

Gesamtpunktzahl: 30

Abgabe der Lösungen bis zum 30.1.2017

Hinweis: Zum Testen können Sie auf den Poolrechnern der Informatik 'drracket' verwenden. Tragen Sie im (oberen) Editor-Fensterteil '#lang scheme' ein und starten Sie dann den Interpreter mit dem Button [Start].

Aufgabe 1: Wertesemantik

9 Punkte

maximale Bearbeitungszeit: 30 Minuten

Geben Sie für die folgenden s-Ausdrücke an, in welcher Reihenfolge und mit welchen Zwischenergebnissen ein Scheme-Interpreter ihren Wert ermittelt. Z.B. lässt sich für den Ausdruck

```
(> (car (quote (2 4))) (car (cdr (quote (1 2 3))))) )
```

die Auswertungsreihenfolge durch folgendes Ablaufprotokoll veranschaulichen:

```
(> (car (quote (2 4)) (car (cdr (quote (1 2 3))))) )
(car (quote (2 4))
  (quote (2 4))
  ==> (2 4)
==> 2
(car (cdr (quote (1 2 3))))
  (cdr (quote (1 2 3)))
    (quote (1 2 3))
    ==> (1 2 3)
  ==> (2 3)
==> 2
==> #f
```

1. (list (cdr (cdr (cdr (quote (1 2 3 4)))))
(car (cdr (quote (1 2 3 4))))))

2. `(if (< (car (quote (5 -3 4 -2))) 0) 0 1)`

Geben Sie für die folgenden Scheme-Ausdrücke an, zu welchem Wert sie evaluieren.

3. `(cons (cdr (quote (1 . 2)))
 (cdr (quote (1 2 . 3))))`

4. `(map (lambda (x) (if (pair? x) (cdr x) x))
 (quote (lambda (x) (if (pair? x) (cdr x) x))))`

5. `(filter (curry < 5)
 (reverse (quote (1 3 5 7 9))))`

6. `(filter (compose negative?
 (lambda (x) (- x 5)))
 (quote (1 3 5 7 9)))`

Aufgabe 2: Programmverstehen

15 Punkte

maximale Bearbeitungszeit: 80 Minuten

Was berechnen die folgenden Funktionen? Reimplementieren Sie jeweils ein analoges Prädikat in Prolog. Diskutieren Sie Unterschiede und Gemeinsamkeiten der beiden Implementationen.

1. `(define (foo1 x y)
 (if (null? x)
 #t
 (if (null? y)
 #f
 (if (eq? (car x) (car y))
 (foo1 (cdr x) (cdr y))
 (foo1 x (cdr y))))))`

```
2. (define (foo2 x y)
    (if (null? x)
        y
        (if (member (car x) y)
            (foo2 (cdr x) y)
            (cons (car x) (foo2 (cdr x) y) ) ) ) )
```

```
3. (define (foo3 x y)
    (if (null? x)
        (quote ())
        (if (member (car x) y)
            (foo3 (cdr x) y)
            (cons (car x) (foo3 (cdr x) y) ) ) ) )
```

```
4. (define (foo4 x)
    (letrec
      ((foo4a (lambda (x y)
                 (if (null? x) y
                     (if (> (car x) y)
                         (foo4a (cdr x) (car x) )
                         (foo4a (cdr x) y) ) ) ) )
      (foo4a (cdr x) (car x)) ) )
```

Hinweis: `letrec` ist eine Variante von `let`, die auch die Verwendung rekursiver Funktionsdefinitionen unterstützt.

```
5. (define (foo5 x y)
    (if (or (null? x) (null? y))
        0
        (+ (* (car x) (car y))
            (foo5 (cdr x) (cdr y)) ) )
```

Aufgabe 3: Programmentwicklung

6 Punkte

maximale Bearbeitungszeit: 40 Minuten

Überlegen Sie sich eine geeignete Repräsentation für die PEANO-Zahlen in Scheme. Reimplementieren Sie die Prädikate `lt/2`, `integer2peano/2` und `add/3` für das Rechnen mit Peano-Zahlen als Scheme-Funktionen. Diskutieren Sie Unterschiede und Gemeinsamkeiten zwischen den Prolog- und Scheme-Implementationen.