

B Exercises 4 Specifications

The objective of this set of tutorial exercises is to develop some specifications.

1. Wordsworth, Section 3.5, exercise 3.4

Define a machine to represent an array. The machine should have two parameters, a natural number to fix the upper limit of the indexes (the lower limit is 1) and a set for the type of values to be stored in the array. Use a partial function from indexes to values for the model. There should be operations to

- read the value at a given index;
- store a value at a given index;
- exchange the values at two indexes.

The initialization and the operations to read values from the array and to exchange values in the array should warn the user of the array not to look at values at an index that has not previously had a value stored.

2. A Vending Machine

We want to produce a simple model of a vending machine that delivers snacks such as Mars Bars, Snickers, Cheezels, etc. We will make this very simple —simpler than most normal vending machines— as we don't yet have enough machinery to handle a more realistic example.

- (a)
- The set SNACK can be modelled using an enumerated set.
 - We need a mapping from snack to price. Price is modelled using natural numbers (cents).
 - We need to have an (enumerated) set of COIN.
 - We need a mapping from coin to value.
 - We will assume that a customer can only deposit one coin and the value of that coin must be at least the price of the chosen snack.
 - We want the following operations:
 - Recharge(snack, quantity)** an operation used by the operator for adding an extra quantity of a snack.
 - UpdatePrice(snack, price)** an operation used by an operator for updating the price of a snack.
 - ChooseSnack(snack, coin)** an operation used by a customer for purchasing a snack. In our model we assume concurrent selection of snack and depositing of coin.
 - Ensure that adequate state invariant and operation preconditions have been specified.
When you get the chance, check the proof obligations and animate the machine to check whether it has any unexpected behaviour.
- (b) Specify a new machine that contains robust versions of the operations in the preceding machine.

3. A Bag machine

A *bag* is a mathematical aggregate construct that can contain multiple instances of a value, but for which there is no ordering. As an example of a bag consider a bag containing jelly beans: the bag will contain a certain number of red jelly beans, black jelly beans etc. Generally with bags we do not explicitly represent zero occurrences of an item: the bag either contains some black jelly beans, or it doesn't.

Develop a machine *Bag* with parameters *VALUE* and *maxbag* being the set of values to be included in bags and the maximum number of bags, respectively.

Model bags as partial functions in $VALUE \rightarrow \mathbb{N}_1$.

Provide the following operations:

- (a) $bag \leftarrow NewBag$: create a new bag.
- (b) $DeleteBag(bag)$: delete an existing bag.
- (c) $AddItem(bag, item)$: add *item* (in set *VALUE*) to the bag, *bag*.
- (d) $in \leftarrow inBag(item, bag)$: return *TRUE* or *FALSE* depending on whether *item* is in *bag* or not, respectively.
- (e) $count \leftarrow Count(item, bag)$ return the number of occurrences of *item* in *bag*, zero if it is not in the bag.
- (f) $BagUnion(bag1, bag2)$: form the bag union of *bag1* and *bag2* with the result in *bag1*.
Note: in bag union the frequencies should be summed.