7. Aufgabenblatt zu Funktionale Programmierung vom 07.12.2005, fällig: 14.12.2005 / 11.01.2006 (jeweils 12:00 Uhr)

Themen: *Polymorphe Funktionen und Funktionale auf Bäumen*

Für dieses Aufgabenblatt sollen Sie Haskell-Rechenvorschriften zur Lösung der nachstehend beschriebenen Aufgabenstellungen entwickeln und für die Abgabe in einer Datei namens `Aufgabe7.hs` ablegen. Sie sollen also für dieses Aufgabenblatt wieder ein Standard-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:

Wir betrachten in der Folge folgenden polymorphen Typ auf Bäumen:

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

1. Analog zu den in Haskell vordefinierten Funktionen auf Listen `zip` und `unzip` wollen wir jetzt Funktionen auf Werten vom Typ `Tree a` mit vergleichbarer Funktionalität schreiben. Im einzelnen:

   - Schreiben Sie eine Haskell-Rechenvorschrift `zipT` mit der Signatur `zipT :: Tree a -> Tree b -> Tree (a,b)`. Angewendet auf zwei Argumentbäume liefert die Funktion `zipT` als Resultat einen neuen Baum, dessen Knoten mit Paaren benannt sind, deren Komponenten die Benennungen der Argumentbäume sind. Hat einer der Teilbäume den Wert `Nil`, während der andere noch einen Wert davon verschieden hat, so bleibt ähnlich wie bei der Funktion `zip` auf Listen dieser Teilbaum unberücksichtigt. Folgende Beispiele illustrieren die gewünschte Funktionalität von `zipT`:

     ```
     zipT (Node 4 (Node 3 Nil Nil) Nil) (Node "abc" (Node "xyz" Nil Nil) Nil)
               == (Node (4,"abc") (Node (3,"xyz") Nil Nil)) Nil)
     zipT (Node 4 (Node 3 Nil Nil) Nil) (Node "abc" Nil Nil)
               == (Node (4,"abc") Nil Nil) Nil)
     ```

   - Schreiben Sie eine Haskell-Rechenvorschrift `unzipT` mit der Signatur `unzipT :: Tree (a,b) -> (Tree a,Tree b)`. Angewendet auf einen Argumentbaum liefert die Funktion `unzipT` als Resultat ein Paar strukturell gleicher Bäume, deren Knoten mit der linken bzw. rechten Komponente des Paares bezeichnet ist, das den entsprechenden Knoten im Argumentbaum bezeichnet. Folgende Beispiele illustrieren die gewünschte Funktionalität von `unzipT`:

     ```
     unzipT (Node (4,"abc") (Node (3,"xyz") Nil Nil)) Nil
      == ((Node 4 (Node 3 Nil Nil)) Nil),(Node "abc" (Node "xyz" Nil Nil)) Nil))
     unzipT (Node (4,"abc") Nil Nil) Nil)
      == ((Node 4 Nil Nil) Nil),(Node "abc" Nil Nil) Nil))
     ```

2. In dieser Aufgabe wollen wir die Funktionen `zipT` und `unzipT` um ein weiteres Argument erweitern, eine Manipulationsfunktion `f`. Auf diese Weise erhalten wir die Funktionale `zipTF` und `unzipTF`.

   - Schreiben Sie eine Haskell-Rechenvorschrift `zipTF` mit der Signatur `zipTF :: (a -> b -> c) -> Tree a -> Tree b -> Tree c`. Angewendet auf eine Manipulationsfunktion `f` und zwei Argumentbäume `s` und `t` liefert die Funktion `zipTF` als Resultat einen Baum, dessen Knotenbenennungen sich als Bild der Anwendung der Funktion `f` auf die Benennungen der entsprechenden Knoten von `s` und `t` ergibt. Hat einer der Teilbäume von `s` oder `t` den Wert `Nil`, während der andere noch einen Wert davon verschieden hat, so bleibt ähnlich wie bei der Funktion `zipT` dieser nichtleere Teilbaum unberücksichtigt. Folgende Beispiele illustrieren die gewünschte Funktionalität von `zipTF`:

     ```
     zipTF (+) (Node 4 (Node 3 Nil Nil) Nil) (Node 2 (Node 7 Nil Nil) Nil)
                             == (Node 6 (Node 10 Nil Nil)) Nil)
     zipTF (++) (Node "abc" (Node "def" Nil Nil) Nil) (Node "xyz" Nil Nil)
                             == (Node "abcxyz") Nil Nil) Nil)
     ```

   - Schreiben Sie eine Haskell-Rechenvorschrift `unzipTF` mit der Signatur `unzipTF :: ((a,b) -> (c,d)) -> Tree (a,b) -> (Tree c,Tree d)`. Angewendet auf eine Manipulationsfunktion `f` und einen Argumentbaum `t` liefert die Funktion `unzipTF` als Resultat einen Baum, dessen Knotenbenennungen sich als Projektion auf die erste bzw. zweite Komponente des Bildes der Anwendung der Funktion `f` auf die Benennung des entsprechenden Knotens von `t` ergibt. Folgende Beispiele illustrieren die gewünschte Funktionalität von `unzipTF`:

     ```
     unzipTF (\x->x) (Node (4,"abc") (Node (3,"xyz") Nil Nil) Nil)
       == ((Node 4 (Node 3 Nil Nil) Nil),(Node "abc" (Node "xyz" Nil Nil)
     unzipTF (\(x,y)->(y,x)) (Node (4,"abc") (Node (3,"xyz") Nil Nil) Nil)
       == ((Node "abc" (Node "xyz" Nil Nil),(Node 4 (Node 3 Nil Nil) Nil))
     ```

Hinweis:

- Verwenden Sie *keine* Module. Wenn Sie Funktionen wiederverwenden möchten, kopieren Sie diese in die Abgabedatei `Aufgabe7.hs`. Andere als diese Datei werden vom Abgabeskript ignoriert.