System Verification Lab 2122 - APP V - A.Y. 2021/2022

Giorgio Paoletti mat. 119927

Lift System

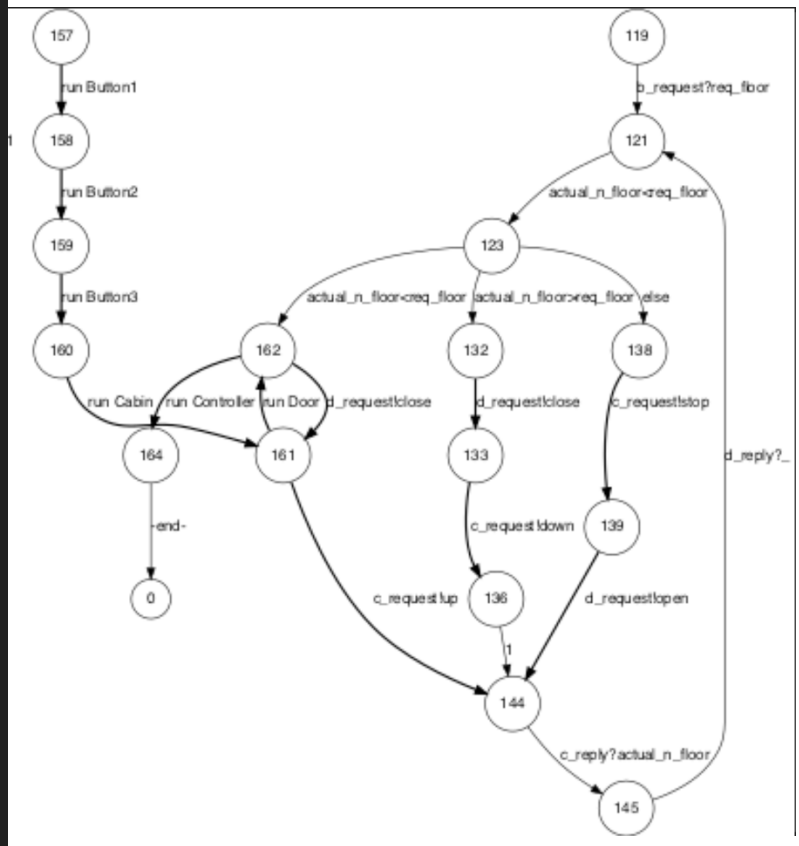
In order to realize the scenario, each component has been defined as a process that can communicate through a channel system composed of a request and a reply channel. This has been done to create a contest in which if a process makes a request it has to wait for a reply to continue its computation.

Following is the list of all processes:

- Button
- Door
- Cabin
- Controller

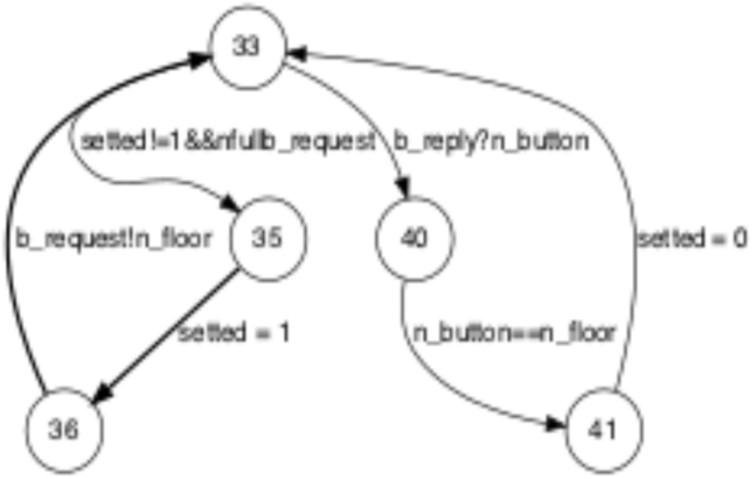
The main process is the Controller whose aim is to manage the request of all instances of the Button process and to send commands to the Cabin process and Door process.
Following the code made in Promela where we can see a do loop in which the process waits for the button request and depending on which floor has been requested the Controller sent a request of opening or closing to the Door process and a direction command to the Cabin. Following the automata generated by using JSpin.

```
/* Controller process */
proctype Controller(){

    int req_floor;
    int actual_n_floor = 1;

end:do

    :: b_request ? req_floor ->
        do
        :: actual_n_floor < req_floor ->
            if
            :: actual_n_floor < req_floor ->
                atomic{
                    d_request ! close;
                    c_request ! up;
                }


            :: actual_n_floor > req_floor ->
                atomic{
                    d_request ! close;
                    c_request ! down;
                }

atReqFlor:  :: else
                atomic{
                    c_request ! stop;
                    d_request ! open;
                }

            fi

            c_reply ? actual_n_floor;
            d_reply ? _;

        od

        b_reply ! req_floor

    od
}
```



After that has been defined the Button process that specifies how an instance of it can non-deterministically generate a floor request.
The Process is composed of a Do loop that manages the non-deterministically pression of the button linked to a specified floor and a request made by the controller to reset the status of the instance.
In addition, during the specification of the request channel has been chosen a buffered channel in a way to represent the booking of the floor. Following the code and automata generated by using JSpin.

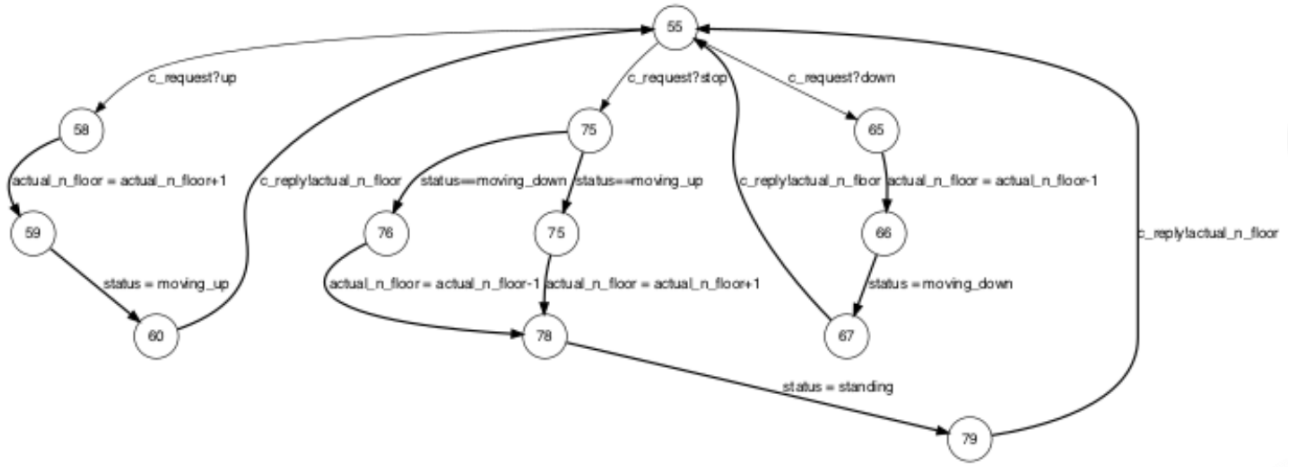
```
/* Button process */
proctype Button(int n_floor){

    int n_button;
    bool setted = false;

end:do

    :: (setted != true  && nfull(b_request)) ->
pt:     atomic{
            setted = true;
            b_request ! n_floor;
        }

    :: b_reply ? n_button ->
       (n_button == n_floor) ->
pf:         setted = false;

    od

}
```

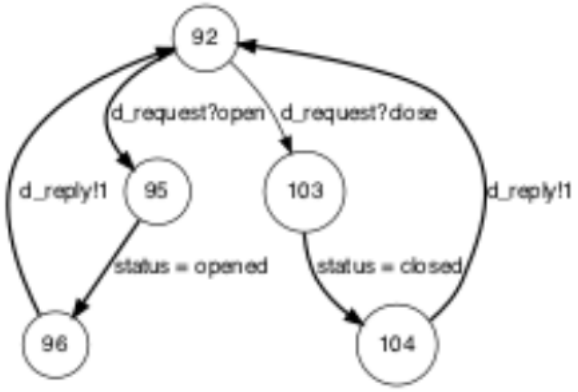


Finally, has been defined the Door and the Cabin that has the same structure depending on the number of the status. The process is made with a Do loop that's aim is to manage the request of the controller and reply in a way to say that the request has been approved. Following the code and automata generated by using JSpin.

```
proctype Cabin(){

    mtype status = standing;
    int actual_n_floor;

end:do

    :: c_request ? up ->

       atomic {
           actual_n_floor++;
cs_up:     status = moving_up;
           c_reply ! actual_n_floor;
       }

    :: c_request ? down ->
       atomic {
           actual_n_floor--;
cs_dw:     status = moving_down;
           c_reply ! actual_n_floor;
       }

    :: c_request ? stop ->


       atomic{
           if
           :: status == moving_up -> actual_n_floor++;
           :: status == moving_down -> actual_n_floor--;
           fi;
cs_s:      status = standing;
           c_reply ! actual_n_floor;
       }

    od
}
```

```
/* Door process */
proctype Door(){

    mtype status = closed;

end:do

    :: d_request ? open ->

ds_o:   atomic{
            status = opened;
            d_reply ! true;
        }


    :: d_request ? close ->

ds_c:   atomic{
            status = closed;
            d_reply ! true;
        }


    od
}
```





In order to verify some properties of the lift system has been defined some LTL properties. Following the formalization using remote references, that has been used to refer to control points in correctness specifications.

1. Whenever the door is open the cabin must be standing.

   ltl ltl1 { [](Door@ds_o -> Cabin@cs_s) }

   With Door@ds_o as Door opened and Cabin@cs_s as Cabin standing.

2. Whenever the cabin is moving the door must be closed.

   ltl ltl2 { [](((Cabin@cs_up || Cabin@cs_dw) -> Door@ds_c) }

   With Door@ds_c as Door closed, Cabin@cs_up as Cabin moving up and Cabin@cs_dw as Cabin moving down.

3. A button cannot remain pressed forever.

   ltl ltl3 { [] (Button@pt -> <> Button@pf) }

   With Button@pt as Button pressed and Button@pf as Button unpressed.

4. The door cannot remain open forever.

   ltl ltl4 { [] (Door@ds_o -> <> Door@ds_c) }

With Door@ds_o as Door opened and Door@ds_c as Door closed.

5. The door cannot remain closed forever.

```
ltl ltl5 { [] (Door@ds_c -> <> Door@ds_o) }
```

6. Whenever the button at floor x (x = 1, 2, 3) becomes pressed then the cabin will eventually be at floor x with the door open.

```
ltl ltl6 { [] (Button@pt -> <> (Controller@atReqFlor && Door@ds_o)) }
```

With Controller@atReqFlor as the lift has arrived at the request floor.

7. Whenever no button is currently pressed and the button at floor x (x = 1, 2, 3) becomes pressed and, afterwards, also the button at floor y (y =! x and y = 1, 2, 3) becomes pressed and, in the meanwhile, no other button becomes pressed then the cabin will be standing at floor x with the door open and, afterwards, the cabin will be standing at floor y with the door open and in the meanwhile the cabin will not be standing at any other floor different from y with the door open.

```
ltl ltl7 {
   [] ((((!Button[x]@pt && !Button[y]@pt && !Button[z]@pt)
            U Button[x]@pt && !Button[y]@pt && !Button[z]@pt)
               U Button[y]@pt && !Button[z]@pt) ->
                  <> ((Cabin@cs_s && Door@ds_o) -> Button[x]@pf)
                     U ((Cabin@cs_s && Door@ds_o) -> Button[y]@pf))
}
```

With Button[x]@pt as Instance of process Button with _pid equal to x with status pressed. Following a screen with the result of the verification made by Spin

```
warning: never claim + accept labels requires -a flag to fully verify
(Spin Version 6.5.2 -- 6 December 2019)
                + Partial Order Reduction
Full statespace search for:
                never claim                   + (ltl7)
                assertion violations          + (if within scope of claim)
                cycle checks                  - (disabled by -DSAFETY)
                invalid end states            - (disabled by never claim)
State-vector 164 byte, depth reached 85, ••• errors: 0 •••
     519 states, stored
     585 states, matched
    1104 transitions (= stored+matched)
     613 atomic steps
hash conflicts:       0 (resolved)
Stats on memory usage (in Megabytes):
   0.095        equivalent memory usage for states (stored*(State-vector + overhead))
   0.379        actual memory usage for states
 128.000        memory used for hash table (-w24)
   0.107        memory used for DFS stack (-m2000)
 128.400        total actual memory usage
```