Fakultät für Informatik und Mathematik Summer semester 2023
Prof. Dr. Martin Griebl

Exercises for Lecture: Functional Programming
Exercise Sheet 6 (Induction Proofs)

Problem 1 (Induction Proofs on Lists)

In Stud.IP you can find the file `ListInduction.hs` with the data types and functions needed for the following exercises. Fill in the proofs for the following exercises and test your proofs in an automated fashion as shown in `ListInduction.hs`.

Consider the inductive definition of `map`, `length` and `(++)`:

```
map :: (a->b) -> [a] -> [b]
map f []     = []                  -- map.1
map f (x:xs) = f x : map f xs      -- map.2

length :: [a] -> Int
length []     = 0                  -- length.1
length (_:xs) = 1 + length xs      -- length.2

(++) :: [a] -> [a] -> [a]
[]     ++ ys = ys                  -- (++).1
(x:xs) ++ ys = x : (xs ++ ys)      -- (++).2
```

(a) Prove using structural induction that for all finite lists `xs :: [a]` and functions `f :: a->b` the following theorem holds:

`length (map f xs)` $=$ `length xs`

Use the equalities for `length` and `map` in your proof.

(b) Prove using structural induction that for all finite lists `xs,ys :: [a]` the following theorem holds:

`length (xs ++ ys)` $=$ `length xs + length ys`

Use the equalities for `length` and `(++)` in your proof.

(c) Let the function `powerlist` be given:

```
powerlist :: [a] -> [[a]]
powerlist []     = [[]]                                       -- powerlist.1
powerlist (x:xs) = map (x :) (powerlist xs) ++ powerlist xs    -- powerlist.2
```

Prove using structural induction that for all finite lists `xs :: [a]` the following holds:

`length (powerlist xs)` $= 2^{(\texttt{length xs})}$

You can reuse the results from the previous subtasks (a) and (b) in the proof for this subtask:

```
length (map f xs) = length xs                     -- lengthmap
length (xs ++ ys) = length xs + length ys         -- lengthconc
```

for arbitrary but finite lists `xs,ys :: [a]` and functions `f :: a->b`.

Problem 2 (Induction Proofs on Trees)

Let the following data type `Tree` be given, together with the functions `sumT` and `sumAcc` (c.f. the file `TreeInduction.hs` in Stud.IP):

```
data Tree  =  Leaf Int  |  Node Tree Int Tree

sumT :: Tree -> Int
sumT (Leaf x) = x                                               -- sumT.1
sumT (Node l x r) = sumT l + x + sumT r                         -- sumT.2

sumAcc :: Tree -> Int -> Int
sumAcc (Leaf x) y     = x+y                                     -- sumAcc.1
sumAcc (Node l x r) y = sumAcc l (sumAcc r (x+y))               -- sumAcc.2
```

Prove using structural induction that for all finite trees `t :: Tree` and arbitrary numbers `y :: Int` the following holds:

`sumT t + y = sumAcc t y`

Due date: Tuesday, May 30, 2023 at 16:00

Upload your Haskell source codes (`*.hs`) to the `Submissions` folder for this exercise in Stud.IP.