Mathematisches Institut
der Universität München
Prof. Dr. Helmut Schwichtenberg
Dr. Christian Urban

Wintersemester 2004/2005
Blatt 6

Übungen zum Schemekurs

Aufgabe 19. Man schreibe eine Prozedur `expr?`, die einen Ausdruck daraufhin überprüft, ob es ein korrekter SCHEME-Ausdruck ist (SCHEME-Ausdrücke sind in den Vorlesungsskripten auf Seite 33 definiert). Zum Beispiel sollen die folgenden Ausdrücke als korrekt klassifiziert werden.

```
(expr? '(set! x 4))
(expr? '(lambda (f) (lambda (x) (f (f x)))))
```

Hinweis: Eine Implementation der Prozedur `const?`, die testet, ob ein Ausdruck eine Konstante ist, d.h. ein Element in $\{\#t, \#f\} \cup I\!N \cup \{\,(\texttt{quote}\ d) \mid d \in \mathsf{Data}\,\}$, könnte wie folgt aussehen.

```
(define (const? x)
    (or (eq? x #t)
        (eq? x #f)
        (and (integer? x) (>= x 0))
        (and (list? x) (eq? (car x) 'quote) (data? (cdr x)))))
```

Lösung.

```
(define (atom? x)
  (or (boolean? x)
      (null? x)
      (and (integer? x) (>= x 0))
      (symbol? x)))

(define (data? x)
  (or (atom? x) (and (pair? x) (data? (car x)) (data? (cdr x)))))

(define (const? x)
  (or (eq? x #t) (eq? x #f)
      (and (integer? x) (>= x 0))
      (and (list? x) (eq? (car x) 'quote) (data? (cdr x)))))


(define keywords '(lambda quote if define set!
                          set-car! set-cdr! begin))

(define (key? x)
  (and (member x keywords) #t))

(define (cifre-symbol? x)
  (char-numeric? (string-ref (symbol->string x) 0)))

(define (var? x)
  (and (symbol? x)
       (not (key? x))
       (not (cifre-symbol? x))))

(var? '3x)


; apply-and returns #t if all entries are true

(define (apply-and li) (not (memq #f li)))

(define (expr? x)
  (or (const? x)
      (var? x)
      (and (list? x)(> (length x) 1)
           (or
            (and (= (length x) 3)
                 (eq? (car x) 'lambda)
                 (list? (cadr x))
                 (apply-and (map var? (cadr x)))
                 (expr? (caddr x)))
            (and (= (length x) 4)
                 (eq? (car x) 'if)
                 (apply-and (map expr? (cdr x))))
            (and (= (length x) 3)
                 (or (eq? (car x) 'define) (eq? (car x) 'set!))
                 (var? (cadr x))
```

```
            (expr? (caddr x)))
       (and (= (length x) 3)
            (or (eq? (car x) 'set-car!)
                (eq? (car x) 'set-cdr!))
            (and (expr? (cadr x))(expr? (caddr x))))
       (and (list? x)  (> (length x) 1)
            (eq? (car x) 'begin)
            (apply-and (map expr? (cdr x))))
       (and (list? x) (> (length x) 1)
            (apply-and (map expr? x)))))))


(expr? '(set! x 4))
(expr? '(lambda (f)(lambda (x) (f (f x)))))
```

Aufgabe 20. Man schreibe ein Programm, das berechnet, auf wie viele Arten man 1 Euro in Münzen wechseln kann.

Lösung.

```
; first solution

(define (change amnt)
  (cond ((= amnt 0) 0)
        ((> amnt 0) (change100 amnt)))
)

;; change of any non-zero amount
(define (change100 amnt)
  (define (change100-iter i max)
    (if (< i (+ max 1))
        (+ (change50 (- amnt (* 100 i))) (change100-iter (+ i 1) max))
        0
    ))
  (change100-iter 0 (floor (/ amnt 100)))
)

(define (change50 amnt)
  (define (change50-iter i max)
    (if (< i (+ max 1))
        (+ (change20 (- amnt (* 50 i))) (change50-iter (+ i 1) max))
        0
    ))
  (change50-iter 0 (floor (/ amnt 50)))
)

(define (change20 amnt)
  (define (change20-iter i max)
    (if (< i (+ max 1))
        (+ (change10 (- amnt (* 20 i))) (change20-iter (+ i 1) max))
        0
    ))
  (change20-iter 0 (floor (/ amnt 20)))
)

(define (change10 amnt)
  (define (change10-iter i max)
    (if (< i (+ max 1))
        (+ (change5 (- amnt (* 10 i))) (change10-iter (+ i 1) max))
        0
    ))
  (change10-iter 0 (floor (/ amnt 10)))
)

(define (change5 amnt)
  (define (change5-iter i max)
    (if (< i (+ max 1))
        (+ (change2 (- amnt (* 5 i))) (change5-iter (+ i 1) max))
        0
    ))
  (change5-iter 0 (floor (/ amnt 5)))
)
```

```
(define (change2 amnt)
  (define (change2-iter i max)
    (if (< i (+ max 1))
        (+ (change1 (- amnt (* 2 i))) (change2-iter (+ i 1) max))
        0
    ))
  (change2-iter 0 (floor (/ amnt 2)))
)

(define (change1 amnt) 1)

(change 100)


; second solution

(define (w b mm)
  (cond ((zero? b) 1)
        ((or (< b 0) (null? mm)) 0)
        (else (+ (w b (cdr mm)) (w (- b (car mm)) mm)))))

(define mm '(100 50 20 10 5 2 1))

(w 100 mm)
```

Abgabe. Dienstag, den 12. Oktober 2004, vor der Vorlesung oder (besser) per Email an `urban@mathematik.uni-muenchen.de`