Exercises for Lecture: Functional Programming
Exercise Sheet 7 (Induction Proofs, Term Representations)

Problem 1 (Induction Proofs)

In Stud.IP you can find the file `InductionProof.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 `InductionProof.hs`.

(a) Prove the following theorem by structural induction.

For an arbitrary associative function `f::a->a->a` with neutral element `e::a` and arbitrary finite lists `xs,ys::[a]` the following holds:

`foldr f e (xs++ys)` = `f (foldr f e xs) (foldr f e ys)`

Use the following equalities in the proof.

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

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

(b) Let the following data type `Tree a` for trees be given together with the two functions `height` and `nrForks`:

```
data Tree a  =  Leaf a  |  Fork (Tree a) (Tree a)

height :: Tree a -> Integer
height (Leaf _)     = 0                                   -- height.1
height (Fork t1 t2) = 1 + max (height t1) (height t2)     -- height.2

nrForks :: Tree a -> Integer
nrForks (Leaf _)     = 0                                  -- nrForks.1
nrForks (Fork t1 t2) = 1 + nrForks t1 + nrForks t2        -- nrForks.2
```

Prove using structural induction that for all finite trees `t :: Tree a` the following holds:

$$\texttt{nrForks t} \leq 2^{(\texttt{height t})} - 1$$

Problem 2 (Operations on Binary Numbers)

Let the natural numbers be given as terms in reversed binary representation without leading zeros (cf. `InvBin.hs` in Stud.IP), e.g., the binary number 1011 is represented as `L (L (O (L Z)))`. The terms use `Z` to end a term, the character `O` represents a 0 und `L` represents a 1.

```
data InvBin = Z         -- 0
            | O InvBin  -- n -> 2*n
            | L InvBin  -- n -> 2*n+1
            deriving Show

fromInvBin :: InvBin -> Integer
fromInvBin Z     = 0                     -- fromInvBin.1
fromInvBin (O x) = 2 * fromInvBin x      -- fromInvBin.2
fromInvBin (L x) = 2 * fromInvBin x + 1  -- fromInvBin.3

inc :: InvBin -> InvBin
inc Z     = L Z         -- inc.1
inc (O x) = L x         -- inc.2
inc (L x) = O (inc x)   -- inc.3
```

(a) Define the inverse function of `fromInvBin`: `toInvBin :: Integer -> InvBin`. The function is defined on the natural numbers only. The returned terms shall be without leading zeros.

(b) Implement functions `add, mul :: InvBin -> InvBin -> InvBin` to add and multiply two numbers, respectively, *without* converting the numbers to `Integer`.

(c) Declare instances of the type classes `Eq`, `Num` und `Ord` for `InvBin`; return error messages for functions which are not defined (e.g., subtraction).

(d) Prove using structural induction that for all finite `x :: InvBin` the following holds:

`fromInvBin (inc x) == fromInvBin x + 1`

Due date: Tuesday, June 06, 2023 at 16:00

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