Funktionale Programmierung SS 2013 - Aufgabenblatt 5

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

Aufgabe 1 (Die Funktionale `secl` und `secr`)

Die Funktionale sind gegeben durch

```
fun secl x f y = f(x,y);
fun secr f y x = f(x,y);
```

a) Die Funktion `dec` vermindert ihr Argument um 1. Geben Sie 2 alternative Definitionen von `dec` an: eine, die `secl` verwendet und eine, die `secr` verwendet.

b) Benutzen Sie `secl` oder `secr` um die Funktion `endmarker` zu definieren, die das Element `255` an eine Liste anhängt.
Bsp.: `endmarker [1,2,3]` liefert `[1,2,3,255]`.

c) Benutzen Sie `secl` und `secr` um Funktionen `lparen` und `rparen` zu definieren, die öffende bzw. schließende runde Klammern vor bzw. hinter eine Zeichkette anfügen.
Bsp.: `rparen "a."` liefert `"a.)"`.

Definieren Sie dann die Funktion `paren` als Komposition dieser beiden Funktionen. Die Funktion `paren` soll runde Klammern um eine Zeichenkette zu setzen. Der Kompositions-Operator `o` ist in SML vordefiniert.

Aufgabe 2 (Reihenberechnung als Schema)

Betrachten Sie die folgende Definition:

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

a) Vergleichen Sie diese Version der Funktion `summation` mit der auf Folie 607. Erstellen Sie dazu zunächst eine Liste nützlicher Vergleichskriterien.

b) Geben Sie die Formeln der berechneten Summen und die Werte für die folgenden Ausdrücke an:

```
summation (fn k => k*k) 5;
summation (summation (fn i => 2*i+1)) 5;
```

Aufgabe 3 (Curry und Uncurry)

Schreiben Sie die Funktionale `curry f a b` und `uncurry f`, die zu einer zweistelligen Funktion in Tupelform die entsprechende Funktion in Curryform bzw. zu einer zweistelligen Curryfunktion die Funktion in Tupelform liefern.

Beispiel-Anwendungen:

```
val double = curry op* 2;
double 21;

val sum = uncurry summation;
sum(fn k => k*k, 5);
```

Aufgabe 4 (Anwendung der Standard-Funktionale)

Verwenden Sie die Definitionen

```
fun map f nil     = nil
  | map f (x::xs) = (f x)::map f xs;

fun filter pred nil     = nil
  | filter pred (x::xs) = if pred(x) then x::filter pred xs else filter pred xs;

fun foldl f e nil   = e
  | foldl f e (x::xs) = foldl f (f (x,e)) xs;

fun exists pred nil     = false
|   exists pred (x::xs) = (pred x) orelse (exists pred xs);

fun all pred nil     = true
|   all pred (x::xs) = (pred x) andalso (all pred xs);

val ewznrw =  [("Koeln", 1007119),           (* (Stadt, Einwohnerzahl) *)
               ("Duesseldorf",588735),
               ("Dortmund",580444),
               ("Essen",574635),
               ("Duisburg",489599),
               ("Bochum",374737)];
```

um die folgenden Funktionen zu definieren.

a) Die Funktion `tausender` reproduziert eine Einwohnerzahlenliste (z.B. `ewznrw`), in dem die Einwohnerzahlen in Tausend Einwohner umgerechnet werden.

   Bsp: `tausender ewznrw` liefert

   ```
   [("Koeln",1007),("Duesseldorf",588),("Dortmund",580),("Essen",574),("Duisburg",489),("Bochum",374)]
   ```

b) Die Funktion `halbemio` reproduziert eine Einwohnerzahlenliste beschränkt auf die Einträge mit Einwohnerzahlen größer gleich 500000.

c) Die komplementäre Funktion `nichthalbemio` reproduziert eine Einwohnerzahlenliste beschränkt auf die Einträge mit Einwohnerzahlen kleiner als 500000. Definieren Sie sie, indem Sie die Filterfunktion von `halbemio` durch Komposition mit der Negationsfunktion `op not` negieren.

d) Die Funktion `totalewz` berechnet aus einer Einwohnerzahlenliste die Gesamtzahl der Einwohner.

e) Die Funktion `dabei` bekommt einen Städtenamen und eine Einwohnerzahlenliste und prüft, ob die Stadt in der Liste vertreten ist.

   Bsp: `dabei "Paderborn" ewznrw` liefert `false`.

f) Die Funktion `millionenstaedte` prüft, ob alle Städte in einer Einwohnerzahlenliste mehr als eine Million Einwohner haben.

g) Erläutern Sie die Funktionsweise der Funktion `ruhrewz` aus folgendem Beispiel:

   ```
   val ruhrgebiet = ["Bochum", "Bottrop", "Dortmund", "Duisburg", "Essen", "Gelsenkirchen", "Hagen"];
   fun ruhrewz l r = filter (fn (s,e) => exists (secr op= s) r) l;
   ruhrewz ewznrw ruhrgebiet;
   ```