Programmierung

02. Übungsblatt

Zeitraum: 18. – 22. April 2016

Übung 1 (AGS 12.1.21)

(a) Geben Sie in Haskell eine boolesche Funktion `comp` einschließlich der Typdefinition an, die zwei Listen mit Elementen des Typs `Int` auf Gleichheit prüft.

(b) Schreiben Sie in Haskell eine Funktion `merge` einschließlich der Typdefinition, die aus zwei aufsteigend geordneten Listen mit Elementen des Typs `Int` durch Mischen eine aufsteigend geordnete Liste erzeugt.

Übung 2

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

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

Hinweise:

- `String = [Char]`.
- `words` und `unwords` sind in der Prelude-Library bereits definiert. Schließen Sie also diese Funktionen beim Einbinden der Prelude mittels der Direktive `hiding` aus (oder verwenden Sie andere Namen, wie z.B. `words'`).

Übung 3 (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 4

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

Zusatzaufgabe 1

(a) Schreiben Sie eine Funktion `pack :: [Char] -> [[Char]]`, welche in einer Liste aufeinander folgende Wiederholungen des gleichen Werts in einer Teilliste zusammenfasst.
Z.B.: `pack ['a','a','b','b','b','a'] = [['a','a'], ['b','b','b'], ['a']]`.

(b) Schreiben Sie eine Funktion `encode :: [Char] -> [(Int, Char)]`, welche eine Liste Lauflängenkodiert.
Z.B.: `encode ['a','a','b','b','b','a'] = [(2, 'a'), (3, 'b'), (1, 'a')]`.

(c) Schreiben Sie eine Funktion `decode :: [(Int, Char)] -> [Char]`, welche eine Lauflängenkodierte Liste wieder dekodiert.
Z.B.: `decode [(2, 'a'), (3, 'b'), (1, 'a')] = ['a','a','b','b','b','a']`.

(d) Schreiben Sie eine Funktion `rotate :: [Int] -> Int -> [Int]`, so dass `rotate xs n` die Liste `xs` um `n` nach links rotiert.
Z.B: `rotate [1,2,3,4] 1 = [2,3,4,1]` oder `rotate [1,2,3] (-1) = [3,1,2]`.

Zusatzaufgabe 2

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

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

also z.B.

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

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