



Institut für Informatik
Prof. Dr. Peter Thiemann
Jochen Walter

Georges-Köhler-Allee 79
D-79110 Freiburg i. Br.

Freiburg, den 8. Dezember 2000

Informatik 1, WiSe 2000/2001

Übungsblatt 6

Die Aufgaben werden in den Übungs- und Programmiergruppen
vom 7.12. bis zum 13.12. besprochen.
Die Lösungen müssen nicht abgegeben werden!

Die Aufgaben auf diesem Blatt können in Teams bearbeitet werden. Bevor drscheme gestartet werden kann, muß setup lang eingegeben werden. Der Sprachumfang sollte auf „Advanced Student“ eingestellt werden. Die Aufgaben sind mit einem bis drei Sternen versehen, wobei die Aufgaben mit einem Stern am einfachsten, die mit dreien am schwierigsten sind.

In den Theorie-Übungsgruppen vom 15.12. bis zum 20.12 werden die Programmierprojekte vorgeführt. Deshalb muß jeder in dieser Woche unbedingt an den Theorie-Übungsgruppen teilnehmen!

Aufgabe 1 (**):

(a) Die Funktion $m : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ ist durch

$$m(x, y) = \begin{cases} y - x & ; \text{ falls } y \geq x \\ 0 & ; \text{ sonst} \end{cases}$$

gegeben. Geben Sie eine Definition dieser Funktion mit dem Scheme der primitiven Rekursion unter Verwendung der Vorgängerfunktion pred. Schreiben Sie Ihre Definition einmal, indem Sie zwei Gleichungen für m angeben und einmal, indem Sie das in der Vorlesung angegebene Funktional PR verwenden.

(b) Die Funktion $\text{mod} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ ist gegeben durch

$$m \bmod n = \begin{cases} m - qn & ; \text{ falls } (\exists q)qn \leq m \wedge (q + 1) \cdot n > m \\ 0 & ; \text{ sonst} \end{cases}$$

Zeigen Sie, daß mod primitiv-rekursiv ist, indem Sie primitiv-rekursive Funktionen f und g angeben, mit denen Sie mod durch das Schema der primitiven Rekursion definieren. Verwenden Sie dazu die Funktion m aus der ersten Aufgabe und die Ganzzahlmultiplikation.

Hinweis: Definieren Sie erst die Hilfsfunktion $\text{sg} : \mathbb{N} \rightarrow \mathbb{N}$:

$$\text{sg}(x) = \begin{cases} 0 & ; \text{ falls } x = 0 \\ 1 & ; \text{ sonst} \end{cases}$$

Lösung zu Aufgabe 1:

(a) Definition durch Gleichungen:

$$\begin{aligned}m(0, y) &= y \\ m(x + 1, y) &= \text{pred}(m(x, y))\end{aligned}$$

Man bezeichnet mit $\text{Prim}^{(n)}(\mathbb{N})$ die Menge der n -stelligen primitiv-rekursiven Funktionen über \mathbb{N} . Das Schar der Funktionale

$$\text{PR}^{(n)} : \text{Prim}^{(n)}(\mathbb{N}) \times \text{Prim}^{(n+2)}(\mathbb{N}) \rightarrow \text{Prim}^{(n)}(\mathbb{N})$$

wurde in der Vorlesung wie folgt definiert:

Falls f eine n -stellige und g eine $(n+2)$ -stellige primitiv-rekursive Funktion ist und h durch das Schema der primitiv-rekursiven Funktion aus f und g definiert ist, dann schreibt man $h = \text{PR}^{(n)}(f, g)$

Die Definition mit Hilfe des Funktionals PR sieht dann wie folgt aus:

$$m = \text{PR}^{(1)}(\Pi_1^{(1)}, \text{pred} \circ \Pi_3^{(3)})$$

(b) Zunächst definieren wir die Signumsfunktion sg nach dem Schema der primitiven Rekursion:

$$\begin{aligned}\text{sg}(0) &= 0^{(0)} \\ \text{sg}(m + 1) &= 0^{(0)'}\end{aligned}$$

$$(h(y) = \text{sg}(y), f() = 0^{(0)}, g(y, z) = 0^{(0)'})$$

Dann definieren wir die einstellige Nullfunktion:

$$\begin{aligned}0^{(1)}(0) &= 0^{(0)} \\ 0^{(1)}(y + 1) &= 0^{(0)}\end{aligned}$$

$$(h(y) = 0^{(1)}(y), f() = 0^{(0)}, g(y, z) = 0^{(0)'})$$

Dann kann man mod wie folgt definieren:

$$\begin{aligned}0 \text{ mod } x &= 0^{(1)}(x) \\ (y + 1) \text{ mod } x &= (y \text{ mod } x)' \cdot \text{sg}(m((y \text{ mod } x)', x))\end{aligned}$$

$$(h(y, x) = y \text{ mod } x, f(x) = 0^{(1)}(x), g(x, y, z) = \text{sg}(m(z', x)) \cdot z')$$

Aufgabe 2 (***):

In der Vorlesung wurde die Ackermannfunktion durch

$$\begin{aligned}A(0, n) &= n + 1 \\ A(m + 1, 0) &= A(m, 1) \\ A(m + 1, n + 1) &= A(m, A(m + 1, n))\end{aligned}$$

definiert. Welche Funktionen sind durch

$$f(n) = A(0, n)$$

$$g(n) = A(1, n)$$

$$h(n) = A(2, n)$$

$$j(n) = A(3, n)$$

gegeben?

Lösung zu Aufgabe 2:

- $f(n) = A(0, n) = n + 1 = n'$
- $g(0) = A(1, 0) = A(0, 1) = 1 + 1$
 $g(n + 1) = A(1, n + 1) = A(0, A(1, n)) = A(1, n) + 1 = g(n) + 1$
 $g(n) = n''$
- $h(0) = A(2, 0) = A(1, 1) = A(0, A(1, 0)) = A(1, 0) + 1 = A(0, 1) + 1 = 3$
 $h(n + 1) = A(2, n + 1) = A(1, A(2, n)) = A(1, h(n)) = h(n)''$
 $h(n) = 2n + 3$
- $j(0) = A(3, 0) = A(2, 1) = h(1) = 5$
 $j(n + 1) = A(3, n + 1) = A(2, A(3, n)) = A(2, j(n)) = h(j(n)) = 2j(n) + 3$
 $j(n) = 2^{n+3} - 3$

(Die Angaben für g, h und j lassen sich durch vollständige Induktion beweisen.)

Aufgabe 3 (**):

Gegeben sei ein Alphabet Σ mit einer totalen Ordnung \leq_{Σ} . Geben Sie eine wohlfundierte Ordnung \leq_{Σ^*} auf Σ^* an.

Lösung zu Aufgabe 3:

Eine mögliche Lösung ist: $w \leq_{\Sigma^*} v$, falls

- $w = v = \epsilon$, oder
- $\text{len}(w) < \text{len}(v)$, oder
- $\text{len}(w) = \text{len}(v) > 0$ und für $w = aw', v = bv'$ gilt $a \leq_{\Sigma} b$, oder
- $\text{len}(w) = \text{len}(v) > 0$ und für $w = aw', v = av'$ gilt $w' \leq_{\Sigma^*} v'$.

In allen anderen Fällen gilt $v \leq_{\Sigma^*} w$.

Beweis der Wohlfundiertheit: Zunächst bezeichnen wir mit \geq_{Σ^*} die Umkehrrelation $\leq_{\Sigma^*}^{-1}$ von \leq_{Σ^*} .

Man betrachtet eine beliebige, nichtleere Teilmenge A von Σ^* und nimmt an, A habe kein minimales Element. Dann muß es eine unendliche absteigende Folge $w_1 \geq_{\Sigma^*} w_2 \geq_{\Sigma^*} \dots$ geben mit $(\forall i \in \mathbb{N}) w_i \in A$ und $(\forall i \in \mathbb{N}) w_i \neq w_{i+1}$.

Man betrachtet ein w_i aus der Folge und definiert $n = \text{len}(w_i)$. Für alle w_j mit $i < j$ gilt $\text{len}(w_j) \leq \text{len}(w_i)$. Da es $|\Sigma|^n$ Zeichenketten der Länge n gibt, muß spätestens für $w_{i+|\Sigma|^n}$ gelten, daß $\text{len}(w_{i+|\Sigma|^n}) < n$ ist.

Auf $w_{i+|\Sigma|^n}$ wendet man dasselbe Argument an. Nach spätestens $n + 1$ Schritten erreicht man ein Element w der Folge mit $\text{len}(w) < 0$. Das ist ein Widerspruch, da $\text{Im}(\text{len}) = \mathbb{N}$. Damit ist unsere Annahme falsch und A hat ein minimales Element.

Aufgabe 4 ():**

Beweisen Sie durch vollständige Induktion über das erste Argument von `app`: Für die in der Vorlesung vorgestellten Funktionen

$$\text{len} : \Sigma^* \rightarrow \mathbb{N}$$

und

$$\text{app} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$$

gilt

$$\text{len}(\text{app}(w, v)) = \text{len}(w) + \text{len}(v)$$

Lösung zu Aufgabe 4:

Induktionsanfang:

$$\begin{aligned} \text{len}(\text{app}(\epsilon, v)) &= \text{len}(v) && \text{Def. von app} \\ &= 0 + \text{len}(v) && \text{Algebraische Umformung} \\ &= \text{len}(\epsilon) + \text{len}(v) && \text{Def. von len} \end{aligned}$$

Induktionsschritt:

Gelte die Behauptung für w , sei $a \in \Sigma$ beliebig:

$$\begin{aligned} \text{len}(\text{app}(aw, v)) &= \text{len}(a \text{ app}(w, v)) && \text{Def. von app} \\ &= 1 + \text{len}(\text{app}(w, v)) && \text{Def. von len} \\ &= 1 + \text{len}(w) + \text{len}(v) && \text{Induktionsannahme} \\ &= \text{len}(aw) + \text{len}(v) && \text{Def. von len} \end{aligned}$$

Aufgabe 5 (Programmierung, *):

In der Vorlesung wurde die Funktion `btree-inorder` angegeben, die eine Liste der Elemente des Baumes zurückliefert. Dabei wurde die Funktion `append` verwendet, wodurch ziemlich viel Speicher verbraucht wird.

Schreiben Sie eine Funktion `btree-fast-inorder`, die ohne `append` auskommt. Verwenden Sie dabei die Funktion zur iterativen Berechnung der Fakultät als Vorbild.

Lösung zu Aufgabe 5:

```
; SIGNATUR
; btree-fast-inorder : btree -> list
; ERKLÄRUNG
; liefert die Liste der Elemente des Baums von links nach rechts
; BEISPIEL
; (btree-fast-inorder empty) == empty
```

```

; (btree-fast-inorder (make-branch (make-branch empty 1 empty) 0 empty))
;   == (list 1 0)
; (btree-fast-inorder
;   (make-branch (make-branch empty 5 (make-branch empty 10 empty))
;                 11 empty))
;   == (list 5 10 11)
; DEFINITION
(define btree-fast-inorder
  (lambda (t)
    (btree-fast-inorder-1 t empty)))

(define btree-fast-inorder-1
  (lambda (t a)
    (cond
      ((empty? t) a)
      ((branch? t)
       (btree-fast-inorder-1
        (branch-left t)
        (cons (branch-elem t)
              (btree-fast-inorder-1 (branch-right t) a)))))))

```

Aufgabe 6 (Programmierung, *):

Schreiben Sie Scheme-Funktionen `btree-preorder` und `btree-postorder`, die als Argument einen `btree`-Wert erwarten und eine Liste der Elemente des Baumes zurückliefern. Bei `btree-preorder` soll dabei zuerst das Element des Wurzelknotens, dann die Elemente des linken Teilbaums und danach die Elemente des rechten Teilbaumes jeweils in Preorder-Reihenfolge angeordnet werden. Bei `btree-postorder` sollen zunächst die Elemente des linken und rechten Teilbaums in Postorder-Reihenfolge und danach das Element des Wurzelknotens aufgelistet werden.

Lösung zu Aufgabe 6:

```

; SIGNATUR
; btree-preorder : btree -> list
; ERKLÄRUNG
; liefert die Liste der Elemente des Baums in Preorder-Reihenfolge
; BEISPIEL
; (btree-preorder empty) == empty
; (btree-preorder (make-branch (make-branch empty 1 empty) 0 empty))
;   == (list 0 1)
; (btree-preorder
;   (make-branch (make-branch empty 5 (make-branch empty 10 empty))
;                 11 empty))
;   == (list 5 10 11)

```

```

;                                     11 empty))
;     == (list 11 5 10)
; DEFINITION
(define btree-preorder
  (lambda (t)
    (cond
      ((empty? t) empty)
      ((branch? t)
       (cons (branch-elem t)
              (append (btree-preorder (branch-left t))
                      (btree-preorder (branch-right t)))))))

; SIGNATUR
; btree-postorder : btree -> list
; ERKLÄRUNG
; liefert die Liste der Elemente des Baums in Postorder-Reihenfolge
; BEISPIEL
; (btree-postorder empty) == empty
; (btree-postorder (make-branch (make-branch empty 1 empty) 0 empty))
;     == (list 1 0)
; (btree-postorder
; (make-branch (make-branch empty 5 (make-branch empty 10 empty))
;              11 empty))
;     == (list 5 10 11)
; DEFINITION
(define btree-postorder
  (lambda (t)
    (cond
      ((empty? t) empty)
      ((branch? t)
       (append (btree-postorder (branch-left t))
                (btree-postorder (branch-right t))
                (list (branch-elem t)))))))

```