Principles of Programming Languages

2013.07.05

Notes

- Total available time: 1h 30'.
- You may use any written material you need.
- You cannot use computers, phones or laptops during the exam.

1 Haskell (13 points)

Consider the *state monad* as seen in class, i.e. defined by:

```
newtype State st a = State (st -> (st, a))

instance Monad (State state) where
  return x = let f t = (t,x)
             in State f
  State f >>= g = State (\oldstate ->
                           let (newstate, val) = f oldstate
                               State f' = g val
                           in f' newstate)

getState :: State state state
getState = State (\state -> (state, state))

putState :: state -> State state ()
putState new = State (\_ -> (new, ()))
```

1. Define the monadic function `mapListM :: (t -> State st a) -> [t] -> State st [a]` that applies its first argument (another monadic function, as you can see by the signature) to every element of the list passed as its second argument (i.e. like a *map*).

2. Define the monadic function `numberList :: Num st => [st] -> State st [(st, st)]`, based on `mapListM`, that takes as input a list of numbers and returns a list of pairs of numbers (x, y), where the first component is the same as the value x at the same position in the input list, while y is the *state* when x was reached. The state is incremented by x, when x is reached. For example:

   ```
   let State f = (numberList [1,3,22,-5])
   in  f 0     -- the initial value of the state is 0
   ```

   should evaluate to `(21, [(1,1),(3,4),(22,26),(-5,21)])`, i.e. the last value of the state is 21.

2 Scheme/Ruby (9 points)

Implement an analogous of the `numberList` function of the previous exercise either in Scheme or in Ruby (it is not necessary to use the same monadic construction).

For instance, the procedure call `(numberlist '(1 3 22 -5) 0)`, where the second parameter is the initial state, should return `'((1 . 1) (3 . 4) (22 . 26) (-5 . 21))`.

3 Prolog (10 points)

Define a Prolog predicate `subsetsum` that checks if there exists a sublist of the first argument such that the sum of all its elements is equal to the second argument.

E.g.: `subsetsum([1,2,3,4],19)` is false, while `subsetsum([1,2,3,4],7)` is true.

Solutions

Haskell

```
mapListM f []     = do return []
mapListM f (x:xs) = do x1 <- f x
                       xs1 <- mapListM f xs
                       return (x1:xs1)

numberList list = mapListM incrState list
                  where incrState v = do cur <- getState
                                         putState (cur+v)
                                         return (v,cur+v)
```

Scheme

```
(define (numberlist lst state)
  (if (null? lst)
      '()
      (let* ((x        (car lst))
             (newstate (+ x state)))
        (cons (cons x newstate)
              (numberlist (cdr lst) newstate)))))
```

Prolog

```
# generates all the possible subsets:
subset([], []).
subset([X|Xs], [X|Ys]) :- subset(Xs, Ys).
subset([_|Xs], Ys)     :- subset(Xs, Ys).

# sum of the elements:
listsum([], 0).
listsum([X|Xs], R1) :- listsum(Xs, R2), R1 is R2+X.

subsetsum(L, X) :- subset(L, S), listsum(S, X).
```