6. Aufgabenblatt zu Funktionale Programmierung vom 24.11.2009. Fällig: 01.12.2009 / 08.12.2009 (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.lhs` ablegen. Wie für die Lösung zum zweiten Aufgabenblatt sollen Sie dieses Mal also wieder ein "literate" 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.

Wir betrachten den folgenden Datentyp für Ausdrücke:

```
data Expr = Opd Int       |
            Add Expr Expr |
            Sub Expr Expr |
            Neg Expr         deriving Show
```

Dabei stehen die Konstruktoren `Add` und `Sub` für Addition und Subtraktion, der Konstruktor `Neg` für Vorzeichenwechsel.

1. Machen Sie den Typ `Expr` mithilfe expliziter Instanzdeklarationen `instance Eq...` und `instance Ord...` zu Instanzen der Typklassen `Eq` und `Ord`.

   Geben Sie dazu Implementierungen der von den Typklassen `Eq` und `Ord` erforderten Funktionen an, so dass gilt:

   - Zwei Ausdrücke sind genau dann *gleich*, wenn sie in Struktur und Benennungen übereinstimmen oder denselben Wert repräsentieren.
   - Ein Ausdruck s ist *echt kleiner* als ein Ausdruck t genau dann, wenn s ein echter Teilausdruck von t ist oder der von s repräsentierte Wert echt kleiner als der von t ist.
   - Ein Ausdruck s ist *kleiner oder gleich* als ein Ausdruck t genau dann, wenn s ein Teilausdruck von t ist oder der von s repräsentierte Wert kleiner oder gleich als der von t ist.

   Für die Implementierung der Funktionen `max`, `min` und `compare` wird nichts gefordert. Nutzen Sie, wo es möglich ist, die default-Implementierungen der Klassenfunktionen aus.

2. Wir betrachten die folgende Variante von Ausdrücken, wobei die Konstruktoren `VOpd`, `VAdd`, `VSub` und `VNeg` dieselbe Bedeutung haben wie die entsprechenden Konstruktoren des Datentyps `Expr`. Der Konstruktor `VVar` erlaubt, Ausdrücke mit Variablen zu modellieren.

   ```
   data (Eq a, Ord a, Num a) => VExpr a = VOpd a                   |
                                          VVar Char                |
                                          VAdd (VExpr a) (VExpr a) |
                                          VSub (VExpr a) (VExpr a) |
                                          VNeg (VExpr a)            deriving Show
   ```

   Weiters betrachten wir den Typ `States` zur Modellierung von Zuständen, d.h. Zuordnungen von Werten zu Variablen(identifikatoren):

   ```
   newtype (Eq a, Ord a, Num a) => States a = State [(Char,a)]
   ```

   (a) Schreiben Sie eine Haskell-Rechenvorschrift `eval` mit der Typsignatur `eval :: (Eq a, Ord a, Num a) => (VExpr a) -> (VExpr a)`, die einen Argumentausdruck maximal auswertet, d.h. Teilbäume, die keine Variablen als Benennungen enthalten, werden ausgewertet und durch einen `VOpd`-Knoten mit entsprechendem Wert ersetzt.

(b) Schreiben Sie eine Haskell-Rechenvorschrift `evalInState` mit der Typsignatur `evalInState :: (Eq a, Ord a, Num a) => (VExpr a) -> (States a) -> (VExpr a)`, die einen Argumentausdruck für einen gegebenen Zustand maximal auswertet, d.h. Variablen im Ausdruck werden durch den entsprechenden Wert im Zustand ersetzt. Dabei wird für eine Variable stets der von links erste Wert im Zustand genommen, falls diese Variable mehrfach im Zustand vorkommt (kommt eine im Variable des Ausdrucks im Zustand gar nicht vor, kann der Ausdruck maximal, aber nicht vollständig ausgewertet werden).

(c) Machen Sie den Typ `VExpr a` mithilfe einer expliziten Instanzdeklaration `instance ...Eq...` zu einer Instanz der Typklasse `Eq`. Zwei Ausdrücke seien dabei *gleich* genau dann, wenn sie strukturgleich sind und sich höchstens in den Variablennamen unterscheiden.

Denken Sie bitte daran, dass für die Lösung von Aufgabenblatt 6 ein "literate" Haskell-Skript gefordert ist!