Programmierung

04. Übungsblatt

Zeitraum: 02. – 05. Mai 2017

Beachten Sie die Übungsverlegungen zum 01. Mai!

Übung 1 (AGS 12.2.15)
Gegeben seien die Terme

$$t_1 = \sigma(\alpha,\ \sigma(\gamma(\alpha),\ \sigma(x_2, x_3))) \quad \text{und}$$
$$t_2 = \sigma(\alpha,\ \sigma(x_1,\ \sigma(x_2, \sigma(x_2, x_1))))$$

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.

Übung 2 (AGS 12.2.12)

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

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

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!

Übung 3 (AGS 12.3.20)
Folgende Definitionen seien gegeben:

```
foo :: [Int] -> [Int]
foo []     = []
foo (x:xs) = x : x : (-1) : foo xs

sum :: [Int] -> Int
sum []     = 0
sum (x:xs) = x + sum xs

length :: [Int] -> Int
length []     = 0
length (x:xs) = 1 + length xs
```

Die folgende Aussage soll mittels struktureller Induktion über Listen bewiesen werden:

(A): Für jede Liste `xs :: [Int]` gilt:

`sum (foo xs) = 2 * sum xs - length xs .`

Bearbeiten Sie die folgenden Teilaufgaben; geben Sie bei jeder Umformung die benutzte *Definition*, bzw. die *Induktionsvoraussetzung* an; quantifizieren Sie alle Variablen.

(a) Zeigen Sie den Induktionsanfang.
(b) Geben Sie die Induktionsvoraussetzung *vollständig* an.
(c) Zeigen Sie den Induktionsschritt.

Übung 4 (AGS 12.3.11)
Folgende Definitionen seien gegeben:

```
data Tree = Leaf Float | Branch Float Tree Tree

add :: Tree -> Float -> Tree
add (Leaf x)       a = Leaf (x+a)
add (Branch x l r) a = Branch (x+a/3) (add l (a/3)) (add r (a/3))

rev :: Tree -> Tree
rev (Leaf x)         = Leaf x
rev (Branch x l r)   = Branch x (rev r) (rev l)

sum :: Tree -> Float
sum (Leaf x)         = x
sum (Branch x l r)   = x + sum l + sum r
```

Zeigen Sie für die oben aufgeführten Definitionen mit Hilfe der strukturellen Induktion, dass die folgende Gleichung für einen beliebigen Baum `t :: Tree` und eine beliebige Zahl `a :: Float` erfüllt ist:

```
sum (add t a) = sum (rev t) + a
```

Geben Sie bei Umformungen die jeweils benutzten Gesetzmäßigkeiten / Definitionen an.

Zusatzaufgabe 1 (AGS 12.1.34)
Gegeben ist der Datentyp

```
data Tree = Node Int [Tree]
```

eines Baumes, bei dem jeder Knoten eine beliebige Anzahl an Kindbäumen haben kann (gegeben in einer Liste vom Typ `[Tree]`).

(a) Geben Sie die Definition der Funktion `noLeaves :: Tree -> Int` an, die ermittelt, wie viele Blattknoten ein gegebener Baum vom Typ `Tree` enthält. Ein Blattknoten ist ein Knoten mit einer leeren Liste an Kindbäumen.
(b) Geben Sie die Definition der Funktion `even :: Tree -> Bool` an, die zu einem gegebenen Baum vom Typ `Tree` ermittelt, ob jeder Knoten eine gerade Anzahl an Nachfolgern hat und in diesem Fall `True` zurückgibt (ansonsten `False`). Sie dürfen dabei auf die Funktion `length :: [Int] -> Int` zurückgreifen, die die Länge einer Liste ermittelt.