LECTURE 16: VENDING MACHINE CASE STUDY

Software Engineering

Mike Wooldridge

1 Specification of a Vending Machine

- In this lecture, we will give a complete specification of a vending machine – the sort you buy cans of coke or cigarettes from.
- First, we need to introduce some types; the first one will be *COIN*, representing all the coins that are accepted by the machine.

  $$\mathit{COIN} == \{100, 50, 20, 10, 5, 2, 1\}$$

- That is, there are coins in denominations of 100, 50, 20, 10, 5, 2, and 1 pence.
- We will also need a type for system messages

  – this is parachuted in:

  $$[\mathit{REPORT}]$$

- Next, we need a type *PROD*, representing all the products that the machine can sell.

$$[PROD]$$

- We can define the state space of the vending machine thus:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{VendingMachine}} \\
\hline
cost : PROD \mathbin{\ooalign{$\rightarrow$\cr\hidewidth$\mid$\hidewidth}} \mathbb{N} \\
stock : \mathrm{bag}\, PROD \\
float : \mathrm{bag}\, COIN \\
\hline
\mathrm{dom}\, stock \subseteq \mathrm{dom}\, cost \\
\hline
\end{array}
$$

- The function *cost* return the cost of a product in pence. For example,

  $$cost(MarsBar) = 25$$
  $$cost(Penguin) = 15$$

- The bag *stock* tells us how many items of each type are in stock. For example,

  $$stock = \{Penguin \mapsto 2\}$$

  means that there are just 2 penguins in the machine.

- The bag *float* records the coins that are currently in the machine; for example

  $$float = \{100 \mapsto 2, 50 \mapsto 8, 5 \mapsto 20\}$$

  means that there are $2 \times$ £1 coins, 8×50p coins and 20×5p coins.

- QUESTION: Why are *stock* and *float* bags and not sets or sequences?

- The invariant dom *stock* $\subseteq$ dom *cost* says that everything in the machine (i.e. in stock) must have a cost associated with it.

Operations

Here are the operations we shall specify:

- initialising the machine;
- pricing goods;
- restocking;
- buying goods.

Initialisation

$$\begin{array}{l} \textit{InitVendingMachine} \\ \hline \Delta \textit{VendingMachine} \\ \hline \textit{cost}' = \{\} \\ \textit{stock}' = [\![]\!] \\ \textit{float}' = [\![]\!] \end{array}$$

- So initially, the machine does not know the cost of anything, contains nothing, and has no float.

Pricing Goods

- This simply means changing the price of an item in stock, or pricing an item that is going to be stocked.
- The inputs are the item and a price.

Price

$$\Delta VendingMachine$$
$$item? : PROD$$
$$price? : \mathbb{N}$$

$$cost' = cost \oplus \{item? \mapsto price?\}$$
$$stock' = stock$$
$$float' = float$$

Restocking

- The next operation to specify is that of restocking the machine with more goods.
- The only input is a new bag of products.
- The precondition $\operatorname{dom} new? \subseteq \operatorname{dom} cost$ is implied by the invariant of $VendingMachine'$.

$$
\begin{array}{l}
\textit{Restock} \\
\hline
\Delta \textit{VendingMachine} \\
new? : \operatorname{bag} PROD \\
\hline
stock' = stock \uplus new? \\
float' = float \\
cost' = cost \\
\hline
\end{array}
$$

- (Note that $\uplus$ is the 'bag union' operator.)

- We shall now make the operation robust. The *Restock* operation fails when an attempt is made to add goods which are not priced. We need a schema to identify this situation.

GoodsNotPriced

$\Xi VendingMachine$
$new? : \mathrm{bag}\ PROD$
$rep! : REPORT$

$\neg(\mathrm{dom}\ new? \subseteq \mathrm{dom}\ cost)$
$rep! = \text{'Some goods are not priced'}$

- We need an operation to report success...

$$\begin{array}{|l}\hline \textit{Success} \\ rep! : REPORT \\ \hline rep! = \text{`Okay'} \\ \hline \end{array}$$

- Now, we simply use the schema calculus to specify a robust version of the Restock operation, called *RestockOp*:

$$RestockOP \mathrel{\widehat=} (Restock \land Success) \lor GoodsNotPriced$$

- This schema expands to ...

$$
\begin{array}{l}
\textit{RestockOp} \\
\hline
\Delta \textit{VendingMachine} \\
\textit{new}? : \mathrm{bag}\, \textit{PROG} \\
\textit{rep}! : \textit{REPORT} \\
\hline
\textit{cost}' = \textit{cost} \\
\textit{float}' = \textit{float} \\
(\textit{stock}' = \textit{stock} \uplus \textit{new}? \wedge \\
\ \textit{rep}! = \text{`Okay'}) \\
\vee \\
(\neg(\mathrm{dom}\, \textit{new}? \subseteq \mathrm{dom}\, \textit{cost}) \wedge \\
\ \textit{stock}' = \textit{stock} \wedge \\
\ \textit{rep}! = \text{`Some goods are not priced'}) \\
\hline
\end{array}
$$

Buying

- The buying operation is a somewhat more complex operation ...
- The inputs are the chosen item and some money.
- We have to check that the item is in stock, and that the user has entered enough money to buy it.
- We may also have to figure out what change to give ...

- We assume that a function

  $sum : \mathrm{bag}\, COIN \rightarrow \mathbb{N}$

  is available, which takes a bag of coins and calculates how much is in the bag. For example, given a bag containing 7×2p, and 3×5p coins,

$$\begin{aligned} sum\{2 \mapsto 7, 5 \mapsto 3\} &= (2 \times 7) + (5 \times 3) \\ &= 14 + 15 \\ &= 29\text{pence} \end{aligned}$$

- The basic *Buy* operation is as follows:

$$
\begin{array}{|l}
\hline
\textit{Buy} \\
\Delta \textit{VendingMachine} \\
\textit{in}?, \textit{out}! : \text{bag}\,\textit{COIN} \\
\textit{item}? : \textit{PROD} \\
\hline
\textit{item}? \ \text{in}\ \textit{stock} \\
\textit{sum}(\textit{in}?) \geq \textit{cost}(\textit{item}?) \\
\textit{out}! \sqsubseteq \textit{float} \\
\textit{sum}(\textit{in}?) = \textit{sum}(\textit{out}!) + \textit{cost}(\textit{item}?) \\
\textit{stock}' \uplus \{\textit{item}? \mapsto 1\} = \textit{stock} \\
\textit{float}' \uplus \textit{out}? = \textit{float} \uplus \textit{in}? \\
\textit{cost}' = \textit{cost} \\
\hline
\end{array}
$$

- in? represents the coins entered; out! represents the change;
- item? is the item dispensed to the user;
- the 1st condition says that the item must be in stock;
- the 2nd condition says that the amount of money entered must be greater than or equal to the cost of the item;
- the 3rd condition says that the change given must have been part of the float;
- the 4th condition says that the the money entered must equal the change given plus the cost of the item;
- the 5th condition says that the stock before must be equal to the stock after, to which is added the dispensed item;
- the 6th condition says that the float after, together with the change dispensed must equal the float before plus the amount entered (i.e. no money disappears)