National Undergraduate Programme in Mathematical Sciences
National Graduate Programme in Computer Science

Functional Programming in Haskell

Mid-semester Examination, I Semester, 2019–2020

Date : September 25, 2019
Time : 0930 – 1230

Marks : 100
Weightage : 30%

This paper has three parts. Each Part A question is worth 4 marks, and each Part B question is worth 6 marks. Part C is worth 50 marks. For Part A, provide answers in the answer sheet, and write your option as well as the answer, like "(a) 225" or "(b) `True`".

Part A

1. What is the result of `length $ filter (>= -10) [35,32..(-25)]`?

   (a) 14 (b) 15 (c) 16 (d) 17

2. What is the result of `length $ takeWhile (>= 5) (reverse [33,31..4])`?

   (a) 0 (b) 5 (c) 10 (d) 15

3. Which of the following is a possible type of the function `foldl (++)`?

   (a) `[a] -> a` (b) `[[a]] -> [a]`
   (c) `a -> [a] -> a` (d) `[a] -> [[a]] -> [a]`

4. Suppose `(++)` is defined as follows:

```
[]     ++ ys = ys
(x:xs) ++ ys = x: xs ++ ys
```

   How many times is the second line of the definition invoked in the computation of the following expression?

```
foldr (++) "" ["abcde", "fghij", "klmno", "pqrst", "uvwxy", "z"]
```

   (a) 75 (b) 101 (c) 25 (d) 26

5. What is the position of `(5,2)` in the following list (counting positions from 0)?

```
[(j,i) | i<-[0..9], j <- [(i+1)..9]]
```

   (a) 17 (b) 19 (c) 24 (d) 26

Part B

1. Fill in values for `f` and `v`: `reverse = foldl' f v`. What are the types of your `f` and `v`?

2. What is the result of `foldr (\x (y:ys) -> x:x+y:ys) [0] [0..9]`?

3. Fill in values for `f`, `v1` and `v2`: `l1 ++ l2 = foldr f v1 v2`. What are the types of your `f`, `v1` and `v2`?

4. Given the following definition of `fib :: Int -> Int`, trace the computation of `fib 6`.

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

   How many times do you evaluate `fib 1` in the course of this computation? [▸]

5. Define the function `subSeq :: [Char] -> [Char] -> Bool` such that `subSeq xs ys` is `True` exactly when `xs` is a subsequence of `ys`, i.e. `xs` is obtained by omitting some characters in `ys` and reading the remaining characters from start to end.

Part C

1. Define the function `elemIndex :: Char -> String -> Maybe Int` with the following behaviour.

   If `x` is not in the list `ys`, the return value is `Nothing`. Otherwise it is `Just i`, where `i` is the least such that `ys!!i == x`.

   (10 marks)

2. Modify the `fib` function given earlier to get a function `fibAndCount :: Int -> (Int, Int)` that on input `n` computes `fib n` as well as the number of times addition is used in the computation of `fib n`.

   For instance: `fibAndCount 5` returns `(5, 7)` and `fibAndCount 19` returns `(4181, 6764)`. (*How do I know those numbers? That's how smart I am!*)

   (10 marks)

3. Trace the computation of `fib 3` for the following definition of `fib`. *(12 marks)*

```
fib n = fibs !! n
fibs = 0:1:zipWith (+) fibs (tail fibs)
```

   Recall that `(!!)` is defined by:

```
(x:xs) !! 0 = x
(x:xs) !! n = xs !! (n-1)
```

and `zipWith` is defined by:

```
zipWith f [] _ = []
zipWith f _ [] = []
zipWith f (x:xs) (y:ys) = f x y: zipWith f xs ys
```

4. A finite list of integers is a *dyadic numeral* if each entry is either 1 or 2. Its value is a natural number defined by the following expression:

```
value :: [Int] -> Int
value ds = sum [(2^(maxInd-i))*(ds!!i) | i <- [0..maxInd] ]
    where maxInd = length ds - 1
```

Note that the empty list is also a dyadic numeral, with value 0. The advantage of dyadic numerals over binary numbers (as you can verify at leisure) is that each natural number has a unique dyadic representation. The representations for 0 to 5 are, respectively, `[]`, `[1]`, `[2]`, `[1,1]`, `[1,2]`, and `[2,1]`.

(a) Give a direct recursive definition of `value :: [Int] -> Int` without using `(!!)` or `(2^)`. *(8 marks)*

(b) Define the function `dyadic :: Int -> [Int]` such that `dyadic n` gives the dyadic representation of the natural number `n`. *(10 marks)*