Programmierung

03. Übungsblatt

Zeitraum: 24. – 28. April 2017

Übung 1 (AGS 12.1.9)
Gegeben sei die folgende Typvereinbarung für binäre Bäume:

```
data Tree = Leaf Int | Branch Tree Tree
```

(a) Schreiben Sie einen Baum dieses Typs mit mindestens 5 Blättern auf.
(b) Schreiben Sie folgende Funktionen für o.g. Datentyp auf:
   1. Zum Ermitteln der Anzahl der Blätter.
   2. Zum Erstellen einer Liste aller Blattinformationen (von links nach rechts gelesen).

Übung 2 (AGS 12.1.41)
Gegeben sei der polymorphe algebraische Datentyp

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

(a) Geben Sie eine Funktion `depth :: Tree a -> Int` an, welche für einen Baum `t` die Länge des kürzesten Pfads von der Wurzel zu einem Blattknoten von `t` berechnet. Die Länge eines Pfads ist dabei die Anzahl der auf ihm vorkommenden Knoten, d.h. der Pfad von der Wurzel zur Wurzel selbst hat die Länge `1`.
(b) Geben Sie eine Funktion `paths :: Tree a -> Tree [a]` an, die in einem Baum die Beschriftung jedes Knotens u durch die Liste der Knotenbeschriftungen auf dem Pfad vom Wurzelknoten zu u ersetzt. Ist der Wurzelknoten z. B. in `t` mit `5` beschriftet, so ist er in `paths t` mit `[5]` beschriftet, und ist sein erster Kindknoten in `t` mit `3` beschriftet, dann ist dieser in `paths t` mit `[5,3]` beschriftet.

Übung 3
In der Vorlesung wurden die Higher-Order-Funktionen

- `map :: (a -> b) -> [a] -> [b]`,
- `filter :: (a -> Bool) -> [a] -> [a]`, und
- `foldr :: (a -> b -> b) -> b -> [a] -> b`

vorgestellt.

Implementieren Sie mithilfe von `map`, `filter` und `foldr` eine Funktion `f :: [Int] -> Int`, die das Produkt der Quadrate der geraden Zahlen in der Eingabeliste berechnet.

Übung 4
Gegeben sei der Datentyp

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

Schreiben Sie eine Funktion `tmap :: (a -> b) -> Tree a -> Tree b`, so dass für alle `f :: a -> b` und `t :: Tree a` gilt, dass `tmap f t` der Baum ist, der entsteht, indem jeder Knoten `v` von `t` durch `f v` ersetzt wird.

Zusatzaufgabe 1 (AGS 12.1.47)

(a) Schreiben sie eine Funktion `avg :: [Float] -> Float` welche den Durchschnitt einer Liste von `Floats` berechnet. Dabei soll die Liste *höchstens einmal durchlaufen* werden. Für die leere Liste soll die Funktion `0` zurückgeben.

(b) Geben Sie eine Funktion `partition :: (a -> Bool) -> [a] -> ([a], [a])` an, welche ein Prädikat `p :: a -> Bool` und eine Liste `xs :: [a]` als Argumente hat. Das Ergebnis ist ein Paar von Listen `(ys, zs)`, so dass `ys` jedes Element aus `xs` enthält, welches das Prädikat `p` erfüllt (d.h.`p y == True` für jedes `y` in `ys`), während `zs` jedes Element aus `xs` enthält, welches das Prädikat `p` nicht erfüllt. Die Reihenfolgen der Elemente in `ys` und `zs` gleichen der in `xs`. Beispielsweise soll gelten:

```
partition even [1,2,3,4,5,6,7,8,9,10] == ([2,4,6,8,10],[1,3,5,7,9])
```

(c) Sei `xs` eine Liste vom Typ `[Int]`. Wir sagen, dass `xs` eine *k-Wiederholung* enthält, falls sich `xs` in Listen `as`, `bs` und `cs` zerlegen lässt, so dass

- `xs == as ++ bs ++ cs` gilt,
- alle Elemente in `bs` gleich sind, und
- die Liste `bs` die Länge k hat.

Schreiben Sie eine Funktion `maxrep :: [Int] -> Int` die das größte k bestimmt, so dass die gegebene Liste eine k-Wiederholung enthält.

Zusatzaufgabe 2

Implementieren Sie eine Funktion `foldl :: (a -> b -> a) -> a -> [b] -> a`, so dass für jedes `f :: a -> b -> a` und $\texttt{a}_0$ `::` a, $\texttt{b}_1, \ldots, \texttt{b}_k$ `::` b, $\texttt{k} \in \mathbb{N}$ gilt, dass

$$\texttt{foldl f a}_0\,[\texttt{b}_1, \ldots, \texttt{b}_k] = \texttt{f}\,(\texttt{f} \cdots (\texttt{f a}_0\,\texttt{b}_1) \cdots \texttt{b}_{k-1})\,\texttt{b}_k\,,$$

also z.B.

$$\texttt{foldl}\,(+)\,5\,[1, 4, 3] = (+)\,((+)\,((+)\,5\,1)\,4)\,3 = ((5 + 1) + 4) + 3.$$

Insbesondere soll `foldl f a [] = a` gelten.

Für Fortgeschrittene: Implementieren Sie `foldl` unter Verwendung von `foldr`!