Programmierung

11. Übungsblatt

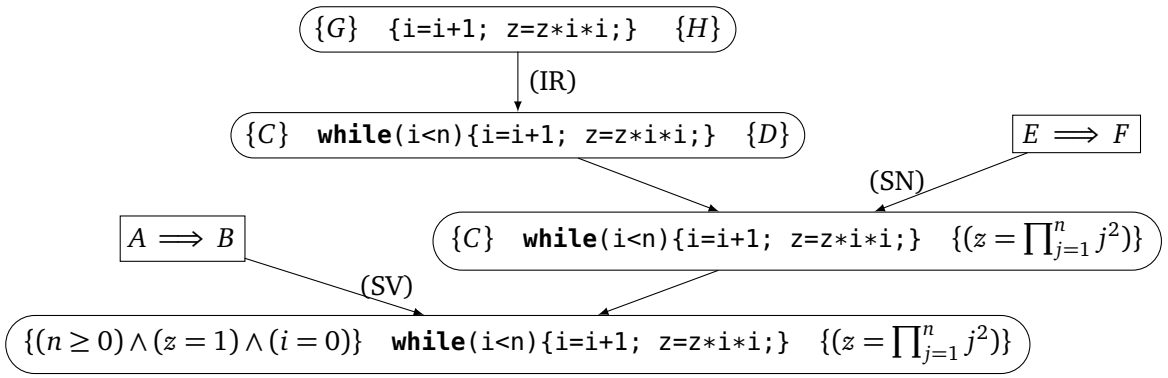
Zeitraum: 26. – 30. Juni 2017

Übung 1 (AGS 15.24 ⋆)
Die Verifikationsformel

$$\{(n \geq 0) \wedge (z = 1) \wedge (i = 0)\}\ \texttt{while(i<n)\{i=i+1; z=z*i*i;\}}\ \{(z = \textstyle\prod_{j=1}^{n} j^2)\}$$

soll mit dem Hoare-Kalkül bewiesen werden.

Ein Teil eines Beweisbaums wurde unten bereits aufgeschrieben, die Ausdrücke A bis H sind jedoch noch unbekannt. Es gelten die Abkürzungen SV = stärkere Vorbedingung, SN = schwächere Nachbedingung und IR = Iterationsregel.



(a) Geben Sie eine geeignete Schleifeninvariante an.
(b) Geben Sie die Ausdrücke A bis H an. Sie können dabei die Schleifeninvariante mit *SI* abkürzen.

Übung 2 (AGS 16.7)
Gegeben sei folgendes C_0–Programm *Fac*:

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

int main() {
  int x1, x2;
  scanf("%i", &x1);
  x2 = 1;
  while (x1 > 0) {
    x2 = x2 * x1;
    x1 = x1 - 1;
  }
  printf("%d", x2);
  return 0;
}
```

Übersetzen Sie anhand der Definition aus der Vorlesung dieses C_0-Programm in ein H_0-Programm. Sie müssen keine Zwischenschritte angeben!

Übung 3 (AGS 16.25)

(a) Transformieren Sie die folgende Funktion `h` eines H_0-Programmes in ein AM_0-Programm mit baumstrukturierten Adressen, berechnen Sie also das Ergebnis *funtrans*(`h :: Int -> ...`). Geben Sie dabei keine Zwischenschritte an.

```
h :: Int -> Int -> Int -> Int
h x1 x2 x3 = if x3 > x1
             then (x2 - 1)
             else h x2 (x1 - x3) x2
```

(b) Folgendes H_0-Programm sei gegeben:

```
module Main where

h :: Int -> Int -> Int
h x1 x2 = if x2 == x1 then 30
                      else x2

g :: Int -> Int -> Int
g x1 x2 = if 10 <= x2 then g (x1-x2) (x2-1)
                      else h (x1+x2) 10

main = do x1 <- readLn
          print (g (3+x1) 5)
```

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

```
#include <stdio.h>

int main() {
  int x1, x2, function = 2, flag, result;
  /*A*/
  while (flag == 1) {
    if (function == 1)
      if (/*B*/) {
        /*C*/
      }
      else {
        /*D*/
      }
    else if (/*E*/) {
      /*F*/
    }
  }
  printf("%d", result);
  return 0;
}
```

Zusatzaufgabe 1 (AGS 16.28)

(a) Transformieren Sie die folgende Funktion `f` eines H_0-Programms in ein AM_0-Programm. Sie können dabei die Adressen frei vergeben, sie müssen nicht baumstrukturiert sein. Geben Sie keine Zwischenschritte an.

```
f :: Int -> Int -> Int
f x1 x2 = if x1 <= 0 then x2
                     else f (x1 / 2) (x2 + x1)
```

(b) Das folgende H_0-Programmstück ist durch Nutzung der Transformationsfunktion aus der Vorlesung aus Statements eines entsprechenden C_0-Programms entstanden. Geben Sie diese Statements an.

```
f1    x1 x2 = if x1 > 0 then f11 x1 x2
                        else f2  x1 x2
f11   x1 x2 = f111 x1 x2
f111  x1 x2 = if x1 `mod` 2 == 0 then f1111 x1 x2
                                 else f112  x1 x2
f1111 x1 x2 = f112 x1 (x2 + x1)
f112  x1 x2 = f1 (x1 - 1) x2
```

Zusatzaufgabe 2 (AGS 12.3.8 ⋆)

Folgender Code sei gegeben.

```
data Tree = Leaf Int | Branch Int Tree Tree

sumTree :: Tree -> Int
sumTree (Leaf i) = i
sumTree (Branch i t1 t2) = i + sumTree t1 + sumTree t2

revTree :: Tree -> Tree
revTree (Leaf i) = Leaf i
revTree (Branch i t1 t2) = Branch i (revTree t2) (revTree t1)

overlay :: Tree -> Tree -> Tree
overlay (Leaf i1) (Leaf i2) = Leaf (i1+i2)
overlay (Branch i1 t11 t12) (Branch i2 t21 t22) =
    Branch (i1+i2) (overlay t11 t21) (overlay t12 t22)
```

Zeigen Sie unter Verwendung der oben aufgeführten Definitionen mit Hilfe der strukturellen Induktion die Gültigkeit der Gleichung

`sumTree (overlay t t) = 2 * (sumTree (revTree t))` für jedes `t :: Tree`.

Geben Sie bei Umformungen die jeweils benutzten Gesetzmäßigkeiten / Definitionen an.

Zusatzaufgabe 3 (AGS 12.4.35)

(a) Berechnen Sie die Normalform des untenstehenden λ-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.

$$\big(\lambda x\, z.\ (\lambda y.y)\ x\ z\ z\big)(\lambda x\ y.y\ z)$$

(b) Gegeben sei der λ-Term

$$\langle F\rangle = \Big(\lambda f\, x\, y.\langle \text{ite}\rangle\ \ \big(\langle \text{iszero}\rangle\ x\big)\ \ y\quad \big(\langle \text{add}\rangle\ x\ \big(f\ (\langle \text{pred}\rangle\ x)\ (\langle \text{add}\rangle\ x\ y)\big)\big)\Big).$$

Geben Sie eine Haskell-Funktion `f` an, so dass `f` $= \langle Y\rangle\langle F\rangle$ gilt!

(c) Betrachten Sie den λ-Term $\langle F\rangle$ aus Teilaufgabe **(b)**. Berechnen Sie die Normalform des Terms $\langle Y\rangle\langle F\rangle\langle 1\rangle\langle 4\rangle$.