Real-world Functional Programming

Coursework Part I Report

14274056 Junsong Yang (psyjy3)

November 4, 2019

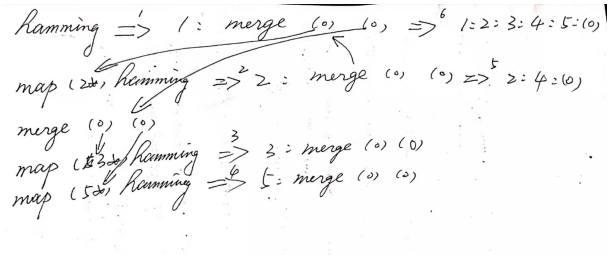
1 Task I.1

The key ideas for this task are to explore the infinite data structure in haskell and to gain a better understanding of lazy evaluation in haskell.

```haskell
hamming :: [Int]
hamming = 1 : merge (map (2*) hamming) (merge (map (3*) hamming) (map (5*) hamming))

merge :: [Int] -> [Int] -> [Int]
merge xss@(x:xs) yss@(y:ys)
  | x == y = x : merge xs ys
  | x < y  = x : merge xs yss
  | x > y  = y : merge xss ys
```

(a) Hamming Function Definition



(b) Cyclic Graph

Figure 1: TaskI.1

Hence, the hamming function can be defined as a infinite list in a recursive manner. As Figure 1 (a) shows, the type of hamming function is a list of Int. This implementation using map function to calculate 2x, 3x and 5x and then merge then together as Figure 1 (b) demonstrated how it was evaluated.

2 Task I.2

The key ideas of this task are extend the current function to caculate sums and averages of range of cells and to explore the weakness of this evaluator then provide a solution.

```haskell
data Exp = Lit Double
         | Ref CellRef
         | Sum CellRef CellRef
         | Avg CellRef CellRef
         | App BinOp Exp Exp



evalCell :: Sheet Double -> Exp -> Double
evalCell _ (Lit v)         = v
evalCell s (Ref r)         = s ! r
evalCell s (Sum c1 c2)     = foldr (+) 0 l
  where
    l = [ s ! (c, r) | c <- [(fst c1)..(fst c2)],
                       r <- [(snd c1)..(snd c2)] ]
evalCell s (Avg c1 c2)     = (foldr (+) 0 l) / (fromIntegral llen)
  where
    l = [ s ! (c, r) | c <- [(fst c1)..(fst c2)],
                       r <- [(snd c1)..(snd c2)] ]
    llen = length l
evalCell s (App op e1 e2) = (evalOp op) (evalCell s e1) (evalCell s e2)
```

Figure 2: Extended Evaluator

Figure 2 demostrates how evalCell function was extended to support Sum and Avg expression. The key idea of this implemetation is to given to CellRef c1 and c2, find every cell in between. Next, lookup corresponding values in the given sheet s. Then put all the values found in sheet s in list l. Finally using foldr to calculate the sum of all values.

The similiar idea was used to implemetate the evaluation of Avg expression but a further step was taken to yield the average value.

The most obvious weakness of this evaluator is that it will stuck in infinite loop when there are cyclic references in the given sheet. As the type signature indicates, the evalCell function will return a Double for every given sheet s and an Exp

3 Task I.3

4 Task I.4

5 Task I.5