Funktionale Programmierung SS 2013 - Aufgabenblatt 7

Prof. Dr. U. Kastens
Institut für Informatik, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn
08.07.2013

Aufgabe 1 (Breitensuche in Lösungsbäumen)

Wir betrachten die Breitensuche in Lösungsbäumen wie auf Folie 712 vorgestellt:

```
(* sequences and function 'take' *)
datatype 'a seq = Nil | Cons of 'a * (unit -> 'a seq);
fun take(xq, 0 ) = []
|   take(Cons(x,xq),n) = x :: take(xq(),n-1)
;
Control.Print.printLength := 150; (* Mehr Ausgabe im interaktiven Modus *)

(* tree description *)
fun next n = ["0"^n, "1"^n];

(* predicate --> task *)
fun pred n = ...;

(* function 'breadthFirst' from slide 7.12 *)
fun breadthFirst (next, pred) root =
   let fun bfs [] = Nil
       |   bfs (x::xs) = if pred x then Cons (x, fn () => bfs (xs @ next x))
                                   else bfs (xs @ next x)
   in bfs [root] end;
```

a) Beschreiben Sie den Suchraum, der von der Funktion `next` aufgespannt wird.

b) Definieren Sie die Funktion `pred` so, dass alle Strings, die den Teil-String `101010` beinhalten, zur Lösung gehören. Verwenden Sie dann `take`, um die ersten 16 Lösungen zu betrachten.

   Hinweis: Die Funktion `String.isSubstring s1 s2` prüft, ob `s1` ein Teil-String von `s2` ist.

Aufgabe 2 (Erste Schritte mit Haskell)

Speichern Sie diese 3 verschiedene Implementierungen der Fakultäts-Funktion in einer Haskell-Datei `Fac.hs`:

```
factorial1 0 = 1
factorial1 n = n * factorial1 (n - 1)

factorial2 n = if n > 0 then n * factorial2 (n-1) else 1

factorial3 n = product [1..n]
```

a) Kommentieren Sie die Funktions-Definitionen sinnvoll.

b) Ergänzen Sie eine Definition, die die in Haskell vordefinierte Funktion `foldl` verwendet, siehe auch Folie 611.

c) Verwenden Sie die Fakultätsfunktionen im interaktiven Haskell-Interpreter `ghci`: die Datei mit den Funktionsdefinitionen wird auf der Kommandozeile übergeben, die Funktionsanwendungen werden interaktiv eingegeben, Bsp.:

```
ghci Fac.hs
> Factorial1 10
```

Aufgabe 3 (Arbeiten mit nicht-endlichen Listen in Haskell)

a) Definieren Sie die Liste `odds` der ungeraden natürlichen Zahlen.

   Tipp: Die Funktion `take (Int -> [a] -> [a])` zum Erzeugen endlicher Listenanfänge ist vordefiniert.

b) Definieren Sie `odds` alternativ mit Hilfe des vordefinierten Funktionals `iterate` (ohne 's'), vergl. auch Folie 705.

c) Schreiben Sie eine Funktion `prefsums`, die die Liste der Präfix-Summen einer Liste liefert.

   Bsp.: `prefums odds` ist `[1,4,9,16,25,36,49,64,81,100, ...]`.

Aufgabe 4 (Näherungsverfahren mit konvergenten Folgen)

Der Kehrwert 1/a einer Zahl a > 0 lässt sich durch die Iterationsfolge

$$x_{n+1} = 2 * x_n - a * x_n^2$$

bestimmen. Die Folge konvergiert, wenn der Startwert x_0 zwischen 0 und 1/a liegt.

Definieren Sie die Haskell Funktion `kehrwert a`, die diese Iterationsfolge generiert und den Kehrwert auf 10^{-9} annähert.

Hier ist die Definition der benötigten Funktion `within`, siehe auch Folie 805:

```
within eps (h1 : (h2: t)) = if abs(h1-h2) < eps
                            then h2
                            else within eps (h2:t)
```

Tipp: Das Haskell-Literal für 10^{-9} ist `1e-9`.