B Exercises 5
Supermarket

The objective of this set of tutorial exercises is to develop a specification of a simple supermarket, involving a number of machines.

In this tutorial we will build a specification case study using a modified version of the Bag machine developed in tutorial 4. The modifications to the Bag machine, now named BagOBJ, consist of

1. The replacement of the parametric set VALUE by a Value_TYPE machine.
2. The separation of the bag mathematics into a BagMath machine. The bag mathematics is also extended.
3. A change to the signature of the NewBag operation to provide a workaround of a problem that can occur when the BagOBJ machine is included in another machine. The problem is mainly due to the lack of sequential composition in specification machines.

1 The machines

1.1 The Value_TYPE machine

MACHINE Value_TYPE
SETS PRODUCT
CONSTANTS maxproduct
PROPERTIES maxproduct : NAT1 &
    card(PRODUCT) = maxproduct
DEFINITIONS VALUE == PRODUCT
END

1.2 The BagMath machine

MACHINE BagMath
SEES Value_TYPE
DEFINITIONS
    Bag(X) == X +-> NAT1
CONSTANTS
    emptyBag,
    BagCount,
    BagUnion
PROPERTIES
    emptyBag : VALUE --> {0} &
    BagCount : Bag(VALUE) --> (VALUE --> NAT) &
    !bb.(bb : Bag(VALUE) => BagCount(bb) = emptyBag <+ bb) &
    BagUnion : Bag(VALUE)*Bag(VALUE) --> Bag(VALUE) &
    !(b1,b2).(b1 : Bag(VALUE) & b2 : Bag(VALUE) => (BagUnion(b1,b2) = {vv,nn | vv : dom(b1)\dom(b2) & (nn : NAT1 & nn = BagCount(b1)(vv) + BagCount(b2)(vv) )})
1.3 The BagOBJ machine

MACHINE BagOBJ(maxbag)

CONSTRAINTS maxbag : NAT1

SEES BagMath, Value_TYPE, Bool_TYPE

SETS BAG

PROPERTIES card(BAG) = maxbag

VARIABLES

bags, freq

INVARIANT

bags <: BAG &
freq : bags --> Bag(VALUE)

INITIALISATION

bags, freq := {}, {}

OPERATIONS

/* Ideally, we would write this operation as \verb|ok, bag <-- NewBag|, returning the new bag as output. Unfortunately, this would limit the use of the operation in machines that include BagOBJ."*/

NewBag(bag) =
PRE bag : BAG & bag /: bags
THEN bags := bags / {bag} || freq(bag) := {}
END;

DeleteBag(bag) =
PRE bag : bags
THEN bags := bags - {bag} || freq := {bag} <<| freq
END;

AddItem(bag, item, quantity) =
PRE bag : bags & item : VALUE & quantity : NAT1 &
   BagCount(freq(bag))(item) + quantity <= maxcount

END
2 THE SUPERMARKET

THEN freq(bag)(item) := BagCount(freq(bag))(item) + quantity
END;

RemoveItem(bag, item, quantity) = 
PRE bag : bags & item : VALUE & quantity : NAT
THEN IF quantity < BagCount(freq(bag))(item)
THEN freq(bag)(item) := freq(bag)(item) - quantity
ELSE freq(bag) := {item} <<| freq(bag)
END
END;

bagUnion(bag1, bag2) = 
PRE bag1 : bags & bag2 : bags
THEN freq(bag1) := BagUnion(freq(bag1),freq(bag2))
END;

DeleteBagContents(bag) = 
PRE bag : bags
THEN freq(bag) := {}
END;

in <-- inBag(bag,item) = 
PRE bag : bags & item : VALUE
THEN in := bool(item : dom(freq(bag)))
END;

count <-- Count(bag,item) = 
PRE bag : bags & item : VALUE
THEN count := BagCount(freq(bag))(item)
END

2 The Supermarket

We will model a supermarket using two machines: one named SuperMarket modeling the products on the shelf of the supermarket and the prices of the products; and the second named Customer modelling the customers, with their trolleys, within the supermarket. Customers checking and obtain a trolley, put products into their trolleys, or put products back on the shelf, and finally checkout, paying for the products in their trolleys, or they may abandon their trolleys.

2.1 The SuperMarket machine

The SuperMarket should be unparameterized and SEES Value>Type and BagMath.

The state should consist of three variables

shelf modelling the supermarket shelf of products,

prices modelling the prices of products. Every product on the shelf must have a price, and
takings  modelling the current takings of the supermarket. This will be the total money value, ignoring currency. This may be modelled as a natural number.

Specify an appropriate invariant, and an initialisation.

The machine should provide the following operations:

AddProduct(product, quantity) add quantity items of product to the supermarket shelf.

UpdatePrice(product, newprice) update the price of product to newprice.

RemoveProduct(product, quantity) remove quantity items of product from the shelf.

AddBagProduct(bagproduct) add a bag of products to the shelf.

AddToTakings(amount) add amount to takings.

2.2 The Customer machine

The customer machine models customers with trolleys. The trolleys will be allocated and controlled by the BagOBJ machines.

The Customer machine should have one parameter, maxcustomer, which will set the limit on the number of trolleys available. The machine INCLUDES SuperMarket and BagOBJ(maxcustomer).

The machine SEES Value_TYPE, BagMath, and Bool_TYPE.

For convenience, include DEFINITIONS TROLLEY == BAG and trolleys == bags.

This supermarket uses the latest hi-tech trolleys that register the total cost of the goods in the trolley.

Declare a variable trolleycost that gives the current total cost of the items of in a trolley for all currently “active” trolleys. Specify an invariant that ensure that the value of trolleycost is consistent with the above. Also ensure that the price of every product item in the trolley is known.

Give an appropriate initialisation.

The machine should have the following operations

ok, trolley <-- CheckIn allocate a trolley to a new customer. The Boolean status ok will indicate whether the allocation is successful.

cost <-- AddToTrolley(trolley, product, quantity) take quantity of product from the supermarket shelf and add to the trolley. The operation shows the customer the total cost of the products in the trolley.

totalcost <-- RemoveFromTrolley(trolley, product, quantity) remove quantity of product from the trolley and return them to the supermarket shelf. Show the total cost.

change <-- CheckOut(trolley, payment) checkout of the supermarket, paying for the goods in the trolley and returning the trolley to the pool.

AbandonTrolley(trolley) abandon the trolley, returning all goods in the trolley to the supermarket shelf.