1. Übungsblatt „Programmiersprachen“ WS 97/98

Leif Kornstaedt, Prof. Gert Smolka

Ausgabe: 30. Oktober 1997 **Abgabe:** 6. November 1997

Aufgabe I.1: Syntax ($2 + 5$ Punkte)

Gegeben sei folgende Zeichenfolge:

```
(define fac
  (lambda (n)
    (if (< n 2)
      1
      (* n (fac (- n 1))))))
```

a. Zerlegen Sie diese Zeichenfolge in eine Wortfolge entsprechend den in der Vorlesung angegebenen lexikalischen Regeln von Scheme-1.

b. Zeichnen Sie den abstrakten Syntaxbaum, der aus der in Teilaufgabe a gewonnenen Wortfolge aufgebaut wird. Markieren Sie dabei die Knoten mit den Symbolen `define`, `lambda`, `formals` (für die formale Argumentliste), `if` und `app` (für Applikationen).

Aufgabe I.2: Auswertung von Ausdrücken (7 Punkte)

Werten Sie folgendes Programm mithilfe der in der Vorlesung angegebenen Transitionen vollständig aus.

```
(define f (lambda (n) (if (< n 1) 17 7)))
(f 2)
```

Aufgabe I.3: Listeniteration ($4 + 3$ Punkte)

a. Schreiben Sie eine dreistellige Scheme-Prozedur `foldr`, die als Eingabe eine Liste $(x_1, \ldots, x_n)$, eine zweistellige Prozedur f und einen Wert z erhält. `foldr` soll den Wert von

$$(f\ x_1\ \ldots\ (f\ x_{n-1}\ (f\ x_n\ z))\ \ldots)$$

berechnen und zurückgeben. Beispielsweise wird

```
(foldr '(1 2 3) + 7)
```

zu $(+\ 1\ (+\ 2\ (+\ 3\ 7))) = 13$ ausgewertet.

b. Drücken Sie die vordefinierte Prozedur `map` (für die in der Vorlesung eine äquivalente Scheme-Prozedur vorgestellt wurde) durch `foldr` aus.

Aufgabe I.4: Freie Variablen ($2 + 4 + 6$ Punkte)

a. Welches sind die freien Variablen des folgenden Lambda-Ausdrucks?

```
(lambda (n) (if (< n 2) 1 (* n (fac (- n 1)))))
```

b. Definieren Sie formal die Menge der freien Variablen $FV(A)$ eines Scheme-1-Ausdrucks A sowie eines Lambda-Ausdrucks.

c. Schreiben Sie ein Scheme-Programm, das einen Lambda-Ausdruck als durch Listen dargestellten Baum als Eingabe erhält und als Ergebnis die Menge seiner freien Variablen liefert.

   Hinweis: Repräsentieren Sie Mengen von Variablen durch Listen von Symbolen. Schreiben Sie zunächst Prozeduren `adjoin` (Hinzufügen eines Elements zu einer Menge, falls es noch nicht enthalten ist), `set-union` (Bilden der Vereinigungsmenge) und `set-difference` (Bilden der Differenzmenge) unter Verwendung von `member`. Diese verwenden Sie dann, um die Prozedur `freevars` zu formulieren.

Starten von Scheme auf den CIP-Rechnern

Auf den CIP-Rechnern kann durch den Befehl

```
~kornstae/bin/scm
```

eine Scheme-Implementierung gestartet werden. Bis das Scheme-System durch die Eingabe von Control-D oder die Auswertung von `(quit)` beendet wird, werden wiederholt Scheme-Ausdrücke über die Tastatur eingelesen, ausgewertet und das Resultat ausgegeben.

Scheme kann auch unter dem Emacs-Editor gestartet werden. Hierzu fügen Sie beispielsweise folgende Zeilen in Ihre `.emacs`-Datei ein und starten Ihren Emacs neu:

```
(setq load-path (cons "/home/stud/kornstae/emacs" load-path))
(require 'my-scheme-mode)
```

Wenn eine Datei mit Endung `.scm` geöffnet wird, sind damit (unter anderem) folgende Tastenkürzel definiert:

`C-c C-r`	Starten des Scheme-Systems
`C-c C-h`	Beenden des Scheme-Systems
`M-r`	Auswerten der markierten Region
`M-l`	Auswerten der Zeile, in der sich der Cursor befindet
`C-c C-p`	Auswerten des Absatzes, in dem sich der Cursor befindet
`M-RET`	Auswerten des gesamten Buffers

Bei Problemen wenden Sie sich bitte an Leif Kornstaedt, `kornstae@ps.uni-sb.de`.