Musterlösung I. Übungsblatt “Informatik I” WS 95/96

Roland Fischer, Jürgen Zimmer, Christian Schulte

Aufgabe I.1 (Exponentialfunktion)

```
(define (exp n m)
   (if (= m 0)
       1
       (* n (exp n (- m 1)))))
```

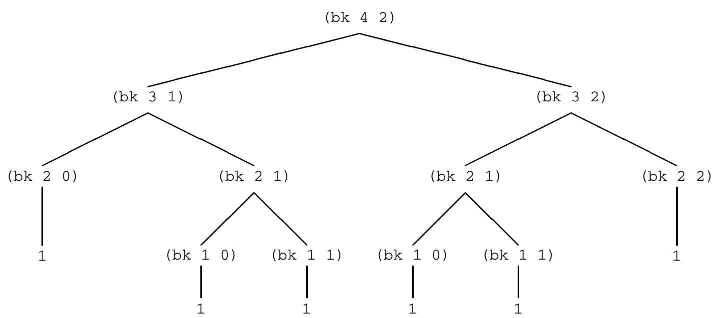
Aufgabe I.2 (Binomialkoeffizienten)

1. Die Bedingung “sonst” beinhaltet alle Fälle, in denen nicht $k > 0$ *und* $n > k$ gilt. Das ist genau dann der Fall, wenn $k \leq 0$ oder $n \leq k$ ist. Insgesamt also:

$$\binom{n}{k} := \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & \text{falls } k > 0 \text{ und } n > k \\ 1 & \text{falls } k \leq 0 \text{ oder } n \leq k \end{cases}$$

2.
```
(define (binomialkoeffizient n k)
     (if (> k 0)
         (if (> n k)
             (+ (binomialkoeffizient (- n 1) (- k 1))
                (binomialkoeffizient (- n 1) k))
             1)
         1)))
```

3. Aus Platzgründen verwenden wir `bk` statt `binomialkoeffizient`.



Aufgabe I.3 (Arithmetik für natürliche Zahlen)

1. Es gilt $n \leq m$ wenn $n = 0$. Ist $n \neq 0$ und $m = 0$, dann ist $n \not\leq m$. Ansonsten ist $n \leq m$ genau dann, wenn $n - 1 \leq m - 1$. Insgesamt haben wir:

$$n \leq m := \begin{cases} \texttt{\#t} & \text{falls } n = 0 \\ \texttt{\#f} & \text{falls } n \neq 0 \text{ und } m = 0 \\ m - 1 \leq n - 1 & \text{sonst} \end{cases}$$

2. `(define inkrement (lambda (n) (+ n 1)))`

3. Die Prozedur `kleiner` geht analog zu `kleiner-gleich`. Lediglich der Fall, bei dem sowohl n als auch m gleich null ist, ist unterschiedlich:

```
(define kleiner
  (lambda (n m)
    (if (null? n)
        (if (null? m) #f #t)
        (if (null? m) #f
            (kleiner (dekrement n) (dekrement m))))))
```

4.
```
(define plus
  (lambda (n m)
    (if (null? n) m
        (inkrement (plus (dekrement n) m)))))
```

5. Es gilt: $n * m = (n-1) * m + m$ und $0 * m = 0$. Damit erhält man folgende induktive Definition für die Multiplikation:

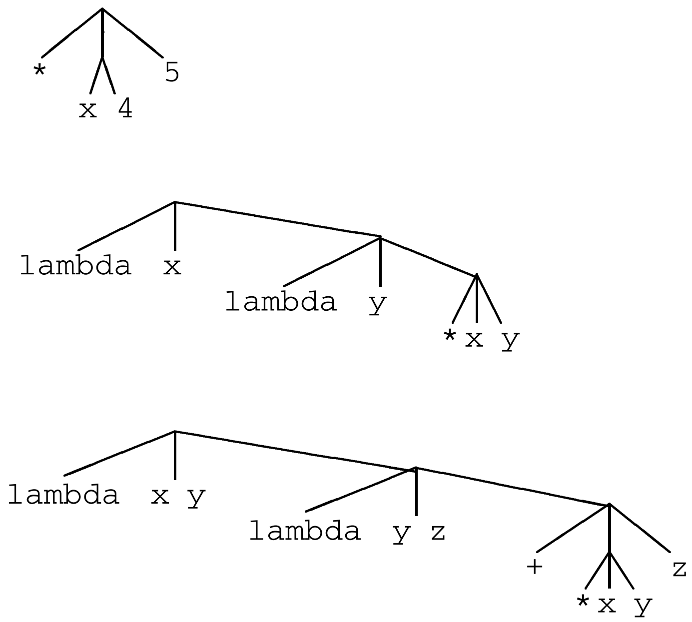
$$n * m := \begin{cases} 0 & \text{falls } n = 0 \\ (n-1) * m + m & \text{sonst} \end{cases}$$

Und als Scheme-Prozedur:

```
(define mult (lambda (n m)
  (if (null? n) 0
      (plus (mult (dekrement n) m) m))))
```

Aufgabe I.4 (Ausdrücke und Namen) In der Vorlesung haben Sie gelernt, daß Abstraktionen Namen binden. Wir wollen in dieser Aufgabe die vier folgenden Ausdrücke einer Bindungsanalyse unterziehen:

- `(* (x 4) 5)`
- `(lambda (x) (lambda (y) (* x y)))`
- `(lambda (x y) (lambda (y z) (+ (* x y) z)))`
- Ist gestrichen, weil der Ausdruck kein Ausdruck von Scheme 1 ist.

1. 

2. Ein Teilausdruck eines Ausdruckes ist der Ausdruck selbst, wir lassen ihn jedoch aus der Liste heraus.

   (a) i. Der Ausdruck selbst.
      ii. `(x 4)`

   (b) i. Der Ausdruck selbst.
      ii. `(lambda (y) (* x y))`
      iii. `(* x y)`

   (c) i. Der Ausdruck selbst.
      ii. `(lambda (y z) (+ (* x y) z))`
      iii. `(+ (* x y) z)`
      iv. `(* x y)`

3. Da jeder Name nur einmal auftritt, wenn man die Auftreten in den Argumenten der `lambda` außer acht läst, geben wir die bindende Abstraktion durch den Teilausdruck an.

Ausdruck	Namen	frei	gebunden	Abstraktion
(a) i.	{*,x}	{*,x}	{}	—
(a) ii.	{x}	{x}	{}	—
(b) i.	{*,x,y}	{*}	{x,y}	x:i, y:ii
(b) ii.	{*,x,y}	{*,x}	{y}	y:ii
(b) iii.	{*,x,y}	{*,x,y}	{}	—
(c) i.	{+,*,x,y,z}	{+,*}	{x,y,z}	x:i, y:ii, z:ii
(c) ii.	{+,*,x,y,z}	{+,*,x}	{y,z}	y:ii, z:ii
(c) iii.	{+,*,x,y,z}	{+,*,x,y,z}	{}	—
(c) iv.	{*,x,y}	{*,x,y}	{}	—