# Programmierung

## 10. Übungsblatt

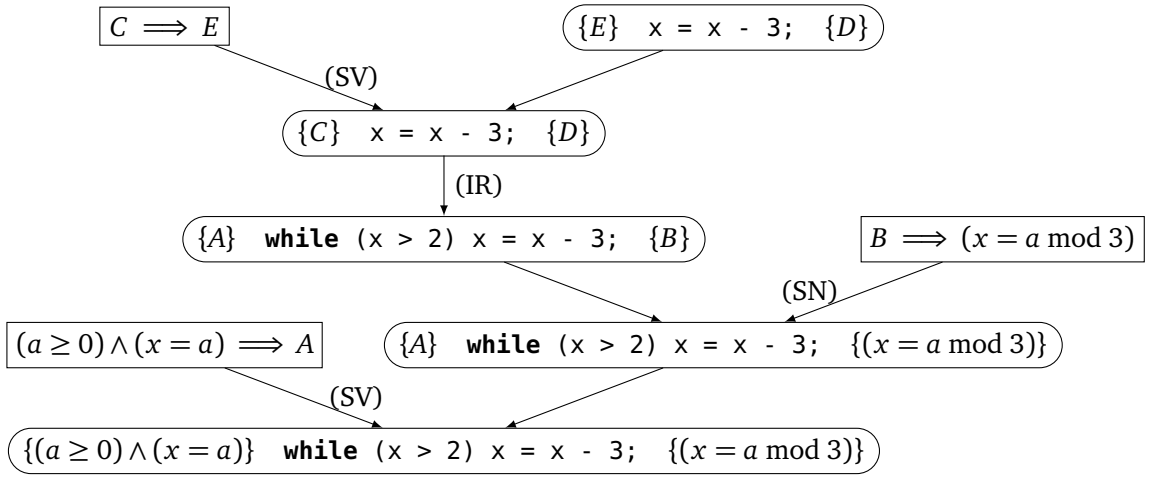
Zeitraum: 20. – 24. Juni 2016

**Übung 1 (AGS 15.23)**
Die Verifikationsformel

$$\{(a \geq 0) \wedge (x = a)\} \quad \texttt{while (x > 2) x = x - 3;} \quad \{(x = a \bmod 3)\}$$

soll mit dem Hoare-Kalkül bewiesen werden, wobei die Operation „mod“ den Rest bei ganzzahliger Division bildet, z. B. $2 \bmod 3 = 2$ und $5 \bmod 3 = 2$.

Der Beweisbaum wurde unten bereits aufgeschrieben, die Ausdrücke $A$ bis $E$ sind jedoch noch unbekannt.

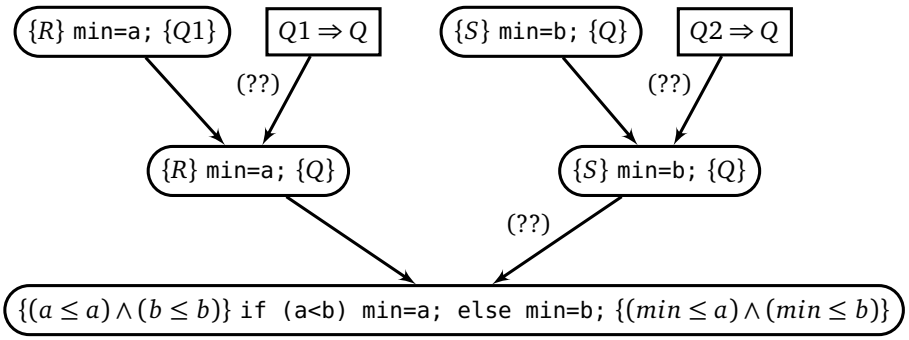



(a) Geben Sie eine geeignete Schleifeninvariante an.
(b) Geben Sie die Ausdrücke $A$, $B$, $C$, $D$, und $E$ an. Sie können dabei die Schleifeninvariante mit $SI$ abkürzen.

**Übung 2 (AGS 15.12 ⋆)**
Mit Hilfe des Hoare–Kalküls wurde für die Verifikationsformel

$$\{(a \leq a) \wedge (b \leq b)\}\ \texttt{if (a<b) min=a; else min=b;}\ \{(min \leq a) \wedge (min \leq b)\}$$

der folgende korrekte Beweisbaum aufgestellt:



Geben Sie für $R$, $S$, $Q$, $Q1$, $Q2$ die konkreten Zusicherungen an. Des Weiteren nennen Sie alle angewendeten Verifikationsregeln und geben Sie explizit die an den Blättern des Beweisbaumes auftretenden Instanzen des Zuweisungsaxioms an.

**Übung 3 (AGS 14.11)**

(a) Folgendes Fragment eines $C_1$-Programms sei bekannt:

```
#include <stdio.h>

int x;

void h(...) {...}

void g(...) {...}

void f(int a; int *b)
{
int c;
if (x>1) g(b);
else h(a,&x);
c=*b + 1;
}

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

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

(b) Gegeben sei folgender $AM_1$-Code:

```
1: INIT 1;          6:  LOADI(-2);        11: READ(lokal,1);    16: PUSH;
2: CALL 10;         7:  ADD;              12: READ(global,1);   17: CALL 4;
3: JMP 0;           8:  STORE(global,1);  13: LOAD(lokal,1)     18: WRITE(global,1);
4: INIT 1;          9:  RET 2;            14: PUSH;             19: RET 0;
5: LOAD(lokal,-3);  10: INIT 1;           15: LOADA(global,1);
```

Betrachten Sie nun die $AM_1$, die sich bereits im Zustand (Konfiguration)

$$\sigma = (12,\ \varepsilon,\ 0:3:0:7,\ 3,\ 5,\ \varepsilon)$$

befindet. Lassen Sie die $AM_1$, beginnend mit $\sigma$, auf dem oben gegebenen $AM_1$-Code solange ablaufen, bis die Maschine stoppt. Dokumentieren Sie den Zustand der $AM_1$ nach Ausführung jedes Befehls.

**Zusatzaufgabe 1 (AGS 14.10)**

(a) Folgendes Fragment eines $C_1$-Programms sei bekannt:

```
#include <stdio.h>

int a, b;

void h(...)  {...}

void g(int *x)
{ int y;
  while (b != 1) {b = *x - 1; h(&y);}
  f(y,x);
}

void f(...)  {...}

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:

```
 1:  INIT 1;      6:  STORE(lokal,2);   11:  RET 1;           16:  CALL 4;
 2:  CALL 12;     7:  LOAD(global,1);   12:  INIT 1;          17:  WRITE(global,1);
 3:  JMP 0;       8:  STORE(lokal,1);   13:  READ(global,1)   18:  RET 0;
 4:  INIT 2;      9:  LIT 5;            14:  LOADA(global,1)
 5:  LOADI(-2);   10:  STORE(global,1);  15:  PUSH;
```

Betrachten Sie nun die $AM_1$, die sich bereits im Zustand (Konfiguration)

$$\sigma = (12,\ \varepsilon,\ 0:3:0,\ 3,\ 9,\ \varepsilon)$$

befindet. Lassen Sie die $AM_1$, beginnend mit $\sigma$, auf dem oben gegebenen $AM_1$-Code solange ablaufen, bis die Maschine stoppt. Dokumentieren Sie den Zustand der $AM_1$ nach Ausführung jedes Befehls.

**Zusatzaufgabe 2 (AGS 12.2.11)**

(a) Wenden Sie den Unifikationsalgorithmus auf die Terme

$$t_1 = \tau\big(\sigma(\alpha, x_1), \sigma(x_4, \gamma(\tau(x_1, x_4)))\big) \qquad \text{und} \qquad t_2 = \tau\big(\sigma(\alpha, \alpha), \sigma(\gamma(x_3), \gamma(x_2))\big)$$

an und bestimmen Sie den allgemeinsten Unifikator!

(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!