Prof. Dr. Christoph Bockisch
Steffen Dick
Fachbereich Mathematik und Informatik
AG Programmiersprachen und -werkzeuge

Philipps Universität Marburg

**Übungen zur Vorlesung**
**Deklarative Programmierung: Sommersemester 2018**

Nr. 8, Abgabe bis 25.5.2017 11:59 Uhr

---

**Aufgabe 8.1:** Hey, LISTEN, Mr. Freeman. Check! | 4 Punkte

Implementieren Sie die folgenden Listenfunktionen:

a) `(zip l1 l2)`, die zwei Listen zu einer Liste von Paaren zusammenführt. Die Länge der Ergebnisliste entspricht der Länge der kürzeren Listen.
Beispiel: `(zip '(1 2 3) '(4 5 6))` → `'((1 4) (2 5) (3 6))`

b) `(zip/with f l1 l2)`, die zwei Listen paarweise mithilfe der Funktion f kombiniert. Verwenden sie hierzu `foldr` und Ihre Implementierung aus Aufgabenteil a).
Beispiel: `(zip/with + '(1 2 3) '(4 5 6))` → `'(5 7 9)`

c) `(group l)`, die aufeinanderfolgende gleiche Elemente innerhalb der Liste l gruppiert. Verwenden Sie hierfür eine `fold`-Funktion.
Beispiel: `(group '(1 2 2 3 4 5 5 6))` → `'(1 (2 2) 3 4 (5 5) 6)`

**Aufgabe 8.2:** Pattern Matching | 4 Punkte

Implementieren Sie folgende (rekursive) Funktionen mithilfe von pattern-matching (verwenden Sie bei Ihrer Implementierung keine Listen-funktionen):

a)
```
; [X] (list-of X) -> Boolean
; Checks whether a list has an even amount of elements
(define (list/even? l) ...)
```

b)
```
;(list-of Number) -> (list-of Posn)
; Converts a list with an even amount of numbers
; (x1 y1 x2 y2 ... xn yn) into a list of posn instances
; pairs of the form (0 0) are omitted.
(define (flat->posn l) ...)
```

c)
```
; A Tree is either:
; - (make-tree value Tree Tree)
; - false
(define-struct tree (value left right))
; Tree -> Number
; Computes the height of a tree
(define (tree/height t) ...)
```

**Aufgabe 8.3:** (EX)TERMINATE! 4 Punkte

Begründen Sie, warum die folgenden Funktionen terminieren:

a)
```
; Number Number -> Number
; Computes the sum from m to n
(define (sumFromTo m n)
        (cond [(> m n) 0]
          [else (+ m (sumFromTo (add1 m) n))]))
```

b)
```
; Number Number -> Number
; Computes the binomial coefficient of n and k
; if 0 <= k <= n
(define (binomi n k)
        (cond [(or (= 0 k) (= n k)) 1]
          [else (+ (binomi (sub1 n) (sub1 k))
          (binomi (sub1 n) k) )]))
```

c)
```
; Number -> Number
; Computes the greatest natural number t with (t*t) <= x
(define (sqrt/nat x)
        (local [(define (sqrt/above x t)
                  (if (> (* t t) x) (sub1 t)
                  (sqrt/above x (add1 t))))]
        (sqrt/above x 1)))
```

**Hinweis**: Finden Sie jeweils eine Funktion $st : T_1 \times \cdots \times T_n \to \mathbb{N}$, welche die Funktionsparameter in die natürlichen Zahlen abbildet und argumentieren Sie, dass diese mit jedem rekursiven Aufruf strikt monoton fällt.

**Aufgabe 8.4:** Schwefelhölzer | 4 Punkte

Das Nimm-2-3-Spiel ist ein Spiel für 2 Personen. Zu Beginn werden Streichhölzer in 2 Reihen ausgelegt, wobei die Anzahl der Streichhölzer in den einzelnen Reihen beliebig gewählt werden kann. Die Spieler ziehen nun abwechselnd. Wer am Zug ist, entnimmt aus der Reihe, die mehr Streichhölzer enthält, 2 oder 3 Streichhölzer. Wer zuerst nicht mehr ziehen kann, hat das Spiel verloren.

a) Schreiben Sie eine Funktion:

```
; Number Number -> Bool
; Checks if one can win the game if the two rows contain
; n and m matches.
(win? n m)
```

`(win? n m)` berechnet, ob der Spieler, **der gerade am Zug ist**, das Spiel gewinnen kann, wenn in den 2 Reihen n und m Streichhölzer liegen.

b) Zeigen Sie, dass ihre `(win? n m)`-Implementierung terminiert.

c) Schreiben Sie eine Funktion:

```
; Number Number -> String
; Suggests a move for the pick-a-match game
(suggest m n)
```

`(suggest m n)` soll dabei dem Spieler den nächsten Zug empfehlen, sodass dieser gewinnt. Greifen Sie dabei auf ihre `(win? n m)`-Funktion zurück.

**Hinweis**: Implementieren Sie Aufgabenteil a) rekursiv: Ich kan gewinnen, falls ich x Hölzer aus Reihe y ziehe und mein Gegenspieler nicht mehr gewinnen kann.