Programmierung

12. Übungsblatt

Zeitraum: 04. – 08. Juli 2016

Hinweis: Die Vorlesung am 08. Juli fällt aus!

Übung 1 (AGS 16.12)

(a) Transformieren Sie ohne Zwischenschritte die folgende Funktion `h` eines H_0-Programmes in ein AM_0-Programm mit baumstrukturierten Adressen.

```
h :: Int -> Int
h x1 = if x1 > 1
       then if x1==2 then x1 else h (x1-1)
       else 0
```

(b) Gegeben sei der folgende Ausschnitt aus einem C_0-Programm:

```
x2 = x1 + 2;
while (x1 < 2)
  if (x2 <= x1) x2 = x2+1;
  else x1 = x2;
printf("%d", x1);
```

Durch Nutzung der Transformationsfunktion aus der Vorlesung/Übung ensteht daraus das folgende Fragment eines H_0-Programms (ohne Angabe der Funktionstypen):

```
f3     x1 x2 = A
f4     x1 x2 = if x1 < 2   then B
                           else C
f41    x1 x2 = if x2 <= x1 then D
                           else E
f411   x1 x2 = F
f412   x1 x2 = G
f5     x1 x2 = H
```

Vervollständigen Sie diesen Programmrumpf durch die Angabe der Ausdrücke für `A` bis `H`!

Übung 2 (AGS 16.15)

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

```
module Main where

f1    x1 x2 = if x1 > 0 then f11 x1 x2
                        else f2 x1 x2
f11   x1 x2 = f111 x1 x2
f111  x1 x2 = f112 (x1-1) x2
f112  x1 x2 = f1 x1 (x2*2)
f2    x1 x2 = x2
```

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

(b) Folgendes H_0-Programm sei gegeben:

```
module Main where

g :: Int -> Int -> Int
g x1 x2 = if x1 > 0 then g (x1-1) (x2*2)
                    else h x2 x2

h :: Int -> Int -> Int
h x1 x2 = x2

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

Vervollständigen Sie die Angaben <A> bis <E> in der folgenden Übersetzung des H_0-Programmes 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);

  return 0;
}
```

Zusatzaufgabe 1

(a) Schreiben Sie die Funktion `unzip :: [(a, b)] -> ([a], [b])`, welche eine Liste von Paaren in zwei Listen aufspaltet. Dabei soll die erste Liste die ersten Komponenten und die zweite Liste die zweiten Komponenten der Paare enthalten, z.B.:

```
unzip [('a', 1), ('b', 2), ('c', 3)] = (['a', 'b', 'c'], [1, 2, 3]).
```

Die Reihenfolge der Elemente soll erhalten bleiben.

(b) Gegeben seien folgende Funktionen.

```
map :: (a -> b) -> [a] -> [b]
map _ []       = []
map f (x : xs) = f x : map f xs

uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (x, y) = f x y
```

Werten Sie den Term

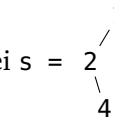
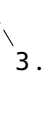
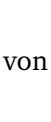
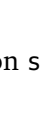
```
map (uncurry (+)) [(1, 2), (3, 4)]
```

schrittweise aus. Sie dürfen dabei sinnvolle Abkürzungen einführen.

(c) Gegeben sei folgender algebraischer Datentyp für binäre Bäume, die leer sein dürfen.

```
data Tree = Null | Node Int Tree Tree
```

Seien `p, t :: Tree`.
Ein *Teilbaum* von `t` ist entweder `t` oder Teilbaum eines Kindbaumes von `t`. Insbesondere ist `Null` Teilbaum von `t`.
Ein *Präfix* von `t` ist ein Baum, der durch Entfernen beliebig vieler Teilbäume aus `t` entsteht. Insbesondere sind `Null` und `t` Präfixe von `t`.
Ein *Pattern* von `t` ist ein Präfix eines Teilbaumes von `t`. Zum Beispiel:

Sei  . Pattern von `s` sind: `Null`,  ,  ,  ,  ,  , ,  ,  , s.

Implementieren Sie die Funktion `isPatternOf :: Tree -> Tree -> Bool`, sodass die Auswertung von `isPatternOf p t` genau dann `True` liefert, wenn `p` ein Pattern von `t` ist.

Zusatzaufgabe 2 (AGS 12.2.6)

(a) Es seien δ ein dreistelliges, σ ein zweistelliges und γ ein einstelliges 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:

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

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

(b) Geben Sie den von Ihnen bestimmten allgemeinsten Unifikator an.