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

WS 2013/14
21.01.2014
Abgabe: 28.01.2014, 15:30

Einführung in die Informatik 2
13. Übung

Aufgabe G13.1 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 G13.2 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 G13.3 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 implementieren. Beachten Sie,

dass auch die Definition einer Variable rekursiv sein kann (vergleiche `inf` aus Aufgabe G13.2). Die Lösung finden Sie auf Folie 345.

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 G13.4 Viel verwürfeln (optional)

Definieren Sie eine Funktion `mix :: [[a]] -> [a]`, die eine (unendliche) Liste von (unendlichen) Listen nimmt und sicherstellt, dass jedes Eingabeelement in der (unendlichen) Ausgabeliste enthalten ist. Die Reihenfolge spielt dabei keine Rolle.

Formal: Mit einer Eingabeliste $xs = [[x_{1,1}, x_{1,2}, \ldots], [x_{2,1}, x_{2,2}, \ldots], \ldots]$ muss für alle m, n gelten: `elem` $x_{m,n}$ (`mix` xs).

Aufgabe H13.1 Wesentlicher Wust von Wörtern (6 Punkte)

Gesucht ist eine Funktion `wordsOf :: [a] -> [[a]]`, 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 H13.2 Sequenzen skippen (8 Punkte)

Definieren Sie eine Funktion `skipper :: [a] -> [[a]]`, die für die (evtl. unendliche) Eingabeliste l eine Liste von nichtleeren Teillisten $t_1, t_2, ...$ zurückgibt, so dass die Teilliste t_k jeweils genau alle k-ten Elemente von l enthält.

```
skipper [1..4] = [[1..4], [2,4], [3], [4]]
skipper [1..6] = [[1..6], [2,4,6], [3,6], [4], [5], [6]]
take 3 (skipper [1..]) = [[1..], [2,4..], [3,6..]]
```

Aufgabe H13.3 Fixes Finden mit Filtern (6 Punkte)

Die Funktion `lookup :: Eq a => a -> [(a, b)] -> Maybe b` aus der Standardbibliothek kennen Sie bereits.

1. Implementieren Sie eine eigene Version `lookupByFilter :: Eq a => a -> [(a, b)] -> Maybe b` mithilfe von `filter`.

   Hinweis: Die Funktion `listToMaybe` könnte hilfreich sein.

2. Wie effizient ist Ihre Funktion im Vergleich zu `lookup`? Probieren Sie ein paar Beispiele und erklären Sie Ihre Ergebnisse.

3. Schreiben sie *einen* angebrachten QuickCheck-Test.

Aufgabe H13.4 Komprimierung ohne Kompromisse (Wettbewerbsaufgabe, 0 Punkte)

In dieser **vorletzten**[1] Wettbewerbsaufgabe geht es um Datenkompression. In der Vorlesung wurde der Huffman-Algorithmus, der einen optimalen Präfixcode erzeugt, vorgestellt. Es gibt aber effizientere Codes, die keine Präfixcodes sind. Sie werden z.B. in Programmen wie `zip` und `gzip` verwendet.

Der Master of Competition wurde von ~~der NSA~~m Bundeskanzleramt beauftragt, die Komprimierungsalgorithmen des Kanzlerinnen-Handys auf ihre Effizienz zu prüfen. Dabei musste der MC feststellen, dass die in Objective-C geschriebene Huffman-Implementierung nicht nur unverständlich ist, sondern auch zu große Dateien erzeugt.

Helfen Sie dem MC, indem Sie zwei Haskell-Funktionen implementieren:

```
compress :: ByteString -> ByteString
decompress :: ByteString -> ByteString
```

[1] Dieses Mal meint es der MC ernst.

Für alle `bs` sollte die folgende Eigenschaft gelten:

```
decompress (compress bs) == bs
```

Sie dürfen annehmen, dass `decompress` nur mit von `compress` erzeugten Strings aufgerufen wird.

Bei der Punktevergabe ist die Größe des erzeugten Strings entscheidend (je kleiner, desto besser). Der MC wird 100 Dateien (sowohl Text- als auch binäre, sowohl kleine als auch riesige Dateien) komprimieren. Sollte ein(e) einzige(r) Teilnehmer(in) eine Lösung einreichen, die für jeden Test einen genauso effizienten oder effizienteren Code als alle anderen eingereichten Lösungen erzeugt, wird diese(r) mit 40 Wettbewerbspunkten (anstatt der üblichen 30) belohnt.

Bei der Jagd nach Effizienz sollten Sie die Korrektheit Ihrer Funktion nicht aus den Augen verlieren. Lösungen, die die Korrektheitstests nicht bestehen, bekommen keine Punkte. 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.