4. Aufgabenblatt zu Funktionale Programmierung vom Mi, 09.11.2016. Fällig: Mi, 16.11.2016 (15:00 Uhr)

Themen: *Funktionen über algebraischen Datentypen, insbesondere Bäumen*

Für dieses Aufgabenblatt sollen Sie die zur Lösung der unten angegebenen Aufgabenstellungen zu entwickelnden Haskell-Rechenvorschriften in einer Datei namens `Aufgabe4.lhs` im home-Verzeichnis Ihres Accounts auf der Maschine g0 ablegen. Wie bei der Lösung zum dritten Aufgabenblatt sollen Sie auch dieses Mal ein "**literate Script**" schreiben. Versehen Sie wie auf den bisherigen Aufgabenblättern alle Funktionen, die Sie zur Lösung benötigen, 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.

Zur Frist der Zweitabgabe: Siehe allgemeine Hinweise auf Aufgabenblatt 1.

1. Wir betrachten folgenden Typ zur Darstellung von Binärbäumen:

   ```
   data Tree a = Nil | Node a (Tree a) (Tree a) deriving (Eq,Ord,Show)
   ```

   Der leere Baum mit Wert `Nil` vom Typ `Tree a` heißt *geordnet*. Ein nichtleerer Baum mit Wert `(Node value t1 t2)` vom Typ `Tree a` heißt *geordnet*, wenn gilt:

   - alle Werte in `t1` sind echt kleiner als `value`
   - alle Werte in `t2` sind echt größer als `value`
   - die Bäume `t1` und `t2` sind gleichfalls geordnet.

   Mit diesen Festlegungen gilt, dass die Markierungen geordneter Bäume paarweise verschieden sind; es gibt keine Duplikate. Geordnete Bäume werden auch als *Suchbäume* bezeichnet.

   Sei weiters der Typ

   ```
   data Order = Up | Down deriving (Eq,Show)
   ```

   gegeben.

   Schreiben Sie nun Haskell-Rechenvorschriften

   ```
   nil           :: Tree a
   isNilTree     :: Tree a -> Bool
   isNodeTree    :: Tree a -> Bool
   leftSubTree   :: Tree a -> Tree a
   rightSubTree  :: Tree a -> Tree a
   treeValue     :: Tree a -> a
   isValueOf     :: Eq a => a -> Tree a -> Bool
   isOrderedTree :: Ord a => Tree a -> Bool
   insert        :: Ord a => a -> Tree a -> Tree a
   delete        :: Ord a => a -> Tree a -> Tree a
   flatten       :: Ord a => Order -> Tree a -> [a]
   ```

   mit folgenden Eigenschaften:

   - Die 0-stellige Funktion `nil` erzeugt den leeren Baum mit Wert `Nil`.
   - Die Wahrheitswertfunktion `isNilTree` liefert den Wert `True`, wenn sie auf den leeren Baum angewendet wird, sonst `False`.
   - Die Wahrheitswertfunktion `isNodeTree` liefert den Wert `False`, wenn sie auf den leeren Baum angewendet wird, sonst `True`.
   - Die Selektorfunktionen `leftSubTree`, `rightSubTree` und `treeValue` brechen angewendet auf den leeren Baum mit dem Aufruf von `error "Empty Tree as Argument"` ab, ansonsten liefern sie den linken bzw. rechten Teilbaum des Arguments bzw. seinen Markierungswert, d.h., die Markierung des Wurzelknotens.

- Die Wahrheitswertfunktion `isValueOf` angewendet auf einen Wert v und einen (geordneten oder nicht geordneten) Baum t liefert den Wert `True`, wenn t einen (oder möglicherweise mehr im Fall nicht geordneter Argumentbäume) mit v markierten Knoten enthält, ansonsten *False*.
- Die Wahrheitswertfunktion `isOrderedTree` liefert den Wert `True`, wenn sie auf einen geordneten Baum angewendet wird, ansonsten `False`.
- Die Funktionen `insert` und `delete` brechen angewendet auf einen nichtgeordneten Baum mit dem Aufruf von `error "Argument Tree not Ordered"` ab.

  Angewendet auf einen Wert v und einen geordneten Argumentbaum t fügt `insert` den Wert v so in Form eines neuen Knotens in t ein, falls v nicht schon als Markierung eines Knotens in t enthalten ist, dass auch der Resultatbaum geordnet ist. Ist v bereits in t enthalten, so wird t selbst als Resultat zurückgegeben.

  Angewendet auf einen Wert v und einen geordneten Argumentbaum t, der einen Knoten mit Markierung v enthält, löscht `delete` den mit v beschrifteten Knoten in t so, dass auch der Resultatbaum geordnet ist. Ist v keine Markierung eines Knotens in t, so gibt `delele` t selbst als Resultat zurück.
- Angewendet auf einen nichtgeordneten Baum bricht die Auswertung von `flatten` mit dem Aufruf von `error "Argument Tree not Ordered"` ab. Angewendet auf einen geordneten Baum t ist der Wert der Funktion `flatten` eine aufsteigend geordnete Liste der in t enthaltenen Werte, falls `flatten` mit `Up` als erstem Argument aufgerufen wird bzw. eine absteigend geordnete Liste der in t enthaltenen Werte, falls `flatten` mit `Down` als erstem Argument aufgerufen wird.

2. Ein Suchbaum sollte in der Praxis möglichst gut balanciert sein, d.h. die Wege von der Wurzel zu den Blättern sollten alle von möglichst gleicher Länge sein. Ein grobes Maß für die Balanciertheit eines Baums ist die Differenz zwischen dem längsten und dem kürzesten Weg von der Wurzel zu einem Blatt.

   Schreiben Sie drei Haskell-Rechenvorschriften

   ```
   maxLength      :: Tree a -> Int
   minLength      :: Tree a -> Int
   balancedDegree :: Tree a -> Int
   ```

   Angewendet auf einen Baum t (gleich, ob geordnet oder nicht) berechnen `maxLength` und `minLength` die Länge des längsten bzw. kürzesten Wegs von der Wurzel zu einem Blatt in t sowie `balancedDegree` den sich daraus ergebenden Hinweis auf die Balanciertheit von t in Form des Absolutbetrags der Differenz dieser beiden Werte.

   Dabei gilt: Im leeren Baum mit Darstellung `Nil` hat die Länge jedes Wegs den Wert 0. In nichtleeren Bäumen mit Darstellung `Node m t1 t2` ist die Länge jedes Wegs von der Wurzel zu einem Blatt durch die Zahl der inneren Knoten auf diesem Weg gegeben, also durch die Zahl der Vorkommen von `Node` auf dem Weg.

Hinweis: Wenn Sie einzelne Rechenvorschriften aus früheren Lösungen wieder verwenden möchten, so kopieren Sie diese in die neue Abgabedatei ein. Ein `import` schlägt für die Auswertung durch das Abgabeskript fehl, weil Ihre alte Lösung, aus der importiert wird, nicht mit abgesammelt wird. Deshalb: Kopieren statt importieren zur Wiederwendung!

Denken Sie bitte daran, dass Sie für die Lösung dieses Aufgabenblatts ein “literate” Haskell-Skript schreiben sollen!

Haskell Live

Am Freitag, den 11.11.2016, werden wir uns in *Haskell Live* mit Lösungsvorschlägen u.a. bereits abgeschlossener Aufgabenblätter beschäftigen, die (gerne auch) von Ihnen eingebracht werden können, sowie mit einigen der schon speziell für *Haskell Live* gestellten Aufgaben.