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.

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

- 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:

  $[REPORT]$
• 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:

\[
V vendingMachine \\
cost : PROD \leftrightarrow \mathbb{N} \\
stock : \text{bag } PROD \\
float : \text{bag } COIN \\
\text{dom } stock \subseteq \text{dom } cost
\]
• The function \textit{cost} return the cost of a product in pence. For example,

\[
\text{cost}(\text{MarsBar}) = 25 \\
\text{cost}(\text{Penguin}) = 15
\]

• The bag \textit{stock} tells us how many items of each type are in stock. For example,

\[
\text{stock} = \{\text{Penguin} \mapsto 2\}
\]

means that there are just 2 penguins in the machine.

• The bag \textit{float} records the coins that are currently in the machine; for example

\[
\text{float} = \{100 \mapsto 2, 50 \mapsto 8, 5 \mapsto 20\}
\]

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

• \text{QUESTION: Why are \textit{stock} and \textit{float} bags and not sets or sequences?}

• The invariant dom \textit{stock} \subseteq dom \textit{cost} says that everything in the machine (i.e. in stock) must have a cost associated with it.
Here are the operations we shall specify:

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

\[ \text{InitVendingMachine} \]
\[ \Delta \text{VendingMachine} \]
\[ cost' = \{ \} \]
\[ stock' = [] \]
\[ float' = [] \]

- 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.

\[
\begin{align*}
\text{Price} \quad & \\
\Delta VendingMachine & \\
\text{item?} : \text{PROD} & \\
\text{price?} : \text{IN} & \\
\text{cost}' = \text{cost} \oplus \{\text{item?} \rightarrow \text{price}?) & \\
\text{stock}' = \text{stock} & \\
\text{float}' = \text{float} &
\end{align*}
\]
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 $\text{dom } new? \subseteq \text{dom } cost$ is implied by the invariant of $VendingMachine'$.

| Restock | $\Delta VendingMachine$
|---------|---------------------------
| $new? : \text{bag PROD}$ | $\text{stock}' = \text{stock } \cup new?$
|                      | $\text{float}' = \text{float}$
|                      | $\text{cost}' = \text{cost}$

- (Note that $\cup$ 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.

\[
\text{GoodsNotPriced} \equiv \text{VendingMachine} \\
\text{new}^? : \text{bag PROD} \\
\text{rep}! : \text{REPORT} \\
\neg (\text{dom new}^? \subseteq \text{dom cost}) \\
\text{rep}! = ‘\text{Some goods are not priced}’
\]
• We need an operation to report success...

\[
\begin{align*}
\text{Success} & \\
\text{rep! : REPORT} & \\
\text{rep! = ‘Okay’}
\end{align*}
\]

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

\[
\text{RestockOP} \equiv (\text{Restock} \land \text{Success}) \\
\lor \text{GoodsNotPriced}
\]
• This schema expands to ...

\[
\begin{align*}
\text{RestockOp} & \quad \Delta \text{VendingMachine} \\
n\text{ew}? & : \text{bag} \text{ PROG} \\
\text{rep!} & : \text{REPORT} \\
\text{cost}' & = \text{cost} \\
\text{float}' & = \text{float} \\
(\text{stock}' & = \text{stock} \uplus \text{new}? \land \\
\text{rep!} & = \text{‘Okay’}) \\
\lor \\
(\neg (\text{dom } \text{new}? \subseteq \text{dom cost}) \land \\
\text{stock}' & = \text{stock} \land \\
\text{rep!} & = \text{‘Some goods are not priced’})
\end{align*}
\]
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

\[ \text{sum} : \text{bag COIN} \rightarrow \mathbb{N} \]

is available, which takes a bag of coins and calculates how much is in the bag. For example, given a bag containing \(7 \times 2\text{p},\) and \(3 \times 5\text{p}\) coins,

\[
\text{sum}\{2 \mapsto 7, 5 \mapsto 3\} = (2 \times 7) + (5 \times 3) \\
= 14 + 15 \\
= 29\text{pence}
\]
• The basic Buy operation is as follows:

\[
\begin{align*}
\text{Buy} & \\
\Delta VendingMachine & \\
\text{in?}, \text{out!} : \text{bag COIN} & \\
\text{item?} : \text{PROD} & \\
\text{item?} \text{ in stock} & \\
\text{sum}(\text{in?}) \geq \text{cost}(\text{item?}) & \\
\text{out!} \subseteq \text{float} & \\
\text{sum}(\text{in?}) = \text{sum}(\text{out!}) + \text{cost}(\text{item?}) & \\
\text{stock}' \uplus \{\text{item?} \mapsto 1\} = \text{stock} & \\
\text{float}' \uplus \text{out?} = \text{float} \uplus \text{in?} & \\
\text{cost}' = \text{cost} & 
\end{align*}
\]
- 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 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)