Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
Lars Noschinski, Dr. Jasmin Blanchette, Dmitriy Traytel

WS 2012/13
15.01.2013
Abgabe: 22.01.2013, 15:30

Einführung in die Informatik 2
12. Übung

Aufgabe G12.1 Rätselraten ums richtige Reduzieren

Identifizieren Sie die Redexe in den folgenden Ausdrücken. Welche davon sind innermost, outermost, beides oder keins von beiden?

1. `1 + (2 * 3)`
2. `(1 + 2) * (2 + 3)`
3. `fst (1 + 2, 2 + 3)`
4. `fst (snd (1, 2 + 3), 4)`
5. `(\x -> 1 + x) (2 * 3)`

Aufgabe G12.2 Arbeitsscheue Auswertung von Argumenten

Sei das folgende Haskell-Programm gegeben:

```
inf :: a -> [a]
inf x = x : inf x

f :: [Int] -> [Int] -> [Int] -> Int
f (x:xs) (y:ys) zs | x > y = y
f (x1:x2:xs) ys (z:zs) = x1
```

Betrachten Sie die Aufrufe:

```
f (inf (1+0)) (inf (1+1)) (inf (1+2))
f (inf (1+2)) (inf (1+1)) (inf (1+0))
f (inf (1+0)) [] (inf (1+1))
```

Wie weit muss Haskell jeweils die Argumente auswerten, um Ihnen das Ergebnis dieser Funktionsaufrufe anzeigen zu können?

Aufgabe G12.3 Faule Fibonacci-Funktionen

In dieser Aufgabe wollen wir Fibonacci-Zahlen als unendliche Liste darstellen.

1. Definieren Sie eine Variable `fib1 :: [Integer]`, die die Liste aller Fibonacci-Zahlen enthält. Zur Erinnerung: Die Fibonacci-Zahlen sind die Folge $(f_n)_{n\in\mathbb{N}}$ mit $f_0 = 0$, $f_1 = 1$ und $f_n = f_{n-1} + f_{n-2}$ für alle $n \geq 2$.

   Hinweis: Nutzen Sie die Funktionen `zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]` und `tail :: [a] -> [a]` um `fib1` ohne Hilfsfunktion zu implemntieren. Beachten Sie,

dass auch die Definition einer Variable rekursiv sein kann (vergleiche `inf` aus Aufgabe G12.2).

2. Geben Sie eine Liste von Schritten an, um die ersten 5 Elemente von `fib1` auszuwerten.
3. Wenn wir eine Fibonacci-ähnliche Liste mit beliebigen Startwerten haben wollen, bietet sich die Funktion

```
fib2 :: Integer -> Integer -> [Integer]
fib2 m n = m : fib2 n (m + n)
```

   an. Geben Sie die Auswertungsschritte an, die notwendig sind, um die ersten 4 Elemente von `fib2 0 1` anzuzeigen.
4. *(optional)* Warum wird sowohl `fib1 !! n` als auch `fib2 0 1 !! n` mit weniger Schritten ausgewertet als `fib n`? `fib :: Integer -> Integer` sei dabei wie folgt definiert:

```
fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)
```

Aufgabe G12.4 Haufenweise Hammingzahlen (optional)

Definieren Sie die unendliche sortierte Liste `hamming :: [Integer]`, die alle positiven Zahlen enthält, deren Primfaktoren ≤ 5 sind. Mit anderen Worten: alle Zahlen, die sich als $2^i \cdot 3^j \cdot 5^k$ für $i, j, k \geq 0$ schreiben lassen.

Aufgabe H12.1 Wesentlicher Wust von Wörtern (5 Punkte)

Gesucht ist eine Funktion `wordsOf :: [Char] -> [[Char]]`, die als Parameter eine Liste von Buchstaben (das Alphabet) erhält und alle endlichen Strings, die aus Buchstaben des Alphabets bestehen, in einer unendlichen Liste zurückgibt. Wenn das Alphabet keine Duplikate enthält, soll die Ausgabeliste ebenfalls keine Duplikate enthalten. Außerdem ist es wichtig, dass die Strings der Ausgabeliste der Länge nach sortiert sind. Strings gleicher Länge sollen lexikographisch bezüglich der Eingabeliste angeordnet sein. Beispiele:

```
take 16 (wordsOf "ab") ==
  ["","a","b","aa","ab","ba","bb",
   "aaa","aab","aba","abb","baa","bab","bba","bbb","aaaa"]

take 10 (wordsOf "ba") ==
  ["","b","a","bb","ba","ab","aa","bbb","bba","bab"]

take 14 (wordsOf "abc") ==
  ["","a","b","c",
   "aa","ab","ac", "ba","bb","bc","ca","cb","cc","aaa"]
```

Dagegen ist die Ausgabe `["","a","aa","aaa","aaaa", ..]` für die Eingabe `[a, b]` verboten!

```
take 5 (wordsOf "ab") /= ["","a","aa","aaa","aaaa"]
```

Aufgabe H12.2 Sequenzen suffixlos summieren (5 Punkte)

Definieren Sie eine Funktion `sumOfPrefixes :: Num a => [a] -> [a]`, die für die (evtl. unendliche) Eingabeliste `[a1,a2,a3,..]` die Ausgabe `[0, a1, a1+a2, a1+a2+a3,..]` produziert. Zum Beispiel:

```
sumOfPrefixes [1,1,1,1,1] == [0,1,2,3,4,5]
sumOfPrefixes [1,2,3,4,5] == [0,1,3,6,10,15]
take 1000 (sumOfPrefixes [1,1 ..]) == take 1000 [0 ..]
```

Aufgabe H12.3 Fixes Finden mit Filtern (5 Punkte)

Die Funktion `find :: (a -> Bool) -> [a] -> Maybe a` aus `Data.List` gibt das erste Element der Liste zurück, das das übergegebene Prädikat erfüllt.

1. Implementieren Sie eine eigene Version `findByFilter :: (a -> Bool) -> [a] -> Maybe a` mithilfe von `filter` (aus `Prelude`).
2. Wie effizient ist Ihre Funktion im Vergleich zu `find`? Probieren Sie ein paar Beispiele und erklären Sie Ihre Ergebnisse.
3. Schreiben sie *einen* angebrachten QuickCheck-Test.

Aufgabe H12.4 Strings strikt ohne Substrings (5 Punkte, Wettbewerbsaufgabe)

In Aufgabe H12.1 wurde eine Funktion `wordsOf` beschrieben. Sie erhält als Parameter eine Liste von Zeichen (das Alphabet) und gibt einer Liste aller Wörter, die aus den Zeichen des Alphabets bestehen, zurück.

Die Wettbewerbsaufgabe diese Woche besteht darin, eine Funktion `censoredWordsOf :: Eq a => [a] -> [[a]] -> [[a]]` zu entwickeln, die `wordsOf` um die Möglichkeit, unerwünschte Teilwörter auszuschließen, erweitert:

Bei einer systematischen Enumerierung besteht die Gefahr, dass unerwüschte Teilwörter mitten in den Wörtern auftauchen. Das zweite Argument gibt die Liste von solchen Ausnahmen an. Alle Wörter, die ein unerwüschtes Teilwort enthalten, sollen ausgelassen werden.

Beispiele:

```
take 10 (censoredWordsOf "ab" []) ==
  ["","a","b","aa","ab","ba","bb","aaa","aab","aba"]

take 10 (censoredWordsOf "ba" []) ==
  ["","b","a","bb","ba","ab","aa","bbb","bba","bab"]
```

```
take 50 (censoredWordsOf "esx" ["sex","xxx"]) ==
  ["","e","s","x","ee","es","ex","se","ss","sx","xe","xs","xx",
   "eee","ees","eex","ese","ess","esx","exe","exs","exx","see",
   "ses","sse","sss","ssx","sxe","sxs","sxx","xee","xes","xex",
   "xse","xss","xsx","xxe","xxs","eeee","eees","eeex","eese",
   "eess","eesx","eexe","eexs","eexx","esee","eses","esse"]

take 10 (censoredWordsOf "ha" ["ah","ha"]) ==
  ["","h","a","hh","aa","hhh","aaa","hhhh","aaaa","hhhhh"]
```

Für den Wettbewerb zählt die Effizienz, vor allem die Skalierbarkeit. Ihre Implementierung sollte mit langen (bzw. kurzen) Alphabet- und Ausnahmelisten sowie mit Duplikaten und anderen redundanten Einträgen möglichst effizient vorgehen.

Lösungen, die sich effizienztechnisch ähnlich verhalten, werden bezüglich ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden. Um teilzunehmen müssen Sie Ihre Lösung innerhalb der Kommentare {-WETT-} und {-TTEW-} eingeben.

Wichtig: Wenn Sie diese Aufgabe als Wettbewerbsaufgabe abgeben, stimmen Sie zu, dass Ihr Name ggf. auf der Ergebnisliste auf unserer Internetseite veröffentlicht wird. Sie können diese Einwilligung jederzeit widerrufen, indem Sie eine Email an `fp@fp.in.tum.de` schicken. Wenn Sie nicht am Wettbewerb teilnehmen, sondern die Aufgabe allein im Rahmen der Hausaufgabe abgeben möchten, lassen Sie bitte die {-WETT-} ... {-TTEW-} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.