Functional and Logic Programming

Home Assignment 4

Due: Saturday, 20.5.2019 - 23:55

Instructions

- Please create a source file called ***hw4.hs*** and put all the answers there.
  The file should start with a comment which contains your **full name** (in English) and **ID**

  ```
  -- John Doe
  -- 654321987
  ```

- Make sure the file **is valid** by loading it into GHCi.
  A valid file will load without any errors or warnings.
- If you need a function but you don't know how to implement it - just write it's signature (name and type) and put `undefined` in the function's body.
  That way you'll be able to load the file even though it contains references to undefined names.
- When writing a function - write both the **type** and the **body** of the function.
- Be sure to write functions with **exactly the specified name** (and **type signature** - if it is provided) for each exercise.
  You may create additional auxiliary/helper functions with whatever names and type signatures you wish.
- Try to write **small functions** which perform just **a single task**, and then **combine** them to create more complex functions.

Exercises

1. This exercise deals with infinite lists.

   Define **naturals-** an infinite sequence of the natural numbers
   (without using [1..])

   Example:
   ```
   *Main> :t naturals
   naturals :: [Integer]
   *Main> naturals
   [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,..continues
   ```

   a) Define **squares**- an infinite sequence of the natural numbers squared

   Example:
   ```
   *Main> :t squares
   squares :: [Integer]
   *Main> squares
   [1,4,9,16,25,36,49,64,81,100,121,144,169,..continues
   ```

   b) Define **threes**- an infinite sequence of multiplications of 3.

   Example:
   ```
   *Main> :t threes
   threes :: [Integer]
   *Main> threes
   [3,6,9,12,15,18,21,24,27,30,33,36,39,..continues
   ```

   d) Define **res-**
   an infinite sequence which mixes together `naturals, squares` and `threes`.
   The first element of the list should be the first element from `naturals`
   The second element of the list should be the first element from `squares`
   The third element of the list should be the first element from `threes`
   The fourth element of the list should be the second element from `naturals`
   The fifth element of the list should be the second element from `squares`
   The sixth element of the list should be the second element from `threes`
   …

   Example:
   ```
   *Main> :t res
   res :: [Integer]
   *Main> res
   [1,1,3,2,4,6,3,9,9,4,16,12,5,25,15,6,36,18,..continues
   ```

c) Implement a function:
`switch` **:: [a] -> [a]**

which takes a list and switches between the i^{th} and the $i + 1^{th}$ elements.

Note: You can assume you will be tested only on infinite lists.

Example:

```
*Main> switch naturals
[2,1,4,3,6,5,8,7,10,9,12,11,14,13,16,15,18,17, ..continues
```

2. This question deals with infinite trees.
   For the binary tree:

   data BinaryTree a = Nil | BNode a (BinaryTree a) (BinaryTree a)

   a) Define the function:
   `infTree` **:: a -> BinaryTree a**
   which produces a full, symmetric, infinite, binary tree of a's.

   b) Define the function:
   `treeMap` **:: (a -> b) -> BinaryTree a -> BinaryTree b**

   Which takes a function and a binary tree, and produces a binary tree in which all nodes are the result of applying the function on the given tree.

   c) Define:
   type Depth = **Int**

   Define the function:
   `treeTake` :: **Depth -> BinaryTree a -> BinaryTree a**

   Which prunes the given tree to produce a tree with at most "depth" levels.

   Example:
   `treeTake` 0 (BNode 6 Nil Nil) = **Nil**

   `treeTake` 3 (BNode 6 Nil Nil) = **BNode 6 Nil Nil**

   `treeTake` 1 (BNode 6 (BNode 3 Nil Nil) (BNode 9 Nil Nil)) =
   BNode 6 Nil Nil

   `treeTake` 2 (BNode 6 (BNode 3 Nil Nil) (BNode 9 Nil Nil)) =
   BNode 6 (BNode 3 Nil Nil) (BNode 9 Nil Nil)

   d) Define the function:
   `treeSort` **:: BinaryTree t -> [t]**

   Which takes a binary search tree and outputs a sorted list of the tree's values.

   Example:

`treeSort` Nil = **[]**
`treeSort` (BNode 6 Nil Nil) = **[6]**
`treeSort` (BNode 6 (BNode 3 Nil Nil) (BNode 9 Nil Nil)) = **[3,6,9]**