B Assignment 4a

Ken Robinson

19th April 2007

Name of assignment: ass4a
Assessment: 10 marks max.
Submission: ~cs2111/bin/giveB ass4a VM
Note: the name of the development (VM) is unimportant

This assignment is the first of two parts. The second part will involve implementation of the machines specified in the first part.

1 Purpose of this assignment

This assignment is concerned with modelling a vending machine. With this assignment we move further down the path to independent specification. There will be less guidance than with previous assignments. This document will give the machines and the operations required and the behaviour of the machines. There will be no statement about what variables are required to model the behaviour. The vending machine described here is a more realistic version of the vending machine in tutorial 4.

1.1 Context Machine

The context contains:

SNACK an enumerated set containing an item called null or nosnack null that doesn’t represent any snack.

COIN an enumerated set of coin denominations. Make them OneDollar, TwoDollar, and FiveDollar, representing one, two and five dollar coins. Assumption: all pricing in this vending machine is in cents.

CoinValue a constant function that converts a coin denomination to its value expressed as a natural number (cents).

<table>
<thead>
<tr>
<th>CONSTANTS</th>
<th>CoinValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROPERTIES</td>
<td>CoinValue ∈ COIN → N ∧ CoinValue = {OneDollar ↦ 100, …}</td>
</tr>
</tbody>
</table>

MONEY a definition denoting a bag of COIN.

MoneyValue a constant function that converts a bag of coins to their value in cents.
MoneyDiff a constant function that produces the difference between two bags of coins.

RESPONSE an enumerated set of appropriate responses.

1.2 VendingAPI machine

We will start at the top-level (API) machine that models the behaviour of a real vending machine. All the operations are robust.

1.2.1 Operations

User interface

\[ \text{status, price } \leftarrow \text{ChooseSnack(snack)} \] Choose a snack.

\[ \text{status } \leftarrow \text{InsertCoin(coin)} \] Insert one coin. The machine may arbitrarily reject the coin.

\[ \text{status, snack, change } \leftarrow \text{DispenseSnack} \] Dispense the chosen snack and give change. This operation should only deliver a real snack (and change) if enough money has been inserted. The change result is the value of the change, not a bag of coins.

\[ \text{status, change } \leftarrow \text{Cancel} \] Cancel the transaction and get back any money that has been inserted into the machine. The machine may malfunction and not return the money! Malfunctioning behaviour should be distinguished from good behaviour by the returned status.

Maintenance operations

\[ \text{status } \leftarrow \text{AddStock(snack, quantity)} \] add quantity of snack to the machine.

\[ \text{status } \leftarrow \text{NewPrice(snack, price)} \] Enter a price for a snack.

\[ \text{status } \leftarrow \text{AddCoins(coin, quantity)} \] Add some quantity of some coin to the coin box.

\[ \text{status } \leftarrow \text{RemoveCoins(coin, quantity)} \] Remove some quantity of some coin from the coin box.

1.3 Behaviour

Obtaining a snack: This vending machine separates choosing, paying and dispensing into separate operations that can be executed in any order. Money inserted into the machine remains available to pay for a snack until a snack is successfully dispensed. Thus a customer can choose a snack, insert some coins, and walk away without receiving a snack. A second customer will get to use the money inserted by the first customer. The new customer can choose a different snack. When a snack is dispensed the coin count is reset to zero.

Change: Change must come from the coins inserted in the machine, plus any supply already in the machine’s coin box. Change given is subject to the coins available, and ideally the machine should give change as near as possible to the correct change, but not exceeding the correct change. We want to make a profit! Malfunctioning behaviour should be distinguished from good behaviour by the returned status.

Note: change cannot be computed at the machine level in B, but it can be specified.
**Machine out of order:** The machine may be out of order due to a full coin box. Any attempt to initiate a purchase from a machine that is out of order should result in an out of order status and the abort of the operation.

If the coin box is empty then a status warning that no change can be given could be issued.

### 1.4 Vending machine

**Machine parameters:** The machine should have parameters for setting the maximum size of the coin box (in number of coins, regardless of denomination), and the maximum stock for any particular snack.

**Fragile Operations** This machine contains fragile versions of the operations in VendingAPI. These operations will have a similar signature with a different operation name without the *status* result. For example:

\[
  \text{price} \leftarrow \text{ChooseSnackF(snack)}
\]

as the fragile version of

\[
  \text{status, price} \leftarrow \text{ChooseSnack(snack)}
\]

**Note:** the fragile version of *Cancel* should not malfunction.

There may be other machines.

### 1.5 The Development

- Use *Vending* and *VendingAPI* as machine names for the machines containing the fragile and robust operations. Other machine names are not important; make your own choices.
- Choose variables and their types carefully.
- Carefully structure the development.
- The AutoProver should be run on all proof obligations.
- Check any remaining POs in order to eliminate errors.