3. Übung (30. April – 4. Mai 2018)

Programmierung

Beachten Sie die Übungsverlegungen zum 1. Mai.

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

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

(a) Schreiben Sie einen Baum des Typs `Tree Int` mit mindestens 5 Blättern auf.

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

Aufgabe 2 (12.1.54)
In der Vorlesung wurden die Higher-Order-Funktionen

- `map :: (a -> b) -> [a] -> [b]`,
- `filter :: (a -> Bool) -> [a] -> [a]`, und
- `foldr :: (a -> b -> b) -> b -> [a] -> b`

vorgestellt. Implementieren Sie eine Funktion `f :: [Int] -> Int` mithilfe von `map`, `filter` und `foldr`, die das Produkt der Quadrate der geraden Zahlen in der Eingabeliste berechnet.

Aufgabe 3 (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. 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 einem beliebigen Alphabet) an, so dass im Laufe der Anwendung des Unifikationsalgorithmus auf t_1 und t_2 der Occur-Check fehlschlägt.

Zusatzaufgabe 1 (AGS 12.1.57)
Implementieren Sie eine Funktion `foldl :: (a -> b -> a) -> a -> [b] -> a`, so dass für jedes `f :: a -> b -> a` und $\texttt{a}_0$ `::` a, $\texttt{b}_1$, …, $\texttt{b}_\texttt{k}$ `::` b, $\texttt{k} \in \mathbb{N}$ gilt, dass

$$\texttt{foldl f a}_0\ [\texttt{b}_1,\ \ldots,\ \texttt{b}_\texttt{k}] \ = \ \texttt{f}\ (\texttt{f}\ \cdots\ (\texttt{f a}_0\ \texttt{b}_1)\ \cdots\ \texttt{b}_{\texttt{k}-1})\ \texttt{b}_\texttt{k}\,,$$

also z.B.

$$\texttt{foldl}\ (+)\ 5\ [1,\ 4,\ 3] \ = \ (+)\ ((+)\ ((+)\ 5\ 1)\ 4)\ 3 \ = \ ((5\ +\ 1)\ +\ 4)\ +\ 3\,.$$

Insbesondere soll `foldl f a [] = a` gelten.
Für Fortgeschrittene: Implementieren Sie **`foldl`** unter Verwendung von **`foldr`**.

Zusatzaufgabe 2 (AGS 12.2.15)
Gegeben seien die Terme

$$\begin{aligned} t_1 &= \sigma(\alpha,\ \sigma(\gamma(\alpha), \sigma(x_2, x_3 \qquad\ ))) \,, \\ t_2 &= \sigma(\alpha,\ \sigma(x_1 \quad , \sigma(x_2, \sigma(x_2, x_1)))) \end{aligned}$$

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.