9. Aufgabenblatt zu Funktionale Programmierung vom 08.12.2004, fällig: 15.12.2004

Themen: *Funktionen auf ganzen Zahlen, Zeichenreihen und Bäumen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der unten angegebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe9.hs` ablegen. Sie sollen also wieder ein "gewöhnliches" Haskell-Script 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 wieder den algebraischen Datentyp für arithmetische Ausdrücke über ganzen Zahlen und (einzeichigen) Variablen von Aufgabenblatt 8 zusammen mit den Funktionen `expCreate` und `expEqual`. Kopieren (nicht per Modul importieren) Sie diese Funktionen aus Ihrer Lösung zu Aufgabenblatt 8 in die neue Abgabedatei `Aufgabe9.hs`.

   - Schreiben Sie eine Haskell-Rechenvorschrift `maxCommonSubExpr`, mit der Signatur `maxCommonSubExpr :: Expr -> Expr -> [Expr]`. Die Funktion `maxCommonSubExpr` liefert als Ergebnis die knotenmaximalen Teilbäume ab, die sowohl Teilbaum des ersten wie des zweiten Arguments sind. Teilbaum ist dabei wie auf Aufgabenblatt 8 zu verstehen. Haben die beiden Argumentbäume keine gemeinsamen Teilbäume, so liefert die Funktion `maxCommonSubExpr` die leere Liste als Resultat. Kommt ein knotenmaximaler Teilbaum mehrfach in einem oder beiden Argumentbäumen als Teilbaum vor, so taucht er dennoch nur einmal in der Ergebnisliste auf. Die Reihenfolge, in der die knotenmaximalen Teilbäume in der Resultatliste auftauchen, spielt keine Rolle.
   - Schreiben Sie eine Haskell-Rechenvorschrift `transform` mit der Signatur `transform :: Expr -> Expr`, die beginnend an der Wurzel jedes Vorkommen eines Teilbaums der Form `(t1*(t2+t3))` durch den Teilbaum `((t1*t2)+(t1*t3))` ersetzt. Durch die Ersetzung kann es geschehen, dass tiefer im Baum ein solches Muster erneut entsteht. Das ist dann ebenfalls zu ersetzen. Andere Muster, etwa `((t1+t2)*t3)` oder `(t1*(t2-t3))` werden nicht ersetzt. Auch werden Ausdrücke von der Funktion `transform` nicht wie von der Funktion `eval` vereinfacht. Arithmetische Gesetze werden ebenfalls nicht ausgenutzt. Die beiden Beispielausdrücke von

Aufgabenblatt 8 blieben also unter der Funktion `transform` unverändert.

- Schreiben Sie eine Haskell-Rechenvorschrift `exportAs` mit der Signatur `exportAs :: Expr -> String -> String`, die einen als Ausdruck übergebenen Ausdruck in Abhängigkeit des zweiten Arguments in üblicher Weise in Infix-, Präfix- oder Postfixnotation ausgibt. Ausdrücke in Infixnotation sollen dabei vollständig, aber nicht überflüssig geklammert sein. Leerzeichen sollen in der Ausgabezeichenreihe nicht enthalten sein. In der Postfix- und Präfixvariante sind Klammern weder erforderlich noch gewünscht. Ist das zweite Argument vom Wert `Infix`, `Praefix` oder `Postfix` wird der entsprechend transformierte Ausdruck zurückgegeben, ansonsten die Zeichenreihe `Ungueltige Eingabe`.

  Hinweis: In Präfixordnung wird zunächst die Marke an der Wurzel ausgegeben, anschließend werden die Marken des linken Teilbaums entsprechend der Präfixordnung, danach die Marken des rechten Teilbaums in Präfixordnung ausgegeben. In Infixordnung werden zunächst die Marken des linken Teilbaums in Infixordnung ausgegeben, danach die Marke der Wurzel, zum Schluss die Marken des rechten Teilbaums in Infixordnung. In Postfixordnung werden zunächst die Marken des linken Teilbaums in Postfixordnung ausgegeben, danach die Marken des rechten Teilbaums in Postfixordnung, zum Schluss die der Wurzel.

- In dieser Aufgabe betrachten wir folgenden algebraischen Datentyp.

  ```
  data Tree a = Leaf Int a |
                Node (Int a) (Tree a) (Tree a)
  ```

  Bäume dieses polymorphen Typs wollen wir als Suchbäume auffassen, wobei die Integer-Komponente den Suchschlüssel repräsentiert. Ein Baum heißt dabei *wohlgeformt* oder kurz ein *Suchbaum*, wenn für jeden Knoten gilt, dass alle Schlüssel im linken Teilbaum einen kleineren Wert und im rechten Teilbaum einen größeren Wert haben als im betrachteten Knoten. Schreiben Sie eine Haskell-Rechenvorschrift `isSearchTree` mit der Signatur `isSearchTree :: Tree a -> Bool`, die angewendet auf einen Baum den Wert `True` liefert, wenn es sich um einen Suchbaum handelt, und `False` sonst.