Informatik III WS2004/2005
Institut Intelligente Systeme
Leiter: Prof. Dr. E. Lehmann
Dietmar Lippold

Übungsblatt 11

Besprechungstermin: 24.01.05 - 28.01.05

Aufgabe 1

Beantworten Sie folgende Fragen zu funktionalen Sprachen im Allgemeinen und ML im Besonderen:

1. Was versteht man unter *polymorphen Funktionen oder Prozeduren*?
2. Was versteht man unter *Pattern–Matching*?
3. Was ist der Unterschied zwischen *statischer Typisierung* und *dynamischer Typisierung*?
4. Was versteht man unter *Typinferenz* und *Typkonstruktoren*?
5. Was ist bei ML der Unterschied zwischen den Bezeichnern *fun* und *fn*?

Aufgabe 2

Im Grundstudiumspool ist das Moskauer ML installiert. Rufen Sie es mit `mosml` auf.

Hinweise zur Eingabeschleife des Moskauer ML:

- `-` ist der Prompt.
- Jede Eingabe muß mit `;` beendet werden.
- `>` bezeichnet die Antwort des Rechners.
- `quit();` beendet das ML.

1. Lesen Sie Kapitel 1 von Robert Harpers *Introduction to Standard ML* [1]. Welche der dort genannten 8 Eigenschaften sind bei Scheme gleich, welche anders?
2. Testen Sie folgende Ausdrücke:

```
3 + 4;          3.0 + 4.0;        3.0 + 4;          val x = 15;
x;              x + 5;            x + 5.0;          fun quad x = x * x;
```

3. Testen Sie folgende Definitionen und Ausdrücke:

```
val wert1 = 1000;
wert1;
fun summe wert2 = wert1 + wert2;
summe(300);
```

[1] Link auf `http://www.informatik.uni-stuttgart.de/ifi/is/Lehre/Vorlesung/info3.html` im Abschnitt Literatur

```
val wert1 = 2000;
summe(300);
fun summe wert2 = wert1 + wert2;
summe(300);
```

Was hat ML da gemacht? Welche Konzepte (welche Art von Bezeichnern oder welche Art des Umgangs damit) aus anderen Sprachen entspricht dem? Welches Konzept (welche Art des Umgangs mit Bezeichnern) fehlt in ML?

4. Testen Sie folgende Listen und Funktionen:

```
[1,2,3];                          1::[2,3];
fn n => n * n * n * n;            (fn n => n * n * n * n) 5;
fun car (head::_) = head;         car [1,2,3];
fun cdr (_::rest) = rest;         cdr [1,2,3];
```

Aufgabe 3

Betrachten Sie folgendes ML–Programm:

```
fun quick[]      = []
  | quick[x]     = [x]
  | quick(a::bs) = (* the head "a" is the pivot *)
    let fun partition(left,right,[]): real list =
            (quick left)@(a::quick right)
          | partition(left,right,x::xs)         =
            if x <= a then partition (x::left, right,xs)
                      else partition (left,x::right,xs)
    in partition([],[],bs) end;
> val quick=fn:real list -> real list

- quick[7.3,3.9,5.0,1.6,8.7,4.2,6.3,2.1]
> val it = [1.6,2.1,3.9,4.2,5.0,6.3,7.3,8.7] : real list
```

Versuchen Sie die Funktion zu verstehen. Testen sie darin vorkommende, Ihnen unbekannte Funktionen oder schlagen Sie sie nach.

1. Was tut die Funktion?
2. Welche Eigenschaft von ML verwendet diese Prozedur, die es in Scheme so nicht gibt?
3. Definieren Sie eine äquivalente Prozedur in Scheme.

Aufgabe 4 (schriftlich)

Prozeduren, deren Rumpf Variablen aus einer äußeren Umgebung (die nicht die globale Umgebung ist) verwenden, nennt man *Closures*. Da die Bindungen an die Variablen aus der äußeren Umgebung länger leben als der Rumpf der Prozedur, kann man so Objekte mit lokalen Zuständen (in den Variablen der äußeren Umgebung gespeichert) und Funktionalität (durch den Prozedurrumpf realisiert) erzeugen. Dieser Effekt wurde beispielsweise schon in Aufgabe 2 von Blatt 5 benutzt.

```
(define (erzeuge-zaehler)
  (let ((zustand 0))
    (lambda ()
      (set! zustand (+ zustand 1))
      zustand )))
```

1. Geben Sie die Ergebnisse von Ausdruck 1 und 4 – 7 an.

```
> (erzeuge-zaehler)              ;; Ausdruck 1
> (define z1 (erzeuge-zaehler))
> (define z2 (erzeuge-zaehler))
> (z1)                           ;; Ausdruck 4
> (z2)                           ;; Ausdruck 5
> (z1)                           ;; Ausdruck 6
> (z2)                           ;; Ausdruck 7
```

2. Erweitern Sie `erzeuge-zaehler` so, daß der erzeugte Zähler ein Argument erhält. Auf die Eingabe `'reset` wird der Zähler auf 0 gesetzt, auf jede andere Eingabe wird der Zähler hochgezählt. Geben Sie ein Aufrufbeispiel an.

3. Definieren Sie eine Prozedur `zcar`, die sich für Listen wie `car` verhält und zusätzlich einen Zähler hochzählt. Bei Eingabe des Symbols `'stand` soll sie den Zählerstand zurückgeben und bei Eingabe von `'reset` den Zähler auf 0 setzen.

4. Definierten Sie eine Prozedur `(zaehl-fn fn)`, die eine beliebige einstellige Prozedur `fn` als Argument erhält und dafür eine mitzählende Prozedur analog zu `zcar` zurückgibt. Diese soll wie `zcar` die Argumente `'stand` und `'reset` verarbeiten können.

   Was ist dabei unschön?