

Graded lab assignment: Elevator controller

1 Overview

The NuSMV model for this exercise describes an elevator that can serve three floors. The model includes the physical system (buttons, cabin, door), and the controller. The goal of this exercise is to

1. complete the model by formalizing the transition relations,
2. formalize the given properties as temporal-logic properties, and
3. ensure that the requirements are satisfied.

The base model has to be complete and correct for passing. Furthermore, you are not allowed to make additional assumptions about the initial state of physical components (in particular, the starting floor and door state) if they are not already given by the model template. If your model passes all mandatory features, you can implement optional advanced features for a higher grade.

Note: For the basic features, no fairness conditions are allowed. For some advanced properties, fairness conditions are permitted if stated.

Deliverables: Submit the source (.smv) file(s) with documentation embedded as comments.

2 Base features (required for P)

2.1 Buttons

There is one button for each floor to request the elevator. Each button can be pressed non-deterministically. A pressed button stays active until the controller resets it.

2.2 Cabin

The cabin can be at any floor between 1 and 3. The controller decides the direction, which the engine translates into moving up or down, or stopping.

2.3 Door

The cabin is equipped with a door that can either be open or closed. The door responds to commands “open”, “close”, and “nop”, which cause it to open, close, or stay in its current state. A request to reach a particular floor is *served* when the cabin arrives at that floor (and stops), and the doors are open.

2.4 Controller

The controller causes the cabin to move, the door to open or close, and resets the corresponding button when the elevator has served a request. It takes as input (sensor data) the current floor, the status of the door, the direction in which the cabin is moving, and the status of all buttons.

When you write the controller, be careful that you do not end up with cases where the elevator “bounces” between, for example, floors 2 and 3, even though there is also a request on floor 1 active (see Figure 1 for an example). To avoid this, you are not allowed to use fairness constraints.

2.5 Safety/liveness properties

The following properties have to be specified in temporal logic:

1. The controller must not reset a button that is not pressed.
2. The cabin can move up only if the floor is not N.
3. The cabin can move down only if the floor is not 1.

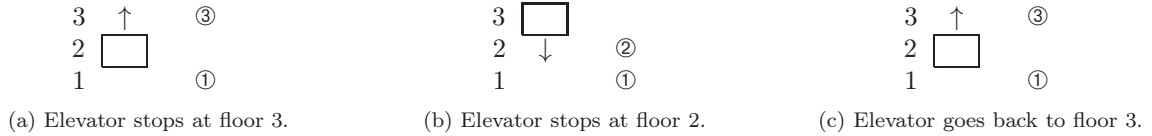


Figure 1: An example where the elevator ends up alternating endlessly between floors 2 and 3.

4. The controller can issue an open command only if the door is closed.
5. The controller can issue a close command only if the door is open.
6. No button can reach a state where it remains pressed forever.
7. No pressed button can be reset until the cabin stops at the corresponding floor and opens the door.
8. A button must be reset as soon as the cabin stops at the corresponding floor with the door open.
9. The cabin can move only when the door is closed.
10. If no button is pressed, the controller must issue no commands and the cabin must be stopped.

You are free to choose LTL or CTL.

3 Tasks for higher grades

Note: It is not necessary to implement all optional features in the *same* model. To work on different features in parallel, you can create multiple models that each implement one or more optional features.

3.1 Secure access

In a secure elevator, a button only responds to a push when a user is authorized to access a given floor. In practice, this may work with a key or keycard, which enables certain buttons; these buttons otherwise appear to be non-functional.

The elevator should work as usual (as specified otherwise) if *access* for a given floor is set, and otherwise never reach that floor.

Modifications: Add variable *access* to the button module; configure the controller such that the buttons are initialized with the extra value being non-deterministic. Add a transition function that ensures that the access never changes over time.¹ Modify the logics of the button module such that it only responds if *access* is set.

Properties: Each floor is only served if access is enabled, or if the elevator happens to start at that floor.

Hints:

1. Define a helper predicate “served_*x*” for each floor to make the properties easier to write.
2. Consider starting with a simpler property: A floor is only served if access is enabled, study the resulting error trace, and extend the property as described above.

3.2 Earthquake safety

The elevator has a sensor that detects strong shaking. In that case, it stops immediately (at the next floor) and opens the doors as soon as the cabin has stopped. It also deactivates all buttons and ignores any button presses. The elevator remains in that state until repair arrives.

An *earthquake* is a non-deterministic event. The corresponding state of the elevator remains set until another input, *repair*, is true. Repair can only reset the elevator state if the door is open. If the elevator door is open and an earthquake happens, repair cannot appear concurrently with the onset of the earthquake.

¹This simplifies the model and its properties. In practice, *access* changes when someone with the right key uses the elevator, but we will only consider time spans in the model where the access permissions of the given users do not change.

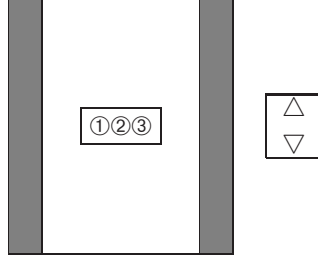


Figure 2: A schematic showing the elevator with open doors, and buttons inside and outside the cabin.

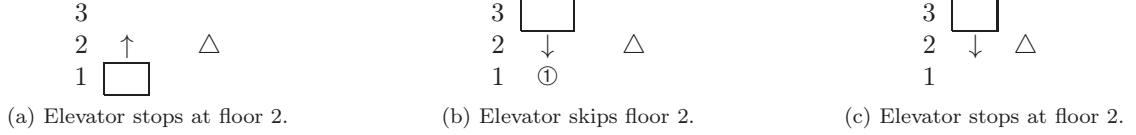


Figure 3: Three scenarios with p_2_up pressed: elevator moving up; elevator moving down with an active request on floor 1; elevator moving down with no other requests active.

Modifications: Add variables *earthquake* and *repair* to the controller. You have to modify the logics of the controller, and possibly other modules, to implement the functionality described above.

Properties:

1. Modify existing properties to allow the controller to reset buttons in case of an earthquake.
2. When an earthquake happens...
 - (a) The elevator stops in the next step.
 - (b) The doors open within a few model transitions. You can use LTL operator $F[\text{min}, \text{max}]$ for this.
 - (c) The elevator remains stopped until repair arrives.

Fairness: Repair eventually arrives.

3.3 Secure access in the presence of earthquakes

If you combine both of the above-mentioned extra features, you will find a conflict between safety and security: The earthquake mode may stop an elevator at a floor where someone normally does not have access. Modify the access-related properties to take this into account and to *allow* this behavior.

3.4 Directional buttons

The initial model is a very simple elevator with just one button (\odot) to stop the elevator whenever it arrives at a given floor. This causes the issue that someone in the second floor may want to return to the first floor, but ends up stopping the elevator when going up, so the elevator stops for no purpose.

In modern elevators, floors in the middle have two buttons, up (\triangle) and down (∇), to ensure the elevator only stops when going in the right direction (see Figure 2). The lowest floor has only one button, up, and the top floor has only one button, down. These two edge cases work like a regular request button; the remainder of this section focuses on up/down buttons. In addition to that, there are still regular request buttons for each floor inside the elevator cabin.

An “up” button outside the cabin has the following semantics, shown in Figure 3:

1. When the elevator moves up, the button has the same effect as the button for that floor inside the elevator cabin. In other words, when the elevator moves up and arrives at the given floor, the elevator is expected to stop and open its doors (see Figure 3a). After that, the button is reset.
2. When the elevator moves down, the cabin passes (see Figure 3b) that floor unless there is no pending request at any lower floor (see Figure 3c). If there is no pending request further down, then the elevator stops at the given floor, opens the doors, and the request (button) is reset.

The “down” button works analogously.

Description	Feature	Property	Points
Secure access is implemented correctly, with correct properties.	medium	medium	+0.75
The earthquake sensor and repair reset are implemented correctly, with at least one correct safety property (if the property is incorrect, the property from our solution will be used to check the model).	medium	easy	+0.5
All properties related to the earthquake sensor are correct.		medium	+0.25
The properties are correct for a combined implementation of earthquake safety and secure access.		difficult	+0.5
Directional buttons are implemented correctly, all old and new properties hold.	difficult	medium	+1.5
Documented error trace of incorrect model.	easy		+0.25
Documented error trace of incorrect property.		easy	+0.25

Table 1: Extra points for advanced features/properties.

Modifications: Add two more inputs, p_2_up and p_2_down , to the controller. Modify the controller to implement the functionality described above. It may be useful to have more “helper definitions” similar to *pending_X* to reduce the amount of code duplication.

Properties: Mandatory properties 7 and 8 need to be copied and adapted for the new buttons. Ensure that a directional floor button only gets reset when the elevator is going in the right direction, or when the cabin is about to change its direction! The final property also needs a minor modification.

3.5 Validation, documentation

An extra half point can be obtained by validating and documenting the model. Commented error traces can be pasted and explained at the bottom of the submission, as a multi-line comment between `--` and `--/`.

Optional: You document an example error trace with a faulty (older?) or mutated *model*. (+0.25)

Optional: You document an example error trace with a faulty (older?) or mutated *property*. (+0.25)

4 Grading criteria

Pass (E):

- The submitted work is your group’s own original work. You may look at examples from the web for inspiration, but you are not allowed to copy code from the web. Note that the given model is a heavily modified version from an example you can find on the web.
- All ten mandatory properties are correct.
- The model correctly implements all ten properties (no error traces found by NuSMV).
- No additional assumptions about the initial state of physical components are made.
- No fairness conditions are used for the mandatory properties and any variables in given the model template.

Extra points are awarded according to Table 1.

Reduced points for late submissions! The maximal number of points is reduced by 1 for each missed deadline. There are two deadlines: for the small lab exercises and for this exercise.