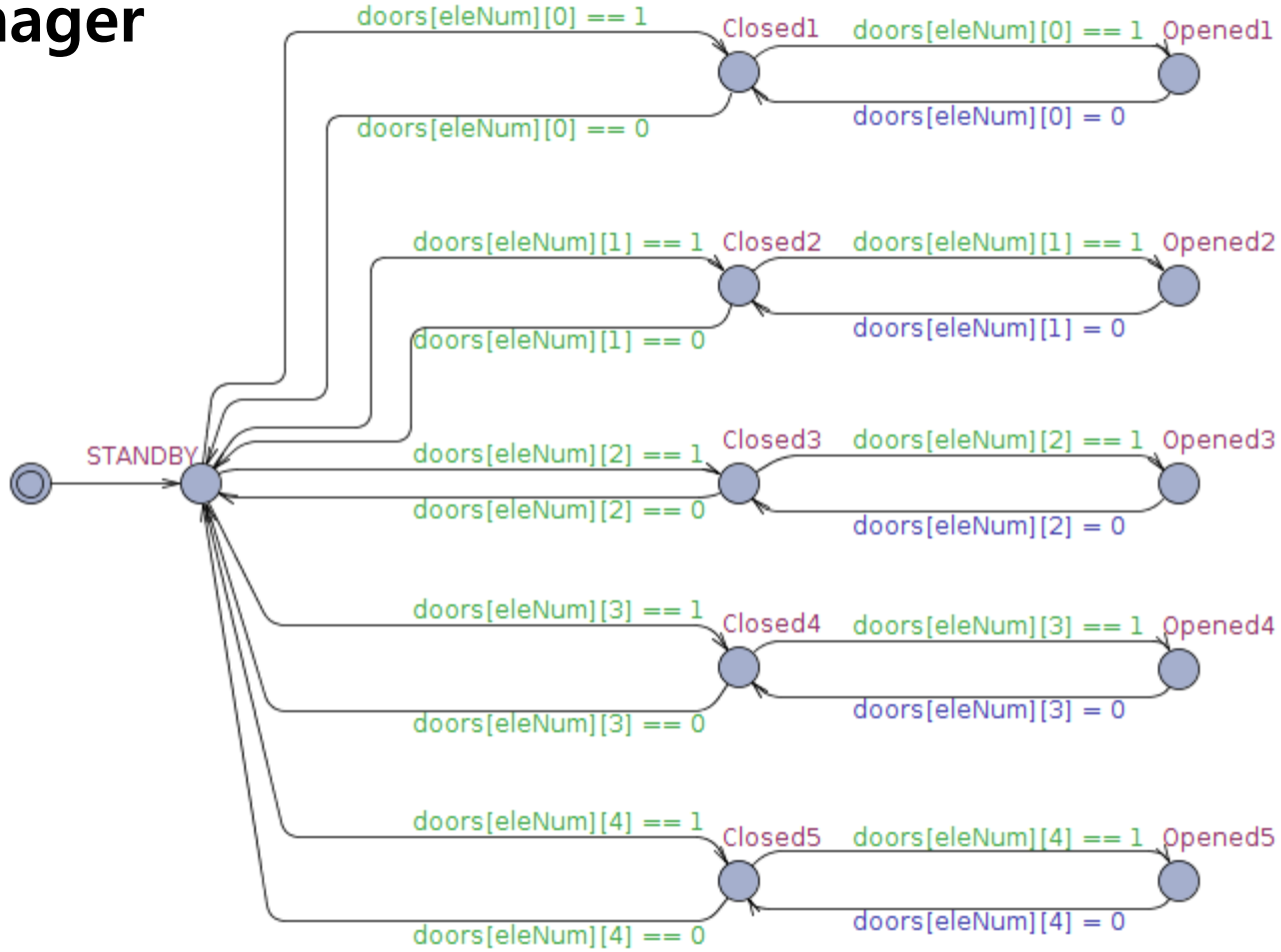


기존 버전의 DoorManager 오토마타

(오토마타의 재사용을 위한 단순한 형태의 오토마타)

- # System Design

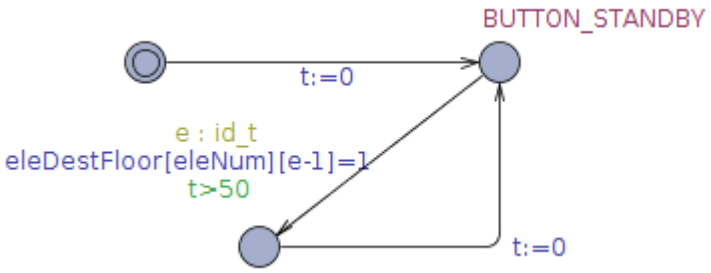
DoorManager



한 Line의 모든 문들을 관리하는 오토마타

• System Design

ElevatorButton



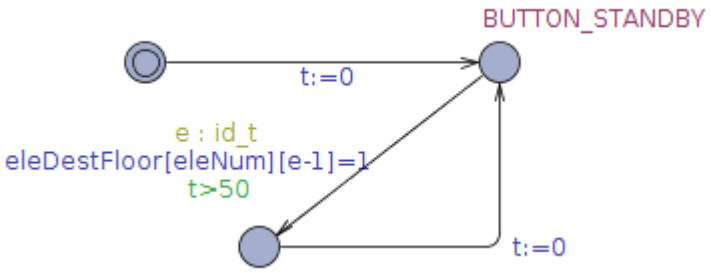
각 엘리베이터 내의 버튼 기능을 수행함

임의의 하나의 버튼을 눌러 엘리베이터를 작동 시킬 수 있도록 설계함

clock을 t로 선언하여 버튼 delay를 주는 데 사용함

• System Design

ElevatorButton



1층부터 5층까지 있다고 가정했을 때,
5!이상의 경우의 수가 존재함

4개의 엘리베이터라면 (5!)^4 이상의 경우의 수 발생함

State Explosion 문제 발생

→ 시뮬레이션에만 사용

• System Design

System Declaration

```
chan openDoor[numberEle][maxFloor];
chan closeDoor[numberEle][maxFloor];
chan upCabin[numberEle];
chan downCabin[numberEle];
chan stopCabin[numberEle];
chan control[numberEle];
cabin1 = Cabin(0, upCabin[0], downCabin[0], stopCabin[0], control[0]);
door1 = DoorManager(0);
cabinmanager1 = CabinController(0, upCabin[0], downCabin[0], stopCabin[0], control[0]);
button1 = ButtonElevator(0);

cabin2 = Cabin(1, upCabin[1], downCabin[1], stopCabin[1], control[1]);
door2 = DoorManager(1);
cabinmanager2 = CabinController(1, upCabin[1], downCabin[1], stopCabin[1], control[1]);
button2 = ButtonElevator(1);

cabin3 = Cabin(2, upCabin[2], downCabin[2], stopCabin[2], control[2]);
door3 = DoorManager(2);
cabinmanager3 = CabinController(2, upCabin[2], downCabin[2], stopCabin[2], control[2]);
button3 = ButtonElevator(2);

cabin4 = Cabin(3, upCabin[3], downCabin[3], stopCabin[3], control[3]);
door4 = DoorManager(3);
cabinmanager4 = CabinController(3, upCabin[3], downCabin[3], stopCabin[3], control[3]);
button4 = ButtonElevator(3);

system
    cabin1, door1, cabinmanager1, button1,
    cabin2, door2, cabinmanager2, button2,
    cabin3, door3, cabinmanager3, button3,
    cabin4, door4, cabinmanager4, button4
;
```

4개의 Cabin과 CabinController에 각 채널을 할당

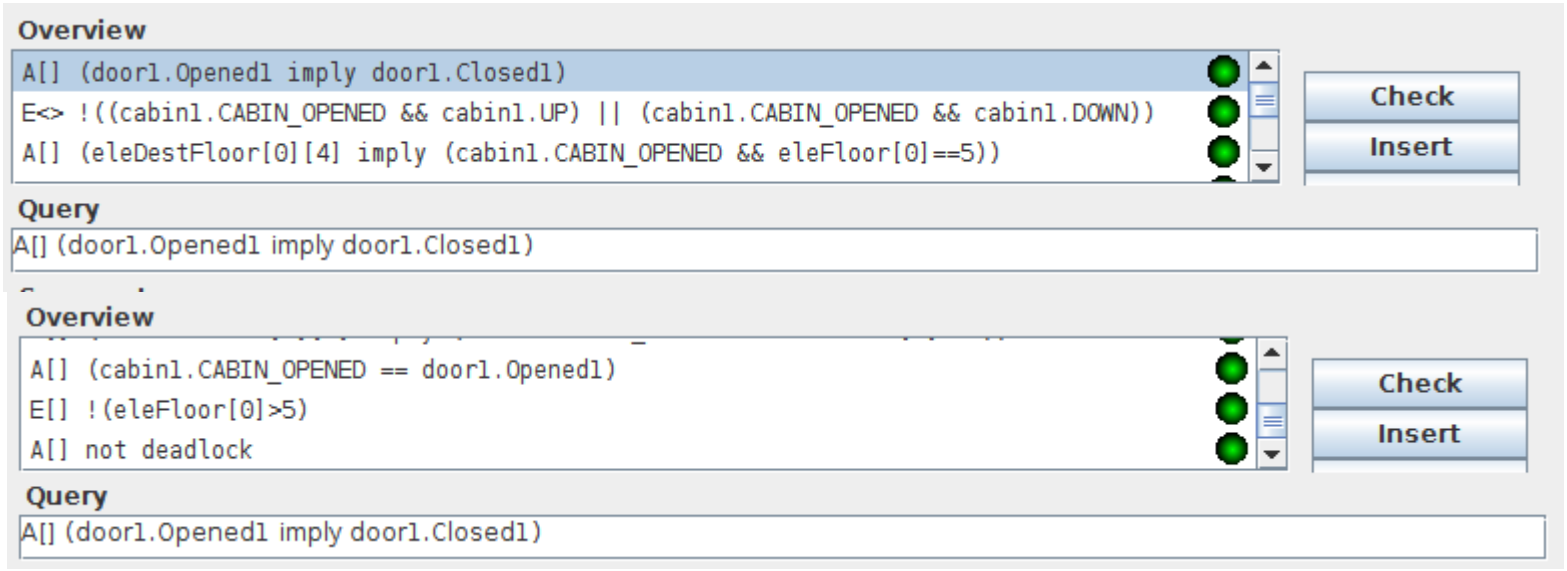
4개의 door를 정의

시뮬레이션을 위한 4개의 button 정의

- ***Demo Play***

- ***System Verification***

• System Verification



- # System Verification

A[] not deadlock

데드락이 발생되지 않는다

True

A[] not deadlock
Verification/kernel/elapsed time used: 6.33s / 0.65s / 6.991s.
Resident/virtual memory usage peaks: 108,892KB / 136,900KB.
Property is satisfied.

- **System Verification**

E[] !(eleFloor[0]>5)

5층을 초과해서 운행하지 않는다

True

```
E[] !(eleFloor[0]>5)
Verification/kernel/elapsed time used: 0s / 0s / 0.001s.
Resident/virtual memory usage peaks: 12,288KB / 47,552KB.
Property is satisfied.
```

- # System Verification

A[] (cabin1.CABIN_OPENED == door1.Opened1)

엘레베이터 문이 열릴때는 복도 측 도어도 함께 열린다

True

```
A[] (cabin1.CABIN_OPENED == door1.Opened1)
Verification/kernel/elapsed time used: 4.36s / 0.7s / 5.065s.
Resident/virtual memory usage peaks: 109,016KB / 138,064KB.
Property is satisfied.
```

• System Verification

A[] (eleDestFloor[0][4] imply (cabin1.CABIN_OPENED && eleFloor[0]==5))

목적지가 5층이라면 엘레베이터가 5층에 도착해서
문이 열리는 경우가 반드시 있다

True

```
A[] (eleDestFloor[0][4] imply (cabin1.CABIN_OPENED && eleFloor[0]==5))
Verification/kernel/elapsed time used: 1.78s / 0.01s / 1.79s.
Resident/virtual memory usage peaks: 109,028KB / 138,072KB.
Property is satisfied.
```

• System Verification

E<> !((cabin1.CABIN_OPENED && cabin1.UP) ||
(cabin1.CABIN_OPENED && cabin1.DOWN))

엘레베이터 문이 열리는 경우엔 엘레베이터가 항상 정지해 있는다.

(엘레베이터 문이 열리는데 엘레베이터가 올라가거나 내려가는 경우는 없다)

True

```
E<> !((cabin1.CABIN_OPENED && cabin1.UP) || (cabin1.CABIN_OPENED && cabin1.DOWN))
Verification/kernel/elapsed time used: 0s / 0s / 0s.
Resident/virtual memory usage peaks: 109,028KB / 138,072KB.
Property is satisfied.
```

- **System Verification**

A[] (door1.Opened1 imply door1.Closed1)

문이 열리면 반드시 문이 다시 닫힌다

True

```
A[] (door1.Opened1 imply door1.Closed1)
Verification/kernel/elapsed time used: 1.89s / 0s / 1.899s.
Resident/virtual memory usage peaks: 109,028KB / 138,072KB.
Property is satisfied.
```

Thank you

Question & Answer