4. Übungsblatt zu Programmiersprachenkonzepte

Manuel Schwarz, Michael Stypa

25. November 2010

Aufgabe 4.1

1. `(FIRST (REST (REST '((A B C) (D E F))))) = NIL`
   `(REST '((A B C) (D E F)))` liefert eine Liste mit einem einzelnen Element. Diese besitzt keinen Dekrement-Teil.

2. `(REST (REST (FIRST '((A B (C D)) (E F))))) = ((C D))`

3. `(REST (REST '((1 2) (3 4) (5 6)))) = ((5 6))`
   Dies funktioniert im Gegensatz zu 1. weil die Ursprungsliste `'((1 2) (3 4) (5 6))` drei Elemente besitzt.

4. `(REST (FIRST ((A (B C)) (D E)))) = Fehler`
   `A` ist keine Funktion.

5. `(REST (FIRST (FIRST '(((A B) (C D)) (E F))))) = (B)`

6. `(REST (FIRST (FIRST '((A B (C D)) (E F))))) = Fehler`
   `A` besitzt keinen Dekrement-Teil.

Aufgabe 4.2

```
(define unknown
  (lambda (ex1 ex2 ex3)
    (cond ((atom ex3) (cond ((eq ex2 ex3) ex1) (T ex3)))
          (T (cons (unknown ex1 ex2 (car ex3))
                   (unknown ex1 ex2 (cdr ex3)))))))
```

Die Funktion unknown ist eine Art replace-Funktion. Ist `ex2` kein Atom, so wird einfach `ex3` zurückgegeben. Der Grund hierfür ist, dass `ex3` solange in Kopf (`FIRST`) und `REST` geteilt wird, bis `ex3` ein Atom ist. Dieses Atom wird dann entsprechend zurückgegeben und rekursiv zu einer neuen Liste (`CONS`) hinzugefügt.
Ist `ex2` nun aber ein Atom und enthält (die Liste) `ex3` nun Atome für die gilt `ex3 == ex2`, dann werden diese durch `ex1` ersetzt.

Beispiele:

```
[1]> (unknown '(1 2 3) '(4 5 6) '(7 8 9))
(7 8 9)
[2]> (unknown '(1 2 3) 8 '(7 8 9))
(7 (1 2 3) 9)
[3]> (unknown '(1 2 3) 8 '(4 8 7 8 9))
(4 (1 2 3) 7 (1 2 3) 9)
```

Aufgabe 4.3

```
(defun kreis (zahl)
  (cond ((or (> zahl 99) (< zahl 1)) "Zahl ist nicht zwischen 1 und 99")
        ((eq zahl 99) 1)
        (T (+ zahl 1))))
```

Aufgabe 4.4

```
(defun mylast (liste)
  (cond ((atom (cdr liste)) (car liste))
        (T (mylast (cdr liste)))))

(defun mylastlist (liste)
  (cond ((atom (cdr liste)) (cons (car liste) '()))
        (T (mylastlist (cdr liste)))))
```