14. Übung (19.–23. Juli 2021)

Programmierung

Aufgabe 1 (AGS 12.1.63 ⋆)

(a) Geben Sie eine Funktion `retrieve :: [a] -> Int -> a` an welche, gegeben eine Liste `xs` und eine *positive* Zahl `n`, das `n`-te Element von `xs` zurückgibt. Falls `n` größer als die Länge von `xs` ist, so ist das Verhalten von `retrieve` nicht definiert.

(b) Gegeben sei der algebraische Datentyp `data CV = C Char | V Int` zur typsicheren Darstellung von Symbolen (`C c`) und Variablen (`V n`). Geben Sie eine Funktion

```
substitute ::  [[Char]] -> [CV] -> [Char]
```

mit zwei Argumenten `sub :: [[Char]]` beziehungsweise `xs :: [CV]` und folgender Funktionalität an. Die Liste `xs` wird von links nach rechts durchlaufen und synchron eine Ausgabeliste wie folgt erzeugt:

- Für jedes Vorkommen eines Symbols `C c` in `xs` wird der Ausgabeliste der Wert `c` angehängt.
- Für jedes Vorkommen einer Variable `V n` in `xs` wird der Ausgabeliste das `n`-te Element von `sub` angehängt.

Gehen Sie davon aus, dass die Liste `sub` ausreichend lang ist. Beispielsweise soll gelten:

```
substitute [['f'], ['c', 'd']] [C 'a', C 'b', V 2, C 'e', V 1]
    == ['a', 'b', 'c', 'd', 'e', 'f']
```

Hinweis: Nutzen Sie die Funktion `retrieve` *aus Teilaufgabe* **(a)**.

Für Teilaufgaben **(c)** und **(d)** sei die folgende Typdefinition zum Aufbau eines Baumes, bei dem jeder Knoten eine beliebige Anzahl an Kindbäumen haben kann, gegeben:

```
data Tree = Node [Tree]
```

Dabei ist das einzige Argument von `Node` eine Liste, welche die *Kindbäume* des Knotens in der Reihenfolge von links nach rechts enthält.

(c) Schreiben Sie in Haskell eine Funktion `maxrank` einschließlich der Typedefinition, die zu einem Baum vom Typ `Tree` den maximalen Rang aller in ihm vorkommenden Knoten bestimmt, wobei der Rang eines Knotens die Anzahl seiner Kindbäume ist.

Hinweis: Benutzen Sie die Funktionen `length` *und* `max`, *welche die Länge einer Liste bzw. das Maximum zweier Werte vom Typ* `Int` *berechnen.*

(d) Schreiben Sie in Haskell eine Funktion `equal` einschließlich der Typdefinition, die zwei Bäume vom Typ `Tree` auf Gleichheit prüft und `True` zurückgibt, falls die Bäume gleich sind, und ansonsten `False` zurückgibt.

Aufgabe 2 (AGS 12.2.17 ⋆)
Sei $\Sigma = \{\delta^{(3)}, \sigma^{(2)}, \gamma^{(1)}, \alpha^{(0)}, \beta^{(0)}\}$ und die folgenden Terme über dem Rangalphabet Σ:

$$t_1 = \delta(\gamma(\sigma(\alpha, x_1)), \beta, x_3) \qquad \text{und}$$
$$t_2 = \delta(\gamma(x_3), \beta, \sigma(x_2, x_1)).$$

(a) 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. Die erste Regelanwendung wurde bereits eingeleitet. Geben Sie anschließend den von Ihnen bestimmten allgemeinsten Unifikator und **zwei weitere Unifikatoren** an.

(b) Geben Sie zwei Paare von Termen über dem Rangalphabet Σ an, sodass für jedes dieser Paare der Occur-Check im Unifikationsalgorithmus fehlschlägt.

Aufgabe 3 (AGS 12.3.33 ⋆)
Folgende Definitionen seien gegeben:

```
1  data BinTree = Node Int BinTree BinTree | Nil Int
2
3  foo :: [Int] -> BinTree
4  foo [] = Nil (-1)
5  foo (x:xs) = Node x (foo xs) (bar (foo xs))
6
7  bar :: BinTree -> BinTree
8  bar t = Node 0 t t
9
10 treesum :: BinTree -> Int
11 treesum (Nil x) = x
12 treesum (Node x l r) = treesum l + x + treesum r
13
14 trip :: [Int] -> Int
15 trip [] = 0
16 trip (x:xs) = x + 3 * trip xs
17
18 length :: [Int] -> Int
19 length [] = 0
20 length (x:xs) = 1 + length xs
```

Zeigen Sie durch strukturelle Induktion, dass für jede Liste `xs :: [Int]` die folgende Gleichung erfüllt ist:

$$\texttt{treesum (foo xs)} = \texttt{trip xs} - 3^{\texttt{(length xs)}}$$

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

Aufgabe 4 (AGS 12.4.38 ⋆)

In der Aufgabe dürfen Sie die folgenden Terme und Beziehungen nutzen:

$$\begin{aligned}
\langle n\rangle &\quad \text{für } n \geq 0\\
\langle succ\rangle\langle n\rangle &\Rightarrow^* \langle n+1\rangle\\
\langle pred\rangle\langle n\rangle &\Rightarrow^* \langle n-1\rangle \quad \text{für } n > 0\\
\langle add\rangle\langle n_1\rangle\langle n_2\rangle &\Rightarrow^* \langle n_1 + n_2\rangle\\
\langle mult\rangle\langle n_1\rangle\langle n_2\rangle &\Rightarrow^* \langle n_1 \cdot n_2\rangle\\
\langle Y\rangle &= (\lambda z.((\lambda u.z(uu))(\lambda u.z(uu))))
\end{aligned}$$

$$\langle gt\rangle\langle n_1\rangle\langle n_2\rangle \Rightarrow^* \begin{cases}\langle true\rangle\,, & \text{wenn } n_1 > n_2\\ \langle false\rangle & \text{sonst}\end{cases}$$

$$\langle iszero\rangle\ s \Rightarrow^* \begin{cases}\langle true\rangle\,, & \text{wenn } s \Rightarrow^* \langle 0\rangle\\ \langle false\rangle & \text{sonst}\end{cases}$$

$$\langle ite\rangle\ s\ s_1\ s_2 \Rightarrow^* \begin{cases}s_1\,, & \text{wenn } s \Rightarrow^* \langle true\rangle\\ s_2 & \text{sonst}\end{cases}$$

(a) Gegeben sei die folgende Haskell-Funktion `g`:

```
g :: Int -> Int -> Int
g 0 y = y
g x y = g (x - 1) (x + y)
```

Geben Sie einen Lambda-Term $\langle G\rangle$ an, sodass $\mathtt{g} = \langle Y\rangle\langle G\rangle$ gilt.

(b) 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.

$$(\lambda f x.f\,x\,x)\ (\lambda y\,z.x\,y\,z)\ w$$

(c) Gegeben sei der λ-Term

$$\langle F\rangle = \Big(\lambda f x\,y\,z.\,\langle ite\rangle\ (\langle gt\rangle(\langle add\rangle\ x\ y)\ z)\ \ z\ \ (f\ (\langle succ\rangle\ x)\ (\langle succ\rangle\ y)\ (\langle pred\rangle\ z)\ )\Big)\,.$$

Berechnen Sie die Normalform des Terms $\langle Y\rangle\langle F\rangle\langle 2\rangle\langle 3\rangle\langle 7\rangle$. Schreiben Sie für jeden Aufruf von $\langle F\rangle$ jeweils zwei Zeilen: eine, in der Sie die Argumente 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.

Aufgabe 5 (AGS 13.14 ⋆)

(a) Natürliche Zahlen können in Prolog$^-$ als Terme über dem einstelligen Funktionssymbol `s` und dem nullstelligen Funktionssymbol `0` dargestellt werden, z.B. steht der Term `s(s(s(0)))` für die Zahl 3. Wir kürzen den Term für die Zahl n mit `<n>` ab, also `s(s(s(0)))` = `<3>`.

Programmieren Sie in Prolog$^-$ eine binäre Relation `leq`, die genau die Paare (x, y) von natürlichen Zahlen enthält, für die $x \leq y$ gilt. Es soll also beispielsweise `leq(<2>, <3>)` gelten. *Hinweis:* Nutzen Sie dafür das Prädikat `nat`, das wie folgt definiert ist:

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

(b) Programmieren Sie in Prolog$^-$ eine binäre Relation `minimum`, die alle Paare (l, x) enthält, sodass l eine nichtleere Liste von natürlichen Zahlen und x das kleinste Element dieser Liste ist. Beispielsweise soll `minimum([<42>,<23>,<1337>], <23>)` gelten. Sie dürfen dazu das Prädikat `leq` aus Teilaufgabe **(a)** benutzen.

(c) Zur Darstellung von Polynomen führen wir die zweistelligen Funktionssymbole `sum`, `prod` und `pow` ein, die für Addition, Multiplikation beziehungsweise Potenz stehen. Das Polynom $x^2 + 4$ wird beispielsweise durch den Term `sum(pow(x,<2>), <4>)` repräsentiert. Gegeben sei das folgende Prolog$^-$-Programm, das für ein Polynom über der Variablen `x` dessen Ableitung nach `x` berechnet:

```
3 derv(x, s(0)).
4 derv(C, 0) :- nat(C).
5 derv(pow(x, s(N)), prod(s(N),pow(x,N))).
6 derv(sum(F,G), sum(DF,DG)) :- derv(F, DF), derv(G, DG).
7 derv(prod(F,G), sum(prod(F,DG),prod(DF,G))) :- derv(F, DF),
      derv(G, DG).
```

Bestimmen Sie für das Goal `?- derv(sum(pow(x,<2>),<4>), D).` eine Belegung der Prolog-Variablen `D` mittels SLD-Refutation. Notieren Sie in jedem Schritt die verwendete Programmzeile. Mehrere Resolutionsschritte unter Anwendung derselben Zeile können Sie mit `?-*` zusammenfassen.

Aufgabe 6 (AGS 15.27 ⋆)

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

```
1  #include <stdio.h>
2
3  int x;
4
5  void g(int *p);
6  void f(...) {...}
7  void main() {...}
8  void g(int *p) {
9    int l;
10   if (x == 0) {
11     x=x+1; g(&l);
12   }
13   f(*p);
14 }
```

Übersetzen Sie die Statements im Rumpf von `g` (Zeilen 10–13) mittels *stseqtrans* in AM_1-Code mit baumstrukturierten Adressen. Geben Sie keine Zwischenschritte an. *Geben Sie zunächst die dazu benötigte Symboltabelle an.*

(b) Gegeben sei folgender AM_1-Code:

```
1:  INIT 1;            6:  LIT 1;            11:  READ(global,1);
2:  CALL 10;           7:  SUB;              12:  LOAD(global,1);
3:  JMP 0;             8:  STOREI(-2);       13:  LIT 0;
4:  INIT 1;            9:  RET 2;            14:  GE;
5:  LOAD(lokal,-3);   10:  INIT 0;           15:  JMC 22;
```

```
16: LOAD(global,1);    19: PUSH;              22: WRITE(global,1);
17: PUSH;              20: CALL 4;            23: RET 0;
18: LOADA(global,1);   21: JMP 12;
```

Führen Sie jede der drei AM_1-Konifgurationen um jeweils vier Schritte weiter:

- $(11, \varepsilon, 0:3:0, 3, 5, \varepsilon)$,
- $(18, \varepsilon, 5:3:0:5, 3, \varepsilon, \varepsilon)$,
- $(7, 1:5, 5:3:0:5:1:21:3:0, 7, \varepsilon, \varepsilon)$.

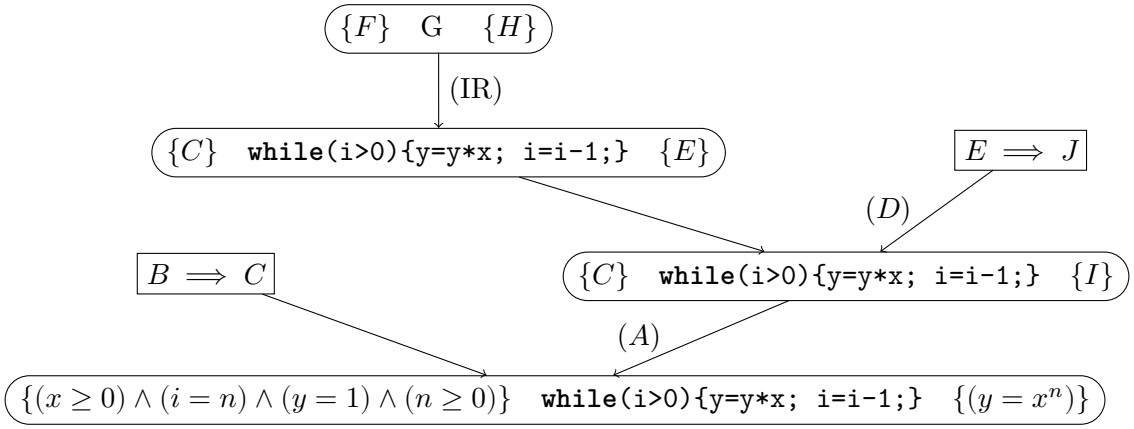
Sie müssen nur Zellen ausfüllen, deren Wert sich im Vergleich zur vorherigen Zeile geändert hat.

Aufgabe 7 (AGS 16.34 ⋆)

Die folgende Verifikationsformel soll mit dem Hoare-Kalkül bewiesen werden:

$$\{(x \geq 0) \wedge (i = n) \wedge (y = 1) \wedge (n \geq 0)\} \quad \texttt{while(i>0)\{y=y*x; i=i-1;\}} \quad \{(y = x^n)\}.$$

Ein Teil eines Beweisbaums wurde unten bereits aufgeschrieben; die Prädikate bzw. Regeln A bis J sind noch unbekannt.



Es gelten die Abkürzungen AR = Alternativregel, CR = Compregel, IR = Iterationsregel, SN = schwächere Nachbedingung, SR = Sequenzregel, SV = stärkere Vorbedingung und ZA = Zuweisungsaxiom.

(a) Geben Sie eine geeignete Schleifeninvariante an.

(b) Geben Sie für die Variablen A bis J entsprechende Prädikate bzw. Regeln an. Sie können die Schleifeninvariante mit *SI* abkürzen.

(c) Mit welcher Regel kann der Beweis fortgeführt werden? Nennen Sie den Namen einer geeigneten Regel und markieren Sie im Beweisbaum oben den Knoten an welchem diese angewandt wird.

Aufgabe 8 (AGS 17.5 a ⋆, 17.6 ⋆)

(a) Geben Sie ein H_0-Programm an, welches die Eingabe einer Zahl $n \in \mathbb{N}$ fordert und dann als Ergebnis $\sum_{j=1}^{n}(j+1)^2$ ermittelt und ausgibt.

(b) Transformieren Sie das folgende H_0-Programm mittels der Funktion *trans* in ein AM_0-Programm mit linearen Adressen. Sie brauchen dabei keine Zwischenschritte anzugeben.

```
1 module Main where
2
3 fac :: Int -> Int -> Int
4 fac x1 x2 = if x1 > 0 then fac (x1 - 1) (x1 * x2)
5                       else x2
6
7 main = do x1 <- readLn
8           print (fac x1 1)
```