5. Aufgabenblatt zu Funktionale Programmierung vom 19.11.2014.
Fällig: 26.11.2014 / 03.12.2014 (jeweils 15:00 Uhr)

Themen: *Funktionen und Funktionen höherer Ordnung auf algebraischen Datentypen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe5.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also wieder ein "gewöhnliches" Haskell-Skript schreiben. Versehen Sie wieder wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung brauchen, mit ihren Typdeklarationen und kommentieren Sie Ihre Programme aussagekräftig. Benutzen Sie, wo sinnvoll, Hilfsfunktionen und Konstanten.

Im einzelnen sollen Sie die im folgenden beschriebenen Problemstellungen bearbeiten.

1. Wir betrachten folgenden algebraischen Datentyp für Bäume:

   ```
   data Tree  = Node (Label -> Label) Label [Tree]
   type Label = Int
   ```

   Schreiben Sie in Anlehnung an die Funktionale `map` und `zip` auf Listen Haskell-Rechenvorschriften

   (a) `tcm :: (Label -> Bool) -> (Label -> Label) -> Tree -> Tree`

   (b) `tzp :: (Label -> Label -> Label) -> Tree -> Tree -> Tree`

   auf Bäumen, die folgendes leisten:

   - Erfüllt die Markierung l eines Knotens das mit dem Funktionsaufruf von `tcm` ("tree conditional map") übergebene Prädikat, so wird l vermöge der mit dem Aufruf von `tcm` übergebenen Transformationsfunktion transformiert, ansonsten mit der Transformationsfunktion des Knotens.
   - Sind die Markierungen l_1 und l_2 sich entsprechender Knoten der Argumentbäume t_1 und t_2 gleich, so ergibt sich die Markierung l des Knotens im Resultatbaum als Resultat der sequentiellen Komposition $f_1 \,.\, f_2$ angewendet auf l_1, wobei f_1 und f_2 die Transformationsfunktionen der jeweiligen Knoten in t_1 und t_2 sind. Sind l_1 und l_2 verschieden, so ergibt sich l aus der Anwendung der mit dem Aufruf von `tzp` übergebenen Transformationsfunktion f auf l_1 und l_2. Die Transformationsfunktion des Resultatknotens ist die Funktion f angewendet auf l_1, falls l_1 und l_2 gleich sind, ansonsten die sequentielle Komposition $f_1 \,.\, f_2$.

2. Schreiben Sie eine Haskell-Rechenvorschrift

   ```
   tmax :: Label -> Tree -> (Label -> Label)
   ```

   Angewendet auf einen Wert l und einen Baum t liefert `tmax` die Markierungsfunktion f eines Knotens aus t als Resultat, so dass der Wert $f\ l$ maximal ist, d.h. keine andere Markierungsfunktion eines Knotens aus t liefert einen echt größeren Wert angewendet auf l als f (f ist i.a. nicht eindeutig bestimmt; welche Markierungsfunktion mit der gewünschten Maximalitätseigenschaft ihre Funktion `tmax` liefert, ist egal.)

3. Wir betrachten abschließend folgenden algebraischen Datentyp für Bäume ("Simple Trees"):

   ```
   data STree = SNode Label [STree] deriving (Eq,Show)
   ```

   - Schreiben Sie eine Haskell-Rechenvorschrift

     ```
     t2st :: Tree -> STree
     ```

     Angewendet auf einen Baum t vom Typ `Tree` liefert `t2st` als Resultat einen Baum st vom Typ `STree`, in dem die Knotenmarkierungen funktionalen Typs aus t fehlen und der ansonsten in Struktur und Markierung mit t übereinstimmt.

- Schreiben Sie in Anlehnung an die Faltungsfunktionale auf Listen eine Haskell-Rechenvorschrift

  (a) `tsum :: STree -> Label`

  die angewendet auf einen Baum die Summe aller Knotenmarkierungen berechnet, sowie eine Haskellrechenvorschrift

  b) `tdepth :: STree -> Label`

  die angewendet auf einen Baum, dessen Tiefe bestimmt. Dem Baumwert `SNode _ []` ist die Tiefe 1 zugeordnet.

Hinweis: Keine früheren Lösungen als Module importieren!

Wenn Sie zur Lösung einzelne Funktionen früherer Lösungen wiederverwenden möchten, so kopieren Sie diese unbedingt explizit in Ihre neue Programmdatei ein. Importieren schlägt im Rahmen der automatischen Programmauswertung fehl. Es wird nicht nachgebildet. Deshalb: Wiederverwendung ja, aber durch kopieren, nicht durch importieren!

Haskell Live

An einem der kommenden *Haskell Live*-Termine, der nächste ist am Freitag, den 21.11.2014, werden wir uns u.a. mit der Aufgabe *World of Perfect Towers* beschäftigen.

World of Perfect Towers

In diesem Spiel konstruieren wir Welten perfekter Türme. Dazu haben wir n Stäbe, die senkrecht auf einer Bodenplatte befestigt sind und auf die mit einer entsprechenden Bohrung versehene Kugeln gesteckt werden können. Diese Kugeln sind ebenso wie die Stäbe beginnend mit 1 fortlaufend nummeriert.

Die auf einen Stab gesteckten Kugeln bilden einen Turm. Dabei liegt die zuerst aufgesteckte Kugel ganz unten im Turm, die zu zweit aufgesteckte Kugel auf der zuerst aufgesteckten, usw., und die zuletzt aufgesteckte Kugel ganz oben im Turm. Ein solcher Turm heißt *perfekt*, wenn die Summe der Nummern zweier unmittelbar übereinanderliegender Kugeln eine Zweierpotenz ist. Eine Menge von n perfekten Türmen heißt *n-perfekte Welt*.

In diesem Spiel geht es nun darum, n-perfekte Welten mit maximaler Kugelzahl zu konstruieren. Dazu werden die Kugeln in aufsteigender Nummerierung, wobei mit der mit 1 nummerierten Kugel begonnen wird, so auf die n Stäbe gesteckt, dass die Kugeln auf jedem Stab einen perfekten Turm bilden und die Summe der Kugeln aller Türme maximal ist.

Schreiben Sie in Haskell oder einer anderen Programmiersprache ihrer Wahl eine Funktion, die zu einer vorgegebenen Zahl n von Stäben die Maximalzahl von Kugeln einer n-perfekten Welt bestimmt und die Türme dieser n-perfekten Welt in Form einer Liste von Listen ausgibt, wobei jede Liste von links nach rechts die Kugeln des zugehörigen Turms in aufsteigender Reihenfolge angibt.