EventB Assignment 2

Ken Robinson

30th March 2012

Name of assignment: ass2
Due date: RailTicketR1 1st April 2012
RailTicketR2 3rd April 2012
Assessment: 15 marks

Submission: use either the web-based give:
https://cgi.cse.unsw.edu.au/~give/Student/give.php?session=12s1
or the cse command
give cs2111 ass2 RailTicket.zip

Please do not submit the assignment as an email attachment.

1 Purpose of this assignment

This assignment is concerned with:

- use of context machines machine and SEES;
- consolidation of the understanding of invariant;
- consolidation of the understanding of guard;
- expansion of general knowledge of and experience with Event-B;
- specifying events;
- experience with refinement, including data refinement.
- using proof obligations to find problems in developments.
- using the animator to check the understanding and capture of requirements.

2 TicketMachine: A Simple Rail Ticket dispensing machine

This assignment is concerned with the modelling of a simple rail ticket dispensing machine. The modelling is to be done in three stages:

atomic the first stage in which the purchase of tickets is an indivisible event.
refinement the second stage in which the purchase of tickets is distributed across a number of events with actions that are typical of what is commonly seen on a real ticket machine. In this refinement payment is made using coins.

<table>
<thead>
<tr>
<th>Event</th>
<th>Parameters</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitPrice</td>
<td>station, price</td>
<td>set initial price of a ticket to station</td>
</tr>
<tr>
<td>ChangePrice</td>
<td>station, price</td>
<td>change the price of a ticket to station</td>
</tr>
<tr>
<td>AddTickets</td>
<td>station, count</td>
<td>provide for restocking of count tickets to destination station</td>
</tr>
<tr>
<td>BuyTickets</td>
<td>station, count, payment</td>
<td>buy count tickets to station. The payment must be the exact cost of the tickets. This machine does not give change.</td>
</tr>
</tbody>
</table>

3 TicketMachineR1: Refinement of TicketMachine

The objective of the refinement is to distribute the single atomic event BuyTicket across a sequence of the following events that might represent the buttons you have to press on a ticket machine to get a number of tickets.

<table>
<thead>
<tr>
<th>Event</th>
<th>Parameters</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose</td>
<td>station,number</td>
<td>a customer chooses a station and the number of tickets required</td>
</tr>
<tr>
<td>Pay</td>
<td>coin</td>
<td>pay with a single coin towards the cost of the tickets. This event can be run a number of times until the customer has paid at least the cost of the tickets.</td>
</tr>
<tr>
<td>GiveChange</td>
<td></td>
<td>give change if the customer has given more than the cost of the tickets</td>
</tr>
<tr>
<td>Cancel</td>
<td></td>
<td>the transaction is cancelled by either the customer or the machine. The requested tickets are not delivered and the amount of money inserted is returned. “Returning money” should be an adjustment of the state of the machine; there is no mechanism for “delivering” money.</td>
</tr>
<tr>
<td>BuyTickets</td>
<td></td>
<td>finally, the refinement of BuyTickets —now with no explicit parameters— delivers the tickets when they have been completely paid for.</td>
</tr>
</tbody>
</table>

While payment is by coin, the moneybox and change are still expressed as numeric values. The moneybox may not necessarily record the current state of the transaction, but should be correct at least by the end of each transaction.

4 TicketMachineR2: Data Refinement of TicketMachineR1

The objective of this refinement is to replace the moneybox, which only records values, to coinbox that should be a box of (bag) of coins. This introduces a complication for giving change, as that now involves the choice of a bag of coins whose value is the required change, compared with simply subtracting the value of the change.
4.1 Initiating RailTicket2B

Follow the following process for RailTicket2B

1. Using the Event-B Explorer to create a refinement of RailTicket2A named RailTicket2B.
2. Add the context CoinBag.
3. Delete moneybox from the variables and invariant.
4. Add the variable coinbox.
5. Add an invariant for coinbox.
6. Add an invariant relating coinbox to moneybox. This is known as the refinement relation; it describes how coinbox models moneybox.

Essentially, the rest of TicketMachine2B consists of replacing occurrences of moneybox in TicketMachine2A with coinbox. Of course, it’s not as direct or simple as that might sound.

Importantly the only references to moneybox should occur in the invariants.

4.2 Contexts and Machines provided

The archive provides:

RailTicket a context defining a STATION, an opaque set of stations, that could be replaced by an enumerated set of stations.

Coin a context defining COIN, a finite set of coins, and CoinValue, a total injective function that maps coins to their value. The set COIN is presented as an opaque set, but could be enumerated. COIN could be replaced by an enumerated set of coins.

TicketMachine a skeletal machine that SEES RailTicket.

CoinBag a context defining the concept of a bag of coins and the functions required to manipulate such a bag.

4.3 Importing Just the CoinBag context

If you have already developed the RailTicket machine then you won’t want to overwrite that with TicketMachine from the archive. The following instructions explain how you can selectively import from an archive, in this case just the CoinBag component of the archive.

On the import menu choose:
General
Archive
Next

From the archive file: choose the archive
then select the options as follows:
Now choose Deselect all

and then select the components of the CoinBag context:

Next select the RailTicket project folder followed by Finish.

This should install only the CoinBag components into your RailTicket project.

4.4 Costs and Coins

**TicketMachine**: payment is by value, not coin;

**TicketMachineR1**: payment is by coin, but change and total payment is by value.

**TicketMachineR2**: a refinement of TicketMachineR1 that uses coins to model the contents of the coinbox, payment and change.

5 Discharge of Proof Obligations

- As usual your invariants and guards should be strong enough to ensure no exceptional behaviour.

- The proof obligations should give a reasonably good indication of the correctness of your model. You should be able to get your POs automatically discharged.

The following table gives the PO statistics for a solution that satisfies the above.
5.1 Other requirements

The machine should not dispense tickets that do no have a known price. The implication of that is the machine should not contain tickets for sale that do not have real price.

Payment is represented by a value; you do not model coins.

5.2 What you have to do

1. Import the provided archive. To do that:
   
   Open Rodin on an existing or new workspace.
   Select Import on the file menu;
   Select General and then “Existing Projects into Workspace”
   Select Next
   Check Copy projects into workspace. Very important: ensures the project is in this workspace, not shared with some other workspace.
   Choose Select archive file and browse to where you have placed the archive. This should list the projects in the archive, in this case RailTicket
   Select choose project and Finish.

   The archive should be installed and you can view the project using the Event-B Explorer
   Important: the archive is offered as a skeleton and apart from adding to that skeleton it may be necessary to make changes.

2. TicketMachineR

   You will notice that the archive does not contain TicketMachineR. This is because the easiest and best way to obtain this machine is to generate it from within the Event-B Explorer by right-clicking on TicketMachine and choosing Refine. This will produce a refinement that is automatically consistent with your version of RailTicket, so is best done when you have filled out that machine.

3. You should monitor the proof obligations very carefully. Attempt to discharge them if possible, but at the very least check them for indications that there is something inconsistent in your model.
4. Remember that the objective is not to reduce the number of POs; the stronger the invariant the more POs you can expect, in general. POs are very useful.

5. Animate your model using AnimB.

6. When you are finished, archive your project and submit as shown at the top of this specification.
CONTEXT RailTicket

SETS

STATION Finite set of stations

AXIOMS

\textbf{axm1} : \textit{finite}(STATION)

END
MACHINE  TicketMachine

SEES  Stations

VARIABLES

stations  Stations known to this machine

ticketprice  Price of tickets

tickets  Number of available tickets

moneybox  amount of all money paid (value not coins)

EVENTS

Initialisation

begin
  skip
end

Event  InitPrice \equiv
Set initial price for tickets to station

any
  station
  price
where
  skip
end

Event  ChangePrice \equiv
Change price for tickets to station

any
  station
  price
where
  skip
end

Event  AddTickets \equiv
Add count tickets to station

any
  station
  count
where
  skip
end
Event $\text{BuyTickets} \equiv$
Request and pay for count tickets to station

any

station
count
payment

where

skip

end

END
CONTEXT Coin

SETS

COIN

CONSTANTS

CoinValue

AXIOMS

axm1: finite(COIN)

axm2: CoinValue ∈ COIN → N₀
CONTEXT CoinBag
EXTENDS Coin
CONSTANTS
COINBAG
bagvalue
emptybag
bagunion
subbag
bagdiff
addcoin
removecoin
AXIOMS
axm1 : COINBAG = COIN → N
axm3 : bagvalue ∈ COINBAG → N
axm4 : ∀ b, c · b ∈ COINBAG ∧ c ∈ dom(b)
     ⇒ bagvalue(b) = b(c) * CoinValue(c) + bagvalue(b ⊣ { c ↦ 0 })
axm5 : ∀ b · b ∈ COINBAG ∧ (b ≠ ∅ ⇒ ran(b) = {0})
     ⇒ bagvalue(b) = 0
axm6 : emptybag = COIN × {0}
axm7 : bagvalue(emptybag) = 0
axm8 : bagunion ∈ COINBAG × COINBAG → COINBAG
axm9 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG
    ⇒ bagunion(b1 ↦ b2) = { c · c ∈ COIN | c ↦ b1(c) + b2(c) }
axm10 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG
       ⇒ bagunion(b1 ↦ b2) = bagunion(b2 ↦ b1)
axm11 : ∀ b · b ∈ COINBAG
     ⇒ bagunion(emptybag ↦ b) = b
axm12 : ∀ b · b ∈ COINBAG ⇒ bagvalue(emptybag ↦ b) = bagvalue(b)
axm13 : subbag ∈ COINBAG × COINBAG → BOOL
axm14 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG
      ∧ (∀ c · c ∈ COIN ∧ b1(c) ≤ b2(c)) ⇔ subbag(b1 ↦ b2) = TRUE
axm15 : ∀ b · b ∈ COINBAG ⇒ subbag(emptybag ↦ b) = TRUE
axm16 : bagdiff ∈ COINBAG × COINBAG → COINBAG
axm17 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG
       ⇒ (b1 ↦ b2) ∈ dom(bagdiff) ⇔ subbag(b2 ↦ b1) = TRUE
axm18 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG ∧ subbag(b2 ↦ b1) = TRUE
       ⇒ bagdiff(b1 ↦ b2) = { c · c ∈ COIN | c ↦ b1(c) - b2(c) }
axm19 : ∀ b, b1, b2 · b ∈ COINBAG ∧ b1, b2 ∈ COINBAG ∧ subbag(b2 ↦ b1) = TRUE
       ⇒ bagvalue(bagdiff(b1 ↦ b2)) = bagvalue(b1) - bagvalue(b2)
axm20: $\forall b_1, b_2 \cdot b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \land \text{subbag}(b_1 \mapsto b_2) = \text{TRUE}$
$\Rightarrow \text{bagvalue}(b_1) \leq \text{bagvalue}(b_2)$

axm21: $\forall b_1, b_2 \cdot b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG}$
$\Rightarrow \text{bagvalue}(\text{bagunion}(b_1 \mapsto b_2)) = \text{bagvalue}(b_1) + \text{bagvalue}(b_2)$

axm22: $\text{addcoin} \in \text{COIN} \times \text{COINBAG} \rightarrow \text{COINBAG}$

axm23: $\forall c, b \cdot c \in \text{COIN} \land b \in \text{COINBAG}$
$\Rightarrow \text{addcoin}(c \mapsto b) = b \triangleleft \{c \mapsto b(c) + 1\}$

axm24: $\text{removecoin} \in (\text{COIN} \times \text{COINBAG}) \rightarrow \text{COINBAG}$

axm25: $\forall c, b \cdot c \in \text{COIN} \land b \in \text{COINBAG} \land b(c) \neq 0$
$\Rightarrow c \mapsto b \in \text{dom}(\text{removecoin})$

axm26: $\forall c, b \cdot c \in \text{COIN} \land b \in \text{COINBAG} \land b(c) \neq 0$
$\Rightarrow \text{removecoin}(c \mapsto b) = b \triangleleft \{c \mapsto b(c) - 1\}$

axm27: $\forall b \cdot b \in \text{COINBAG}$
$\Rightarrow b = \{c \cdot \top | c \mapsto b(c)\}$

END