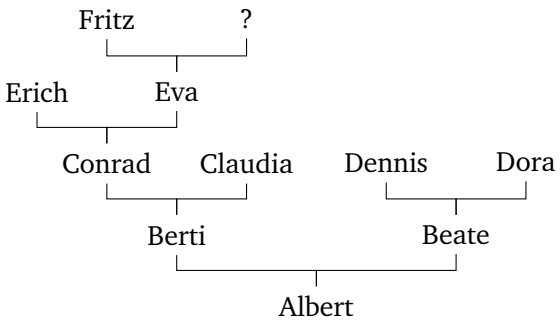
Programmierung

12. Übungsblatt

Zeitraum: 03. – 07. Juli 2017

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

Übung 1
Folgender Stammbaum sei gegeben:



Bilden Sie diesen Stammbaum in Prolog ab. Implementieren Sie die Prädikate

- "Vater von",
- "Vorfahre von",
- "weibliche Vorfahrin von".

Übung 2
Gegeben ist folgender Prolog-Code.

```
sum(0,X,X).
sum(s(X),Y,s(Z)) :- sum(X,Y,Z).

prod(0,_,0).
prod(s(X),Y,Z) :- prod(X,Y,W), sum(Y,W,Z).
```

Geben Sie eine SLD-Refutation für `?- prod(s(s(0)), s(0), X).` an.

Übung 3
Gegeben sei der Prolog-Code

```
prefix([],L) :- is_list(L).
prefix([A|AS], [A|BS]) :- prefix(AS,BS).
```

Dabei gilt das vordefinierte Prädikat `is_list(L)` genau dann wenn `L` eine Liste ist. Geben Sie alle möglichen SLD-Refutationen für `?- prefix([a|L], [a,c]).` an. Behandeln Sie dabei `is_list` ohne Zwischenschritte.

Übung 4 (AGS 16.18)

(a) Schreiben Sie ein C_0-Programm auf, das durch Anwendung der aus der Vorlesung bekannten Transformationsfunktion in das folgende H_0-Programm (auf die Typdefinition wurde hier verzichtet) überführt werden kann:

```
module Main where

f1     x1 x2 = f2 x1 1
f2     x1 x2 = if x1 > 0 then f21 x1 x2
                         else f3 x1 x2
f21    x1 x2 = f211 x1 x2
f211   x1 x2 = f212 x1 (x2*2)
f212   x1 x2 = f2 (x1-1) x2
f3     x1 x2 = x2

main = do x1 <- readLn
          print (f1 x1 0)
```

(b) Folgendes H_0-Programm sei gegeben:

```
module Main where

f :: Int -> Int -> Int
f x1 x2 = if x1 > 0  then f (x1-x2) (x2+1)
                     else g x2 (x2-1)

g :: Int -> Int -> Int
g x1 x2 = if x1 == x2 then x1
                      else x2
main = do x1 <- readLn
          print (f x1 x1)
```

Vervollständigen Sie die Angaben `/*A*/` bis `/*E*/` in der folgenden Übersetzung des H_0-Programms in ein äquivalentes C_0-Programm:

```
#include <stdio.h>

int main() {
  int x1, x2, function, flag, result;
  /*A*/
  while (flag == 1) {
    if (function == 1)
      if (/*B*/) {
        /*C*/
      } else {
        /*D*/
        function = 2;
      }
    else if (function == 2) {
      /*E*/
    }
  }
  printf("%d", result);
}
```

Zusatzaufgabe 1 (AGS 16.17)

(a) Transformieren Sie die folgende Funktion `h` eines H_0-Programmes in ein AM_0-Programm

mit baumstrukturierten Adressen, berechnen Sie also

$$\underline{functrans}(\texttt{h :: Int -> Int -> Int h x1 x2 = ...}).$$

Sie brauchen dabei keine Zwischenschritte anzugeben.

```
h :: Int -> Int -> Int
h x1 x2 = if x1 <= x2 then x1 else h x2 x1
```

(b) Das folgende H_0-Programm (ohne Funktionstypen) ist durch die Anwendung von trans auf ein C_0-Programm P entstanden:

```
module Main where

f1      x1 x2 x3 = if x2 > 0   then f11 x1 x2 x3
                               else f2 x1 x2 x3
f11     x1 x2 x3 = if x1 >= x2 then f111 x1 x2 x3
                               else f112 x1 x2 x3
f111    x1 x2 x3 = f1 (x1-x2) x2 x3
f112    x1 x2 x3 = f1121 x1 x2 x3
f1121   x1 x2 x3 = f1122 x1 x2 x1
f1122   x1 x2 x3 = f1123 x2 x2 x3
f1123   x1 x2 x3 = f1 x1 x3 x3
f2      x1 x2 x3 = x1

main = do x1 <- readLn
          x2 <- readLn
          print (f1 x1 x2 0)
```

Geben Sie das Programm P an.

Zusatzaufgabe 2

Es seien δ ein dreistelliges, σ ein zweistelliges, γ ein einstelliges und α ein nullstelliges Funktionssymbol. $V = \{x_1, x_2, x_3\}$ sei eine Menge von Variablen.

Wenden Sie den Unifikationsalgorithmus auf die Terme t_1 und t_2 an und ermitteln Sie deren allgemeinsten Unifikator:

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

Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an.

Zusatzaufgabe 3 (AGS 12.3.23)
Folgende Definitionen seien gegeben:

```
 1  data Tree a = Leaf | Branch a (Tree a) (Tree a)
 2
 3  breadth :: Tree a -> Int
 4  breadth Leaf           = 1
 5  breadth (Branch x l r) = breadth l + breadth r
 6
 7  length :: [a] -> Int
 8  length []     = 0
 9  length (x:xs) = 1 + length xs
10
11  toTree :: [a] -> Tree a
12  toTree []     = Leaf
13  toTree (x:xs) = Branch x (toTree xs) (toTree xs)
14
15  pow :: Int -> Int
16  pow 0 = 1
17  pow n = 2 * pow (n - 1)
```

Sei `a` ein beliebiger Typ. Die folgende Aussage soll mittels Induktion über `[a]` bewiesen werden:
(A) Für jede Liste `xs :: [a]` gilt:

```
breadth (toTree xs) = pow (length xs).
```

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

(a) Zeigen Sie den Induktionsanfang.
(b) Zeigen Sie den Induktionsschritt inklusive der Induktionsvoraussetzung.