7. Aufgabenblatt zu Funktionale Programmierung vom 20.11.2007. Fällig: 27.11.2007 / 04.12.2007 (jeweils 15:00 Uhr)

Themen: *Typklassen und überladene Funktionen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.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. In dieser Aufgabe knüpfen wir noch einmal an die Situation von Aufgabenblatt 6 an, wollen dieses Mal durch die Einführung einer Typklasse eine gefälligere Lösung erhalten.

   Definieren Sie dazu eine Typklasse `Sortable` wie folgt:

   ```
   class Sortable a where
     sortIncr :: a -> a
     sortDecr :: a -> a
   ```

   Für Instanzen der Typklasse `Sortable` soll stets gelten, dass die Funktion `sortIncr` ihr Argument aufsteigend sortiert, die Funktion `sortDecr` absteigend.

   Wir betrachten neben polymorphen Listen vom Typ `[a]` folgenden Baumtyp:

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

   Machen Sie die Typen `Tree a` und `[a]` zu Instanzen der Typklasse `Sortable`, wobei `a` bei der Instanzbildung jeweils aus der Typklasse `Ord` vorausgesetzt werden soll (entsprechend dem Vorbild bei der Instanzbildung für Baumtypen für die Klasse `Eq` im Vorlesungsteil vom 20.11.2007)

   Auf Bäumen soll gelten, dass die Funktion `sortIncr` (`sortDecr`) die Knoten im Baum so umordnet (d.h. einen neuen Baum mit umgeordneten Knoten aufbaut und zurückgibt), dass die Benennungen in aufsteigender (absteigender) Reihenfolge ausgegeben werden, wenn der Resultatbaum in Präfixordnung durchlaufen wird. Die Resultatbäume brauchen anders als auf Aufgabenblatt 6 nicht ausbalanciert zu sein.

   Auf Listen soll gelten, dass die Funktion `sortIncr` (`sortDecr`) als Resultat eine Liste zurückliefert, die die Elemente der Argumentliste in aufsteigender (absteigender) Folge enthält.

2. Definieren Sie eine Typklasse `UniqueSorted` wie folgt:

   ```
   class Sortable a => UniqueSorted a where
     usortIncr :: a -> a
     usortDecr :: a -> a
   ```

   Machen Sie analog zur vorigen Teilaufgabe die Typen `Tree a` und `[a]` auch zu Instanzen der Typklasse `UniqueSorted`.

   Für die Funktionen `usortIncr` (`usortDecr`) gelten die gleichen Anforderungen wie für ihre Gegenstücke aus der vorigen Teilaufgabe, jedoch sollen die Resultate frei von Duplikaten sein. Für Bäume heißt das, dass keine Benennung doppelt im Resultatbaum enthalten sein darf.

3. Betrachten Sie jetzt folgenden heterogenen Baumtyp:

   ```
   data HTree a b = HNil |
                    Node1 a (HTree a b) (HTree a b) |
                    Node2 a b (HTree a b) (HTree a b) deriving Show
   ```

   Machen Sie auch Baumtyp `HTree a b` zu einer Instanz der Typklassen `Sortable` und `UniqueSorted`.

(a) Auf `HTree`-Bäumen soll gelten, dass die Funktion `sortIncr` (`sortDecr`) die Knoten im Baum so umordnet (d.h. einen neuen Baum mit umgeordneten Knoten aufbaut und zurückgibt), dass die `a`-Benennungen in aufsteigender (absteigender) Reihenfolge ausgegeben werden, wenn der Resultatbaum in Präfixordnung durchlaufen wird. Wie im vorigen Aufgabenteil brauchen die Resultatbäume nicht ausbalanciert zu sein.

(b) Für die Funktionen `usortIncr` (`usortDecr`) gelten die gleichen Anforderungen wie für ihre Gegenstücke aus dieser Teilaufgabe, jedoch sollen die Resultate frei von Duplikaten sein. Für `HTree`-Bäume heißt das, dass keine inneren Knoten von der Art `Node1` mit gleicher Benennung vorkommen und keine inneren Knoten von der Art `Node2`, die gleichzeitig in `a`- und `b`-Komponente übereinstimmen.