Exercise Session 9 : Refinement and Proof Obligations in Event-B

Ingmar Dasseville *ingmar.dasseville@cs.kuleuven.be*
Thomas Vanstrydonck *thomas.vanstrydonck@cs.kuleuven.be*
Simon Marynissen *simon.marynissen@cs.kuleuven.be*

December 5, 2017

1 Rodin Handbook

Last session you went over the Rodin tutorial in the Rodin Handbook, located at `http://handbook.event-b.org/current/pdf/rodin-doc.pdf`. This session you will use what you learned to model and refine some systems using the Rodin platform.

1.1 A Beverage Vending Machine

Import the BeverageVendingMachine archive, the same way you would import a .zip in eclipse. Simply create a new empty Event-B project and import all the files from the archive.

The project contains two files: the Beverages context and the VendingMachine0 machine. These are listed below (you can not copy-paste this code into Rodin). Rodin uses a tree-structure to maintain the source code. New elements can be added by right-clicking the desired branch and choosing the type of node to add. Information can be added to these nodes by clicking the desired location.

Context

```
context Beverages

sets drinks

constants cola fanta no_drink
```

```
axioms
  @drinks_partition partition(drinks , {cola}, {fanta}, {
     no_drink})
end
```

This context specifies which drinks are available in the vending machine. the set `drinks` consists three constants `cola`, `fanta` and `no_drink`. This is specified in the Axiom `drinks_partition`. The partition command has a strict form, as shown, ie. partition(set-name, element1, element2, ...).

Axioms are used to define the type of the constants as well as provide additional information about the constants and sets.

Remark : Sets can only consist of user-defined constants, for sets of predefined elements you should use constants. For example to specify the set of even numbers between 1 and 10, you create the constant `even` and constrain it via an axiom, as in the following example.

```
context Even_example

constants even

axioms
  @even_def even = {2,4,6,8}
end
```

The BeverageVendingMachine offers a very simple vending machine, which stocks cola and fanta. After inserting a coin (event insert_coin) it is possible to select a drink, either cola (select_cola) or fanta (select_fanta). After which said drink is provided and the payment completed.

Questions

- Refine the vending machine. Start by right-clicking the VendingMachine0 machine and selecting refine. Notices all events are copied and designated *extended*, this means all information about the event is kept and you only add to it. In other words, there is no data refinement. For this exercise, leave all events on extended.

- Vending machines tend to have a limited stock. Refine the machine so that it has a certain *quantity* of cola and fanta and refine the appropriate events so that these quantities are reduced when you get a certain drink.

- Does the machine you just created constitute a correct refinement of the original vending machine? You can check by right-clicking the machine and selecting "Start Animation / Model Checking". This works just as the ProB Animator. If you ignore the new variables, the machines should have exactly the same behaviour. Do they? Why is that?

- Add a refill event to solve the problem. Make sure this event is not enabled continuously by adding an appropriate guard. Otherwise it would all the machine to go into a loop and never return to one of the original events.

---

2 A simple elevator

Create a new Event-B project called Elevator. You're asked to design a simple elevator and make sure it is safe. To do so you can use refinement to address each aspect of the elevator separately.

The specifications of the elevator are as follows :

- The elevator can stop at all floors from -2 until 5.
- The doors of the elevator can be openened and closed.
- The elevator can only start moving when the doors are closed.
- When the elevator is moving, the doors can not be opened.

The following steps guide you through the proces of designing this elevator.

Extremely simple elevator

From the specifications which property is most elementary and essentially describes the elevator in its entirety (abstracting some of the details)?

- The elevator can stop at all floors from -2 until 5.

Abstractly, an elevator can be described as something which moves from one floor to another. This is the first step in modelling using Event-B : Identify the core abstraction and model it.

Tip : Use an event parameter to select the desired destination floor, do not make a different event for each floor.

Adding some details

Next, a refinement of this abstract machine can be made to add some details about the doors. This refinement handles the next two properties :

- The doors of the elevator can be openened and closed.
- The elevator can only start moving when the doors are closed.

A major safety concern for an elevator is making sure the doors can not be opened while it is moving. This should be an invariant of the system. In your current representation of the elevator it is not yet possible to specify the invariant which states the doors of the elevator are closed while its moving.

Safety measure

A last refinement is needed to support the final property :

- When the elevator is moving, the doors can not be opened.

To be able to describe this property, you need additional information about the elevator, namely whether it is moving or not. In Event-B events are assumed to take no time. Clearly moving from one floor to another takes some time. This can be modeled by acknowledging that moving to a certain floor isn not a single event. It can be divided into several smaller events: choosing which floor to go to, starting to move to that floor and reaching the destination. As such the elevator can be said to be in two different states, it can be moving or not.

Questions

- Does this new state information make it possible to write down the invariant?
- Does the elevator satisfy this invariant? You can check this by seeing whether the proof obligation fired successfully.

---

3 Data refinement : Parking Lot

The next exercise will show you the use of gluing/linking invariants when performing data refinement. Import the Parking Lot project and open it.

The ParkingLot0 machine sketches the following situation :

- The parking lot has a limited number of spaces.

- There can not be more cars in the parking lot than there are spaces.
- Cars can enter or leave the parking lot.

Obviously this is a overly simplified model of a parking lot. Imagine the parking lot has a single access road which is not wide enough to allow cars to enter and exit the parking lot at the same time. By refining the given machine add this functionality to the modeling of the parking lot, use the following questions as a guide.

Questions

- Can the cars in the parking lot be divided into different groups? How many? Which ones?
- How do these groups relate to the number of cars in the parking lot?
- How does this influence the events of the machine?
- Use these groups to make sure the access road is used correctly, ie. cars are either entering or leaving the parking lot.
- Check whether entering and leaving happens correctly by specifying an invariant.