Smart Elevator

Advanced Software Engineering

2017.05.02

KONKUK UNIVERSITY ITCS

노은방, 심우진

CONTENTS

Overall Description

- The Number of Elevator – 4, Elevator Floor – 5.
- 사용자가 Floor에서 엘리베이터를 요청했을 때 적합한 엘리베이터가 해당 Floor로 가도록 한다.
- 엘리베이터에 탑승한 사용자가 선택한 Floor에는 가능한 빠른 시간 내에 도착해야 한다.
- 1층 로비에서 사용자가 원하는 층을 선택하면, 동적으로 가장 빠른 엘리베이터가 도착하도록 한다.

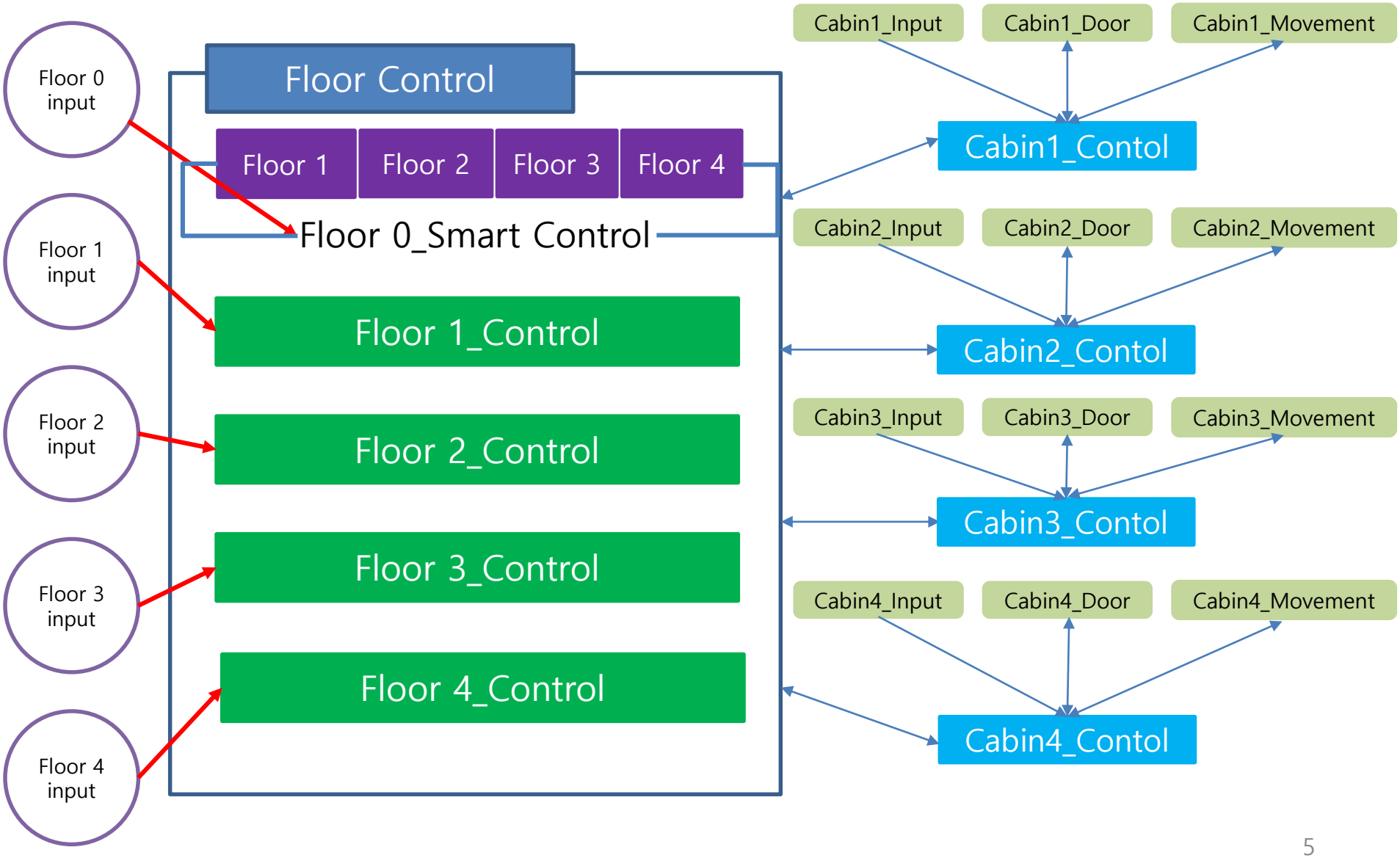
Elevator Component

4 Cabin

Controller(Floor, Cabin)

20 Door(4 x 5)

Structure of Elevator



Implementaion Code

Cabin1_Input

```
proctype cab1_input(){
    if
    ::(cab1_floor!=0) -> cab1_input_num[0] = true
    ::(cab1_floor!=1) -> cab1_input_num[1] = true
    ::(cab1_floor!=2) -> cab1_input_num[2] = true
    ::(cab1_floor!=3) -> cab1_input_num[3] = true
    ::(cab1_floor!=4) -> cab1_input_num[4] = true
    fi
}
```

Implementaion Code

Cabin1_Door

```
proctype cab1_door(byte i; chan openClose)
{
    cab1_doorClose:
    do
    ::  atomic{openClose?eval(i),OPEN} goto cab1_doorOpen
    ::  atomic{openClose?eval(i),CLOSE} goto cab1_doorClose
    od

    cab1_doorOpen:
    do
    ::  atomic{openClose?eval(i),OPEN} goto cab1_doorOpen
    ::  atomic{openClose?eval(i),CLOSE} goto cab1_doorClose
    od
}
```

Implementaion Code

Cabin1_Movement

```
proctype cab1_movement(chan upDown)
{
    cab1_floor0:
    do
    :: atomic{upDown?UP; cab1_state=UP; cab1_floor = 1; goto cab1_floor1}
    :: atomic{upDown?DOWN; cab1_state=STOP; cab1_floor = 0; goto cab1_floor0}
    od

    cab1_floor1:
    do
    :: atomic{upDown?UP; cab1_state=UP; cab1_floor = 2; goto cab1_floor2}
    :: atomic{upDown?DOWN; cab1_state=DOWN; cab1_floor = 0; goto cab1_floor0}
    od

    cab1_floor2:
    do
    :: atomic{upDown?UP; cab1_state=UP; cab1_floor = 3; goto cab1_floor3}
    :: atomic{upDown?DOWN; cab1_state=DOWN; cab1_floor = 1; goto cab1_floor1}
    od

    cab1_floor3:
    do
    :: atomic{upDown?UP; cab1_state=UP; cab1_floor = 4; goto cab1_floor4}
    :: atomic{upDown?DOWN; cab1_state=DOWN; cab1_floor = 2; goto cab1_floor2}
    od

    cab1_floor4:
    do
    :: atomic{upDown?UP; cab1_state=STOP; cab1_floor = 4; goto cab1_floor4}
    :: atomic{upDown?DOWN; cab1_state=DOWN; cab1_floor = 3; goto cab1_floor3}
    od
}
```

Implementaion Code

Cabin1_Control

Cab1_waiting:

```
proctype cab1_control(chan openClose, upDown)
{
    cab1_waiting:

    if
    ::(cab1_input_num[0]==false && cab1_input_num[1]==false && cab1_input_num[2]==false
                               && cab1_input_num[3]==false && cab1_input_num[4]==false)->
        cab1_state = STOP; goto cab1_waiting
    ::else-> goto cab1_process
    fi
```

Implementaion Code

Cabin1_Control

Cab1_process : Cabin에서 0층 누르고, State(UP, Down, Stop)에 따라 분류, 해당 층 별로 다시 CASE화

우선순위 1 : 동일한 층의 입력이 들어온 경우 문을 열고 닫음

```
::(cab1_input_num[0]==true && cab1_state==UP && cab1_floor==0)->
    goto cab1_on0
::(cab1_input_num[0]==true && cab1_state==DOWN && cab1_floor==0) ->
    goto cab1_on0
::(cab1_input_num[0]==true && cab1_state==STOP && cab1_floor==0) ->
    goto cab1_on0
::(cab1_input_num[1]==true && cab1_state==UP && cab1_floor==1) ->
    goto cab1_on1
::(cab1_input_num[1]==true && cab1_state==DOWN && cab1_floor==1) ->
    goto cab1_on1
::(cab1_input_num[1]==true && cab1_state==STOP && cab1_floor==1) ->
    goto cab1_on1
```

Implementaion Code

Cabin1_Control

Cab1_process:

우선순위 2 : 들어온 입력에서 이동 방향이 동일하거나 멈춘 상태에서 바로 다음 층에서 멈추는 경우

```
if
::(cab1_input_num[0]==true && cab1_state==DOWN && cab1_floor==1) ->
    atomic{upDown!DOWN} -> goto cab1_on0
::(cab1_input_num[0]==true && cab1_state==STOP && cab1_floor==1) ->
    atomic{upDown!DOWN} -> goto cab1_on0

::(cab1_input_num[1]==true && cab1_state==UP && cab1_floor==0) ->
    atomic{upDown!UP}; goto cab1_on1
//0층이라서 특수한 케이스
::(cab1_input_num[1]==true && cab1_state==DOWN && cab1_floor==0) ->
    atomic{upDown!UP}; goto cab1_on1
::(cab1_input_num[1]==true && cab1_state==STOP && cab1_floor==0) ->
    atomic{upDown!UP}; goto cab1_on1
::(cab1_input_num[1]==true && cab1_state==DOWN && cab1_floor==2) ->
//꼭대기 이전층이라 특수한 케이스
::(cab1_input_num[3]==true && cab1_state==UP && cab1_floor==4) ->
    atomic{upDown!DOWN}; goto cab1_on3
::(cab1_input_num[3]==true && cab1_state==DOWN && cab1_floor==4) ->
    atomic{upDown!DOWN}; goto cab1_on3
::(cab1_input_num[3]==true && cab1_state==STOP && cab1_floor==4) ->
    atomic{upDown!DOWN}; goto cab1_on3
```

Implementaion Code

Cabin1_Control

Cab1_process:

우선순위 3 : 이동방향 동일하고 멈춰 있고, 한층 이상 떨어진 경우

```
::(cab1_input_num[1]==true && cab1_state==DOWN && cab1_floor>=3) ->
    goto cab1_down_not_stop
::(cab1_input_num[1]==true && cab1_state==STOP && cab1_floor>=3) ->
    goto cab1_down_not_stop

::(cab1_input_num[2]==true && cab1_state==UP && cab1_floor<=0) ->
    goto cab1_up_not_stop
//0층이라서 특수한 케이스
::(cab1_input_num[2]==true && cab1_state==DOWN && cab1_floor<=0) ->
    goto cab1_up_not_stop
::(cab1_input_num[2]==true && cab1_state==STOP && cab1_floor<=0) ->
    goto cab1_up_not_stop
//꼭대기층이라 특수한 케이스
::(cab1_input_num[2]==true && cab1_state==UP && cab1_floor>=4) ->
    goto cab1_down_not_stop
::(cab1_input_num[2]==true && cab1_state==DOWN && cab1_floor>=4) ->
    goto cab1_down_not_stop
::(cab1_input_num[2]==true && cab1_state==STOP && cab1_floor>=4) ->
    goto cab1_down_not_stop
```

Implementaion Code

Cabin1_Control

Cab1_process:

우선순위 4 : 방향이 반대이며, 바로 이전 층인 경우

```
if
::(cab1_input_num[0]==true && cab1_state==UP && cab1_floor==1
        && cab1_input_num[2]==false && cab1_input_num[3]==false
        && cab1_input_num[4]==false)->
    atomic{upDown!DOWN}; goto cab1_on0

::(cab1_input_num[1]==true && cab1_state==UP && cab1_floor==2
        && cab1_input_num[3]==false && cab1_input_num[4]==false)->
    atomic{upDown!DOWN}-> goto cab1_on1

::(cab1_input_num[2]==true && cab1_state==UP && cab1_floor==3
        && cab1_input_num[4]==false) ->
    atomic{upDown!DOWN}; goto cab1_on2
::(cab1_input_num[2]==true && cab1_state==DOWN && cab1_floor==1
        && cab1_input_num[0]==false) ->
    atomic{upDown!UP}; goto cab1_on2

::(cab1_input_num[3]==true && cab1_state==DOWN && cab1_floor==2
        && cab1_input_num[0]==false && cab1_input_num[1]==false) ->
    atomic{upDown!UP}; goto cab1_on3

::(cab1_input_num[4]==true && cab1_state==DOWN && cab1_floor==3
        && cab1_input_num[0]==false && cab1_input_num[1]==false
        && cab1_input_num[2]==false) ->
    atomic{upDown!UP}; goto cab1_on4
```

Implementaion Code

Cabin1_Control

Cab1_process:

우선순위 5 : 방향이 반대이며, 층도 한층 이상 나는 경우

```
::else->
    //방향도 반대, 층도 한층이상 나는 경우
    if
    ::(cab1_input_num[0]==true && cab1_state==UP && cab1_floor>=2
            && cab1_input_num[3]==false && cab1_input_num[4]==false)->
        goto cab1_down_not_stop
    ::(cab1_input_num[1]==true && cab1_state==UP && cab1_floor>=3
            && cab1_input_num[4]==false)->
        goto cab1_down_not_stop
    ::(cab1_input_num[3]==true && cab1_state==DOWN && cab1_floor<=1
            && cab1_input_num[0]==false) ->
        goto cab1_up_not_stop
    ::(cab1_input_num[4]==true && cab1_state==DOWN && cab1_floor<=2
            && cab1_input_num[0]==false && cab1_input_num[1]==false) ->
        goto cab1_up_not_stop
    ::else -> goto cab1_waiting
    fi
```

Implementaion Code

Cabin1_Control

Cab1_process:

컨트롤 입력 처리 중 입력이 더 없는지 동적으로 체크.

```
:: (cab1_input_num[0]==false && cab1_input_num[1]==false && cab1_input_num[2]==false
                             && cab1_input_num[3]==false && cab1_input_num[4]==false)->
    cab1_state=STOP; goto cab1_waiting
//동적 체크
```

Implementaion Code

Floor1~4_input

```
proctype floor1_input(){
    if
    ::(floor1_up_input != true) -> floor1_up_input=true
    ::(floor1_down_input != true) -> floor1_down_input=true
    fi
}
proctype floor2_input(){
    if
    ::(floor2_up_input != true) -> floor2_up_input=true
    ::(floor2_down_input != true) -> floor2_down_input=true
    fi
}
proctype floor3_input(){
    if
    ::(floor3_up_input != true) -> floor3_up_input=true
    ::(floor3_down_input != true) -> floor3_down_input=true
    fi
}
proctype floor4_input(){

    (floor4_down_input != true) -> floor4_down_input=true

}
```

Implementaion Code

Floor0_input

```
proctype floor0_input(){

    if
    ::(floor0_input_floor1 != true) -> floor0_input_floor1=true
    ::(floor0_input_floor2 != true) -> floor0_input_floor2=true
    ::(floor0_input_floor3 != true) -> floor0_input_floor3=true
    ::(floor0_input_floor4 != true) -> floor0_input_floor4=true
    fi
}
```

Implementaion Code

Floor1_Control

Floor1_waiting :

```
floor1_waiting:

if
::(floor1_up_input==false && floor1_down_input==false) ->
    goto floor1_waiting
::else -> goto floor1_process
fi
```

Implementaion Code

Floor1_Control

Floor1_process :

1층에 바로 엘리베이터가 멈춰 있는 경우

```
if
//1층에 바로 엘베가 멈춰있던 경우
::(floor1_up_input==true && cab1_state==STOP && cab1_floor==1)->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
::(floor1_down_input==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_down_input=false; goto floor1_waiting
```

Implementaion Code

Floor1_Control

Floor1_process :

1층으로 가기 한 층 전이거나, 한층 차이의 상태에서 멈춰 있는 엘리베이터

```
//1층에서 올라가고자하고, 0층에 엘베가 있는 경우
::(floor1_up_input==true && cab1_state==UP && cab1_floor==0) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
//0층이기에 특수한 케이스
::(floor1_up_input==true && cab1_state==DOWN && cab1_floor==0) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
::(floor1_up_input==true && cab1_state==STOP && cab1_floor==0) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting

//1층에서 올라가고자하고, 2층에 엘베가 있는 경우
::(floor1_up_input==true && cab1_state==UP && cab1_floor==2
        && cab1_input_num[3]==false && cab1_input_num[4]==false) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
::(floor1_up_input==true && cab1_state==DOWN && cab1_floor==2) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
::(floor1_up_input==true && cab1_state==STOP && cab1_floor==2) ->
    cab1_input_num[1]=true; (cab1_floor==1)->
    floor1_up_input=false; goto floor1_waiting
```

Implementaion Code

Floor1_Control

Floor1_process :

2층 차이나는 경우

```
::else ->
    //2층차이나는 경우
    if
    //1층에서 올라가고자하고, 3층에 엘베 있는 경우
    ::(floor1_up_input==true && cab1_state==UP && cab1_floor==3
            && cab1_input_num[4]==false) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting
    ::(floor1_up_input==true && cab1_state==DOWN && cab1_floor==3) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting
    ::(floor1_up_input==true && cab1_state==STOP && cab1_floor==3) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting

    //1층에서 내려가고자하고, 3층에 엘베 있는 경우
    ::(floor1_down_input==true && cab1_state==UP && cab1_floor==3
            && cab1_input_num[4]==false) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
    ::(floor1_down_input==true && cab1_state==DOWN && cab1_floor==3) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
    ::(floor1_down_input==true && cab1_state==STOP && cab1_floor==3) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
```

Implementaion Code

Floor1_Control

Floor1_process :

3층 차이나는 경우

```
::else ->
    //3층 차이나는 경우
    if
    //꼭대기층이라 특수한 케이스
    ::(floor1_up_input==true && cab1_state==UP && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting
    ::(floor1_up_input==true && cab1_state==DOWN && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting
    ::(floor1_up_input==true && cab1_state==STOP && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_up_input=false; goto floor1_waiting

    //꼭대기층이라 특수한 케이스
    ::(floor1_down_input==true && cab1_state==UP && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
    ::(floor1_down_input==true && cab1_state==DOWN && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
    ::(floor1_down_input==true && cab1_state==STOP && cab1_floor==4) ->
        cab1_input_num[1]=true; (cab1_floor==1) ->
        floor1_down_input=false; goto floor1_waiting
```

Implementaion Code

Floor2_Control

Floor2_process :

2층으로 가기 한 층 전이거나, 한층 차이의 상태에서 멈춰 있는 엘리베이터

```
if
//2층에서 올라가고자하고, 1층에 엘베가 있는 경우
::(floor2_up_input==true && cab1_state==UP && cab1_floor==1) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==DOWN && cab1_floor==1
        && cab1_input_num[0]==false) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting

//2층에서 올라가고자하고, 3층에 엘베가 있는 경우
::(floor2_up_input==true && cab1_state==UP && cab1_floor==3
        && cab1_input_num[4]==false) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==DOWN && cab1_floor==3) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==STOP && cab1_floor==3) ->
    cab1_input_num[2]=true; (cab1_floor==2)->
    floor2_up_input=false; goto floor2_waiting
```

Implementaion Code

Floor2_Control

Floor2_process :

2층 차이나는 경우

```
if
//2층에서 올라가고자하고, 0층에 엘베 있는 경우
::(floor2_up_input==true && cab1_state==UP && cab1_floor==0) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting
//0층이라 특수한 케이스
::(floor2_up_input==true && cab1_state==DOWN && cab1_floor==0) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==STOP && cab1_floor==0) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting

//2층에서 올라가고자하고, 4층에 엘베 있는 경우
::(floor2_up_input==true && cab1_state==UP && cab1_floor==4) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==DOWN && cab1_floor==4) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting
::(floor2_up_input==true && cab1_state==STOP && cab1_floor==4) ->
    cab1_input_num[2]=true; (cab1_floor==2) ->
    floor2_up_input=false; goto floor2_waiting
```

Implementaion Code

Floor3_Control

Floor3_process :

3층에서 올라가고자 하고 2층에 엘리베이터가 있는 경우 및
3층에서 올라가고자 하고, 4층에 엘리베이터가 있는 경우

```
if
//3층에서 올라가고자하고, 2층에 엘베가 있는 경우
::(floor3_up_input==true && cab1_state==UP && cab1_floor==2) ->
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting
::(floor3_up_input==true && cab1_state==DOWN && cab1_floor==2
        && cab1_input_num[1]==false && cab1_input_num[0]==false) ->
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting
::(floor3_up_input==true && cab1_state==STOP && cab1_floor==2) ->
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting

//꼭대기층이라 특수한 케이스
::(floor3_up_input==true && cab1_state==UP && cab1_floor==4) ->
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting
::(floor3_up_input==true && cab1_state==STOP && cab1_floor==4) ->
    cab1_input_num[3]=true; (cab1_floor==3)->
    floor3_up_input=false; goto floor3_waiting
```

Implementaion Code

Floor3_Control

Floor3_process :

2층 차이나는 경우

```
if
//3층에서 올라가고자하고, 1층에 엘베 있는 경우
::(floor3_up_input==true && cab1_state==UP && cab1_floor==1) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_up_input=false; goto floor3_waiting
::(floor3_up_input==true && cab1_state==DOWN && cab1_floor==1
        && cab1_input_num[0]==false) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_up_input=false; goto floor3_waiting
::(floor3_up_input==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_up_input=false; goto floor3_waiting

//3층에서 내려가고자하고, 1층에 엘베 있는 경우
::(floor3_down_input==true && cab1_state==UP && cab1_floor==1) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_down_input=false; goto floor3_waiting
::(floor3_down_input==true && cab1_state==DOWN && cab1_floor==1
        && cab1_input_num[0]==false) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_down_input=false; goto floor3_waiting
::(floor3_down_input==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[3]=true; (cab1_floor==3) ->
    floor3_down_input=false; goto floor3_waiting
```

Implementaion Code

Floor4_Control

Floor4_process :

4층에서 한층 전이거나, 한층 차이 있는 상태에서 멈춰 있는 엘리베이터

```
:: else ->
    //4층으로 가기 한층전이거나, 한층차이난 상태에서 멈춰있는 엘베
    if
    //4층에서 내려가고자하고, 3층에 엘베가 있는 경우
    ::(floor4_down_input==true && cab1_state==UP && cab1_floor==3) ->
        cab1_input_num[4]=true; (cab1_floor==4)->
        floor4_down_input=false; goto floor4_waiting
    ::(floor4_down_input==true && cab1_state==DOWN && cab1_floor==3
            && cab1_input_num[2]==false && cab1_input_num[1]==false
            && cab1_input_num[0]==false) ->
        cab1_input_num[4]=true; (cab1_floor==4)->
        floor4_down_input=false; goto floor4_waiting
    ::(floor4_down_input==true && cab1_state==STOP && cab1_floor==3) ->
        cab1_input_num[4]=true; (cab1_floor==4)->
        floor4_down_input=false; goto floor4_waiting
```

Implementaion Code

Floor4_Control

Floor4_process :

2층 차이나는 경우

```
if
//4층에서 내려가고자하고, 2층에 엘베 있는 경우
::(floor4_down_input==true && cab1_state==UP && cab1_floor==2) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==DOWN && cab1_floor==2
        && cab1_input_num[1]==false && cab1_input_num[0]==false) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==STOP && cab1_floor==2) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
```

Implementaion Code

Floor4_Control

Floor4_process :

3층 차이나는 경우

```
if
//4층에서 내려가고자하고, 1층에 엘베 있는 경우
::(floor4_down_input==true && cab1_state==UP && cab1_floor==1) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==DOWN && cab1_floor==1
        && cab1_input_num[0]==false) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
```

Implementaion Code

Floor4_Control

Floor4_process :

4층 차이나는 경우

```
if
//4층에서 내려가고자하고, 0층에 엘베 있는 경우
::(floor4_down_input==true && cab1_state==UP && cab1_floor==0) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==DOWN && cab1_floor==0) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
::(floor4_down_input==true && cab1_state==STOP && cab1_floor==0) ->
    cab1_input_num[4]=true; (cab1_floor==4) ->
    floor4_down_input=false; goto floor4_waiting
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 1층으로 가는 버튼에 대한 컨트롤러

```
proctype floor0_smart_to1_control()
{
    //0층에서 1층 가는 버튼에 대한 컨트롤러
    floor0_to1_waiting:

    if
    ::(floor0_input_floor1==false) ->
        goto floor0_to1_waiting
    ::else -> goto floor0_to1_process
    fi

    floor0_to1_process:
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 가장 가까운 순서대로만 고려하고, 도착한 후 해당하는 층에 true 해준다.

```
if
//0층까지 가장 가까운 순서대로만 고려하고, 도착한 후 해당하는 층에 true 해준다
//0층에 이미 멈춰있는 경우
::(floor0_input_floor1==true && cab1_state==STOP && cab1_floor==0)->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting

::(floor0_input_floor1==true && cab2_state==STOP && cab2_floor==0)->
    cab2_input_num[0]=true; (cab2_floor==0)-> cab2_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting

::(floor0_input_floor1==true && cab3_state==STOP && cab3_floor==0)->
    cab3_input_num[0]=true; (cab3_floor==0)-> cab3_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting

::(floor0_input_floor1==true && cab4_state==STOP && cab4_floor==0)->
    cab4_input_num[0]=true; (cab4_floor==0)-> cab4_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 1층 눌렀고, 엘리베이터가 1층에 있는 경우

```
if
::(floor0_input_floor1==true && cab1_state==UP && cab1_floor==1
        && cab1_input_num[2]==false && cab1_input_num[3]==false
        && cab1_input_num[4]==false) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==DOWN && cab1_floor==1) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==STOP && cab1_floor==1) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 1층 눌렀고, 엘리베이터가 2층에 있는 경우

```
if
//0층에서 1층 누르고, 2층에 엘베 있는 경우
::(floor0_input_floor1==true && cab1_state==UP && cab1_floor==2
    && cab1_input_num[3]==false && cab1_input_num[4]==false) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==DOWN && cab1_floor==2) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==STOP && cab1_floor==2) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 1층 눌렀고, 엘리베이터가 3층에 있는 경우

```
if
//0층에서 1층 누르고, 3층에 엘베 있는 경우
::(floor0_input_floor1==true && cab1_state==UP && cab1_floor==3
    && cab1_input_num[4]==false) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==DOWN && cab1_floor==3) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==STOP && cab1_floor==3) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
```

Implementaion Code

Floor0_Smart Control

Floor0_smart_to1_process :

0층에서 1층 눌렀고, 엘리베이터가 4층에 있는 경우

```
if
//0층에서 1층 누르고, 4층에 엘베 있는 경우
::(floor0_input_floor1==true && cab1_state==UP && cab1_floor==4) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==DOWN && cab1_floor==4) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
::(floor0_input_floor1==true && cab1_state==STOP && cab1_floor==4) ->
    cab1_input_num[0]=true; (cab1_floor==0)-> cab1_input_num[1]=true;
    floor0_input_floor1==false; goto floor0_to1_waiting
//이도저도 아니면 적합한 엘베 나타날때까지 반복
::else -> goto floor0_to1_process
fi
```

LTL Properties

1. 엘리베이터는 0층에서 5층까지 움직인다.

- ltl p1 {[] ((cab1_floor>=0) & (cab1_floor<=4))}

```
spin -a elevator_v5.pml
ltl p1: [] (((cab1_floor>=0)&(cab1_floor<=4)))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 7884
Depth=     513 States=    1e+06 Transitions= 8.67e+06 Memory=   294.129 t=    6.81 R=   1e+05
Depth=     654 States=    2e+06 Transitions= 1.75e+07 Memory=   523.328 t=    13.5 R=   1e+05
Depth=     654 States=    3e+06 Transitions= 2.64e+07 Memory=   752.625 t=    20.9 R=   1e+05
Depth=     654 States=    4e+06 Transitions= 3.52e+07 Memory=   981.824 t=    28.3 R=   1e+05
pan: reached -DMEMLIM bound
            1.07365e+09 bytes used
            102400 bytes more needed
            1.07374e+09 bytes limit
hint: to reduce memory, recompile with
  -DCOLLAPSE # good, fast compression, or
  -DMA=412   # better/slower compression, or
  -DHC # hash-compaction, approximation
  -DBITSTATE # supertrace, approximation

(Spin Version 6.4.6 -- 2 December 2016)
Warning: Search not completed
            + Partial Order Reduction

Full statespace search for:
            never claim         - (not selected)
            assertion violations +
            acceptance   cycles  + (fairness disabled)
            invalid end states +

State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
 36847832 transitions (= stored+matched)
       49 atomic steps
hash conflicts:   1541706 (resolved)
```

LTL Properties

2. Cabin에서 눌린 층에는 언젠가 도착한다.

- ltl p2 {<> ((cab1_input_num[0]==1) -> (cab1_floor==0 && cab1_input_num[0]==0))}
- ltl p2 {<> ((cab1_input_num[1]==1) -> (cab1_floor==1 && cab1_input_num[1]==0))}
- ltl p2 {<> ((cab1_input_num[2]==1) -> (cab1_floor==2 && cab1_input_num[2]==0))}
- ltl p2 {<> ((cab1_input_num[3]==1) -> (cab1_floor==3 && cab1_input_num[3]==0))}
- ltl p2 {<> ((cab1_input_num[4]==1) -> (cab1_floor==4 && cab1_input_num[4]==0))}

```
ltl p2: <> ((! ((cab1_input_num[0]==1))) || (((cab1_floor==0)) && ((cab1_input_num[0]==0))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

3. 문이 열린 다음에는 닫힌다.

- ltl p3 {[] ((cab1_door@cab1_doorOpen) -> (cab1_door@cab1_doorClose))}

```
spin -a  elevator  v5.pml
ltl p3: [] ((! ((cab1_door@cab1_doorOpen))) || ((cab1_door@cab1_doorClose)))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
 36847832 transitions (= stored+matched)
```

LTL Properties

4. 문이 열린 상태에선 움직이지 않는다.

- ltl p4 {[] !((cab1_door@cab1_doorOpen) -> ((cab1_movement@cab1_floor0->cab1_movement@cab1_floor1) || (cab1_movement@cab1_floor1->cab1_movement@cab1_floor2) || (cab1_movement@cab1_floor2->cab1_movement@cab1_floor3) || (cab1_movement@cab1_floor3->cab1_movement@cab1_floor4)))}

```
ltl p4: [] (! ((! ((cab1_door@cab1_doorOpen))) || (((((! ((cab1_movement@cab1_floor0))) || ((cab1_
movement@cab1_floor1))) || ((! ((cab1_movement@cab1_floor1))) || ((cab1_movement@cab1_flo
or2)))) || ((! ((cab1_movement@cab1_floor2))) || ((cab1_movement@cab1_floor3)))) || ((! ((cab1_m
ovement@cab1_floor3))) || ((cab1_movement@cab1_floor4))))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

5. O층에서 요청하면 도착하고, O층에서 누른 층에 도착한다.

- ltl p5 {[](floor0_input_floor1==true -> <>(cab1_floor==0 || cab2_floor==0 || cab3_floor==0 || cab4_floor==0) -> <>(cab1_floor==1 || cab2_floor==1 || cab3_floor==1 || cab4_floor==1))}

```
ltl p5: [] ((! ((! ((floor0_input_floor1==1))) || (<> (((((cab1_floor==0)) || ((cab2_floor==0))) || ((cab3_flo
or==0))) || ((cab4_floor==0)))))) || (<> (((((cab1_floor==1)) || ((cab2_floor==1))) || ((cab3_floor==1)))
|| ((cab4_floor==1)))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 4384
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

6. 층에서 들어온 입력을 순차적으로 처리한다.

- ltl p6 {[](((cab1_floor==0) && (cab1_input_num[2]==true) && (cab1_input_num[1]==true)) -> ((cab1_floor==1)->(cab1_floor==2)))}
- ltl p6 {[](((cab1_floor==1) && (cab1_input_num[2]==true) && (cab1_input_num[3]==true)) -> ((cab1_floor==2)->(cab1_floor==3)))}
- ltl p6 {[](((cab1_floor==4) && (cab1_input_num[1]==true) && (cab1_input_num[3]==true)) -> ((cab1_floor==3)->(cab1_floor==1)))}

```
ltl p6: [] ((! ((((cab1_floor==0)) && ((cab1_input_num[2]==1))) && ((cab1_input_num[1]==1)))) || ((!
((cab1_floor==1))) || ((cab1_floor==2))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 4368
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

7. 층에서 입력이 들어오면 언젠가 해당 층에 도착한다.

- ltl p7 {<>((floor1_up_input==true || floor1_down_input==true) -> (cab1_floor==1 || cab2_floor==1 || cab3_floor==1 || cab4_floor==1))}
- ltl p7 {<>((floor2_up_input==true || floor2_down_input==true) -> (cab1_floor==2 || cab2_floor==2 || cab3_floor==2 || cab4_floor==2))}
- ltl p7 {<>((floor3_up_input==true || floor3_down_input==true) -> (cab1_floor==3 || cab2_floor==3 || cab3_floor==3 || cab4_floor==3))}
- ltl p7 {<>((floor4_up_input==true || floor4_down_input==true) -> (cab1_floor==4 || cab2_floor==4 || cab3_floor==4 || cab4_floor==4))}

```
ltl p7: <> ((! (((floor1_up_input==1)) || ((floor1_down_input==1)))) || (((((cab1_floor==1)) || ((cab2_flo
or==1))) || ((cab3_floor==1))) || ((cab4_floor==1))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 8488
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

8. 아무런 입력이 없다면 엘리베이터는 멈춘다.

- ltl p8 {[](((cab1_input_num[0]==0) && (cab1_input_num[1]==0) && (cab1_input_num[2]==0) && (cab1_input_num[3]==0) && (cab1_input_num[4]==0) && (floor0_input_floor1==0) && (floor0_input_floor2==0) && (floor0_input_floor3==0) && (floor0_input_floor4==0) && (floor1_up_input==0) && (floor2_up_input==0) && (floor3_up_input==0) && (floor1_down_input==0) && (floor2_down_input==0) && (floor3_down_input==0) && (floor4_down_input)) -> (cab1_state==STOP))}

```
ltl p8: [] ((! ((((((((((((((((cab1_input_num[0]==0)) && ((cab1_input_num[1]==0))) && ((cab1_input_num[2]==0))) && ((cab1_input_num[3]==0))) && ((cab1_input_num[4]==0))) && ((floor0_input_floor1==0))) && ((floor0_input_floor2==0))) && ((floor0_input_floor3==0))) && ((floor0_input_floor4==0))) && ((floor1_up_input==0))) && ((floor2_up_input==0))) && ((floor3_up_input==0))) && ((floor1_down_input==0))) && ((floor2_down_input==0))) && ((floor3_down_input==0))) && (floor4_down_input))) || ((cab1_state==STOP)))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 4196

State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

9. Cabin은 올라가거나 내려가는 상태에서 언젠가 멈추게 된다.

- ltl p9 {[](((cab1_state==UP) || (cab1_state==DOWN)) -> (cab1_state==STOP))}

```
ltl p9: [] ((! (((cab1_state==UP)) || ((cab1_state==DOWN)))) || ((cab1_state==STOP)))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 7048
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

LTL Properties

9. Cabin은 올라가거나 내려가는 상태에서 언젠가 멈추게 된다.

- ltl p10 {<>((cab1_control@cab1_up_not_stop || cab1_control@cab1_down_not_stop) ->(cab1_control@cab1_on0 || cab1_control@cab1_on1 || cab1_control@cab1_on2 || cab1_control@cab1_on3 || cab1_control@cab1_on4))}

```
ltl p10: [] ((! (((cab1_control@cab1_up_not_stop)) || ((cab1_control@cab1_down_not_stop)))) || (<>
 ((((((cab1_control@cab1_on0)) || ((cab1_control@cab1_on1))) || ((cab1_control@cab1_on2))) || ((
cab1_control@cab1_on3))) || ((cab1_control@cab1_on4)))))
gcc-4 -DMEMLIM=1024 -O2 -DXUSAFE -DNOCLAIM -w -o pan pan.c
./pan -m10000  -a
Pid: 3632
```

```
State-vector 412 byte, depth reached 654, errors: 0
  4183738 states, stored
 32664094 states, matched
```

Simulate

영상 대체

THANK YOU