Advanced Programming (I00032)
Modeling a state machine

Assignment 7

1 Modeling the FNWI Vending Machine

In the restaurant at the ground floor of the *Huygens* building of FNWI, there is a vending machine that is able to produce cooled items like soda and energy bars. Make a model of this machine of type `State In → [Trans Out State]`. The possible inputs for this vending machine are:

- Press a key on the interface. The keys are buttons on a touch screen, but the buttons seem to be rather stable. The real machine highlights keys that are 'useful', you can ignore this. Just specify that pressing these buttons has no effect.
- Insert a coin of some value. Ignore the fact that this machine processes bank passes.

The output is either:

- a message on the screen, or
- a product, or
- some change in case the price of the product was less than the inserted amount of money.

A single input can yield several outputs, e.g. producing an energy bar and some change.

The following types can be convenient for your specification. You are free to use different types if that suits your model better.

```
:: In    = D Digit | Start | Reset | Return | C Coin
:: Out   = Display [String] | Product Product | Change [Coin]
:: Digit = Digit Int // numbers 0..9
:: Coin  = Coin  Int // 5, 10, 20, 50, 100, 200 cents
```

The specification state keeps track of available products. It is not necessary to list the complete assortment of the actual FNWI vending machine. Your specification does not have full knowledge of the state of the machine. The amount of items in stock and the amount of change are unknown. Your specification has to anticipate that items are out of stock.

It is not necessary that your specification checks the contents of all messages on the display.

The inputs `D Digit` are intended as as digits on the keypad. The new type `Digit` makes it easier to generate only the digits between 0 and 9. Deriving generation for values for this type would give us values like `Digit −1` and `Digit` *maxint*. Generating only the digits between 0 and 9 in a pseudo random order can be done with:

```
ggen {|Digit|} n r = randomize (map Digit [0..9]) r 10 (λ_.[])
```

A simple application of this generic generation is:

```
list :: [Digit]
list = ggen {|*|} 2 aStream
```

The expression `list` yields [`Digit 9`, `Digit 3`, `Digit 0`, `Digit 2`, `Digit 8`, `Digit 5`, `Digit 6`, `Digit 1`, `Digit 7`, `Digit 4`]. Similar definitions can be given for the generation of other input values. If you provide a tailor made instance for the generation of `Coins`, the generation of values for the type `In` can be derived.

Allow non-deterministic behavior of the vending machine since in the specification we do not know if all items are in stock and if there are enough coins to give change.

2 An Implementation of the Vending Machine

Make a simple implementation of vending machines satisfying the specification above. Keep this as simple as possible. The only purpose of this implementation is that there is something to test. It might be convenient to reuse the specification of sellable items from the specification. This implementation is of course deterministic. It knows for instance the amount of items available and the number of coins of each kind. Assume for instance that there are initially two items of each product available in the machine. You are free to make some differences between this implementation and the specification.

It is probably most convenient if you implement this machine as a function of type `MachineState In` → (`[Out]`, `MachineState`) in Clean. There is nothing wrong with implementing this machine in an arbitrary programming language like C++ or Java, but that requires a foreign function interface in the tests.

3 MBT of the Vending Machine Implementation

Do a model-based test of the implementation of the vending machine with G∀st using your specification. You have to use the function `testConfSM` (test conformance for state machines).

Check if the differences between the model and the specification made are found.

Deadline

The deadline for this exercise is October 28, 23:59. Note that there are no lectures in week 44 and 45 due to the autumn break.