1 Purpose of this assignment

This assignment is concerned with:

- use of contexts;
- consolidation of the notion of invariant;
- consolidation of the notion of guard;
- expansion of knowledge and experience with eventB;
- specifying events;
- satisfaction of requirements;
- consolidation of the concept of refinement;
- using proof obligations to find problems in developments.

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 two stages: the first stage in which the operation of buying tickets is atomic. In the refinement the purchase of tickets will be distributed across a number of actions that are typical of what is commonly seen on a real ticket machine. In the following list of events, names after the event name represent parameters. The machine has the following events:

InitPrice station price set initial price of a ticket to station.

ChangePrice station price change the price of a ticket to station.
AddTickets station count provide for restocking of count tickets to destination station.

BuyTickets station count payment) buy a count tickets to station. The payment must be at exactly the cost of the tickets.

Requirement Tickets that are available for sale must have a valid price. A valid price is one set by the events InitPrice or ChangePrice. Conversely, if the price of a ticket to any station is not known then tickets to that station must not be available for sale.

3 TicketMachineR: Refinement of TicketMachine

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

Choose(station number) a customer chooses a station and the number of tickets required.

Pay coin pay all or part of the remaining cost of the tickets using a coin. The value of the coin is given by coinvalue(coin).

Cancel the customer wishes to cancel the purchase. 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.

GiveChange for a customer who has paid more than the price of the tickets, this event causes the excess money to be returned. As with Cancel “giving change” only adjust the total amount paid.

BuyTickets finally, the refinement of BuyTickets, now with no explicit parameters delivers the tickets when they have been exactly paid for.

Payment is represented by a coin, whose value is given by the coinvalue(coin). Although payment is represented by coins, only the value of the coin is added to a moneybox, which for this assignment is represented simply by a numeric amount.

3.1 Machines required

The assignment requires the completion of:

TicketMachine a machine that SEES RailTicket context.

TicketMachineR a refinement of TicketMachine.

3.2 Discharge of Proof Obligations

The proof obligations should give a reasonably good indication the correctness of your model. You get approximately the following POs

TicketMachine: 15 all automatically discharged;

TicketMachineR1: 42 all automatically discharged

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

If you have significantly less POs than shown above your invariants are almost certainly too weak.
3.3 Animation
You might find it useful to animate your model using AnimB.

4 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
   - 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. Complete TicketMachine consistent with the requirements.

3. In completing the refinement TicketMachineR1 you may find it better to delete the RailTicketR1 obtained from the archive and right-click on TicketMachine in the Event-B explorer and choose Refine, naming the refinement TicketMachineR1. This will preserve what you have entered in RailTicket. However, there will be new partially completed events, shown in the listings in this assignment specification, that you will have to add to the newly-created refinement.

4. You should monitor the proof obligations very carefully, and frequently. Attempt to discharge them if possible, but at the very least check them for indications that there is something inconsistent in your model.

5. 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.

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

COIN
CONSTANTS

coinvalue
AXIOMS

axm1: \text{finite}(\text{COIN})
axm3: coinvalue \in \text{COIN} \to \mathbb{N}_1

END
MACHINE TicketMachine

SEES RailStation

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 ≡ Set initial price for tickets to station

any station
price
when skip
end

Event ChangePrice ≡ Change price for tickets to station

any station
price
when skip
end

Event AddTickets ≡ Add count tickets to station

any station
count
when skip
Event  \textit{BuyTickets} \equiv \text{Request and pay for count tickets to station}

\textbf{any} \textit{station} \\
\textbf{count} \\
\textbf{payment} \\
\textbf{when} \text{skip} \\
\textbf{end}

\text{END}
MACHINE TicketMachineR1

REFINES TicketMachine

SEES RailStation, COINS

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

  extended
  begin skip
  end

Event InitPrice ≜ Set initial price for tickets to station

  extends InitPrice
    any station
    price
    when skip
    end

Event ChangePrice ≜ Change price for tickets to station

  extends ChangePrice
    any station
    price
    when skip
    end
**Event** \( \text{AddTickets} \) \( \triangleq \) Add count tickets to station

extends \( \text{AddTickets} \)

\begin{align*}
\text{any} & \quad \text{station} \\
\text{count} & \\
\text{when} & \quad \text{skip} \\
\text{end} \\
\end{align*}

**Event** \( \text{Choose} \) \( \triangleq \) Choose tickets

\begin{align*}
\text{any} & \quad \text{station} \\
\text{number} & \\
\text{when} & \quad \text{skip} \\
\text{end} \\
\end{align*}

**Event** \( \text{Pay} \) \( \triangleq \) Pay towards cost of tickets

\begin{align*}
\text{any} & \quad \text{coin} \\
\text{when} & \quad \text{skip} \\
\text{end} \\
\end{align*}

**Event** \( \text{GiveChange} \) \( \triangleq \) Give change if needed

\begin{align*}
\text{begin} & \quad \text{skip} \\
\text{end} \\
\end{align*}

**Event** \( \text{Cancel} \) \( \triangleq \) Abort purchase

\begin{align*}
\text{begin} & \quad \text{skip} \\
\text{end} \\
\end{align*}

**Event** \( \text{BuyTickets} \) \( \triangleq \) Finalise purchase of tickets

refines \( \text{BuyTickets} \)

\begin{align*}
\text{begin} & \quad \text{skip} \\
\text{end} \\
\end{align*}

END
5 RailTicketR2: the Second Refinement

In addition to splitting the BuyTickets event into multiple events, the first refinement refined from payment expressed in amounts to payment expressed in discrete coins. However, the model does not move completely to coins:

- the contents of moneybox is expressed in value, rather than coins;
- change is expressed as a value, rather than coins.

The next refinement is intended to address that inconsistency in the modelling.

The essential differences:

- there is now a coinbox as well as moneybox and these two must be kept consistent, but you will have to think carefully about what that means;
- change must be given in coins.

5.1 Coin bags

We now need to be able to model bags of coins. A bag is a collection in which members of the collection have a multiplicity. That is, unlike sets that have only membership, a member of a bag has a multiplicity. Coin bags are modelled in the context COINBAGS. An outline of COINBAGS follows.

For illustrative purposes we will define the following variables and their values:

**coin, a Set of Coin:**

\[
\text{coin} = \{\text{One, Two, Five}\}
\]

**\(b_1, b_2\), two bags of coin:**

\[
\begin{align*}
\text{\(b_1\)} & = \{\text{One} \mapsto 4, \text{Two} \mapsto 2, \text{Five} \mapsto 2\}, \\
\text{\(b_2\)} & = \{\text{One} \mapsto 5, \text{Two} \mapsto 2, \text{Five} \mapsto 3\}
\end{align*}
\]

**COINBAG:** COINBAG is our model of a bag of coins. We use a total function \(\text{COIN} \rightarrow \mathbb{N}\). Thus, if \(b\) is a bag and \(c\) is a coin, then \(b(c)\) is the number of \(c\) coins in the bag—sometimes called the frequency—and that could be zero.

**emptybag** emptybag is a bag that is essentially empty, that is the frequency of all coins in the bag is zero.

\[
\{\text{One} \mapsto 0, \text{Two} \mapsto 0, \text{Five} \mapsto 0\}
\]

**bagvalue** bagvalue is a total surjective function that takes the name of a bag variable and returns the total value of all the coins in the bag.

*Important:* we define bagvalue as a total surjective function from COINBAG to \(\mathbb{N}\) because we must be able to model any natural number value.

\[
\text{bagvalue} \in \text{COINBAG} \rightarrow \mathbb{N}
\]

- \(\text{bagvalue}(b_1) = 18\)
- \(\text{bagvalue}(b_2) = 24\)
**bagunion**  
`bagunion(b1 → b2)` takes the two bag variables `b1` and `b2` and forms the union of the bag contents of the two variables. For example, if `b1` contains `c → m` and `b2` contains `c → n`, then `bagunion(b1 ↦ b2)` will contain `c → m + n`.

\[ \text{bagunion}(b_1 \mapsto b_2) = \{ \text{One} \mapsto 9, \text{Two} \mapsto 4, \text{Five} \mapsto 5 \} \]

**subbag**  
`subbag(b1 → b2)` determines if `b1` is a subbag of `b2`, that is, whether the contents of `b1` are contained in `b2`. For this to be true all the coins in `b1` should be present in `b2` with frequency at least as great as their frequency in `b1`.

subbag gives a BOOL value, that is `TRUE` if `b1` is a subbag of `b2` or `FALSE` if `b1` is not a subbag of `b2`.

\[ \text{subbag}(b_1 \mapsto b_2) = \text{TRUE} \]
\[ \text{subbag}(b_2 \mapsto b_2) = \text{FALSE} \]

**bagdiff**  
`bagdiff(b1 → b2)` yields a bag that is the difference between `b1` and `b2`, that is `b1 = bagsum(b2 ↦ bagdiff(b1 → b2))`.

`bagdiff(b1 → b2)` is only valid if `subbag(b2 ↦ b1) = TRUE`.

\[ \text{bagdiff}(b_2 \mapsto b_1) = \{ \text{One} \mapsto 1, \text{Two} \mapsto 0, \text{Five} \mapsto 1 \} \]
**CONTEXT** COINBAGS

This context models a bag of coins, in which there can be multiple instances of each coin.

**EXTENDS** COINS

**CONSTANTS**

- COINBAG
- bagvalue
- emptybag
- bagunion
- subbag
- bagdiff

**AXIOMS**

1. \[ \text{COINBAG} = \text{COIN} \rightarrow \mathbb{N} \]
2. \[ \text{emptybag} = \text{COIN} \times \{0\} \]
3. \[ \text{bagvalue} \in \text{COINBAG} \rightarrow \mathbb{N} \]
4. \[ \forall b \cdot b \in \text{COINBAG} \land \text{ran}(b) = \{0\} \rightarrow \text{bagvalue}(b) = 0 \]
5. \[ \forall b, c \cdot b, c \in \text{COINBAG} \land c \in \text{COIN} \land b(c) \neq 0 \rightarrow \text{bagvalue}(b) = b(c) \ast \text{coinvalue}(c) + \text{bagvalue}(b \setminus \{c \mapsto 0\}) \]
6. \[ \forall c, n \cdot c \in \text{COIN} \land n \in \mathbb{N} \rightarrow \text{bagvalue}(\text{emptybag} \setminus \{c \mapsto n\}) = n \ast \text{coinvalue}(c) \]
7. \[ \text{bagunion} : \text{COINBAG} \times \text{COINBAG} \rightarrow \text{COINBAG} \]
8. \[ \forall b_1, b_2 \cdot b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \rightarrow \text{bagunion}(b_1 \mapsto b_2) = \text{bagunion}(b_2 \mapsto b_1) \]
9. \[ \forall b \cdot b \in \text{COINBAG} \rightarrow \text{bagunion}(b \mapsto \text{emptybag}) = b \]
10. \[ \forall b_1, b_2, b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \land b_1 \neq \text{emptybag} \land b_2 \neq \text{emptybag} \rightarrow \text{bagvalue}(\text{bagunion}(b_1 \mapsto b_2)) = \text{bagvalue}(b_1) + \text{bagvalue}(b_2) \]
11. \[ \text{subbag} \in \text{COINBAG} \times \text{COINBAG} \rightarrow \text{BOOL} \]
12. \[ \forall b_1, b_2, b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \land \forall c \cdot c \in \text{COIN} \land b_1(c) \leq b_2(c) \rightarrow (\text{subbag}(b_1 \mapsto b_2) = \text{TRUE}) \]
13. \[ \forall b \cdot b \in \text{COINBAG} \rightarrow \text{subbag}(\text{emptybag} \mapsto b) = \text{TRUE} \]
14. \[ \forall b_1, b_2, b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \rightarrow \text{bagdiff}(b_1 \mapsto b_2) = \text{TRUE} \]
\textit{axm15}: \forall b_1, b_2. b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \land \text{subbag}(b_2 \mapsto b_1) = \text{TRUE} \\
\Rightarrow \text{bagdiff}(b_1 \mapsto b_2) = \{ c \cdot c \in \text{COIN} | c \mapsto b_1(c) - b_2(c) \}

\textit{axm16}: \forall b_1, b_2. b_1 \in \text{COINBAG} \land b_2 \in \text{COINBAG} \land \text{subbag}(b_2 \mapsto b_1) = \text{TRUE} \\
\Rightarrow \text{bagvalue}(\text{bagdiff}(b_1 \mapsto b_2)) = \text{bagvalue}(b_1) - \text{bagvalue}(b_2)

\text{END}
MACHINE TicketMachineR2  

Extends the modelling of coins in all transactions rather than values

REFINES TicketMachine

SEES RailStation, COINBAGS

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)

coinbox  bag of coins in machine

EVENTS

Initialisation

begin  skip
end

Event $InitPrice \doteq$

Set initial price for tickets to station

doesn't change $InitPrice$

any station
price
when  skip
end

Event $ChangePrice \doteq$

Change price for tickets to station

doesn't change $ChangePrice$

any station
price
when  skip
end

**Event**  \textit{AddTickets} \equiv \quad \text{Add count tickets to station}

\textbf{extends} \textit{AddTickets} \\
\quad \textbf{any} \textit{station} \\
\quad \text{count} \\
\quad \textbf{when} \text{skip} \\
\quad \textbf{end}

\textbf{Event}  \textit{Choose} \equiv \\
\quad \textbf{any} \textit{station} \\
\quad \text{number} \\
\quad \textbf{when} \text{skip} \\
\quad \textbf{end}

\textbf{Event}  \textit{Pay} \equiv \\
\quad \textbf{Status} \textit{convergent} \\
\quad \quad \textbf{any} \textit{coin} \\
\quad \quad \textbf{when} \text{skip} \\
\quad \quad \textbf{end}

\textbf{Event}  \textit{GiveChange} \equiv \\
\quad \textbf{any} \textit{change} \\
\quad \textbf{when} \text{skip} \\
\quad \textbf{end}

\textbf{Event}  \textit{Cancel} \equiv \\
\quad \textbf{begin} \text{skip} \\
\quad \textbf{end}

\textbf{Event}  \textit{BuyTickets} \equiv \quad \text{Request and pay for count tickets to station}

\textbf{refines} \textit{BuyTickets} \\
\quad \textbf{begin} \text{skip} \\
\quad \textbf{end}

\textbf{END}