2. Übung (13.–17. April 2020)

Programmierung

Aufgabe 1 (AGS 12.1.7, 12.1.8)
Schreiben Sie die folgenden Haskell-Funktionen:

(a) `prod :: [Int] -> Int`, welche die Zahlen in einer Liste aufmultipliziert.

(b) `rev :: [Int] -> [Int]`, welche eine Liste umkehrt.

(c) `excl :: Int -> [Int] -> [Int]`, so dass `excl x xs` die Liste ist, die aus `xs` hervorgeht, indem alle Vorkommen von `x` gelöscht werden.

(d) `isOrd :: [Int] -> Bool`, welche für eine Liste prüft, ob sie aufsteigend sortiert ist.

(e) `merge :: [Int] -> [Int] -> [Int]`, welche zwei aufsteigend sortierte Listen zu einer aufsteigend sortierten Liste vereinigt.

(f) Implementieren Sie die (unendliche) Liste `fibs :: [Int]` der Fibonacci-Zahlen f_0, f_1, ….

Aufgabe 2 (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.

Hinweis: `String = [Char]`

Aufgabe 3 (AGS 12.1.30)
Gegeben sei der Typ `data BinTree = Branch Int BinTree BinTree | Nil`.

(a) Geben Sie eine Funktion `insert :: BinTree -> [Int] -> BinTree` an, die alle Werte einer Liste von Integer-Zahlen in einen bereits bestehenden *Suchbaum* des Typs `BinTree` so einfügt, dass die Suchbaumeigenschaft erhalten bleibt. In einem Suchbaum 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) Geben Sie eine Haskell-Funktion einschließlich der Typ-Definition an, die testet, ob zwei Binärbäume des Typs `BinTree` identisch sind.

Hinweis: Der Datentyp `BinTree` definiert keine Ausgabefunktion und kann damit in `ghci` nicht angezeigt werden. Wenn Sie `deriving Show` hinter diese Typdefinition schreiben wird automatisch eine Ausgabefunktion generiert und Sie können Elemente dieses Datentyps wie gewohnt anzeigen lassen.

Zusatzaufgabe 1 (AGS 12.1.55)

(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']]`.

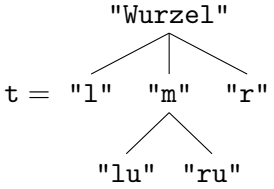
(b) Schreiben Sie eine Funktion `encode :: [Char] -> [(Int, Char)]`, welche eine Liste lauflängenkodiert.
Z.B.: `encode ['a','a','b','b','b','a'] = [(2, 'a'), (3, 'b'), (1, 'a')]`.

(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 (AGS 12.1.49 b–c)

(a) Gegeben sei der algebraische Datentyp `RoseTree` zur Darstellung von Bäumen, deren Knoten jeweils beliebig viele Kindbäume haben dürfen: `data RoseTree = Node String [RoseTree]`. Geben Sie für den rechtsstehenden Baum das entsprechende Element des Typs `RoseTree` an.



(b) Implementieren Sie die Funktion `level`, die für ein `n :: Int` und ein `t :: RoseTree` von links nach rechts alle Knoten aus `t` liefert, die `n` Kanten von der Wurzel entfernt sind, z.B.: `level 1 t = ["l", "m", "r"]`.