

Petite Chez Scheme Version 7.0a
 Copyright (c) 1985-2005 Cadence Research Systems

```

;; =====
;; Übung 5, Aufgabe 2
;; =====

;; -----
;; Die Funktion intersection lässt sich effizient implementieren: Wenn (car s1) und (car s2)
;; nicht übereinstimmen, so kann man das kleinere der beiden Elemente sofort wegwerfen.
;; -----

> (define (intersection s1 s2)
  (if (or (null? s1) (null? s2))
      ()
      (let ((x (car s1)) (y (car s2)))
        (cond ((= x y) (cons x (intersection (cdr s1) (cdr s2))))
              ((< x y) (intersection (cdr s1) s2))
              ((> x y) (intersection s1 (cdr s2)))))))

;; -----
;; big-intersection () lässt sich nicht vernünftig definieren, da die Operation "Durchschnitt"
;; kein neutrales Element besitzt (zumindest nicht unter den endlichen Mengen), also wird in
;; diesem Fall eine Fehlermeldung zurückgeliefert.
;; -----

> (define (big-intersection l)
  (if (null? l)
      "undefiniert"
      (let ((s1 (car l)) (l1 (cdr l)))
        (if (null? l1)
            s1
            (intersection s1 (big-intersection l1))))))

;; -----
;; Auch subset? und set-equal? lassen sich effizient implementieren, indem man ausnutzt, dass
;; s1 und s2 aufsteigend geordnet sind. set-equal? ist nichts anderes als die Gleichheit von
;; Listen.
;; -----

> (define (subset? s1 s2)
  (or (null? s1)
      (and (not (null? s2))
            (or (and (= (car s1) (car s2))
                     (subset? (cdr s1) (cdr s2)))
                (and (> (car s1) (car s2))
                     (subset? s1 (cdr s2)))))))

> (define (set-equal? s1 s2)
  (or (and (null? s1) (null? s2))
      (and (not (null? s1))
            (not (null? s2))
            (= (car s1) (car s2))
            (set-equal? (cdr s1) (cdr s2)))))

;; -----
;; minimum liefert einfach das erste und maximum das letzte Element der aufsteigend geordneten
;; Liste. Im Falle der leeren Menge lässt sich kein vernünftiges Resultat finden.
;; -----

> (define (minimum s)
  (if (null? s)
      "undefiniert"
      (car s)))

> (define (maximum s)
  (if (null? s)
      "undefiniert"
      (let ((s1 (cdr s)))
        (if (null? s1)
            (car s)
            (maximum s1))))))

```

```
;; -----  
;; Beispiel: s i n = {0, i, i * 2, ... i * n}  
;; -----  
  
> (define (s i n)  
  (letrec  
    ((s (lambda (i j)  
          (if (> j n)  
              ()  
              (cons (* i j) (s i (+ j 1))))))  
      (s i 0)))  
  
> (big-intersection  
  (list (s 2 100) (s 3 100) (s 4 100) (s 6 100)))  
  
(0 12 24 36 48 60 72 84 96 108 120 132 144 156 168 180 192)  
  
> (big-intersection  
  (list (s 7 100) (s 11 100) (s 9 100)))  
  
(0 693)  
  
> (subset? (s 12 100) (s 3 400))  
  
#t  
  
> (subset? (s 12 100) (s 3 399))  
  
#f  
  
> (subset? (s 6 50) (s 3 100))  
  
#t  
  
> (subset? (s 3 100) (s 6 50))  
  
#f  
  
> (set-equal? (s 3 100) (s 3 99))  
  
#f  
  
> (set-equal? (s 3 99) (s 3 100))  
  
#f  
  
> (set-equal? (s 3 100) (s 3 100))  
  
#t  
  
> (set-equal? (s 3 100) (s 6 50))  
  
#f  
  
> (set-equal? (s 6 50) (s 3 100))  
  
#f  
  
> (minimum (s 7 100))  
  
0  
  
> (maximum (s 7 100))  
  
700  
  
>
```