3. Übung (23.–26 April)

Programmierung

Aufgabe 1 (AGS 12.1.30)
Gegeben sei der Typ `data Tree = Node Int Tree Tree | Nil`.

(a) Geben Sie eine Funktion `insert :: Tree -> [Int] -> Tree` an, die alle Werte einer Liste von Integer-Zahlen in einen bereits bestehenden *Suchbaum* des Typs `Tree` so einfügt, dass die Suchbaumeigenschaft erhalten bleibt.

(b) Geben Sie eine Haskell-Funktion einschließlich der Typ-Definition an, die testet, ob zwei Binärbäume des Typs `Tree` identisch sind.

Aufgabe 2 (AGS 12.1.41)
Gegeben ist der Datentyp `data Tree = Node Int [Tree]` eines Baumes, bei dem jeder Knoten eine beliebige Anzahl an Kindbäumen haben kann (gegeben in einer Liste vom Typ `[Tree]`).

(a) Geben Sie die Definition der Funktion `noLeaves :: Tree -> Int` an, die ermittelt, wie viele Blattknoten ein gegebener Baum vom Typ `Tree` enthält. Ein Blattknoten ist ein Knoten mit einer leeren Liste an Kindbäumen.

(b) Geben Sie die Definition der Funktion `even :: Tree -> Bool` an, die zu einem gegebenen Baum vom Typ `Tree` 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 :: [Tree] -> Int` zurückgreifen, die die Länge einer Liste von Bäumen ermittelt.

Aufgabe 3 (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.

Zusatzaufgabe 1 (AGS 12.1.11)
Gegeben sei eine Liste der Bauart $[l_1, l_2, \ldots, l_n]$ mit $l_1, l_2, \ldots, l_n$ jeweils vom Typ `[Int]`. Es soll von jeder Liste l dieses Typs die Länge der längsten Liste, also $\max\{\text{length}(l_i) \mid 1 \leq i \leq n\}$, berechnet werden.

(a) Geben Sie ein Beispiel für eine Liste dieses Listentyps an und nennen Sie das zugehörige Ergebnis.

(b) Schreiben Sie in Haskell eine Funktion `maxLength :: [[Int]] -> Int`, die diese Aufgabe erfüllt und werten Sie abschließend einen Funktionsaufruf aus.

Zusatzaufgabe 2 (AGS 12.1.57)

Implementieren Sie eine Funktion `foldl :: (Int -> Int -> Int) -> Int -> [Int] -> Int`, 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

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

also z.B.

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

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