CSC4020Z: Functional Programming

Practical Assignment: Haskell

Department of Computer Science
University of Cape Town, South Africa

DUE: Monday, 16th March, 2020, 10.00 AM

Assignment Instructions and Description

The *Glasgow Haskell Compiler* (GHC) provides an interactive interpreter (GHCi), which will be the main Haskell tool used in this module. The usual way to write Haskell programs is to have two windows open: one for a text editor to write your code, and the other for GHCi so that you can regularly load and test your code. For example, a Haskell script defining the following function:

$double\ x = x + x$

And named: *script1.hs* can be compiled via typing:

ghci script1.hs

GHCi should load and you should see something like:

...
[1 of 1] Compiling Main (script1.hs, interpreted)
Ok, one module loaded.
$*$*Main>*

In this case *script1* can then be tested via typing the function name and some value, for example:

double 7

Implement Haskell functions that provide solutions to the following computational problems given in each of the three (3) parts of this assignment:

Part A: Four (4) questions: 10 marks.

Part B: Six (6) questions: 25 marks.

Part C: One (1) question: 15 marks.

Submit your scripts in a single ZIP file via VULA assignments tab, using your student number as the ZIP file name (e.g.: XYZZYX001.ZIP) and each script named according to the corresponding part and question number (e.g.: *partA-question1.hs*, *partB-question2.hs*, ...).

Part A [10 Marks]

1. Define a function product that produces the product of a list of numbers. For example: *product [2,3,4]*, should produce the solution: *24*.

   (2 Marks)

2. The library function *last* selects the last element of a non-empty list; for example: *last [1,2,3,4,5] = 5*. Write another definition for the *last* function in terms of the other library functions.

   (2 Marks)

3. Using library functions, define a function *halve :: [a] − > ([a],[a])* that splits an even length list into two halves. For example: *halve [1,2,3,4,5,6]*, should produce: *([1,2,3],[4,5,6])*.

   (2 Marks)

4. Consider a function *safetail :: [a] − > [a]* that behaves in the same way as the *tail* function except that it maps the empty list to itself rather than producing an error. Using *tail* and the function *null :: [a] − > Bool* that decides if a list is empty or not, define *safetail* using only: (1) a conditional expression, (2) guarded equations, and (3) pattern matching. Defining three different functions for solving these three different problems is an acceptable approach.

   (4 Marks)

Part B [25 Marks]

1. Suppose that arithmetic expressions built up from integers, addition and multiplication are represented using the following types:

   $\textbf{\textit{data Expr}} = \textit{Val Int} \mid \textit{App Op Expr Expr}$

   $\textbf{\textit{data Op}} = \textit{Add} \mid \textit{Mul}$

   Define functions:

   $\textbf{\textit{eval}} :: \textit{Expr} \rightarrow \textit{Int}$

   $\textbf{\textit{values}} :: \textit{Expr} \rightarrow [\ \textit{Int}\ ]$

   that respectively evaluate an expression to its integer value, and return the list of integer values contained in an expression.

   (3 Marks)

2. Define a function:

   $\textbf{\textit{delete}} :: \textit{Int} \rightarrow [\ \textit{Int}\ ] \rightarrow [\ \textit{Int}\ ]$

   that deletes the first occurrence (if any) of a value from a list. For example, *delete* 2 [1, 2, 3, 2] should give the result [1, 3, 2].

   (3 Marks)

3. Using *delete*, define a function:

   $\textbf{\textit{perms}} :: [\ \textit{Int}\ ] \rightarrow [\ [\ \textit{Int}\ ]\ ]$

   that returns all permutations of a list, given by all possible re-orderings of its elements. For example, *perms* [1, 2, 3] should return:

   [[1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]

   (5 Marks)

4. Define a function:

   $\textbf{\textit{split}} :: [\ \textit{Int}\ ] \rightarrow\ [([\textit{Int}], [\textit{Int}])]$

   that returns all splits of a list into two non-empty parts that append to give the original list. For example, *split* [1, 2, 3, 4] should return:

   [([1], [2, 3, 4]), ([1, 2], [3, 4]), ([1, 2, 3], [4])]

   (4 Marks)

5. Using *split*, define a function:

   $\boldsymbol{exprs} :: [\ Int\ ] \rightarrow\ [\ Expr\ ]$

   that returns all expressions whose list of values is a given list. For example, *exprs* [1, 2, 3] should return all e for which *values* e = [1, 2, 3].

   (6 Marks)

6. Using your answers to the previous parts, define a function:

   $\boldsymbol{solve} :: [\ Int\ ] \rightarrow Int \rightarrow\ [\ Expr\ ]$

   that returns all expressions whose list of values is a permutation of the given list and whose value is the given value.

   For example, *solve* [1, 2, 3, 4] 10 should return all expressions e for which *values* e is a permutation of [1, 2, 3, 4] and *eval* $e = 10$.

   (4 Marks)

Part C [15 Marks]

Design and implement a *Turing machine* that operates only on the following input and output symbols: $\{a,\ b\}$ and computes a function $f(x)$, where $f(x)$ outputs only the symbol a if x is a palindrome and outputs the symbol b otherwise. A palindrome is a symmetrical string, i.e., if we reverse the order of its symbols, it is still the same string.

For example:

$f(bba) = b$

$f(baab) = a$

$f(abab) = b$

$f(aaa) = a$

$f(babab) = a$

The input string x can be up to length $N = 4$, but must contain only the symbols: a, b. The Turing machine terminates after the last symbol in the string of symbols has been processed or if it is given an empty string $\{\ \}$. In the latter case it terminates immediately with the output a.

In a text-file (named README), give the sequence of possible Turing machine state configurations that are used during the computation of f with the input string: aba

The list of possible configurations should be in the following format:

$< CurrentState, InputSymbol, OutputSymbol, NewState >$