Functional Programming SS 2020 LVA 703025

Exercise Sheet 13, 10 points Deadline: **Tuesday, June 16, 2020, 10am**

- Please write all the Haskell code into a single .hs-file and upload it in OLAT.
- You can use the template .hs-files that are provided on the proseminar page[1].
- Your .hs-files should be compilable with ghci.
- Don't forget to mark your completed exercises in OLAT.
- Feel free to import functions from the Haskell standard library.

Exercise 13.1 *Simplifying Formulas* 5 p.

In this exercise we want to build functions for working with propositional logic formulas[2].

1. We want to be able to let the user simplify a formula, so given `Formula a` from the template file, write a function `simp :: Formula a -> Formula a` which simplifies a formula recursively according to (at least) the following rules:

$$\begin{array}{ccccc} \top \wedge x \longrightarrow x & x \wedge \top \longrightarrow x & \top \vee x \longrightarrow \top & x \vee \top \longrightarrow \top & \neg\top \longrightarrow \bot \\ \bot \wedge x \longrightarrow \bot & x \wedge \bot \longrightarrow \bot & \bot \vee x \longrightarrow x & x \vee \bot \longrightarrow x & \neg\bot \longrightarrow \top \\ \neg\neg x \longrightarrow x & & & & \end{array}$$

   Examples:

```
simp (Neg (Con Bot (Atom 'a'))) = Top
simp (Con (Atom 'a') (Neg (Neg Top))) = Atom 'a'
simp (Dis (Neg Bot) (Atom 'a')) = Top
```

   Note: First do a pattern match on the formula to see what rules apply. Then simplify the inner terms and check which rule exactly applies. You are going to need to use case expressions for that.

   (3 points)

2. The user should be able to fix the assignment[3] of a specific variable (that is, make it `True` or `False`). Write a function `substitute :: Formula a -> a -> Bool -> Formula a` which takes a formula, a variable identifier and a `Bool` and replaces all occurences of the variable in the formula with $\top$ or $\bot$ respectively. Examples:

```
substitute (Con (Atom 1) (Atom 2)) 2 False = Con (Atom 1) Bot
substitute (Neg (Con Bot (Atom 'a'))) 'a' True = Neg (Con Bot Top)
```

   (2 points)

[1] http://cl-informatik.uibk.ac.at/teaching/ss20/fp/index.php#exercises

[2] see *Aussagenlogik* as explained in the Introduction to Theoretical Computer Science course http://cl-informatik.uibk.ac.at/teaching/ws19/eti/ohp/2-beamer.pdf or for instance https://en.wikipedia.org/wiki/Propositional_calculus

[3] see *Wahrheitswertbelegung* in the previously linked lecture slides or for instance https://en.wikipedia.org/wiki/Valuation_(logic)

Exercise 13.2 *Folds* **2 p.**

1. Implement a function `prefixes` that computes all prefixes of a list using `foldr`.

   Example: `prefixes [1,2,3,4] = [[],[1],[1,2],[1,2,3],[1,2,3,4]]`

   (1 point)

2. Implement a function `suffixes` that computes all suffixes of a list using `foldl`.

   Example: `suffixes [1,2,3,4] = [[1,2,3,4],[2,3,4],[3,4],[4],[]]`

   (1 point)

Note: You have to use `foldl` and `foldr` in this exercise.

Exercise 13.3 *First missing positive* **2 p.**

Write a function `firstMissing` that finds the smallest missing positive number in a list of integers. In this exercise `0` is not a positive number.
Examples:

```
firstMissing [1,2,0] = 3
firstMissing [5,6,-8,1] = 2
firstMissing [-3,-2,7,8,9] = 1
```

Exercise 13.4 *Height of binary tree* **1 p.**

Write a function `treeHeight` that calculates the height of a binary tree. The height of a node is the number of edges on the longest path between the root and a leaf (so `Empty` has height `0`).
Examples:

```
treeHeight (Node Empty 4 Empty) = 1
treeHeight (Node (Node Empty 4 Empty) 1 (Node Empty 3 (Node Empty 2 Empty))) = 3
```