# Correctness by Construction

# Term Project: Simple Elevator

The deadline to turn in the Event B model and related documents is:

**Tuesday, May 21st 2024, 23:59**

The project presentations will take place on:

**Wednesday May 22nd 2024, 15:30-20:00**

The order of presentations is not yet decided. It is likely that we will need to use all the evening.

Manuel Carro
manuel.carro@upm.es

April 27th, 2024

## 1 General Description

The objective of this project is to develop an Event B model for the controller of a simplified elevator.

There is has a button in every level, outside the cabin, to call the elevator. Inside the cabin, there is a panel with buttons (one per level) to request that elevator goes to that level and stops there. The cabin has doors that can be opened and closed.

When a person arrives at the elevator entry, if the cabin is there and the doors are open, they can just enter and press a button to state where they want to go. If the doors are closed, they can call the elevator with the external button. When the elevator arrives, it will open the doors so that they can enter and select their destination.

We will assume that the doors take some time to open and some time to close, and that this is the amount of time that people has to enter the elevator. There are no sensors to detect whether people enter or leave the cabin, and therefore no way of controlling the presence / absence of people in it. Therefore we just need to receive and process signals from the buttons, act upon the doors, and start / stop the elevator.

In order to simplify the design and focus on the controller, solutions only need to make a minimal representation of the environment: only what is necessary to capture the non-determinism related to where people want to go. Given that the only signals we can receive from the outside are the buttons being pressed, we can assume that when a button is pressed, the controller receives that information in the form more convenient for our design. You have freedom to decide how.

## 2 Requirements and Environment

| | |
|---|---|
| ENV 1 | We have an elevator that is used to transport people up and down inside a building. |

| | |
|---|---|
| ENV 2 | The building has a fixed number of levels, numbered from **0** (the ground level) to **N_Levels** (the last level). |

| | |
|---|---|
| EQP 3 | In every level, there is a button outside the elevator cabin to call the elevator. |

| | |
|---|---|
| EQP 4 | Inside the elevator cabin there is a panel with buttons (one per level) to request that the elevator goes to the level associated to the button. |

| | |
|---|---|
| EQP 5 | The elevator has doors. |

| | |
|---|---|
| FUN 6 | The elevator can be sent directly to any level. |

| | |
|---|---|
| FUN 7 | The elevator doors can be opened and closed from the controller. |

| | |
|---|---|
| FUN 8 | The elevator doors should be closed when the elevator moves between levels. |

| | |
|---|---|
| FUN 9 | When the elevator reaches a level to serve a request, it should open the doors. |

| | |
|---|---|
| FUN 10 | After the elevator stops at a level and the doors are open, it can start moving again if it has requests to serve. |

| | |
|---|---|
| FUN 11 | The doors should not be closed if there are no requests to serve. |

| | |
|---|---|
| FUN 12 | The status of the buttons can be accessed from the controller. |

| | |
|---|---|
| FUN 13 | All pending requests should eventually be served. |

| | |
|---|---|
| FUN 14 | Requests to go to a level for which there is already a pending request can be ignored. |

| | |
|---|---|
| FUN 15 | The elevator should not move if there are no requests waiting to be served. |

| | |
|---|---|
| FUN 16 | The elevator should not move in a direction where there are no pending requests to attend. |

## 3 Controls

As mentioned before, we do not require to focus on the representation of the environment in this project. Therefore we can make some simplifying assumptions regarding the communication with the outside world:

- We can assume that there is a variable that we can set to, for example, `Door_Open` and `Door_Closed` to, resp., open and close the door.
- We can assume that there is a variable that we can set to a level number from `0` to `N_Levels` which will make the elevator go there.
- Telling the elevator to move to a different level does **not** cause the doors to close. Closing and opening doors must be done explicitly.
- We can assume that we can register which buttons are being pressed without explicitly reading from communication lines and / or resetting the state of lines. However, we have to model the fact that buttons can be pressed at any moment (and that this must be registered) or not pressed at all.

You are not expected to optimize the movement of the elevator. However, the elevator should not move randomly between levels. A good compromise is to follow the so-called *elevator algorithm*: the elevator moves upwards as long as there are requests in levels above it, serving them in the order in which they appear, and when they are all served, the elevator starts moving down following a similar idea. It does not change direction when there are still pending requests in the direction it is moving.

## 4 Tasks

Your task is to develop an Event B model to control an elevator respecting the requirements presented in Section 2, according to the signals read from the buttons. Use invariants to capture these requirements when possible. You can decide whether to perform model refinement or not. All the proof obligations that Rodin generates should be proven or reviewed. You should prove absence of deadlocks.

If you think the requirements are insufficient (for example, if you believe that some conditions are missing), you are free to suggest new requirements as long as they are reasonable, do not contradict other requirements or they do not severely limit the functionality of the system. Likewise, any simplification you may want to introduce will have to be motivated in the presentation you will prepare.

## 5 Teams, Submission, and Presentation

The project is to be done (and turned in) by **teams of three students**. Please get in touch with me if a team of three cannot be assembled. The work developed has to be presented in the presentation session (see the first page of this document).

The material to be submitted before the presentation is:

1. A Rodin project with the model for the problem, exported as we did in previous homeworks. The proofs necessary for Section 4 must be discharged. If some proof is not discharged, it must be reviewed and a justification why it ought to hold must be given as part of the document described in point 3, below.
2. The slides to be used for the presentation of the project (see later).

3. A document, **in PDF format**, explaining:

   - How the requirements were addressed in the model.
   - If necessary, a (convincing) explanation of why any property whose proof was not discharged with Rodin is true.

   This document can be the slides used for the presentation (see the paragraph below) if you think they are self-contained and clear enough, or a separate document if you think that additional explanations / clarifications are needed. Comments for key points in the Rodin model can also be used to explain your decisions and the intended meaning of the components of the model.

Every team should make a presentation of at most 20 minutes (including questions; I suggest 15 min. presentation plus 5 min. questions) explaining the strategy to solve the problem and anything else that you think is worth mentioning (for example, difficulties found, etc.) Every team member should present part of the work, ideally dividing the time equally among team members. Your classmates will have the chance to make questions related to the presentation and its contents. This presentation and the slides can be

In order to work around any problem with the projector, connections, etc. I recommend to bring the presentation in a PDF file that can be transferred between computers if necessary.

## 6 Additional Information

**Using Additional Theorem Provers** You will most likely need the *Atelier B* provers installed (which you should have, anyway). It should be possible to design a model for the problem for which Rodin can discharge all the proof obligations (almost) automatically — maybe clicking some buttons in the *Proving View*.

If, despite interacting with the theorem provers, you cannot discharge a PO that you are convinced is correct, you can try with the SMT solvers. Go to Help → Install new software, select Rodin plugins → Prover Extensions → SMT Solvers, and install them. In the Prover View a new button will appear with which you can select additional provers which use the hypothesis that appear in the Selected Hypothesis sub-window. The SMT solvers can in many cases prove sequents that PP or ml cannot prove. Likewise, the Atelier B provers can sometimes discharge proofs that the SMT provers cannot.

If you cannot discharge some PO that you are convinced is correct, please mark it as *reviewed* (with the Ⓡ button) and follow the instructions in point 3 of Section 5.

**Remember not to use the NewPP theorem prover.** It is unsound: it is known to have bugs enough so that it is not reliable for normal use.

**Seeing the Whole Model** A large model can sometimes be difficult to work with — the display may be cluttered with large formulas. Rodin can export models to LaTeX which can then be processed to generate a PDF that is displayed / printed. That is performed by a Rodin plugin — see http://wiki.event-b.org/index.php/B2Latex for an explanation of how to install the plugin and use the generated LaTeX file.

**Theorems and Order of Formulas**

- Although the different parts of the guards are in logical conjunction, sometimes changing the order of the guards helps the theorem provers to do their job (this is actually necessary in some cases — see the next bullet).

- The order of formulas is relevant to write "lemmas". Flagging a formula as a "theorem" in a context / invariant section / guard section makes Rodin to try to prove it from the *previous* formulas within its scope. If it is proven, it becomes available to further proofs. It would therefore work as a lemma that helps prove additional properties.

**Deadlock Freedom** To prove deadlock freedom, the disjunction of the guards has to be always true — i.e., there is always some event that can be executed. This disjunction can be written as an invariant or, alternatively, as a `theorem` which can be proved based on the previous invariants. Both are equivalent logically. The latter is preferred because its proof does not depend on changes in the actions / guards of the events, but it may need additional work. Try both.

To use the `theorem` approach:

- In the **INVARIANT** section, add a formula with the disjunction of the guards. Make sure it is the **last** in the list of the invariants.
- Mark it as a theorem: after the formula there should be a label reading `not theorem`. Click on it and it will change to mark it as a `theorem`.

You may want to use events with parameters to solve this problem. A parameter is conceptually an existentially quantified variable and a guard with an existential variable is enabled when there is a value for the variable that makes the guard true. Therefore, to represent this guard in a self-contained way, we have to existentially quantify some variable(s). In an event such as

**MACHINE** Example Any
**EVENTS**
**Event** sample ⟨ordinary⟩ $\widehat{=}$
  **any**
    a
  **where**
    grd1: $a \in X$
    grd2: $a > f(b)$
  **then**
    act1: ...
  **end**
**END**

the formula to express that the guard is true would be

$$\exists a \cdot (a \in X \land a > f(b))$$

Note that the formula needs to include the range of the quantified variable, as in the guard of the corresponding event.

**Event-B constructs** You may want to use Event-B constructs that we have mentioned in the lectures on passing, but not used them in any example. Have a look at the reference card available at the course web site.

Sets are one of the type of objects that are available in Event-B but that we did not use a lot in our examples. Sets are very useful to model and reason about many real-life cases, and Event B can use them as first-level citizens to assign them, state that an element belongs or not to a set, make the union, intersection, difference, build sets using comprehension, etc.

In particular, you may want to record whether a level has to be served or not by using a function

$$f \in \texttt{Levels} \to \texttt{BOOL}$$

that registers whether there is a request for every level. A function like this actually represents a set $S \subseteq \texttt{Levels}$ where $x \in S \Leftrightarrow f(x) = \texttt{TRUE}$, and is called the *characteristic* function of $S$. So one can use, instead of $f$, the set $S$ and substitute expressions as follows:

| Using $f_S$ | Using $S$ |
|---|---|
| $f_S(x) = \texttt{TRUE}$ | $x \in S$ |
| $f_S(x) = \texttt{FALSE}$ | $x \notin S$ |
| $f_S(x) := \texttt{TRUE}$ | $S := S \cup \{x\}$ |
| $f_S(x) := \texttt{FALSE}$ | $S := S \setminus \{x\}$ |
| $\forall x \cdot x \in \mathrm{dom}(f_S) \Rightarrow f_S(x) = \texttt{FALSE}$ | $S = \varnothing$ |
| $\forall x \cdot x \in \mathrm{dom}(f_S) \Rightarrow f_S(x) = \texttt{TRUE}$ | $S = \texttt{Levels}$ |