Programmierung

13. Übungsblatt

Zeitraum: 11. – 15. Juli 2016

Übung 1 (AGS 12.3.16)

Folgende Definitionen seien gegeben:

```
1  data Tree = Node Tree Tree | Leaf Int
2
3  sumTree :: Tree -> Int
4  sumTree (Leaf x)     = x
5  sumTree (Node t1 t2) = (sumTree t1) + (sumTree t2)
6
7  prod :: [Int] -> Int
8  prod []              = 1
9  prod (a:r)           = a * (prod r)
10
11 toTree :: [Int] -> Int -> Tree
12 toTree [] x         = Leaf x
13 toTree (a:r) x      = Node (toTree r (2*a*x)) (toTree r (-a*x))
```

Die folgende Aussage soll mittels struktureller Induktion über Listen bewiesen werden: Für jede Liste `r :: [Int]` und für jedes `x :: Int` gilt

```
sumTree (toTree r x) = x * (prod r) .
```

(Hinweis: Zeigen Sie den Induktionsanfang, geben Sie die Induktionsvoraussetzung vollständig an und zeigen Sie den Induktionsschritt.)

Übung 2 (AGS 12.4.13)

(a) Gegeben sei folgender λ–Term:

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

Reduzieren Sie diesen Term solange, bis seine Normalform erreicht ist. Schreiben Sie – bevor Sie einen Ableitungsschritt ausführen – für die relevanten (Teil-)Ausdrücke die Mengen der freien bzw. der gebundenen Vorkommen von Variablen auf.

(b) Eine Funktion $g : \mathbb{N} \times \mathbb{N}^+ \to \mathbb{N}^+$ sei wie folgt definiert:

$$\begin{aligned} g(x,y) &= y && \text{für } x = 0 \\ g(x,y) &= 3 + g(x-1, y+2) && \text{wenn } x \text{ teilbar durch 2 und } x > 0 \\ g(x,y) &= 2 * g(x-1, y+1) && \text{sonst für } x > 0 \end{aligned}$$

Geben Sie zur Funktion g den zugehörigen λ-Term $\langle G \rangle$ an, so dass $\langle g \rangle = \langle Y \rangle \langle G \rangle$ gilt.

(c) Folgender λ–Term sei gegeben:

$$\begin{aligned} \langle F \rangle = (\lambda z x y.\langle ite \rangle(\langle iszero \rangle x)\ (\langle succ \rangle(\langle succ \rangle y)) \\ (\langle add \rangle\ y\ (\ z(\langle pred \rangle x)(\langle succ \rangle x)))) \end{aligned}$$

Berechnen Sie in sinnvollen Schritten $\langle Y \rangle \langle F \rangle \langle 1 \rangle \langle 5 \rangle$. Führen Sie im Rechenprozess zweckmäßige Abkürzungen der λ–Terme ein.

Übung 3 (AGS 14.14)

(a) Gegeben sei folgendes Fragment eines C_1-Programms:

```
#include <stdio.h>
int x, y;
void f(...)  {...}

void g(int *b) {
    int a;
    while (x > 0) {
        x = x - a;
        f(y, &a);
    }
    g(b);
}

void main()  {...}
```

Übersetzen Sie die Sequenz der Statements im Rumpf von g in entsprechenden AM_1-Code mit baumstrukturierten Adressen (mittels *stseqtrans*). Sie brauchen keine Zwischenschritte anzugeben. Geben Sie zunächst die dazu benötigte Symboltabelle $tab_{g+lDecl}$ an.

(b) Gegeben sei folgender AM_1-Code:

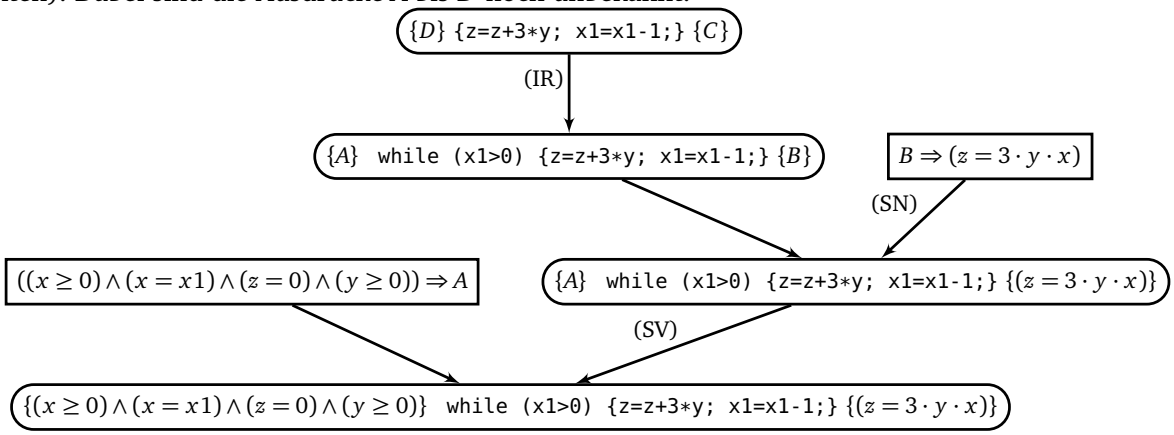
```
01: INIT 1;              08: STORE(global,1);     15: LOADA(lokal,1);
02: CALL 10;             09: RET 2;               16: PUSH;
03: JMP 0;               10: INIT 2;              17: CALL 4;
04: INIT 0;              11: READ(lokal,2);       18: WRITE(global,1);
05: LOADI(-2);           12: READ(lokal,1);       19: RET 0;
06: LOAD(lokal,-3);      13: LOAD(lokal,2);
07: ADD;                 14: PUSH;
```

Die AM_1 habe bereits die Konfiguration $(13, \varepsilon, 0 : 3 : 0 : 9 : 7, 3, \varepsilon, \varepsilon)$. Führen Sie die Berechnung weiter bis das Programm terminiert.

Übung 4 (AGS 15.3)
Für die Verifikationsformel

$$\{(x \geq 0) \wedge (x = x1) \wedge (z = 0) \wedge (y \geq 0)\}\texttt{while (x1>0) \{z=z+3*y; x1=x1-1;\}}\{(z = 3 \cdot y \cdot x)\}$$

wurden die ersten drei (korrekten) Regelanwendungen des Beweisbaumes aufgeschrieben (siehe unten). Dabei sind die Ausdrücke A bis D noch unbekannt.



(a) Geben Sie die Schleifeninvariante an.
(b) Geben Sie die Ausdrücke für A, B, C, D an.

Zusatzaufgabe 1 (AGS 12.1.17)

(a) Schreiben Sie in Haskell eine Funktion `expo` einschließlich Funktionstypdefinition zur Berechnung von x^y mit $x, y \in \mathbb{N}$. Den vordefinierten Operator `^` dürfen Sie hierbei nicht nutzen. Geben Sie eine schrittweise Auswertung für die Berechnung von 2^2 an.

(b) Gegeben sei die Datentypdefinition

```
data Tree a = Branch (Tree a) (Tree a) | Leaf a
```

Schreiben Sie in Haskell eine Funktion `check` einschließlich der Funktionstypdefinition, die für einen beliebigen Binärbaum des oben genannten Typs testet, ob mindestens ein Pfad von der Wurzel zu einem Blatt genau eine vorgegebene Länge `k` hat. Existiert ein solcher Pfad, so soll die Funktion den Wert `True` liefern, sonst `False`.
Hinweis: Die Pfadlänge (oder Tiefe) eines Knotens n ist die Anzahl der Kanten auf dem Weg vom Wurzelknoten zum Knoten n.

(c) Schreiben Sie in Haskell eine Funktion `test` einschließlich der Funktionstypdefinition, die für eine beliebige Liste `x` von Integer–Werten testet, ob alle darin enthaltenen Werte jeweils nur einmal vorkommen. Standardfunktionen sind nicht erlaubt. Die Liste `x` darf nicht verändert werden.

Zusatzaufgabe 2 (AGS 12.2.10)

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

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

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 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!