LECTURE 16: VENDING MACHINE CASE STUDY

Software Engineering

Mike Wooldridge

- 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:

  VendingMachine

  $cost : PROD \pfun \mathbb{N}$

  $stock : \mathrm{bag}\, PROD$

  $float : \mathrm{bag}\, COIN$

  $\mathrm{dom}\, stock \subseteq \mathrm{dom}\, cost$

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:

  $[REPORT]$

- 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 $\mathrm{dom}\, stock \subseteq \mathrm{dom}\, cost$ 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

InitVendingMachine

$\Delta VendingMachine$

$cost' = \{\}$

$stock' = [\![]\!]$

$float' = [\![]\!]$

- 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.

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 $\mathrm{dom}\, new? \subseteq \mathrm{dom}\, cost$ is implied by the invariant of *VendingMachine*′.

Restock

$\Delta VendingMachine$

$new? : \mathrm{bag}\ PROD$

$stock' = stock \uplus new?$

$float' = float$

$cost' = cost$

- (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

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

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

- We need an operation to report success...

Success

$rep! : REPORT$

$rep! =$ 'Okay'

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

$$RestockOP \cong (Restock \wedge Success) \vee GoodsNotPriced$$

- This schema expands to ...

RestockOp

$\Delta VendingMachine$
$new? : \mathrm{bag}\, PROG$
$rep! : REPORT$

$cost' = cost$
$float' = float$
$(stock' = stock \uplus new? \wedge$
$rep! =$ 'Okay'$)$
$\vee$
$(\neg(\mathrm{dom}\, new? \subseteq \mathrm{dom}\, cost) \wedge$
$stock' = stock \wedge$
$rep! =$ 'Some goods are not priced'$)$

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

Buy

$\Delta VendingMachine$
$in?, out! : \mathrm{bag}\, COIN$
$item? : PROD$

$item? \in stock$
$sum(in?) \geq cost(item?)$
$out! \sqsubseteq float$
$sum(in?) = sum(out!) + cost(item?)$
$stock' \uplus \{item? \mapsto 1\} = stock$
$float' \uplus out? = float \uplus in?$
$cost' = cost$

- 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)