14. Übung (13.–17. Juli 2020)

Programmierung

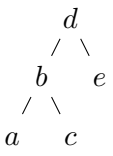
Hinweis In dieser Übung werden nur Aufgaben gestellt, die bisherige Themen wiederholen. Anstelle der Besprechung wird am 17. Juli um 9:20 Uhr ein Repetitorium angeboten, in dem Fragen zu vergangenen Themen ausführlich beantwortet bzw. Themen wiederholt besprochen werden können.

Aufgabe 1 (AGS 12.1.60)

Gegeben sei ein algebraischer Datentyp für Binärbaume:

```
data BinTree = Branch Char BinTree BinTree | Leaf Char
```

(a) Eine *Position* in einem Binärbaum beschreibt den Pfad von der Wurzel zu einem Knoten des Baumes als Liste von Einsen und Zweien. Dabei bedeutet eine `1`, dass sich der Knoten im linken Teilbaum befindet, und eine `2`, dass sich der Knoten im rechten Teilbaum befindet. Die Position `[]` zeigt auf den Wurzelknoten.



Im oben gezeigten Binärbaum steht z.B. d an Position `[]`, b an Position `[1]`, e an Position `[2]` und c an Position `[1,2]`.

Schreiben Sie eine Funktion `labelAt :: BinTree -> [Int] -> Char`, welche den `Char` an der gegebenen Position des gegebenen Binärbaumes ausgibt. Sie können davon ausgehen, dass die gegebene Liste eine valide Position im Baum ist.

(b) Ein Binärbaum `s` ist *Präfix* eines Binärbaumes `t` wenn folgende zwei Bedingungen gelten:

- Die Wurzelbeschriftungen von `s` und `t` stimmen überein.
- Falls `s` kein Blatt ist, dann ist auch `t` kein Blatt und
  - der linke Teilbaum von `s` ist Präfix des linken Teilbaumes von `t` und
  - der rechte Teilbaum von `s` ist Präfix des rechten Teilbaumes von `t`.

Schreiben Sie eine Funktion `isPrefixOf :: BinTree -> BinTree -> Bool`, die genau dann `True` zurückgibt, wenn der erste Baum Präfix des zweiten Baumes ist.

(c) Alternativ zur Definition in Aufgabe 1 (a) kann man Positionen wie folgt definieren: Sei `t :: BinTree`. Eine Liste von positiven natürlichen Zahlen ist eine *Position in* `t`,

- wenn sie gleich `[]` ist, oder
- wenn sie von der Form `(i : is)` ist (für ein `i` $\in \{1, 2\}$) und `is` eine Position im `i`-ten Teilbaum von `t` ist.

Geben Sie eine Funktion `positions :: BinTree -> [[Int]]` an, die eine Liste aller Positionen des gegebenen Baumes zurückgibt.

Aufgabe 2 (AGS 12.2.15)
Gegeben seien die Terme $t_1 = \sigma(\alpha,\ \sigma(\gamma(\alpha),\ \sigma(x_2, x_3)))$ und $t_2 = \sigma(\alpha,\ \sigma(x_1,\ \sigma(x_2, \sigma(x_2, x_1))))$ 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.

Aufgabe 3 (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.

Aufgabe 4 (AGS 12.4.33)
(a) Berechnen Sie die Normalform des λ-Terms $(\lambda f x.x(f(fx)))(\lambda y.xy)$, indem Sie ihn *schrittweise* reduzieren.

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

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

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

(c) Betrachten Sie den λ-Term $\langle F \rangle$ aus Teilaufgabe 4 (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 λ-Terme ein.

Aufgabe 5 (AGS 13.15)
Ein *Binärbaum* in Prolog$^-$ ist entweder von der Form `nil` oder von der Form `tree(X, L, R)`, wobei `X` eine natürliche Zahl, `L` ein Binärbaum und `R` ein Binärbaum sind. Die *Menge der Schlüsselwerte* eines Binärbaums `T` ist leer, falls `T` von der Form `nil` ist, und, falls `T` von der Form `tree(X, L, R)` ist, dann umfasst sie genau `X` sowie alle Schlüsselwerte von `L` und `R`.
Natürliche Zahlen stellen wir in Prolog$^-$ als Terme über dem einstelligen Funktionssymbol `s` und dem nullstelligen Funktionssymbol `0` dar:

```
1 nat(0).
2 nat(s(X)) :- nat(X).
```

Dabei kürzen wir wie in der Vorlesung den Term für die natürliche Zahl `n` mit `<n>` ab, z.B. `s(s(s(0)))` = `<3>`.

(a) Ein Binärbaum `T` hat die *Suchbaumeigenschaft*, wenn `T` von der Form `nil` ist oder wenn `T` von der Form `tree(X, L, R)` ist und `X` größer ist als jeder Schlüsselwert von `L`, `X` kleiner ist als jeder Schlüsselwert von `R` und sowohl `L` als auch `R` die Suchbaumeigenschaft haben. Programmieren Sie in Prolog$^-$ eine einstellige Relation `search_tree`, die genau die Binärbäume enthält, für die die Suchbaumeigenschaft gilt.

Hinweise:

- Verwenden Sie die zweistellige Relation `lt` (*echt kleiner als*):

```
3 lt(0, s(X)) :- nat(X).
4 lt(s(X), s(Y)) :- lt(X, Y).
```

- Programmieren Sie die zweistelligen Hilfsrelationen `all_less` und `all_greater`, die alle Paare `(T, N)` von Binärbäumen `T` und natürlichen Zahlen `N` enthalten, sodass alle Schlüsselwerte in `T` echt kleiner (bzw. echt größer) als `N` sind.

(b) Zur Bestimmung der Höhe eines Binärbaums sei folgendes Prolog$^-$-Programm gegeben:

```
5  max(X, X, X) :- nat(X).
6  max(X, Y, X) :- lt(Y, X).
7  max(X, Y, Y) :- lt(X, Y).
8  height(nil, 0).
9  height(tree(T, L, R), s(H))
10   :- height(L, HL), height(R, HR), max(HL, HR, H).
```

Bestimmen Sie für das Goal `?- height(tree(<7>, nil, nil), X).` eine Belegung der Prolog-Variablen `X` mittels SLD-Refutation. Notieren Sie in jedem Schritt die verwendete Programmzeile. Mehrere Resolutionsschritte unter Anwendung derselben Programmzeile können Sie mit `?-*` zusammenfassen.

Geben Sie abschließend die auf diese Art ermittelte Belegung der Variablen `X` an.

Aufgabe 6 (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(){...}
```

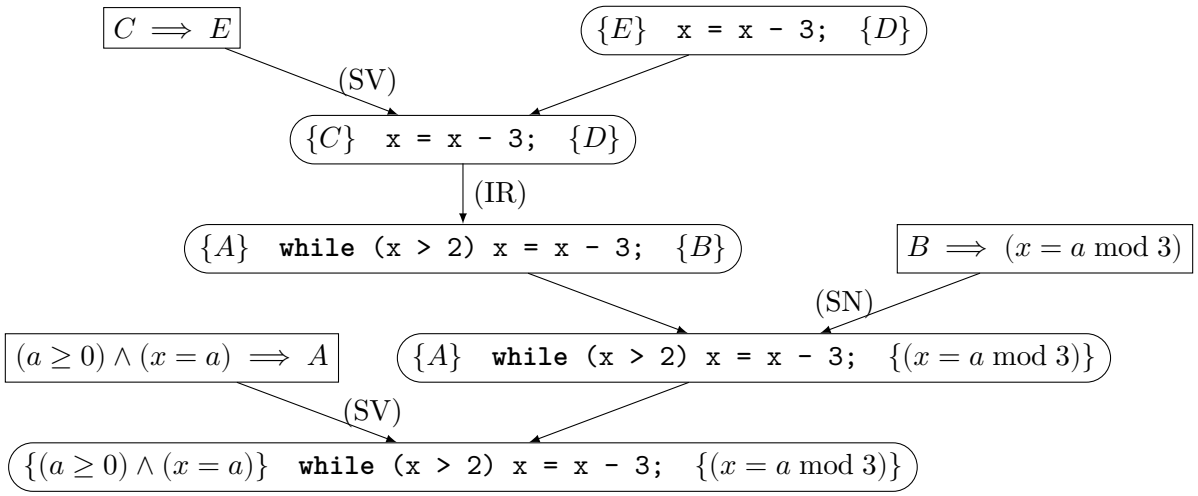
Übersetzen Sie die Sequenz der Statements im Rumpf von `f` in entsprechenden AM_1-Code mit baumstrukturierten Adressen (mittels *stseqtrans*). Sie brauchen keine Zwischenschritte anzugeben. Geben Sie zunächst die dazu benötigte Symboltabelle $tab_{\texttt{f}}$ an.

(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;
```

Aufgabe 7 (AGS 16.23)

Die Verifikationsformel $\{(a \geq 0) \wedge (x = a)\}$ `while (x > 2) x = x - 3;` $\{(x = a \bmod 3)\}$ 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.

Aufgabe 8 (AGS 17.29 b)

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
f x1 x2 = if x1 > x2 then x2
          else if x1 == 0 then f (x2 `div` 2) x2
               else x1
```