XII. Übungsblatt "Informatik I" WS 95/96

Christian Schulte, Prof. Gert Smolka

Ausgabe: 19. Januar 1996

Abgabe: 26. Januar 1996 vor der Vorlesung (Programme per E-Mail)

Aufgabe XII.1 (Strom von natürlichen Zahlen (3))

1. Vervollständigen Sie die folgende Definition eines Stromes `nats` von natürlichen Zahlen:

   ```
   (define nats (cons-stream 1 (stream-map ??? nats)))
   ```

2. Geben Sie Definitionen von `stream-map` und `nats` an, die die Notationen `cons-stream` und `delay` *nicht* verwenden. Benutzen Sie dazu die Prozeduren `force` und `memo-proc` aus der Vorlesung. Siehe auch die WWW-Seite:

   ```
   http://ps-www.dfki.uni-sb.de/courses/info-i/programme.html
   ```

   Diese Seite enthält neben den Definitionen von `force` und `memo-proc` ein Beispiel aus der Vorlesung und wie man Ströme in MIT Scheme benutzen kann.

Aufgabe XII.2 (Skalieren von Strömen (1)) Schreiben Sie eine Prozedur `(stream-scale n s)`, die einen Strom zurückliefert, der die Elemente des Stroms von Zahlen `s` mit `n` multipliziert enthält.

Aufgabe XII.3 (Mischen von Strömen (2)) Gegeben seien zwei Ströme `s1` und `s2` von Zahlen in aufsteigender Reihenfolge ohne Wiederholungen. Schreiben Sie eine Prozedur `(stream-merge s1 s2)` die einen Strom zurückliefert, der die Elemente von `s1` und `s2` in aufsteigender Reihenfolge ohne Wiederholungen enthält. (Siehe dazu auch Aufgabe VIII.3)

Aufgabe XII.4 (Hamming-Zahlen (6)) Ein berühmtes Problem, das zuerst von R. Hamming formuliert wurde, ist alle Zahlen in aufsteigender Ordnung ohne Wiederholungen aufzuzählen, die nur die Primfaktoren 2, 3 und 5 haben. Ein offensichtliches, aber auch offensichtlich sehr ineffizientes, Verfahren ist, alle Zahlen aufzuzählen und zu testen, ob sie nur die Primfaktoren 2, 3 und 5 haben.
Wir wollen das Problem dadurch lösen, indem wir einen Strom von Hamming-Zahlen berechnen. Dabei sind die beiden folgenden Tatsachen wichtig:

1. Eine Zahl ist genau dann eine Hamming-Zahl, wenn sie sich in der Form $2^i 3^j 5^k$ darstellen läßt, wobei i, j, k natürliche Zahlen sind.

2. Die Menge der Hamming-Zahlen H ist die einzige Menge, die die folgende rekursive Gleichung erfüllt:
   $$H = \{1\} \cup \{2h | h \in H\} \cup \{3h | h \in H\} \cup \{5h | h \in H\}$$

Geben Sie mit Hilfe der Prozeduren `stream-merge` und `stream-scale` aus den vorherigen Aufgaben eine Definition eines Stromes `h` an, der alle Hamming-Zahlen in aufsteigender Ordnung enthält.

Aufgabe XII.5 (Umordnen von Strömen (6)) Die Prozedur `(reorder s1 s2)` ordnet Elemente vom Strom `s1` in der Ordnung an, wie sie vom Strom `s2` angegeben wird. Beginnt `s1` mit den Elementen 4, 13, 2, 8 und beginnt `s2` mit den Elementen 3, 1, 4, 2, dann soll der Ergebnisstrom von `reorder` gerade mit den Elementen 2, 4, 8, 13 beginnen (Das erste Element des Ergebnisstroms soll gerade das dritte Element von `s1` sein, usw.)

1. Implementieren Sie `reorder`. Benutzen Sie dabei die Prozedur `stream-nth` für Ströme aus der Vorlesung.

2. Beschreiben Sie die Ströme zu denen `(reorder nats s)` und `(reorder s nats)` evaluieren, wobei `nats` der Strom aus der Aufgabe XII.1 und `s` ein beliebiger Strom von natürlichen Zahlen ist.

3. Sei `fibs` ein Strom der Fibonacci-Zahlen. Was sind die ersten sieben Zahlen des Ausgabestromes von `(reorder fibs fibs)`?

Aufgabe XII.6 (Akkumulieren über Listen (6)) Im folgenden wollen wir eine Prozedur `foldr` betrachten, die Elemente einer Liste bezüglich einer zweistelligen Prozedur und eines Startwertes akkumuliert:

```
(define (foldr p i xs)
   (if (null? xs) i
       (p (car xs) (foldr p i (cdr xs)))))
```

Dabei berechnet `(foldr p i (`x_1 `...` x_n`))` gerade `(p` x_1 `(` ... `(p` x_n `i)` ... `))`. Daher auch der Name: Die Operation `p` wird von rechts nach links geklammert.

1. Schreiben Sie analog zu `foldr` eine Prozedur `(foldl p i (`$x_1 \ldots x_n$`))`, die von "links klammert" und `(p (p ... (p i` x_1`) ...)` x_n`)` berechnet.

2. Welche Eigenschaft muß die durch die Prozedur `p` implementierte Funktion haben, so daß `(foldl p i xs)` und `(foldr p i xs)` zu äquivalenten Ergebnissen evaluieren.

3. Sei `xs` eine Liste von Zahlen. Was berechnet die Summe von `xs` effizienter: `(foldr + 0 xs)` oder `(foldl + 0 xs)`? Wieso?

4. Sei `xs` eine Liste von Listen. Welche Berechnung ist effizienter: `(foldr append () xs)` oder `(foldl append () xs)`? Wieso?