2. Übung (19.–23. April 2021)

Programmierung

Aufgabe 1 (AGS 12.1.7 b,d,e; 12.1.8)

Schreiben Sie die folgenden Haskell-Funktionen:

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

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

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

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

Aufgabe 2 (AGS 12.1.64 b,c ⋆)

(a) Schreiben Sie eine Funktion `isPrefix :: String -> String -> Bool` welche überprüft, ob die erste übergebene Zeichenkette ein Präfix der zweiten übergebenen Zeichenkette ist. Die Zeichenkette `xs` ist ein *Präfix* von `ys`, wenn es eine Zeichenkette `zs` gibt, sodass `ys == xs ++ zs` gilt. Zum Beispiel ist `"abc"` ein Präfix von `"abcde"` aber nicht von `"abbcd"`.

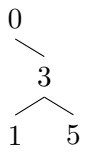
Hinweise: `String = [Char]`, Zeichenketten sind also Listen von Zeichen. Die leere Liste ist ein Präfix von jeder Liste.

(b) Schreiben Sie eine Funktion `countPattern :: String -> String -> Int`, welche zwei Zeichenketten `xs` und `ys` als Argumente nimmt und die Anzahl der Paare (`ys1`,`ys2`), für die `ys = ys1 ++ xs ++ ys2` gilt, zurückgibt. Intuitiv wird die Anzahl der Vorkommen von `xs` in `ys` ermittelt. Zum Beispiel gilt `countPattern "aba" "ababa" == 2` und `countPattern "" "abc" == 4`.

Aufgabe 3 (AGS 12.1.30)

Gegeben sei der Typ `data BinTree = Branch Int BinTree BinTree | Nil`.

(a) Gegeben ist die rechtsstehende graphische Darstellung eines Binärbaums t. Geben Sie einen Haskell-Ausdruck `mytree` des Typs `BinTree` für t an.



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

(c) 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.

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 12.1.66 c ⋆)

Geben Sie eine Funktion `unwind :: BinTree -> [Int]` an, welche einen gegebenen Baum ebenenweise traversiert und dabei die Liste der Knotenbeschriftungen in der Reihenfolge der Traversierung berechnet. Dabei soll zunächst die Wurzel besucht werden, dann die direkten Nachfolger der Wurzel von links nach rechts, dann deren direkte Nachfolger von links nach rechts, usw. Zum Beispiel soll für den in Teilaufgabe (b) betrachteten Baum gelten: `unwind mytree == [1,2,3,7,6,5,4]`.