Funktionale Programmierung SS 2013 - Lösung 5

Prof. Dr. U. Kastens
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
10.06.2013

Lösung zu Aufgabe 1

a)

```
val dec1 = secr op- 1;
val dec2 = secl ~1 op+;
```

b)

```
val endmarker = secr op@ [255];
```

c)

```
val lparen = secl "(" op^;
val rparen = secr op^ ")";
val paren = lparen o rparen;
```

Lösung zu Aufgabe 2

a) Wir betrachten

```
fun summation f 0 = 0
|   summation f m = f (m-1) + summation f (m-1)
;
```

im Vergleich zu

```
fun summation607 f m =
    let fun sum (i, z):real =
        if i=m then z else sum (i+1, z + (f i))
    in sum (0, 0.0) end;
```

Kriterien des Vergleichs

1. Typ

   ```
   summation: fn (int -> int) -> int -> int
   summation607: fn (int -> real) -> int -> real
   ```

2. Berechnete Funktion
   `summation f m` berechnet in beiden Fällen die Summe f(0) + f(1) + ... + f(m-1).
3. Berechnungsreihenfolge
   summation607 berechnet 0.0 + f(0) + f(1) + ... + f(m-1) von links nach rechts.
   summation berechnet f(m-1) + ... + f(0) von links nach rechts.
4. Zeitkomplexität
   Die Komplexität ist gleich. Die Funktion `summation607` könnte durch die end-rekursive Hilfsfunktion Laufzeitvorteile haben.

b)

$$\sum_{k=0}^{4} (k*k) = 30$$

$$\sum_{k=0}^{4} \sum_{i=0}^{k-1} (2*i+1) = 30$$

Lösung zu Aufgabe 3

```
fun curry f a b = f(a,b);
val double = curry op* 2;
double 21;

fun uncurry f(a,b) = f a b;
val sum = uncurry summation;
sum(fn k => k*k, 5);
```

Lösung zu Aufgabe 4

a)

```
fun tausender l = map (fn (stadt,einw) => (stadt,einw div 1000)) l;
```

Bemerkung: Natürlicher wäre eine Definition der Form

```
val tausender = map (fn (stadt,einw) => (stadt,einw div 1000));
```

Die wird aufgrund von Beschränkungen des SML-Typsystems vom SML-Interpretierer nicht akzeptiert. Ausführliche Erläuterung dieses und ähnlicher Phenomene finden Sie unter SML's Value Restriction.

b)

```
fun halbemio l = filter (fn (s,e) => e > 500000) l;
```

c)

```
fun nichthalbemio l = filter (op not o (fn (s,e) => e > 500000)) l;
```

d)

```
fun totalewz l = foldl (fn ((stadt,ewz),erg) => ewz+erg) 0 l;
```

e)

```
fun dabei stadt l = exists (fn (a,b) => a = stadt) l;
```

f)

```
fun millionenstaedte l = all (fn (a,b) => b >= 1000000) l;
```

g) Die Funktionsweise der Funktion `ruhrewz`:

```
fun ruhrewz l r = filter (fn (s,e) => exists (secr op= s) r) l;
```

Die Funktion filtert diejenigen Elemente aus einer Einwohnerzahlenliste, deren Städtenamen in der Liste enthalten ist, der als zweiter Parameter übergeben wird. Um dies zu prüfen, wird das Funktional `exists` mit einem Prädikat verwendet, das die Gleichheit der Städtenamen prüft.