Universität Siegen
Lehrstuhl Theoretische Informatik
Carl Philipp Reh

Funktionales Programmieren
SS 2020

Übungsblatt 2

Aufgabe 1. Gegeben sei der Typ `Nat` aus der Vorlesung, definiert als

```
data Nat = Zero | Succ Nat
```

(a) Implementieren Sie eine geeignete `Show`-Instanz für `Nat`.

(b) Implementieren Sie die Funktion

```
fromInt :: Int -> Nat
```

welche einen `Int` in `Nat` umwandelt. Dabei soll gelten, dass fromInt für negative Zahlen Zero liefert und für positive Zahlen $n \geq 0$ soll $\text{Succ}^n(\text{Zero})$ geliefert werden.

(c) Implementieren Sie die Funktion

```
fromInt' :: Int -> Maybe Nat
```

welche im Gegensatz zu fromInt bei negativen Zahlen Nothing liefert.

(d) Implementieren Sie die Funktion

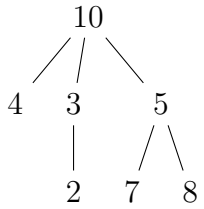
```
add :: Nat -> Nat -> Nat
```

welche zwei `Nat` addiert. Für $x, y \in \mathbb{N}$ soll gelten, dass

$$\text{add}(\text{Succ}^x(\text{Zero}), \text{Succ}^y(\text{Zero})) = \text{Succ}^{x+y}(\text{Zero}).$$

(e) Implementieren Sie eine Funktion, die es Ihnen erlaubt, explizite Rekursion bei `Nat` zu vermeiden. Implementieren Sie anschließen `add`, was wir nun `add'` nennen, darüber.

Aufgabe 2. Betrachten Sie den folgenden Baum:



Dieser besteht aus einem Wurzelknoten, der mit 10 beschriftet ist. Der Wurzelknoten hat drei Kindknoten, welche mit 4, 3 und 5 beschriftet sind, usw.

(a) Definieren Sie eine geeignete Data-Deklaration für Bäume. Es soll möglich sein, diese auch mit anderen Beschriftungen als Werte vom Typ `Int` zu versehen. Beschreiben Sie, welche Typkonstruktoren und Wertkonstruktoren Sie definieren.

(b) Implementieren Sie den obigen Baum.

(c) Implementieren Sie eine geeignete `Show`-Instanz für Ihre Bäume.

(d) Implementieren Sie die Funktion

```
preorder :: Tree a -> [a]
```

welche die *Preorder* eines Baums als Liste liefert. Die Preorder vom Beispielbaum wäre $10, 4, 3, 2, 5, 7, 8$.

(e) Implementieren Sie die Funktion

```
sumTree :: Tree Int -> Int
```

welche die Werte eines Baums von `Int` aufsummiert.