I. Übungsblatt "Informatik I" WS 95/96

Christian Schulte, Prof. Gert Smolka

Ausgabe: 20. Oktober 1995 **Abgabe:** 27. Oktober 1995 vor der Vorlesung

Aufgabe I.1 (Exponentialfunktion (4)) Die Exponentialfunktion $exp(n, m) = n^m$ ist für natürliche Zahlen induktiv wie folgt definiert:

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

Zum Beispiel ist $exp(4, 2) = 16$, $exp(3, 8) = 6561$ und $exp(1, 1000) = 1$. Definieren Sie eine rekursive Scheme-Prozedur `(exp n m)`, die die Exponentialfunktion berechnet.

Aufgabe I.2 (Binomialkoeffizienten (6)) Aus der Schule werden Sie vielleicht den Binomialkoeffizenten $\binom{n}{k}$ (gelesen: "n über k") kennen. Dieser ist induktiv wie folgt definiert:

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

Einige Binomialkoeffizienten sind zum Beispiel:

$$\binom{4}{2} = 6 \quad \binom{5}{2} = 10 \quad \binom{9}{3} = 84 \quad \binom{6}{3} = 20$$

1. Ersetzen Sie in der obigen Definition die Bedingung "sonst" durch eine gleichwertige Bedingung in denen n und k explizit vorkommen.
2. Definieren Sie eine rekursive Scheme-Prozedur mit dem Namen `binomialkoeffizient`, die der obigen induktiven Definition folgt.
3. Zeichnen Sie den Aufrufbaum für `(binomialkoeffizient 4 2)` (siehe auch im Buch Abbildung 1.5).

Aufgabe I.3 (Arithmetik für natürliche Zahlen (8)) In dieser Aufgabe lernen Sie Funktionen über den natürlichen Zahlen induktiv zu definieren und sie dann rekursiv in Scheme zu programmieren. Dabei stehen Ihnen nur die folgenden definierten Prozeduren zur Verfügung:

Test auf null: Eine Prozedur, die testet, ob ihr Argument null ist:

```
(define null? (lambda (n) (= n 0)))
```

Dekrementieren: Eine Prozedur, die ihr Argument um eins verringert:

```
(define dekrement (lambda (n) (- n 1)))
```

Kleiner gleich: Eine Prozedur `(kleiner-gleich n m)`, die testet, ob `n` kleiner oder gleich `m` ist:

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

1. Geben Sie eine induktive Definition für die Prozedur `kleiner-gleich` im Stile der Definitionen in den Aufgaben I.1 und I.2 an.

2. Geben Sie eine Prozedur `(inkrement n)` an, die `n+1` ergibt.

3. Definieren Sie eine rekursive Prozedur `(kleiner n m)`, die testet ob `n` echt kleiner als `m` ist. Verwenden Sie dabei nur `dekrement` und `null?`.

4. Die Addition zweier natürlicher Zahlen können wir induktiv wie folgt definieren:

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

   Schreiben Sie dafür eine Prozedur `(plus n m)`, die nur von `inkrement`, `dekrement` und `null?` Gebrauch macht.

5. Analog zu der obigen Definition für die Addition, kann man auch die Multiplikation zweier ganzer Zahlen induktiv definieren. Geben Sie eine induktive Definition an, und schreiben Sie dafür eine rekursive Scheme-Prozedur `(mult n m)`.

Aufgabe I.4 (Ausdrücke und Namen (8)) 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)))`
- ```
  (lambda (x)
    (define y 4)
    (define z 8)
    (if (< y x) (lambda () z) (lambda (z) (z y))))
  ```

1. Zeichnen Sie für die Ausdrücke jeweils einen Baum, wie Sie ihn in der Vorlesung kennengelernt haben.

2. Konstanten und Namen heißen *primitive* Ausdrücke. Alle anderen Ausdrücke heißen *zusammengesetzt*. Geben Sie alle zusammengesetzten Teilausdrücke der obigen Ausdrücke an.

3. Geben Sie zu jedem zusammengesetzten Teilausdruck an:

   (a) Die darin auftretenden Namen.

   (b) Die freien Auftreten eines Namens.

   (c) Die gebundenen Auftreten eines Namens und die Abstraktion, die das Auftreten des Namens bindet.

Bitte denken Sie daran, Ihre Aufgabenblätter mit Namen, Matrikelnummer und Übungsgruppennummer deutlich zu kennzeichnen!
  ```