Funktionale Programmierung, SS 2018, Blatt 1

Fabian Kunze, Steven Schäfer, Kathrin Stark,
Prof. Dr. Gert Smolka
https://courses.ps.uni-saarland.de/fp_ss18/

Grundlagen funktionaler Programmierung, Auswertungsstrategien, Streams

Aufgabe 1.1 (Rückblick) Erinnern Sie sich an die zentralen Ideen zur funktionalen Programmierung aus der Vorlesung Programmierung 1:

- Polymorphismus,
- höherstufige Prozeduren,
- Muster und Konstruktoren,
- Listen, Bäume, Faltungen,
- (abstrakte) Datenstrukturen.

(Dies entspricht in etwa den Kapiteln 3, 4, 6, 7 und 14 aus *Programmierung – Eine Einführung in die Informatik mit Standard ML* von Gert Smolka.)

Aufgabe 1.2 (Programmieren in Haskell) Lesen Sie Kapitel 1–3 in *A Gentle Introduction to Haskell* (www.haskell.org/tutorial/) von Hudak, Peterson und Fasel, insbesondere Abschnitte 3.3 und 3.4 zu nicht-strikter ('lazy' oder 'by need') Auswertung. Verwenden Sie einen Haskell Interpreter, um mit einigen Beispielen aus dem Text zu experimentieren.

Aufgabe 1.3 (Streams) Wir bezeichnen eine unendliche Liste als *stream*. Abschnitt 3.4 in *A Gentle Introduction...* zeigt, wie streams als zyklische Listen definiert werden können. Geben Sie Definitionen

a) für $twos = [2, 2, 2, ...]$,

b) für $powers = [1, 2, 4, 8, 16, ...]$, bestehened aus den Potenzen 2^n,

c) für $fact = [1, 1, 2, 6, 24, ...]$, bestehend aus den n-ten Fakultäten $n!$.

Aufgabe 1.4 (Sieve of Eratosthenes) Ein Verfahren, mit dem Primzahlen bestimmt werden können, ist das folgende: Ausgehend von $[2, 3, 4, ...]$ werden zunächst die Vielfachen 4,6,8,... von 2 gestrichen. Anschliessend werden die Vielfachen 6,9,12,... von 3 (der nächst größeren Zahl der verbleibenden Elemente) gestrichen, sofern sie noch vorkommen, dann die Vielfachen von 5.... Wird dieser Prozess fortgeführt, so ergibt sich als 'Grenzwert' der stream $[2, 3, 5, 7, ...]$ aller Primzahlen.

Realisieren Sie dieses Verfahren in Haskell mittels zweier Deklarationen $primes :: [Int]$ und $sieve :: [Int] \to [Int]$.

Aufgabe 1.5 (Hamming sequence)

a) Deklarieren Sie in Haskell den stream $hamming = [1, 2, 3, 4, 6, 8, 9, 12, ...]$ aller Vielfachen $2^n 3^m$ von 2 und 3, aufsteigend angeordnet.
Hinweis 1 Nutzen Sie aus, dass $1 = 2^0 3^0$ ein Element des streams ist, und dass k enthalten ist genau dann wenn $2k$ und $3k$ ebenfalls enthalten sind.
Hinweis 2 Verwenden Sie eine Hilfsprozedur $merge :: [Int] \to [Int] \to [Int]$, zum Mischen zweier sortierter streams.

b) Verallgemeinern Sie die obige Deklaration zu einer Deklaration $ghamming :: [Int] \to [Int]$, welche zu einer nichtleeren endlichen Liste von natürlichen Zahlen $p_1, \ldots, p_n$ alle Vielfachen ${p_1}^{m_1} \ldots {p_n}^{m_n}$ in aufsteigender Reihenfolge anordnet.

Aufgabe 1.6 (Auswertungsstrategien in Standard ML) Sei eine Prozedurdeklaration *fun p x = e* gegeben. Erinnern Sie sich daran, dass die Auswertungsreihenfolge in Standard ML im Gegensatz zu Haskell 'strikt' ist: Bei einer Prozeduranwendung $p\,a$ wird das Argument a zu einem Wert v ausgewertet, bevor der Prozedurrumpf e ausgeführt wird (mit dem formalen Parameter x der Prozedur ersetzt durch v).

a) Überlegen Sie sich an dem Beispiel *fun const1 x = 1*, dass bedarfsgesteuerte und strikte Auswertung zu unterschiedlichem Verhalten führen können.

b) Implementieren Sie (potentiell) unendliche Listen in Standard ML.

c) *Zusatzaufgabe:* Implementieren Sie Haskell's bedarfsgesteuerte Auswertung für Listen möglichst originalgetreu in Standard ML.

Hinweis: In Standard ML ist *fn* () ⇒ *e* ein Wert. Der Rumpf *e* wird erst evaluiert, sobald die Abstraktion auf () angewendet wird.
Hinweis 2: Sie benötigen für die beiden Implementierungen in Standart ML vermutlich einen neuen, induktiven Datentypen, der Funktionen als Werte enthält.