Aufgabenblatt zur 3. Übung

Zeitraum: 27.04. bis 30.04.2009

1. Aufgabe: (Klausuraufgabe 02.2009)

(a) Gegeben sei der Typ `data Tree = Node Tree Tree | Leaf Int | NIL`.

Geben Sie eine Funktion `delete :: Tree -> Int -> Tree` an, die aus einem Baum des Typs `Tree` alle die Blattknoten entfernt, deren Int-Wert mit dem Eingabewert (zweites Argument der Funktion) übereinstimmt.

(b) Gegeben seien die Typen

```
data TreeA = NodeA Int TreeA TreeA | NILA
data TreeB = NodeB (Int,Int) TreeB TreeB | Leaf Int | NILB
```

Geben Sie eine HASKELL-Funktion `compose :: TreeA -> TreeA -> TreeB` einschließlich der Typdefinition an, die durch knotenweise Transformation aus den zwei Bäumen (Argumenten) des Typs `TreeA` einen Baum des Typs `TreeB` nach folgendem Prinzip erzeugt:

- Falls beide Argumentbäume (an der gleichen Position) leer sind, ist das Resultat (an dieser Position) ein leerer Baum,
- falls nur ein Argumentbaum leer ist, ist das Resultat ein Blattknoten (Leaf), wobei dessen Int-Wert die Summe der Integerwerte des anderen Argumentbaumes erhalten soll,
- falls beide Argumentbäume (an der gleichen Position) nichtleere Knoten enthalten, ist das Resultat ein Knoten mit dem Tupel der beiden Integerwerte in der Reihenfolge 1. Argument - 2. Argument.

Hinweis: Schreiben Sie gegebenenfalls eine Hilfsfunktion `summe` zum Berechnen der Summe der `Int`-Werte eines Baumes.

2. Aufgabe: (AGS 11.19)

Gegeben sei der polymorphe Typ:

```
data Tree t = Leaf t | Branch t  (Tree t)  (Tree t)
```

(a) Schreiben Sie eine polymorphe Funktion `liste` (einschließlich Typdefinition von `liste`), die aus jedem Baum B o. g. Typs jeweils eine Liste des Typs `[t]` aller Knotenbewertungen des Baumes B in der Reihenfolge $<$ linker Teilbaum , Wurzelbewertung , rechter Teilbaum $>$ erzeugt („inorder“–Durchlauf).

(b) Seien `f::t -> t` eine Funktion, `x` eine Variable vom Typ `Tree t` und `map` die Ihnen bekannte Haskell–Funktion.

Schreiben Sie eine Funktion `g` (einschließlich Typdefinition von `g`), so dass folgende Gleichung gilt:

```
map f (liste x) = liste (g f x)
```

3. Aufgabe: (AGS 11.27*)

(a) Es seien σ ein zweistelliges, γ ein einstelliges und α ein nullstelliges Funktionssymbol.
$V = \{x_1, x_2, x_3\}$ sei eine Menge von Variablen.
Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an und ermitteln Sie deren allgemeinsten Unifikator:

$$t_1 = \sigma(\sigma(\gamma(x_1), x_2), \gamma(\gamma(\alpha)))$$
$$t_2 = \sigma(\sigma(\gamma(\alpha), \gamma(\gamma(x_1))), \gamma(x_3))$$

Geben Sie dabei jeweils die benutzte Regel an.

(b) Identifizieren Sie die Typsymbole σ, γ und α mit den Typkonstruktoren $()^2$, $[\,]$ und `Int`.
Geben Sie nun zu t_1, t_2 aus (a) die äquivalenten Typterme und Typausdrücke an.

4. Aufgabe: (AGS 11.26)

Es sei δ ein dreistelliges, σ ein zweistelliges, γ ein einstelliges und α ein nullstelliges Basisfunktionssymbol. Des Weiteren sei $V = \{x_1, \ldots, x_3\}$ eine Menge von Variablen.

Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an und ermitteln Sie den allgemeinsten Unifikator:

$t_1 = \delta(\gamma(x_1), \gamma(\gamma(\alpha)), \sigma(x_2, \alpha))$

$t_2 = \delta(\gamma(\alpha), \gamma(x_2), x_3)$

Zusatzaufgabe: (AGS 11.20*)

(a) Definieren Sie eine Funktion `addL`, welche die Elemente einer Liste von ganzen Zahlen aufsummiert. Geben Sie den Typ von `addL` an.

(b) Gegeben sei der Datentyp `data UTree = UI [UTree] | UL Int` zur Repräsentation von Bäumen bei denen i) die inneren Knoten (beschriftet mit `UI`) beliebig viele Nachfolger haben und ii) die Blattknoten (beschriftet mit `UL`) Integer-Werte tragen. Definieren Sie eine Funktion `addU :: UTree -> Int`, welche alle `Int`-Werte in einem solchen Baum aufsummiert. Nutzen Sie dafür die Funktion `addL` und die Funktion `map :: (a -> b) -> [a] -> [b]` aus der Vorlesung, die eine Liste elementweise transformiert.

(c) Definieren Sie einen polymorphen Datentyp `Tree a` zur Repräsentation von Bäumen, bei denen i) die inneren Knoten genau drei Nachfolger haben und ii) die Blattknoten Werte eines beliebigen, aber festen Typs `a` tragen. Geben Sie außerdem für diesen Datentyp (nur) den Typ einer polymorphen Funktion `mapTree` an, welche analog zur bekannten Funktion `map` für Listen, einen Baum blattweise transformiert. (D.h. `mapTree` besitzt zwei Argumente - eins für die Transformation der Blattwerte und eins für den Eingabebaum.)