3. Übung (26.–30 April 2021)

Programmierung

Aufgabe 1 (AGS 12.1.41)
Gegeben ist der Datentyp `data RoseTree = Node Int [RoseTree]` eines Baumes, bei dem jeder Knoten eine beliebige Anzahl an Kindbäumen haben kann (gegeben in einer Liste vom Typ `[RoseTree]`).

(a) Geben Sie die Definition der Funktion `countLeaves :: RoseTree -> Int` an, die ermittelt, wie viele Blattknoten ein gegebener Baum vom Typ `RoseTree` enthält. Ein Blattknoten ist ein Knoten ohne Kindbäume.

(b) Geben Sie die Definition der Funktion `evenNodes :: RoseTree -> Bool` an, die zu einem gegebenen Baum vom Typ `RoseTree` ermittelt, ob jeder Knoten eine gerade Anzahl an Nachfolgern hat und in diesem Fall `True` zurückgibt (ansonsten `False`). Sie dürfen dabei auf die Funktion `length :: [RoseTree] -> Int` zurückgreifen, die die Länge einer Liste von Bäumen ermittelt.
Hinweis: Die Funktion `length` ist in der Haskell-Standardbibliothek `Prelude` bereits implementiert, Sie können sie ohne eigene Definition verwenden.

Aufgabe 2 (12.1.54 ⋆)
In der Vorlesung wurden die Higher-Order-Funktionen

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

vorgestellt. Implementieren Sie eine Funktion `f :: [Int] -> Int` mithilfe von `map`, `filter` und `foldr`, die das Produkt der Quadrate der geraden Zahlen in der Eingabeliste berechnet.
Hinweis: Die Funktionen `map`, `filter` und `foldr` sind in der Haskell-Standardbibliothek `Prelude` bereits implementiert, Sie können sie ohne eigene Definition verwenden.

Aufgabe 3 (AGS 12.1.57)
Geben Sie eine Funktion `foldleft :: (Int -> Int -> Int) -> Int -> [Int] -> Int` an, so dass für jedes `f :: Int -> Int -> Int` und $\texttt{a}_0$ `:: Int`, $\texttt{b}_1$, ..., $\texttt{b}_\texttt{k}$ `:: Int`, $\texttt{k} \in \mathbb{N}$ gilt, dass

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

also z.B.

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

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

Zusatzaufgabe 1 (AGS 12.1.53 ⋆)

(a) Schreiben Sie eine Funktion `join :: [String] -> String`, welche eine Liste von Wörtern aneinanderfügt. Die einzelnen Wörter sollen dabei durch ein Leerzeichen voneinander getrennt werden.

(b) Schreiben Sie eine Funktion `unjoin :: String -> [String]`, welche den Eingabestring in seine Teilwörter zerlegt. Nehmen Sie dabei zur Vereinfachung an, dass Wörter nur durch Leerzeichen voneinander getrennt sind.

Zusatzaufgabe 2 (AGS 12.1.19 ⋆)

Gegeben sei der folgende Datentyp `BinTree` zur Realisierung von Binärbäumen mit Knotenbeschriftungen vom Typ `Int` in Haskell: `data BinTree = Branch Int BinTree BinTree | Nil`.

(a) Definieren Sie unter Verwendung der Haskellfunktion `collapse`

```
collapse :: BinTree -> [Int]
collapse Nil = []
collapse (Branch x y z) = (collapse y) ++ [x] ++ (collapse z)
```

eine Haskellfunktion `check :: BinTree -> Bool`, die überprüft, ob es sich bei einem Baum vom Typ `BinTree` um einen binären Suchbaum handelt, das heißt jeder Knoten ist mit einer ganzen Zahl beschriftet und es muss für jeden Knoten `x` gelten, dass seine Beschriftung größer oder gleich (bzw. kleiner oder gleich) allen Beschriftungen im linken (bzw. rechten) Teilbaum von `x` ist.

(b) Definieren Sie eine Funktion `insert :: Int -> BinTree -> BinTree`, die einen Wert in einen binären Suchbaum einfügt, so dass der resultierende Baum ebenfalls ein binärer Suchbaum ist.

(c) Definieren Sie eine Funktion `merge :: BinTree -> BinTree -> BinTree`, die zwei binäre Suchbäume zu einem verschmilzt. Benutzen Sie dafür die Funktion `insert` aus dem vorherigem Aufgabenteil.