4. Übung (7.–11. Mai 2018)

Programmierung

Beachten Sie die Übungsverlegungen zum 10. Mai.

Aufgabe 1 (AGS 12.2.13)

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

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

(a) Wenden Sie den Unifikationsalgorithmus auf t_1 und t_2 an. Geben Sie den allgemeinsten Unifikator an.

(b) Geben Sie zwei weitere Unifikatoren an.

Aufgabe 2 (AGS 12.3.20)

Zeigen Sie unter Verwendung der folgenden Definitionen durch strukturelle Induktion die Gültigkeit der Gleichung `sum (foo xs) = 2 * sum xs - length xs` für jedes `xs :: [Int]`.

```
1  foo :: [Int] -> [Int]
2  foo []     = []
3  foo (x:xs) = x : x : (-1) : foo xs
4
5  sum :: [Int] -> Int
6  sum []     = 0
7  sum (x:xs) = x + sum xs
8
9  length :: [Int] -> Int
10 length []     = 0
11 length (x:xs) = 1 + length xs
```

Aufgabe 3 (AGS 12.3.11)

Zeigen Sie unter Verwendung der folgenden Definitionen durch strukturelle Induktion die Gültigkeit der Gleichung `sum (add t a) = sum (rev t) + a` für jedes `t :: Tree` und `a :: Float`.

```
1  data Tree = Leaf Float | Branch Float Tree Tree
2
3  add :: Tree -> Float -> Tree
4  add (Leaf x)       a = Leaf (x+a)
5  add (Branch x l r) a = Branch (x+a/3) (add l (a/3)) (add r (a/3))
6
7  rev :: Tree -> Tree
8  rev (Leaf x)         = Leaf x
9  rev (Branch x l r)   = Branch x (rev r) (rev l)
10
11 sum :: Tree -> Float
12 sum (Leaf x)         = x
13 sum (Branch x l r)   = x + sum l + sum r
```

Aufgabe 4 (AGS 12.4.1 $\star$)

(a) Bestimmen Sie für jeden der folgenden λ-Terme t die Mengen $FV(t)$ und $GV(t)$:

- $(\lambda x.x\ y)\ (\lambda y.y)$
- $(\lambda x.(\lambda y.z\ (\lambda z.z\ (\lambda x.y))))$
- $(\lambda x.(\lambda y.x\ z\ (y\ z)))\ (\lambda x.y\ (\lambda y.y))$

(b) Reduzieren Sie die folgenden λ-Terme zu Normalformen. Schreiben Sie – bevor Sie einen Ableitungsschritt ausführen – für die relevanten (Teil-)Ausdrücke die Mengen der freien bzw. der gebundenen Vorkommen von Variablen auf.

- $(\lambda x.(\lambda y.x\ z\ (y\ z)))\ (\lambda x.y\ (\lambda y.y))$
- $(\lambda x.(\lambda y.(\lambda z.z)))\ x\ (+\ y\ 1)$
- $(\lambda x.(\lambda y.x\ (\lambda z.y\ z)))\ (((\lambda x.(\lambda y.y))\ 8)\ (\lambda x.(\lambda y.y)\ x))$
- $(\lambda h.(\lambda x.h\ (x\ x))\ (\lambda x.h\ (x\ x)))\ ((\lambda x.x)\ (+\ 1\ 5))$
- $(\lambda f.(\lambda a.(\lambda b.f\ a\ b)))\ (\lambda x.(\lambda y.x))$

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.