

Übungen zu Informatik I (Lösungsvorschlag)

Aufgabe 1 Auswertung und Terminierung (12 Punkte)

a) Die Auswertung in vereinfachter Notation ist:

$$\begin{aligned} pascal(3, 2) &\rightsquigarrow pascal(2, 1) + pascal(2, 2) \\ &\rightsquigarrow (pascal(1, 0) + pascal(1, 1)) + pascal(2, 2) \\ &\rightsquigarrow (1 + pascal(1, 1)) + pascal(2, 2) \\ &\rightsquigarrow (1 + 1) + pascal(2, 2) \\ &\rightsquigarrow 2 + pascal(2, 2) \\ &\rightsquigarrow 2 + 1 \\ &\rightsquigarrow 3 \end{aligned}$$

b) Zu zeigen: Die Aussage gilt für alle n, k mit $k \leq n$. Wir zeigen die Aussage durch Induktion nach n .

Induktionsanfang: $n = 0$. Da $0 \leq k \leq n$, gilt $k = 0$, also $pascal(n, k) \rightsquigarrow 1$, d.h. die Auswertung von $pascal(n, k)$ terminiert.

Induktionsschritt: $n > 0$. Falls $k = 0$ oder $k = n$, gilt $pascal(n, k) \rightsquigarrow 1$, also terminiert die Auswertung von $pascal(n, k)$. Sei also $0 < k < n$. Dann gilt $pascal(n, k) \rightsquigarrow pascal(n - 1, k - 1) + pascal(n - 1, k)$. Nach Induktionsvoraussetzung terminiert die Auswertung von $pascal(n - 1, k - 1)$ und von $pascal(n - 1, k)$, also terminiert auch die Auswertung von $pascal(n, k)$.

Aufgabe 2 Polynome als Listen (12 Punkte)

a) Eine mögliche Definition von *polyeval* ist die folgende:

```
fun polyeval nil z = 0
  | polyeval (x::xs) z = x+(z*polyeval(xs)(z));;
```

b) Die Funktion *polyderiv* kann mit Hilfe der Technik der Einbettung wie folgt definiert werden:

```
fun deriv n nil = nil
  | deriv n (x::xs) = (n*x) :: (deriv (n+1) xs);;

fun polyderiv nil = nil
  | polyderiv (x::xs) = deriv 1 xs;;
```

Aufgabe 3**BNF-Grammatiken**

(12 Punkte)

- a) Wir setzen $G = (\Sigma, V, S, P)$ wobei $V = \{\langle S \rangle, \langle C \rangle, \langle D \rangle\}$ und P die Produktionsregeln

$$\langle S \rangle ::= [a\langle S \rangle \mid b\langle S \rangle \mid c\langle C \rangle \mid d\langle D \rangle]$$

$$\langle C \rangle ::= b\langle S \rangle \mid d\langle D \rangle$$

$$\langle D \rangle ::= c\langle C \rangle$$

umfasst.

Idee: Die Nichtterminalzeichen $\langle C \rangle$ und $\langle D \rangle$ sorgen dafür, dass nach einem c bzw. d nur die "richtigen" Zeichen erzeugt werden können.

- b) Ableitung von $aabba$:

$$\langle S \rangle \rightsquigarrow a\langle S \rangle$$

$$\rightsquigarrow aa\langle S \rangle$$

$$\rightsquigarrow aab\langle S \rangle$$

$$\rightsquigarrow aabb\langle S \rangle$$

$$\rightsquigarrow aabba\langle S \rangle$$

$$\rightsquigarrow aabba$$

Ableitung von $dcdeb$:

$$\langle S \rangle \rightsquigarrow d\langle D \rangle$$

$$\rightsquigarrow dc\langle C \rangle$$

$$\rightsquigarrow dcd\langle D \rangle$$

$$\rightsquigarrow dcdc\langle C \rangle$$

$$\rightsquigarrow dcdeb\langle S \rangle$$

$$\rightsquigarrow dcdeb$$

Aufgabe 4**Tupel-Selektoren**

(4 Punkte)

Eine mögliche Herleitung ist die folgende:

(1)	$\{x : \mathbf{int}\} \triangleright x : \mathbf{int}$	Typaxiom
(2)	$\emptyset \triangleright 2 : \mathbf{int}$	Typaxiom
(3)	$\emptyset \triangleright 7 : \mathbf{int}$	Typaxiom
(4)	$\emptyset \triangleright 3 : \mathbf{int}$	Typaxiom
(5)	$\emptyset \triangleright * : \mathbf{int} * \mathbf{int} \rightarrow \mathbf{int}$	Typaxiom
(6)	$\emptyset \triangleright + : \mathbf{int} * \mathbf{int} \rightarrow \mathbf{int}$	Typaxiom
(7)	$\{x : \mathbf{int}\} \triangleright x * 2 : \mathbf{int}$	(R2) mit Prämissen (1), (2) und (5)
(8)	$\emptyset \triangleright 7 + 3 : \mathbf{int}$	(R2) mit Prämissen (3), (4) und (6)
(9)	$\{x : \mathbf{int}\} \triangleright (x * 2, 7 + 3) : \mathbf{int} * \mathbf{int}$	(R1) mit Prämissen (7), (8)
(10)	$\emptyset \triangleright \mathbf{fn} x \Rightarrow (x * 2, 7 + 3) : \mathbf{int} \rightarrow \mathbf{int} * \mathbf{int}$	(R4) mit Prämisse (9)

Aufgabe 5**Künstlervermittlung**

(20 Punkte)

Folgender SML-Code enthält die Lösung aller Teilaufgaben:

(* Jeder Künstler hat einen Namen, eine Gage und eine Auftrittsdauer in Minuten. Bei Akrobaten wird die minimale Höhe angegeben, die die Halle, in der sie auftreten, haben muss.

*)

```

val foldr = List.foldr;;
val exists = List.exists;;
val all = List.all;;
val filter = List.filter;;

datatype kuenstler = Zauberer of string * int
  | Akrobat of string * int * int;;

val z1 = Zauberer("Zauberer 1", 8);;
val z2 = Zauberer("Zauberer 2", 7);;
val a1 = Akrobat("Akrobat 1", 3, 15);;
val a2 = Akrobat("Akrobat 2", 5, 6);;
val a3 = Akrobat("Akrobat 3", 2, 2);;

val ks = [z1,z2,a1,a2,a3];;

fun dauer (Zauberer(_,d)) = d
  | dauer (Akrobat(_,d,_)) = d;;

val gesamtdauer = (foldr (op +) 0) o (map dauer);;

fun hoch_genug h (Zauberer(_, _)) = true
  | hoch_genug h (Akrobat(_, _, mh)) = mh <= h;;

fun koennen_auftreten h ks =
  all (hoch_genug h) ks;;

```

```
fun sublisten nil = [[]]
  | sublisten (x::xs) = (sublisten xs) @ (map (fn l => (x::l)) (sublisten xs));

fun engagement h = (filter (koennen_auftreten h)) o sublisten;
```