6. Aufgabenblatt zu Funktionale Programmierung vom 20.11.2013.
Fällig: 27.11.2013 / 04.12.2013 (jeweils 15:00 Uhr)

Themen: *Typklassen, Polymorphie und Überladung*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften für die Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe6.hs` ablegen. Sie sollen für die Lösung dieses Aufgabenblatts also 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. Haskell stellt mit `Int` und `Integer` zwei Datentypen zur Repräsentation ganzer Zahlen da. Mit dem algebraischen Datentyp `Nat` wollen wir eine Repräsentation natürlicher Zahlen in Haskell einführen:

   ```
   data Nat = Z | S Nat deriving Show
   ```

   Der Konstruktorname `Z` erinnert an "Zero", der Konstruktorname `S` an "Successor". Entsprechend steht der `Nat`-Wert `Z` für die natürliche Zahl 0, der `Nat`-Wert `S Z` für die 1, der `Nat`-Wert `S (S Z)` für die 2 usw. Auf diese Weise besitzt jede natürliche Zahl eine eindeutige Darstellung als Wert des Datentyps `Nat`.

   Machen Sie den Typ `Nat` mithilfe expliziter Instanzdeklarationen zu einer Instanz der Haskell-Typklassen

   - `Eq`
   - `Ord`
   - `Num`

   so dass die in diesen Klassen vorgesehenen Operatoren und Relatoren auf Werten vom Typ `Nat` in derselben Weise operieren wie auf den entsprechenden Werten vom Typ `Int` bzw. `Integer`. Nutzen Sie dabei, wo dies möglich ist, in den Klassen für einzelne Operatoren und Relatoren vorgegebene Protoimplementierungen aus. Resultate von in der Klasse `Num` vorgesehenen Funktionen ($(-)$ ('minus') und `negate` ('Vorzeichenwechsel')), die bei Rechnung auf ganzen Zahlen negativ wären (d.h. kleiner als 0), werden durch den `Nat`-Wert für 0 ersetzt. Argumente von in der Klasse `Num` vorgesehenen Funktionen (`fromInteger` (Konversionsfunktion)), die bei Rechnung auf ganzen Zahlen negativ sein dürften (d.h. kleiner als 0), werden wie das Argument 0 behandelt. Beides dient dazu den Zahlbereich von `Nat` nicht zu verlassen. Die Funktion `signum` in `Num` liefert das Vorzeichen ihres Arguments: Für den `Nat`-Wert `Z` also das Resultat `Z`, für alle anderen `Nat`-Werte das Resultat `S Z`.

2. Wir führen eine Typklasse `SimNf` ein, die Typen umfassen soll, deren Werte auf Ähnlichkeit verglichen werden können und eine Normalform besitzen:

   ```
   class Ord a => SimNf a where
     similar    :: a -> a -> Bool
     normalform :: a -> a
   ```

   Machen Sie die Typen `Integer`, `String` und `Tree b` mit

   ```
   data Tree b = Null | Node b (Tree b) (Tree b) deriving (Eq,Ord,Show)
   ```

   zu Instanzen der Typklasse `SimNf`. Dabei seien Ähnlichkeit und Normalform wie folgt festgelegt:

   - Für `Integer`: Zwei Zahlen heißen ähnlich, wenn ihre Absolutbeträge dieselbe Quersumme besitzen. Die Normalform einer Zahl ist diejenige ähnliche Zahl gleichen Vorzeichens, bei der die Ziffern von links nach rechts aufsteigend angeordnet sind. Die Normalform der Zahl 343254131 ist also die Zahl 112333445, die der Zahl 0 die Zahl 0, die der Zahl -34234312 die Zahl -12233344.

- Für `String`: Zwei Zeichenreihen `s` und t heißen ähnlich, wenn die Anzahl der Vorkommen der Kleinvokale `'a'`, `'e'`, `'i'`, `'o'` und `'u'` paarweise in `s` und `t` gleich sind. Andere möglicherweise in `s` oder `t` vorkommende Zeichen sind für die Eigenschaft ähnlich ohne Bedeutung. Die Zeichenreihen `"arOpFae;:6iiKlooPsAo"` und `"fe3.oiDEalKaoRoi"` sind also ähnlich. Die Normalform einer Zeichenreihe `s` ist diejenige eindeutig bestimmte ähnliche Zeichenreihe, die ausschließlich Kleinvokale enthält, die alphabetisch aufsteigend angeordnet sind. Die gemeinsame Normalform der Zeichenreihen `"arOpFae;:6iiKlooPsAo"` und `"fe3.oiDEalKaoRoi"` ist also die Zeichenreihe `"aaeiiooo"`.
- Für `Tree b`: Zwei Bäume heißen ähnlich, wenn sie dieselbe Struktur besitzen und die gleichen Marken tragen, jedoch möglicherweise anders über die Knoten verteilt. So sind z.B. die Bäume

  ```
  Node 2 (Node 3 Null Null) (Node 5 Null Null)
  Node 3 (Node 2 Null Null) (Node 5 Null Null)
  Node 5 (Node 3 Null Null) (Node 2 Null Null)
  ```

  paarweise ähnlich, die Bäume

  ```
  Node 2 (Node 3 (Node 5 Null Null) Null) Null
  Node 7 (Node 3 Null Null) (Node 5 Null Null)
  ```

  sind jedoch weder zueinander noch einem der obigen drei Bäume ähnlich.

  Die Normalform eines Baums `t` ist derjenige ähnliche Baum `tn`, auf dem die Marken von `t` so verteilt sind, dass sie bei Infix-Durchlauf (d.h., linker Teilbaum, Wurzel, rechter Teilbaum) von `tn` in aufsteigender Ordnung gefunden werden. Die gemeinsame Normalform der Bäume

  ```
  Node 2 (Node 3 Null Null) (Node 5 Null Null)
  Node 3 (Node 2 Null Null) (Node 5 Null Null)
  Node 5 (Node 3 Null Null) (Node 2 Null Null)
  ```

  ist also der Baum

  ```
  Node 3 (Node 2 Null Null) (Node 5 Null Null)
  ```

Haskell Live

Der nächste *Haskell Live*-Termin ist am Freitag, den 22.11.2013.