National Undergraduate Programme in Mathematical Sciences
National Graduate Programme in Computer Science
Functional Programming in Haskell
Mid-semester Examination, I Semester, 2018–2019

Date : 26 September, 2018 | Marks : 100
Time : 0930 – 1230 | 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.

Part A

1. What is the result of `length $ filter odd [35,32..(-19)]`?

   (a) 9 (b) 10 (c) 11 (d) 18

2. What is the result of `length $ takeWhile (>= 5) ([33, 31..4] ++ [20..1])`?

   (a) 14 (b) 30 (c) 15 (d) 29

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

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

4. How many times is the `(:)` function applied in the computation of the following expression?

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

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

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

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

   (a) 25 (b) 26 (c) 24 (d) 17

Part B

1. Give values of `f` and `v` such that `foldl f v` is the same as `reverse`? What are the types of `f` and `v`?

2. Trace the computation of `foldr f [0] [0..4]` where `f x (y:ys) = x+y:y:ys`.

3. How would you express `length` in terms of `map`, `const` and `sum`?

4. Given the following definition of `fib :: Int -> Int`, trace the computation of `fib 5`.

```
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 `isPrefixOf :: [Char] -> [Char] -> Bool` such that `isPrefixOf xs ys` is `True` exactly when `xs` is a prefix of `ys`.

   For instance, `isPrefixOf "abc" "abcdef"` is `True`, while `isPrefixOf "abc" "adebfc"` is `False`.

Part C

1. Define the function `elemIndex :: Int -> [Int] -> 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. Define the function `splitAt :: Int -> [a] -> ([a],[a])` such that `splitAt n xs` is the same as `(take n xs, drop n xs)`. You should not use `take` or `drop` in your definition, and should traverse the list exactly once. *(10 marks)*

3. Trace the computation of `fib 3` for the following definition of `fib`. *(10 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 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^)`. *(6 marks)*

(b) Give a definition of `value` that uses `foldl'`. *(4 marks)*

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