# Programmierung

## 13. Übungsblatt

Zeitraum: 10. – 14. Juli 2017

*Lernraum Programmierung* am Sa., 15.07.2017, 13:00–15:00 Uhr, APB/E023!

**Übung 1**

Folgendes Prolog-Programm sei gegeben.

```
pair(X, Y, [X, Y | _]) :- Y is X + 1.
pair(X, Y, [_ | L])    :- pair(X, Y, L).
```

Dabei wertet das Prädikat `is` arithmetische Formeln aus und prüft die Ergebnisse auf Gleichheit. Bestimmen Sie durch SLD-Refutation *alle* Belegungen von `X` und `Y` für das Goal

```
?- pair(X, Y, [1,2,4,5]).
```

**Übung 2**

Folgendes Prolog-Programm sei gegeben.

```
subt(X,X).
subt(S1,s(_,T2)) :- subt(S1,T2).
subt(S1,s(T1,_)) :- subt(S1,T1).
```

Intuitiv wird durch `subt` die Teilbaum-Relation kodiert.

(a) Bestimmen Sie durch SLD-Refutation alle Belegungen von `X` und `Y` für das Goal

```
?- subt(s(X,Y), s(s(a,b), s(b,a))).
```

(b) Bestimmen Sie durch SLD-Refutation drei verschiedene Lösungen für das Goal

```
?- subt(s(a,a), X).
```

**Zusatzaufgabe 1 (AGS 12.1.45 ⋆)**

(a) Sei `xs` eine Liste. Eine Teilliste von `xs` ist eine Liste, die durch Löschen von beliebig vielen Elementen aus `xs` entsteht. Die Teillisten von `[1, 2, 3]` sind beispielsweise `[]`, `[1]`, `[2]`, `[3]`, `[1, 2]`, `[1, 3]`, `[2, 3]` und `[1, 2, 3]`.
Schreiben Sie in Haskell die Funktion `isSubseqOf :: Eq a => [a] -> [a] -> Bool`, die prüft, ob die erste übergebene Liste eine Teilliste der zweiten ist.

(b) Gegeben seien folgende Funktionen.

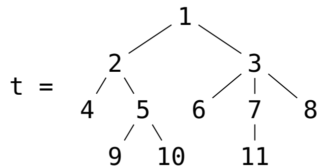
```
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
zipWith f (x : xs) (y : ys) = f x y : zipWith f xs ys
zipWith _ _        _        = []

h :: [Int] -> [Int] -> [Int]
h xs ys = zipWith (\ x y -> 2 * x + y) xs ys
```

Berechnen Sie schrittweise `h [1, 2] [3, 4, 5]`. Sie dürfen dabei sinnvolle Abkürzungen einführen.

(c) Gegeben sei folgender algebraischer Datentyp für beliebig verzweigende Bäume.

```haskell
data Tree a = Node a [Tree a]
```

Ein Pfad eines Baumes ist die Liste aller Knotenbeschriftungen auf dem Weg von der Wurzel des Baumes zu einem Knoten. (Daraus folgt, dass jeder Pfad mindestens ein Element enthält, nämlich die Beschriftung des Wurzelknotens.)
Implementieren Sie die Funktion `isPathOf :: Eq a => [a] -> Tree a -> Bool`, welche prüft, ob ein Baum einen gegebenen Pfad beinhaltet. Zum Beispiel:




```haskell
isPathOf [] t = False
isPathOf [1, 2] t = True
isPathOf [1, 2, 9] t = False
isPathOf [1, 2, 5, 9] t = True
```

**Zusatzaufgabe 2 (AGS 12.4.34)**

(a) Berechnen Sie die Normalform des unten stehenden $\lambda$-Terms, indem Sie ihn *schrittweise* reduzieren. Geben Sie dabei vor jedem Schritt für die relevanten Teilausdrücke die Mengen der gebunden bzw. frei vorkommenden Variablen an.

$$(\lambda xy.yx)\,(\lambda x.xy)\,(\lambda x.x)$$

(b) Gegeben sei folgende Haskell-Funktion:

```haskell
f :: Int -> Int -> Int
f x y
  | x <= y    = if y <= x then x else f (y - 1) x
  | otherwise = f (x - 1) y
```

Geben Sie einen $\lambda$-Term $\langle F\rangle$ an, so dass $\texttt{f} = \langle Y\rangle\langle F\rangle$ gilt.

(c) Gegeben sei der $\lambda$-Term

$$\langle G\rangle = \Big(\lambda gxyz.\langle\text{ite}\rangle\,(\langle\text{iszero}\rangle x)\;y\;\big(g\,(\langle\text{pred}\rangle x)\,(\langle\text{mult}\rangle yz)\,(\langle\text{pred}\rangle z)\big)\Big)$$

Berechnen Sie die Normalform des Terms $\langle Y\rangle\langle G\rangle\langle 1\rangle\langle 5\rangle\langle 4\rangle$. Schreiben Sie für jeden Aufruf von $\langle G\rangle$ jeweils **zwei Zeilen**: eine in der Sie die Werte der Parameter des Aufrufs protokollieren, und eine in der Sie ihre Auswertung skizzieren.
Stellen Sie zuerst die Wirkungsweise des Fixpunktkombinators $\langle Y\rangle$ in einer Nebenrechnung dar. Führen Sie zweckmäßige Abkürzungen der $\lambda$-Terme ein.

**Zusatzaufgabe 3 (AGS 12.2.8)**

(a) Wenden Sie den Unifikationsalgorithmus auf die Terme $t_1$ und $t_2$ an:

$$t_1 = \sigma\big(x_3, \sigma(\gamma(x_3), x_4, x_1), x_2\big)\,,$$
$$t_2 = \sigma\big(x_3, \sigma(x_1, x_3, x_2), \gamma(x_4)\big)\,.$$

Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Die erste Regelanwendung wurde bereits eingeleitet. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

(b) Geben Sie zwei weitere Unifikatoren an.

**Zusatzaufgabe 4 (AGS 13.10)**

(a) Geben Sie für das folgende $C_0$-Programm *Test* die Übersetzung in ein linearisiertes $AM_0$–Programm an. Zwischenschritte der Übersetzung brauchen Sie nicht anzugeben.

```
/* Test */
#include <stdio.h>

int main() {
  int x, y, a;
  scanf("%i", &y);
  scanf("%i", &a);
  x = 0;
  while (x < a) {
    x = x + 1;
    y = y * y;
  }
  printf("%d", y);
  return 0;
}
```

(b) Folgendes $AM_0$–Programm sei gegeben:

```
1:  READ 1;        6:  SUB;
2:  READ 2;        7:  JMC 9;
3:  LOAD 1;        8:  JMP 5;
4:  LOAD 2;        9:  WRITE 2;
5:  LIT 0;
```

Protokollieren Sie den schrittweisen Ablauf dieses Programms auf der $AM_0$ mit der Anfangskonfiguration $(1, \varepsilon, [\ \ ], 0 : 1, \varepsilon)$.