universität innsbruck

Functional Programming | SS 2020 | LVA 703025

Exercise Sheet 8, 10 points | Deadline: Monday, May 11, 2020, 10am

- Watch the lecture of week 8 and study part 5 of the slides[1].
- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- For Exercise 8.1.1. you have to upload a drawing. Please upload the drawing as a `.jpg`, `.png` or `.pdf`-file alongside your .hs-file.
- You can use the template .hs-file that is provided on the proseminar page[2].
- Your .hs-file should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 8.1 *Lists* 3 p.

In this exercise the goal is to provide more insight about `foldl` and `foldr`. Consider the list `xs` of integers `[1,2,3,4]` without syntactic sugar `1 : (2 : (3 : (4 : [])))`.

1. Draw the abstract syntax tree of the latter representation of the list above and explain the difference between `foldl` and `foldr`.

   Substitute `+` for the `Cons` operator `:` and `0` for `[]` in the above list representation.
   Does this represent the calculation done with `foldl (+) 0 xs` or `foldr (+) 0 xs`?
   What happens if we take `(*)` instead of `(+)`? Justify your answers. (1 point)

2. The powerlist of a list `ys` contains every list, where arbitrary elements were removed from `ys`. For this exercise the only allowed functions are `foldl`, `foldr`, `(++)` and `map`. Give their most general type signatures.

   (a) Implement a function `powerlistFR` using `foldr`. (1 point)

   (b) Implement a function `powerlistFL` using `foldl`. (1 point)

   Examples:
   ```
   powerlistFL [1,1,2] = [[1,1,2],[1,1],[1,2],[1],[1,2],[1],[2],[]]
   powerlistFR [1,2,3] = [[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]
   ```
   Note that the order of the lists in these examples is not relevant, i.e., your solution may return the elements of the powerlist in a different order than in the examples.

Exercise 8.2 *Lists, Folds, Insertion Sort* 2 p.

Consider the function `insertS x ys` which adds an element `x` before the first element in `ys` which is greater. Use `insertS` to implement insertion sort in two ways in Haskell. Insertion sort works by iteratively inserting elements at the correct position into a sorted list[3].

1. Implement insertion sort by writing a recursive function `iSort`.

   Example: `iSort [3,1,2,1,6,12] = [1,1,2,3,6,12]` (1 point)

2. Implement insertion sort using `foldr` in a function `iSortFold`. (1 point)

[1] `http://cl-informatik.uibk.ac.at/teaching/ws19/fp/slides/05x1.pdf`
[2] `http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises`

Exercise 8.3 *foldr-function* **2 p.**

Write the following functions using `foldr`. You may find lambda expressions useful.

1. Consider the prelude function `all` which defined in the slide Part 5, page 20. Define `all` without recursion, but via `foldr` as `all_fold`. (1 point)

2. Consider a function which converts a list of digits (represented as `Integers`) into an `Integer`:

   ```
   dig2int :: [Integer] -> Integer
   dig2int [] = 0
   dig2int (x:xs) = x + 10 * dig2int xs
   ```

   where the output of `dig2int [2, 1, 5]` is `512`. Define `dig2int` without recursion, but via `foldr` as `dig2int_fold`. (1 point)

Exercise 8.4 *Zip, List Comprehensions* **3 p.**

1. Write a function `number :: [a] -> [(Int,a)]` which numbers all elements of a list with their position. For instance, `number "hello" = [(0,'h'),(1,'e'),(2,'l'),(3,'l'),(4,'o')]`.

   Define `number` with the help of `zip` or `zipWith`. It is neither allowed to use recursion nor is the `(!!)` operator allowed. (1 point)

2. Write a function `evenProdSum` which given a list of `Ints` $[x_1, \ldots, x_n]$, computes the sum $2x_2 + 4x_4 + 6x_6 + \ldots + nx_n$ (if n is even), or $2x_2 + 4x_4 + 6x_6 + \ldots + (n-1)x_{n-1}$ (if n is odd).

   For instance, `evenProdSum [3,17,8,9,5,7] = 2 * 17 + 4 * 9 + 6 * 7 = 112`.

   Define `evenProdSum` without using recursion. Instead, list comprehensions, and functions like `number` from part 1 of this exercise and the Prelude function `sum` might be useful. (2 points)