Programmierung

Aufgabe 1 (AGS 12.1.16 a, 12.1.48 b,c, 12.1.56)
Gegeben sei der polymorphe algebraische Datentyp

```
data BinTree a = Branch a (BinTree a) (BinTree a) | Leaf a
```

(a) Schreiben Sie einen Baum des Typs `BinTree Int` in dem der Konstruktor `Leaf` mindestens 5-Mal vorkommt.

(b) Geben Sie eine Funktion `depth :: BinTree a -> Int` an, welche für einen Baum `t` die Länge des kürzesten Pfads von der Wurzel zu einem Blattknoten von `t` berechnet. Die Länge eines Pfads ist dabei die Anzahl der auf ihm vorkommenden Knoten, d.h. der Pfad von der Wurzel zur Wurzel selbst hat die Länge `1`.

(c) Geben Sie eine Funktion `paths :: BinTree a -> BinTree [a]` an, die in einem Baum die Beschriftung jedes Knotens u durch die Liste der Knotenbeschriftungen auf dem Pfad vom Wurzelknoten zu u ersetzt. Ist der Wurzelknoten z. B. in `t` mit `5` beschriftet, so ist er in `paths t` mit `[5]` beschriftet, und ist sein erster Kindknoten in `t` mit `3` beschriftet, dann ist dieser in `paths t` mit `[5,3]` beschriftet.

(d) Schreiben Sie eine Funktion `tmap :: (a -> b) -> BinTree a -> BinTree b`, so dass für alle `f :: a -> b` und `t :: BinTree a` gilt, dass `tmap f t` der Baum ist, der entsteht, indem jeder Knoten `v` von `t` durch `f v` ersetzt wird.

Aufgabe 2 (ABS 12.1.51 a,b)

(a) Schreiben Sie die Funktion `unpairs :: [(a, b)] -> ([a], [b])`, welche eine Liste von Paaren in zwei Listen aufspaltet. Dabei soll die erste Liste die ersten Komponenten und die zweite Liste die zweiten Komponenten der Paare enthalten, z.B.:

```
unpairs [('a', 1), ('b', 2), ('c', 3)] == (['a', 'b', 'c'], [1, 2, 3]).
```

Die Reihenfolge der Elemente soll erhalten bleiben.

(b) Gegeben seien folgende Funktionen.

```
map :: (a -> b) -> [a] -> [b]
map _ []       = []
map f (x : xs) = f x : map f xs

uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (x, y) = f x y
```

Werten Sie den Term `map (uncurry (+)) [(1, 2), (3, 4)]` *schrittweise* aus.

Hinweis: Beide Funktionen sind bereits in der Prelude-Bibliothek von Haskell implementiert und müssen somit nicht neu definiert werden.

Hinweis In den folgenden Aufgaben soll der Unifikationsalgorithmus schrittweise dokumentiert werden. Sie können die Aufgaben auf Papier lösen und als Scan bzw. Foto über Opal einreichen.

Aufgabe 3 (AGS 12.2.14)

(a) Gegeben seien die Terme $t_1 = \delta(\alpha, \sigma(x_1, \alpha), \sigma(x_2, x_3))$ und $t_2 = \delta(\alpha, \sigma(x_1, x_2), \sigma(x_2, \gamma(x_2)))$ über dem Rangalphabet $\Sigma = \{\delta^{(3)}, \sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an. Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

(b) Gegeben seien die Haskell-Typterme

$$t_1 = \texttt{(a, [a])}, \quad t_2 = \texttt{(Int, [Double])} \quad \text{und} \quad t_3 = \texttt{(b, c)}.$$

Welche Paare dieser Terme sind unifizierbar? Geben Sie ggf. einen allgemeinsten Unifikator an!

Aufgabe 4 (AGS 12.2.12)

Gegeben seien folgende Terme über dem Rangalphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$:

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

(a) Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an. Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

(b) Geben Sie zwei weitere Unifikatoren an.

(c) Geben Sie zwei Terme t_1 und t_2 über dem Alphabet Σ an, so dass im Laufe der Anwendung des Unifikationsalgorithmus auf t_1 und t_2 der Occur-Check fehlschlägt.

Zusatzaufgabe 1 (ABS 12.1.21 a–c)

Nutzen Sie den Datentyp `BinTree` aus Aufgabe 1.

(a) Programmieren Sie eine Funktion `check :: BinTree Bool -> Bool`, die genau dann `True` liefert, wenn der Eingabebaum mindestens ein Blatt mit dem gespeicherten Wert `False` besitzt.

(b) Programmieren Sie eine Funktion `toList :: BinTree a -> [a]`, die aus einem Baum des Typs `Tree a` eine Liste der gespeicherten Werte der Blätter generiert, und zwar in der Reihenfolge von rechts nach links gesehen.

(c) Programmieren Sie eine Funktion `toTree :: [Int] -> BinTree Int`, die eine Liste von Zahlen als Argument nimmt und einen Baum erzeugt, welcher – zusätzlich zu einem Blatt mit dem gespeicherten Wert 42 und inneren Knoten mit dem gespeicherten Wert 23 – für jedes Element der Liste genau ein Blatt mit dessen Wert besitzt. Die Form des Baumes spielt keine Rolle.

Zusatzaufgabe 2 (AGS 12.2.15)
Gegeben seien die Terme

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

über dem Rangalphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an. Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.