Correctness by Construction

Third Event-B Exercise

Deadline: Tuesday, April 29th 2025, 23:59

Manuel Carro
manuel.carro@upm.es

March 31st, 2025

1 A Doctor's Office

We have to design a system to control the access to a doctor's office, where only one patient can be at a time. The office has an external status light that can be red (meaning that the office cannot be accessed) or green (with the opposite meaning). A sensor in front of the entrance door can detect when someone is waiting. A sensor in front of the exit detects when a patient heading out of the office leaves. The doctor has a button to signal the s/he is ready to accept a new patient.

The inside of the office cannot be seen from the outside, and the other way around. All communication takes place through the external status light. Patients to be attended should wait until the status light is green to enter the office. When the light is red, patients wait standing on the sensor. The persons inside the office can leave at any time through an exit door, different from the entrance door.[1] Figure 1 shows a schematic representation of the office.

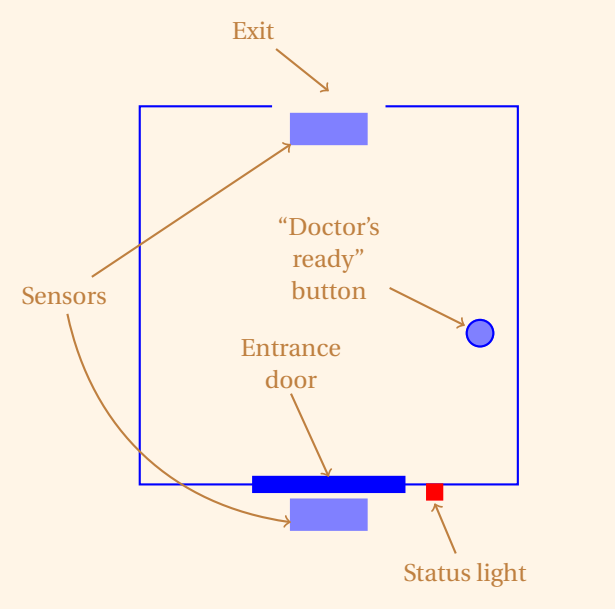

Figure 1: Doctor's Office.

Patients are expected to follow some common-sense rules, among which we can list:

- The status light is respected.

[1] In real life, patients would wait for the doctor to examine them. To simplify, patients can leave whenever they wish to do so.

- Patients stand in front of the door waiting for the light to turn green.
- Only one patient stands in front of the door.
- Only one patient enters the office at a time, and only when the light is green.
- Anyone waiting at the entrance door does not walk away, but eventually enters the office when the status light indicates that this is possible.

The office has a button / web interface / ... that doctors press to let the controller know that they are ready to see another patient. This may happen when the patient is still in the office and getting ready to leave or at any moment after the patient has left (the doctor may have to take care of a number of administrative tasks). In any case, a new patient should not be able to enter the office until the previous patient has left the office **and** the doctor has signaled that they are ready to attend a new patient.

If necessary, you may introduce behavior that can be reasonable in a realistic environment. If so, you should explain and motivate it (see Section 3). If the rules you introduce simplify the problem too much, you may not get full marks, even if your model completely implements them.

1.1 Requirements

FUN 1	This system deals with the access control of a doctor's office.

FUN 2	The system must not deadlock.

SAF 3	At most one patient can be in the office at any given time.

EQP 4	Doctors have a device which is used to signal the controller when they are ready to attend a new patient.

FUN 5	After a patient has entered the office, doctors can signal at any time that they are ready to see a new patient.

EQP 6	There is a status light outside the office with two colors: green and red.

FUN 7	When the status light is red, patients cannot enter. When the status light is green, patients can enter.

FUN 8	The status light can be green only when there is no patient in the office and the doctor has signaled that they are ready to see a new patient.

ENV 9	Patients obey the status light.

EQP 10	There is a sensor at the entrance of the office that detects when a person enters.

EQP 11	There is a sensor at the exit of the office that detects when a person leaves. The exit of the office is different from and independent of the entrance of the office.

FUN 12	The sensors produce an *on* signal when a person is standing on / in front of them and an *off* signal otherwise.

FUN 13	Anyone wishing to enter the office must step on the sensor, and wait there until the status light is green, if it is not already.

ENV 14	Anyone who stands on the sensor will wait there for the status light to turn green and enter the office.

FUN 15	A patient inside the office can leave at any moment.

ENV 16	The inside of the office cannot be seen from its outside and vice-versa.

1.2 Tasks

You have to:

1. Develop an Event-B model that captures the requirements in Section 1.1 and the general description in Section 1. Make sure that the events and variables are clearly identified as belonging to the environment or to the controller (Section 1.3.2).

2. If you have made any changes or additions to the behavior rules, write a **short** document stating them and the motivation to propose these changes. Make sure that your model takes care of them. If you have new requirements, give them a number/name to identify them and make it easier to refer to them in the comments of the model.

1.3 Additional Information

1.3.1 The Sensors

The sensors work similarly to those we presented in the *One-Way Bridge* example. When a person steps on the sensor, its state (which we access through a variable in the model) goes from *off* to *on*. When someone standing on the sensor walks away, the state returns to *off* (see Figure 2).

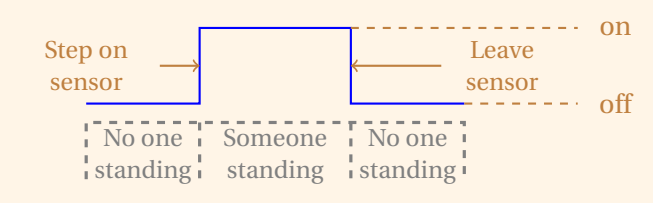

Figure 2: Behavior of the sensor when someone stands on it. Time goes from left to right.

1.3.2 Separation of Concerns

In order to have a model that can be implemented, the controller cannot manipulate data that models the environment. Likewise, the events that model the environment cannot (directly) read / change data that belongs to the controller. The final design should have a clear separation between variables and events that model the behavior of the environment and those that model the controller. Please state clearly which events and variables pertain to the environment simulation and which to the controller. It is advisable to adopt a naming scheme that reflects this. Also, every variable needs to have a clear *meaning* in the model: what it represents and what it is used for. Respect this meaning throughout the development of the model and it will help avoid mistakes.

The events that model the behavior of patients can:

- Change the sensor state, because that change is triggered by a patient standing on or leaving the sensor.
- Read the sensor state, because that corresponds to physically seeing a patient standing on it. You may also have a specific (environment) variable to denote a patient standing on a sensor if you think that this is cleaner.
- See and react to the color of the status light, but not change it.

The events that model the controller can:

- Read the sensor status, because it would be (physically) connected to the controller.
- Read and set the light status.
- Read and write any additional variables necessary to keep an internal state.

However, they cannot:

- Set variables that respond to actions by patients (e.g., the sensor state).
- Read or set variables that are only necessary to model the behavior of the persons.

An exception to this division of concerns comes from the need to model that the hardware / software is fast enough to register and process signals from the sensors (Section 1.3.3.)

1.3.3 Processing Speed of Hardware Equipment

Real hardware / software takes some time to process signals and actuate on external devices. But this is often so fast that it is safe to assume it is immediate from a human point of view. In our case, we will assume the software and hardware is so fast that the signals generated by the sensor are received and treated by the controller before they change, and they are not lost (because, e.g., someone walks very fast past a sensor).

Since we are not representing (real) time in our models, a simple specification can allow interleavings that would not take place when humans are involved, due to the huge speed difference between persons and machines.[2] To prevent these unreal interleavings to be considered by the theorem provers, you can add additional variables to block / delay (environment) events that would not

[2] This is not bad, as if we were modeling a system purely based on software, these interleavings might very well happen and, if they are problematic, they would need to be detected.

immediately take place in real life and disallow schedulings that would not happen due to the aforementioned speed difference.

In particular, it is admissible to include the variables and logic necessary to reflect that events corresponding to a patient leaving the sensor can only be enabled after the *on* signal has been registered by the controller. In the real world we cannot enforce this, but if we assume that it is what happens in reality, enforcing it in the model does not violate reality.

However, this logic must not unnecessarily serialize the design, which has to be concurrent by nature. For example, a person may be standing at the entrance of the office while another person is inside and/or exiting and the doctor is pressing the "next patient" button. These events may take place in any order, and none of these orders should to be removed.

1.3.4 Some Recommendations

- The system can be designed using a single Event-B model. However, progressive refinements may help separate concerns and consider requirements incrementally.
- You can follow an approach similar to that of the One-Way Bridge. However, the behavior of the office is simpler (there is only one area to control and a maximum of one person) and considerable simplifications can be made.
- Use invariants / guards to capture the problem requirements. In the model, please add comments stating which requirement in Section 1.1 each guard / invariant is supposed to address, when they address requirements directly.[3]
- Since at most one patient can be in the office at any time, it is tempting to use a BOOL to model this. That may not be a good choice, because controlling a violation of the maximum occupancy requirement will not be straightforward then. A number with suitable bounds will be more appropriate (and less error-prone) to capture these violations and easier to modify if the requirements of the problem change.
- The proofs are likely to be rather easy. Some of them may need invoking directly the PP predicate prover. However, coming up with the right model may be delicate. The interplay of events makes it necessary to think carefully the role of each variable, and stick to it.

2 Phone Agenda

During the lectures we say an example of the design of a simple phone agenda (available here and listed at the of this homework, in Section A). Your task is to **refine** it to change the definition of the preferred set to become a function from persons to phone numbers that registers the preferred phone number (if any) for some people in the agenda. In particular, you should:

- **Refine** the initial machine. Refining it is very important to ensure that Rodin connects the invariants of the old model with the mechanics of the new model.
- Remove the declaration of preferred, which will be accessed but not updated.

[3] You will surely need auxiliary invariants to help prove some of the invariants / refinements that are used to capture the requirements.

- Use instead a new variable (let us call it prefopt, but you can use any name you find appropriate) which should be a function from people to phone numbers, to access immediately the preferred phone number for a person.
- Establish the gluing invariant between prefopt and preferred which shows that prefopt contains the same information as preferred.
- Write the invariants necessary to ensure that:
  - Every pair person / preferred number corresponds to a relation number / person already in the agenda.
  - A person does not have two different favorite numbers (this is carried over from the initial machine).
- Change the necessary events in new model to use prefopt instead of preferred.

3 To Be Turned In

The documents / files mentioned below must be sent to manuel.carro@upm.es by Tuesday, April 29th 2025, 23:59, as stated below:

- **For the Doctors' Office and the Phone Agenda**: you should turn in the final models, developed using Rodin. The name of model you use in Rodin should follow the scheme *HwName*_HW3_2025_*Initials*. As an example, in my case they would be DoctorOffice_HW3_2025_MCL and PhoneAgenda_HW3_2025_MCL. If you have two family names, please use the initials for both. Ideally, all the proof obligations should be discharged. Export the models to two zip files (one for each model) and send me these zip files as mail attachments.

  Please follow the naming guidelines above. They will help me sort out and process your homework and avoid mistakes.

- **For the Doctors' Office only**: If you suggested changes to the problem description, please submit the document mentioned in point 2 of Section 1.2 in PDF format. You can hand-write it **very clearly** and scan it or take a **good** picture and convert it to a PDF document. Please do not store it as part of the zip file where you pack the Event-B model; send it as a separate attachment instead. Similarly to the model, please name it DoctorOfficeNewReqs_HW3_2025_*Initials*.pdf.

A Phone Agenda – Initial Machine

CONTEXT phone_Ctx

SETS

People Infinite. We can make it finite, all the POs can be discharged anyway

Phones Infinite. We can make it finite, all the POs can be discharged anyway

END

MACHINE phone_Mch0

SEES phone_Ctx

VARIABLES

agenda The agenda where we store names and phone numbers

preferred A set of numbers that we prefer for calling some people

INVARIANTS

`invAgenda`: $agenda \in Phones \nrightarrow People$

Every phone belongs to one person only

There are no phones without an owner

There are no persons in the agenda without a phone

`invPref`: $preferred \subseteq dom(agenda)$

The preferred numbers have to be in the agenda

`uniquePref`: $\forall p1, p2 \cdot (p1 \in preferred \land p2 \in preferred \land p1 \neq p2) \Rightarrow agenda(p1) \neq agenda(p2)$

Every person has at most one preferred contact

EVENTS

Initialisation

We start with an emtpy agenda

begin

`act1`: $agenda := \varnothing$

`act3`: $preferred := \varnothing$

end

Event AddPhone ⟨ordinary⟩ $\widehat{=}$

Insert a new phone and person to who it is associated.

any

person External parameters

phone

where

`grd1`: $person \in People$

`grd2`: $phone \in Phones$

`grd3_2`: $phone \notin dom(agenda)$

Needed because phones cannot be repeated

then

`act1`: $agenda(phone) := person$

end

Event RemovePhone ⟨ordinary⟩ $\widehat{=}$

Remove a phone number.

If it's the last one for a person, then the owner also needs to be removed.

any

phone

where

grd1: $phone \in Phones$

We do not need to require that it is already in the agenda (but we might).
Nothing will happen if it is not. If it is the last phone of a person, the
person has to be removed as well.

then

act1: $agenda := \{phone\} \lhd agenda$

Note: "agenda := agenda \ {phone ↦ agenda(phone)}" would also work

act2_2: $preferred := preferred \setminus \{phone\}$

We cannot have orphan phone numbers

end

Event RemovePerson ⟨ordinary⟩ $\widehat{=}$

If person not in agenda, nothing changes

any

person

where

grd1: $person \in People$

then

act1: $agenda := agenda \rhd \{person\}$

Remove from agenda all entries associated with that person

act2_2: $preferred := preferred \setminus agenda^{-1}[\{person\}]$

If the person had a preferred phone number, we have to remove it.
Get the phone numbers associated with the person, remove them
from the list of preferred phone numbers.

end

Event MakePreferred ⟨ordinary⟩ $\widehat{=}$

Remember at most one preferred phone number per person!

any

phone

where

grd2: $phone \in dom(agenda)$

We cannot make a phone preferred if it is not in the agenda

then

act1: $preferred := (preferred \setminus agenda^{-1}[\{agenda(phone)\}]) \cup \{phone\}$

If we just do preferred := preferred ∪ {phone}
we might end up with more than one preferred phone #
per person!

end

Event RemovePreferred ⟨ordinary⟩ $\widehat{=}$

any

phone

where

grd1: $phone \in Phones$

It could also be "phone ∈ dom(agenda)"

If it is not in the agenda, nothing will happen.

then

act1: $preferred := preferred \setminus \{phone\}$

end

Event TrasferPhone ⟨ordinary⟩ $\widehat{=}$

Change the owner of a phone #. We do not set it as preferred.

any

phone

next_owner

where

grd1: $phone \in dom(agenda)$

grd2: $next\_owner \in People$

grd3: $next\_owner \neq agenda(phone)$

It does not make sense to transfer a phone to its current owner.

We could accept it, but it complicates the specification. For the sake of clarity, it seems simpler just not to allow that transition.

then

act1: $agenda(phone) := next\_owner$

Or "agenda := agenda ⩤ {phone ↦ new_owner}"

act2: $preferred := preferred \setminus \{phone\}$

end

END
