13. Übung (8.–12. Juli 2019)

# Programmierung

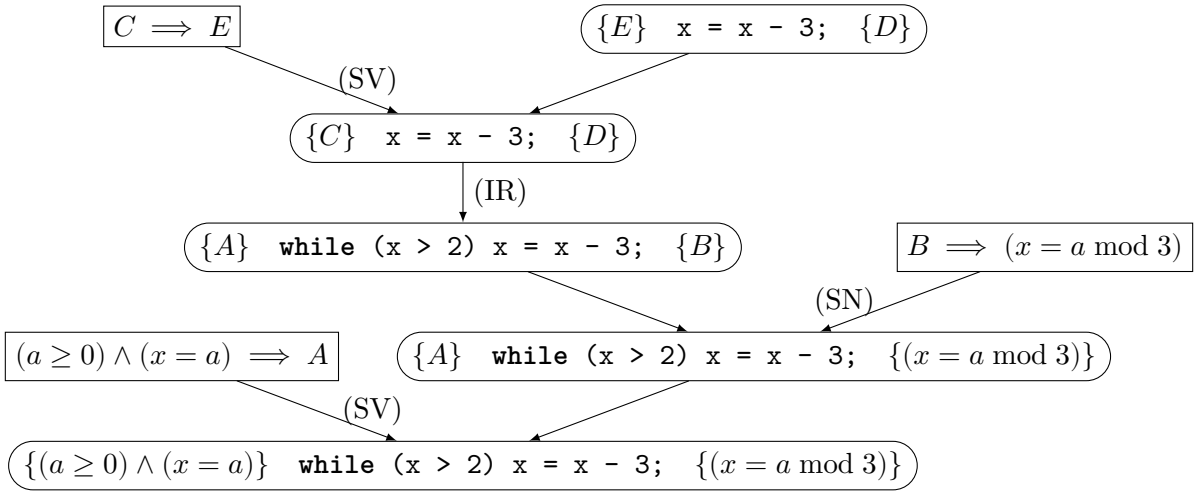
*Bitte beachten Sie die Klausurtermine und Raumzuordnungen auf der Lehrveranstaltungswebseite. Am 24. Juli um 15:00–17:00 wird ein Lernraum zur Vorbereitung der Klausur angeboten.*

**Aufgabe 1 (AGS 16.23)**

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.

**Zusatzaufgabe 1 (AGS 15.11)**

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

```
1  #include <stdio.h>
2
3  int x;
4
5  void h(...) {...}
6  void g(...) {...}
7
8  void f(int a, int *b) {
9    int c;
10   if (x>1) g(b);
11     else h(a,&x);
12   c=*b + 1;
13 }
14 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 $tab_{\texttt{f}}$ an.

(b) Lassen Sie die $AM_1$, beginnend mit $(12, \varepsilon, 0{:}3{:}0{:}7, 3, 5, \varepsilon)$, auf dem unten gegebenen $AM_1$-Code solange ablaufen, bis die Maschine terminiert.

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

**Zusatzaufgabe 2 (AGS 12.1.52)**

(a) Schreiben sie eine Funktion `avg :: [Float] -> Float` welche den Durchschnitt einer Liste von `Float`s berechnet. Dabei soll die Liste *höchstens einmal durchlaufen* werden. Für die leere Liste soll die Funktion `0` zurückgeben.

(b) Geben Sie eine Funktion `partition :: (a -> Bool) -> [a] -> ([a], [a])` an, welche ein Prädikat `p :: a -> Bool` und eine Liste `xs :: [a]` als Argumente hat. Das Ergebnis ist ein Paar von Listen `(ys, zs)`, so dass `ys` jedes Element aus `xs` enthält, welches das Prädikat `p` erfüllt (d.h. `p y == True` für jedes `y` in `ys`), während `zs` jedes Element aus `xs` enthält, welches das Prädikat `p` nicht erfüllt. Die Reihenfolgen der Elemente in `ys` und `zs` gleichen der in `xs`. Beispielsweise soll gelten:
`partition even [1,2,3,4,5,6,7,8,9,10] == ([2,4,6,8,10],[1,3,5,7,9])`

(c) Sei `xs` eine Liste vom Typ `[Int]`. Wir sagen, dass `xs` eine *k-Wiederholung* enthält, falls sich `xs` in Listen `as`, `bs` und `cs` zerlegen lässt, so dass

- `xs == as ++ bs ++ cs` gilt,
- alle Elemente in `bs` gleich sind, und
- die Liste `bs` die Länge $k$ hat.

Schreiben Sie eine Funktion `maxrep :: [Int] -> Int` die das größte $k$ bestimmt, so dass die gegebene Liste eine $k$-Wiederholung enthält.

**Zusatzaufgabe 3 (AGS 12.2.15)**

Gegeben seien die Terme

$$t_1 = \sigma(\alpha,\ \sigma(\gamma(\alpha),\ \sigma(x_2, x_3\quad\quad))) \quad \text{und}$$
$$t_2 = \sigma(\alpha,\ \sigma(x_1\quad,\ \sigma(x_2, \sigma(x_2, x_1))))$$

über dem Rangalphabet $\Sigma = \{\sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}\}$. Wenden Sie den Unifikationsalgorithmus auf die Terme $t_1$ und $t_2$ an. Wenden Sie bei jedem Umformungsschritt nur eine Regelsorte an und geben Sie diese jeweils an. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator an.

**Zusatzaufgabe 4 (AGS 12.3.26)**
Folgende Definitionen seien gegeben:

```
1 mul :: Int -> [Int] -> [Int]
2 mul a []     = []
3 mul a (x:xs) = (a * x) : mul a xs
```

Zeigen Sie mittels struktureller Induktion über listen, dass für jede Liste `ys :: [Int]` gilt:

> Für alle `a, b :: Int` gilt `mul b (mul a ys)` = `mul (b * a) ys`.

Zeigen Sie dazu den Induktionsanfang und den Induktionsschritt; geben Sie beim Induktionsschritt die Induktionsvoraussetzung an. Geben Sie bei jeder Umformung die benutzte *Definition* bzw. die *Induktionsvoraussetzung* an. Quantifizieren Sie alle Variablen.

**Zusatzaufgabe 5 (AGS 12.4.33)**

(a) Berechnen Sie die Normalform des $\lambda$-Terms $(\lambda fx.x(f(fx)))(\lambda y.xy)$, indem Sie ihn *schrittweise* reduzieren.

(b) Gegeben sei der $\lambda$-Term $\langle F \rangle$:

$$\Big(\lambda fxy.\langle\text{ite}\rangle\ (\langle\text{iszero}\rangle\, y)\ (\langle\text{mult}\rangle\, x\, \langle 2\rangle)\ \Big(\langle\text{ite}\rangle\ (\langle\text{iszero}\rangle\, (\ \ \text{mod}\ \ y\, \langle 2\rangle))\ (f\, x\, (\langle\text{pred}\rangle\, y))\ (f\, (\langle\text{mult}\rangle\, x\, \langle 2\rangle)\, (\langle\text{pred}\rangle\, y))\Big)\Big)$$

Geben Sie eine Haskell-Funktion `f` an, so dass $\langle$`f x y`$\rangle = \langle Y\rangle\langle F\rangle\langle$`x`$\rangle\langle$`y`$\rangle$ für alle `x`, `y` $\in \mathbb{N}$ gilt!

(c) Betrachten Sie den $\lambda$-Term $\langle F \rangle$ aus Teilaufgabe 1 (b). Berechnen Sie die Normalform des Terms $\langle Y\rangle\langle F\rangle\langle 2\rangle\langle 1\rangle$. Schreiben Sie für jeden Aufruf von $\langle F \rangle$ jeweils **zwei Zeilen**: eine in der Sie die Werte der Parameter des Aufrufs protokollieren, und eine in der Sie ihre Auswertung skizzieren.

Stellen Sie zuerst die Wirkungsweise des Fixpunktkombinators $\langle Y \rangle$ in einer Nebenrechnung dar. Führen Sie zweckmäßige Abkürzungen der $\lambda$-Terme ein. Die spitzen Klammern $\langle\cdots\rangle$ dürfen Sie weglassen. Sie können die in der AGS gegebenen Terme und Beziehungen benutzen.

**Zusatzaufgabe 6 (AGS 13.3 a)**
Wir betrachten Binärbäume in Prolog$^-$ als Terme der Form `nil` oder `tree(X, L, R)`, wobei `X` für ein Symbol steht und `L` und `R` Binärbäume sind. Das Funktionssymbol `nil` steht für den leeren Baum. Gegeben seien die zwei Bäume

```
<t1> = tree(a, tree(b, nil, nil), tree(v, nil, nil))
```
und
```
<t2> = tree(c, nil, tree(d, nil, nil))
```

und das folgende Prolog$^-$-Programm:

```
1 istree(nil).
2 istree(tree(_, L, R)) :- istree(L), istree(R).
3
4 insert(nil, _, nil).
5 insert(tree(v, _, _), T,                T) :- istree(T).
6 insert(tree(X, L, R), T, tree(X, LT, RT)) :- insert(L, T, LT),
      insert(R, T, RT).
```

Bestimmen Sie durch SLD-Refutation für das Goal `?- insert(<t1>, <t2>, X).` eine Belegung der Variablen `X`.