8. Aufgabenblatt zu Funktionale Programmierung vom 14.12.2005, fällig: 11.01.2006 / 18.01.2006 (jeweils 12:00 Uhr)

Themen: *Polymorphe Funktionen, Funktionale und Automaten*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe8.hs` ablegen. Sie sollen also ein gewöhnliches Haskellskript schreiben. Versehen Sie 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 folgende Problemstellungen bearbeiten:

1. In dieser Aufgabe betrachten wir folgenden algebraischen Typ von Binärbäumen:

   ```
   data Tree a = Leaf a |
                 Branch (Tree a) (Tree a) deriving (Eq,Ord,Show)
   ```

   Schreiben Sie eine Haskell-Rechenvorschrift `sortTree` mit der Signatur `sortTree :: Ord a => (a -> a -> Bool) -> Tree a -> Tree a`, die angewendet auf einen Relator auf Werten vom Typ `a` und einen Baum vom Typ `Tree a` einen neuen Baum vom Typ `Tree a` als Resultat liefert, der die gleiche Struktur wie der Argumentbaum besitzt, in dem aber die Werte an den Blättern entsprechend der durch den Relator vorgegebenen Ordnung von links nach rechts angeordnet sind. Zwei Beispiele: `sortTree (>=) (Branch (Branch (Leaf 4) (Leaf 3)) (Leaf 5)) == (Branch (Branch (Leaf 5) (Leaf 4)) (Leaf 3))`, `sortTree (<=) (Branch (Branch (Leaf 4) (Leaf 3)) (Leaf 5)) == (Branch (Branch (Leaf 3) (Leaf 4)) (Leaf 5))`.

   Ist der Relator nicht reflexiv, z.B. `<`, aber sind die Blattmarkierungen paarweise verschieden, so soll der Baum wie oben beschrieben umsortiert werden, anderenfalls soll der Argumentbaum unverändert ausgegeben werden. Zwei Beispiele: `sortTree (>) (Branch (Branch (Leaf 4) (Leaf 3)) (Leaf 5)) == (Branch (Branch (Leaf 5) (Leaf 4)) (Leaf 3))`, `sortTree (>) (Branch (Branch (Leaf 4) (Leaf 3)) (Leaf 4)) == (Branch (Branch (Leaf 4) (Leaf 3)) (Leaf 4))`.

2. In dieser Aufgabe betrachten wir folgenden algebraischen Typ von Binärbäumen:

   ```
   data STree a b = Nil |
                    SLeaf a b |
                    Node a b (STree a b) (STree a b) deriving (Eq,Ord,Show)
   ```

   In dieser Aufgabe wollen wir eine Funktion zum Sortieren der Elemente von Listen schreiben. Diese Funktion heiße `listSort` und habe die Signatur `listSort :: Ord a => (a -> a -> Bool) -> [a] -> [a]`. Über den Relator ist dabei das Sortierkriterium vorgegeben. Die Sortierfunktion soll dabei so vorgehen, dass zunächst die Liste von links nach rechts gelesen wird und ihre Elemente in einem Suchbaum vom Typ `data STree a b` abgelegt werden. Dabei kann die zweite Benennungskomponente an jedem Blatt oder inneren Knoten eines Baums dazu genutzt werden, sich die Anzahl unter Umständen in der Argumentliste mehrfach vorkommender Werte zu merken. Für den Suchbaum gilt, dass alle Werte im linken Teilbaum mit dem Wert in

der Wurzel gemäss dem Relator vergleichbar sein müssen, und umgekehrt der Wert in der Wurzel mit allen Werten im rechten Teilbaum. Ist der Baum als Zwischenergebnis konstruiert, so erhält man die Resultatliste eines Baumdurchlaufs in Infixordnung. Ein Beispiel: `listSort (>=) [3,2,6,2,1] == [6,3,2,2,1]`, `listSort (<=) [3,2,6,2,1] == [1,2,2,3,6]`. Ist der Relator nicht reflexiv und sind gleichzeitig die Elemente der Argumentliste nicht paarweise verschieden, so soll die Argumentliste unverändert zurückgegeben werden, ansonsten wie oben beschrieben sortiert.

3. In dieser Aufgabe betrachten wir endliche Automaten, die wir in Haskell durch folgenden algebraischen Datentyp repräsentieren:

```
newtype Aut a = Aut ([(a,[(a,Char)])],a,[a]) deriving (Eq,Show)
```

Die erste Komponente im Dreitupel eines Automaten (`[(a,[(a,Char)])]`) codiert die Automatenstruktur als gerichteten Graphen, die zweite Komponente bezeichnet den ausgezeichneten Startzustand des Automaten (`a`), die dritte Komponente die Menge der ausgezeichneten Endzustände des Automaten (`[a]`). Die Kanten des Automaten sind gerichtet und mit einem Buchstaben bezeichnet. Ein Automat heißt *wohlgeformt* genau dann, wenn der Anfangszustand und die Endzustände in der Zustandsliste des Automaten vorkommen, die Nachfolgerliste eines jeden Knoten nur im Automaten vorkommende Zustände enthält und alle von einem Zustand ausgehenden Kanten paarweise verschieden beschriftet sind. Die *Zustandsliste* eines Automaten ist dabei implizit gegeben durch die in der ersten Automatenkomponente genannten Zustände, d.h. genannt als Zustand mit einer Liste von Nachfolgern oder genannt in der Nachfolgerliste eines Zustands. Ein Zustand, der in der Nachfolgerliste eines Zustands genannt ist, aber selbst keine Nachfolger hat, muss keinen eigenen Eintrag in der ersten Komponente eines Automaten haben, also keinen Eintrag, in der er als erste Komponente dieses Eintrags auftaucht. Die Zustandsliste des Automaten `(Aut ([(1,[(2,'a')]),(2,[(3,'b')])],1,[2,3]))` enthält also die Zustände `1`, `2` und `3`.

- Schreiben Sie eine Wahrheitswertfunktion `isWellformed :: Eq a => Aut a -> Bool`, die genau dann den Wert `True` liefert, wenn der Argumentautomat wohlgeformt ist und sonst `False`. Zwei Beispiele:
  `isWellformed (Aut ([(1,[(2,'a')]),(2,[(3,'b')])],1,[2,3])) == True`,
  `isWellformed (Aut (([(1,[(2,'a')]),(2,[(3,'b')])],5,[2,3])) == False`
- In dieser Aufgabe wollen wir uns mit dem Wortproblem für unsere Automaten beschäftigen. Ein endlicher Automat *akzeptiert* ein Wort, wenn ausgehend vom Anfangszustand beim zeichenweisen Lesen des Worts der Automat in einem Endzustand endet, ansonsten akzeptiert er das Wort nicht. Schreiben Sie eine Haskell-Rechenvorschrift `accept :: Eq a => Aut a -> String -> Bool`, die angesetzt auf einen wohlgeformten Automaten und eine Zeichenreihe den Wert `True` liefert, wenn der Automat das Wort akzeptiert, und `False` sonst. Insbesondere akzeptiert der Automat ein Wort nicht, wenn er für ein zu lesendes Zeichen keinen passenden Zustandsübergang machen kann, d.h. wenn keine vom aktuell erreichten Zustand ausgehende Kante mit dem passenden Zeichen beschriftet ist. Ist der Automat nicht wohlgeformt, soll ebenfalls der Wert `False` zurückgegeben werden. Zwei Beispiele: `accept (Aut ([(1,[(2,'a')]),(2,[(3,'b')])],1,[2,3]))`

`''ab'' == True`, `accept (Aut ([(1,[(2,'a')]),(2,[(3,'b')])],1,[2,3]))`
`''acb'' == False`

Hinweis:

- Verwenden Sie *keine* Module. Wenn Sie Funktionen wiederverwenden möchten, kopieren Sie diese in die Abgabedatei `Aufgabe8.hs`. Andere als diese Datei werden vom Abgabeskript ignoriert.