Andreas Podelski
Fachbereich 14 – Informatik
Universität des Saarlandes,
Max-Planck-Institut für Informatik

Programmiersprachen (WS 1996/97)

21. November 1996

Übungsblatt 5

Abdelwaheb Ayari
Hubert Baumeister

Aufgabe 1: (3 Punkte)

1. Definiere folgende Funktionen
   - `take : int -> 'a stream -> 'a list`
     `take n s` liefert eine Liste der ersten `n` Elemente von `s` zurück.
   - `drop : int -> 'a stream -> 'a stream`
     `drop n s` liefert den Strom `s` ohne die ersten `n` Elemente zurück.
   - `fromList : 'a list -> 'a stream`
     `fromList` konvertiert eine Liste in einen Strom.
   - `fromInteger : (int*int) -> (int*int) stream`
     `fromInteger(n,m)` liefert den unendlichen Strom $[(n,m),\ (n,m+1),\ \ldots]$
2. Definiere eine Funktion `Flatten 'a stream stream -> 'a stream`, die einen Strom von Strömen zu einem einzigen Strom vereinigt. Beachte, daß die Vereinigung fair ist, d.h. daß jeder Strom in der Eingabe in endlicher Zeit Elemente zu dem Ergebnis beisteuert, auch wenn einige Ströme in der Eingabe unendlich sind.
   Gegeben seien folgende Funktion:
   ```
   fun stream1 0 = Nil
   | stream1 n = Cons(fromInteger(n,1),fn () => stream1 (n-1))
   ```
   und den unendlichen Strom:
   ```
   val stream2 =
   let fun makestream n = Cons(fromInteger(n,1),fn () => makestream (n+1))
   in makestream 0 end
   ```
   Werte folgende Ausdrücke aus:
   ```
   - val s = stream1 100;
   - take 20 s;
   - take 20 (Flatten s);
   - take 20 (drop 10 s);
   - take 20 (drop 10 (Flatten s));
   - val s = stream2;
   - take 10 s;
   - take 10 (Flatten s);
   - take 10 (drop 100 s);
   - take 10 (drop 100 (Flatten s));
   ```

Aufgabe 2: (3 Punkte)
Die Aufgabe 2 ist die Aufgabe 1 auf dem 6. Übungsblatt.

Aufgabe 3: (4 Punkte)
Löse Aufgabe 3 vom dritten Übungsblatt aber stelle zusätzlich sicher, daß der Binärbaum *fast vollständig ist*. Ein Binärbaum heißt *fast vollständig* wenn ein k existiert so daß jedes Blatt auf Stufe k oder Stufe $k+1$ ist.

Zum Testen steht in `/home/stud/abdelwaheb/pub/random.sml` eine Funktion `make_random_list : int * int -> int list` zur Verfügung. Ein Liste mit n Elementen zwischen 1 und m wird erzeugt durch `make_random_list(n,m)`.
(Hinweis: Implementiere einen Heap: Die Grundidee ist, daß mit Hilfe von fast vollständigen Binärbäumen ein *Array* implementiert werden kann, das logarithmische Zugriffszeit auf die Elemente bietet. Sei z.B. $i = 9$ ein Index eines Arrays. Wir betrachten die Binärexpansion von $i = 1001$ und gehen bei einer 0 in den linken Unterbaum und bei einer 1 in den rechten. Beachte, daß so jedem Element des Arrays ein eindeutiger Knoten im Baum zugeordnet ist. Falls wir auf ein Element des Arrays zugreifen wollen, dann können wir die Binärexpansion von rechts nach links lesen und somit haben wir einen einfachen Test ob wir nach rechts oder links gehen müssen, nämlich: Gehe nach rechts wenn $i \bmod 2 = 1$, sonst nach links.)