XIV. Übungsblatt "Informatik I" WS 95/96

Prof. Gert Smolka, Christian Schulte

Ausgabe: 2. Februar 1996

Abgabe: 9. Februar 1996 vor der Vorlesung (Programme per E-Mail)

Aufgabe XIV.1 (Erzwingen von Auswertung (4)) Schreiben Sie in Scheme 3 eine Prozedur

```
(erzwinge-tief x)
```

die alle von `x` durch eventuell mehrmalige Anwendung von `car` und `cdr` erreichbaren Knoten erzwingt. Um die Sache einfach zu machen, dürfen Sie annehmen, daß der von `x` aus erreichbare Graph nicht zirkulär ist. Testen Sie Ihre Prozedur mit dem Scheme 2+3 Interpretierer.

Aufgabe XIV.2 (Delay und force (4)) Erweitern Sie den Scheme 2+3 Interpretierer so, daß er die zwei Sonderformen

```
(delay x)
(force x)
```

richtig behandelt (also mit `delay` ein Versprechen baut und mit `force` ein Versprechen auswertet). Denken Sie daran, `delay` und `force` als neue reservierte Symbole einzuführen.
Zeigen Sie, daß Sie in Scheme 3 `delay` und `force` als Prozeduren definieren können. Argumentieren Sie, warum dies in Scheme 2 nicht geht.

Aufgabe XIV.3 (Verzögerte Auswertung und Zuweisung (4)) Programme, die sowohl verzögerte Auswertung als auch Zuweisung benutzen, können sehr schwer zu verstehen sein. Überzeugen Sie sich davon, indem sie in dem Ausdruck

```
(new ys
   (seq (:= ys (new mul (new x
                (seq (:= x 2)
                     (seq (:= mul (lambda (y)
                                    (seq (:= x (* x y)) x)))
                          (cons (mul 2)
                             (cons (mul 3)
                                (cons (mul 4) ()))))))))))
        ???))
```

die Fragezeichen durch die folgenden Ausdrücke ersetzen:

1. `(+ (car ys) (car (cdr ys)))`
2. `(+ (car (cdr ys)) (car ys))`
3. `(seq (+ (car (cdr (cdr ys))) 1) (+ (car (cdr ys)) (car ys)))`
4. `(seq (car (cdr (cdr ys))) (+ (car (cdr ys)) (car ys)))`
5. `(seq (+ 1 (car (cdr ys))) (+ (car (cdr ys)) (car ys)))`
6. `(seq (eq? 1 (car ys)) (+ (car (cdr ys)) (car ys)))`

Geben Sie den Wert an, zu dem der resultierende Ausdruck dann in Scheme 3 evaluiert. Welchen Wert hat `x` nach der Auswertung?

Aufgabe XIV.4 (Ströme in Scheme 3 (4)) Schreiben Sie ein Scheme 3 Programm, daß den Strom der Tripel positiver ganzer Zahlen i, j, k, so daß $i + j > k$ ist, erzeugt. Wenn Sie dies auf die einfachste Weise täten, wobei i, j und k jeweils alle positiven ganzen Zahlen durchliefen, würde Ihr Programm äußerst ineffizient. Warum?
Benutzen Sie daher die Tatsache, daß k in dem Intervall zwischen 1 und $i + j$ liegen muß.

Aufgabe XIV.5 (Sequentialisierung (4)) Zeigen Sie, daß in Scheme 2 die Sequentialisierung als eine Prozedur

```
(begin x y)
```

definierbar ist, die die Sonderform `(seq ...)` nicht benutzt.
Warum ist dies in Scheme, so wie es im Buch beschrieben ist, nicht möglich?

Aufgabe XIV.6 (Sequentialisierung (4)) Nehmen Sie an, daß in Scheme 3 die Prozedur

```
(define begin
  (lambda (x y)
    (if (eq? (eq? x x) (eq? y y)) y (()))))
```

definiert ist. Geben sie eine Definition für `x` an, so daß

```
(seq (cdr x) x)
```

zu `(5 ("VERSPRECHEN" (cons 2 ())))` und

```
(begin (cdr x) x)
```

zu `(5 2)` auswertet.
Wie müßte man die Prozedur `werte-aus-sequentialisierung` im Scheme 2+3 Interpretierer abändern, damit

```
(begin a b)
```

und

```
(seq a b)
```

für beliebige Ausdrücke äquivalent sind? (Nebenbei: die zu `(begin a b)` äquivalente Definition von (seq a b) scheint die bessere zu sein.)

Der Scheme 2+3 Interpretierer auf den ausgeteilten Blättern verläßt sich darauf, daß `map` von links nach rechts vorgeht: Dies ist in MIT Scheme leider nicht so. Eine korrigierte Version des Interpretierers finden Sie im WWW:

`http://ps-www.dfki.uni-sb.de/courses/info-i/scheme-3.scm`