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

WS 2013/14
22.10.2013
Abgabe: 29.10.2013, 15:30

Einführung in die Informatik 2
2. Übung

Aufgabe G2.1 Listenkomprehensionen

Benutzen Sie Listenkomprehensionen um die folgenden Funktionen zu implementieren:

1. Schreiben Sie eine Funktion `all_sums :: [Integer] -> [Integer] -> [Integer]`. Das Ergebnis von `all_sums xs ys` soll eine Liste sein, die für jede Kombination von x in `xs` und y in `ys` den Wert $x + y$ enthält.
2. Schreiben Sie eine Funktion `evens :: [Integer] -> [Integer]`, die alle ungerade Zahlen aus einer Liste entfernt. Sie können die Funktion `mod` verwenden.
3. Schreiben Sie eine Funktion `n_lists :: [Integer] -> [[Integer]]`, die eine Liste von Zahlen so auf eine Liste von Listen abbildet, dass n auf die Liste von 1 bis n abgebildet wird.
4. Schreiben Sie mit nur einer Listenkomprehension die Funktion

   ```
   all_even_sum_lists :: [Integer] -> [Integer] -> [[Integer]]
   ```

   die für Eingaben `xs` und `ys` die Ausgabe `n_lists (evens (all_sums xs ys))` berechnet.

   Formulieren Sie einen QuickCheck-Test, der Ihre Funktion mit der angegeben Referenz vergleicht.

Aufgabe G2.2 Mengenoperationen

Wir verwenden Listen um Mengen darzustellen. Schreiben Sie Funktionen für die üblichen Mengenoperationen:

1. Vereinigung: `union :: [Integer] -> [Integer] -> [Integer]`
2. Schnitt: `intersection :: [Integer] -> [Integer] -> [Integer]`
3. Differenz: `diff :: [Integer] -> [Integer] -> [Integer]`.

   Sie können die Funktion `elem :: Integer -> [Integer] -> Bool` verwenden, die bestimmt ob ein Wert Element einer Liste ist.
4. Für den Schnitt von Mengen gilt in der Mathematik $x \in A \cap B \longleftrightarrow x \in A \wedge x \in B$. Prüfen sie diese Eigenschaft analog für `intersection`.
5. Wie lässt sich `elem` mit Listenkomprehensionen schreiben?

Hinweis: Verwenden Sie `quickCheckWith (stdArgs { maxSize=4, maxDiscardRatio=40 })` um ihre Properties zu testen. Dies konfiguriert QuickCheck, nur kleine Parameter zu generieren und mehr Werte auszuprobieren. Warum ist dies hier sinnvoll? (Sie können `verboseCheckWith` verwenden, um sich alle generierten Parameter anzuschauen und mit den Parametern zu spielen.)

Aufgabe G2.3 Brüche

Wir können Brüche als Tupel darstellen, d.h. das Tupel `(a,b)` repräsentiert den Bruch a/b. Dabei sollen zwei Tupel als gleich behandelt werden, wenn sie den gleichen Bruch repräsentieren, d.h. `(1,2)` und `(3,6)` repräsentieren den gleichen Wert.

1. Schreiben Sie eine Funktion

   ```
   eq_frac :: (Integer,Integer) -> (Integer,Integer) -> Bool
   ```

   die entscheidet, ob zwei Brüche gleich sind.
2. Prüfen Sie mittels QuickCheck-Tests, ob sich die `eq_frac`-Funktion korrekt verhält. Sinnvolle Eigenschaften sind z.B. Symmetrie und $m/n = (m \cdot k)/(n \cdot k)$.

Aufgabe G2.4 Potenzen von 2

Die Funktion

```
pow2 :: Integer -> Integer
pow2 0 = 1
pow2 n | n > 0 = 2 * pow2 (n - 1)
```

implementiert $n \mapsto 2^n$ für $n \geq 0$. Für einen gewissen n benötigt die Berechnung n Schritte:

$$2^{100} = 2 \cdot 2^{99} = 2 \cdot 2 \cdot 2^{98} = 2 \cdot 2 \cdot 2 \cdot 2^{97} = \cdots = \overbrace{2 \cdot 2 \cdot \ldots \cdot 2}^{100 \text{ mal}} \cdot 1$$

Gesucht ist eine effizientere Implementierung, die maximal $\lceil 2 \log_2 n \rceil$ Schritte benötigt. Die Gleichungen $2^{2n} = (2^n)^2$ und $2^{2n+1} = 2 \cdot 2^{2n}$ können dabei hilfreich sein. Zum Beispiel:

$$\begin{aligned} 2^{100} &= (2^{50})^2 = ((2^{25})^2)^2 = ((2 \cdot 2^{24})^2)^2 = ((2 \cdot (2^{12})^2)^2)^2 = ((2 \cdot ((2^6)^2)^2)^2)^2 \\ &= ((2 \cdot (((2^3)^2)^2)^2)^2)^2 = ((2 \cdot (((2 \cdot 2^2)^2)^2)^2)^2)^2 \end{aligned}$$

Aufgabe H2.1 Zählen (4 Punkte)

Implementieren Sie eine Funktion `count :: [Char] -> Char -> Integer`, die zählt, wie oft ein Buchstabe in einem Wort vorkommt. Zum Beispiel:

```
count "Abrakadabra" 'a' == 4
count "Abrakadabra" 'x' == 0
```

Sie dürfen hierbei Basisfunktionen der `List`-Standardbibliothek verwenden, insbesondere auch `sum` und `genericLength`.[1]

[1] Ärgerlicherweise liefert die in den Folien erwähnte `length`-Funktion einen Maschinen-`Int` und keinen unbegrenzten `Integer` zurück.

Aufgabe H2.2 Aschenputtel (6 Punkte)

Der Prinz hat nach dem Tanz den Schuh seiner unbekannten Tanzpartnerin in der Hand und möchte herausfinden, wer sich hinter der Maske verbirgt. Glücklicherweise hatte seine Mutter, die Königin, schon vor einiger Zeit für Steuerzwecke eine Liste erstellen lassen, die die Schuhgröße eines jeden Bürgers des Reiches enthält.

1. Da Sie selber auf großem Fuße leben und auf einen königlichen Steuererlass spekulieren, bieten Sie dem Prinz an, die möglichen Kandidaten aus dieser Liste herauszusuchen. Schreiben Sie dafür eine Haskell-Funktion
   `reverseLookup :: Integer -> [(String,Integer)] -> [String]`,
   die ihnen für eine gegebene Schuhgröße die Liste aller Bürger mit dieser Schuhgröße berechnet.

   Zum Beispiel:

   ```
   reverseLookup 36
       [ ("A. Puttel", 36)
       , ("Gretel", 42)
       , ("Der kleine Muck", 45)
       , ("Cin de Rella", 36)
       , ("Haensel", 38)
       ] ==
   ["A. Puttel", "Cin de Rella"]
   ```

   Verwenden Sie lediglich Listenkomprehensionen.

2. Bevor Sie dem Prinzen die Ergebnisse überreichen, sollten Sie sicherstellen, dass Ihre Funktion richtig arbeitet. Schreiben Sie dazu QuickCheck-Properties, die testen, ob

   a) jeder Bürger mit der passenden Schuhgröße gefunden wird und ob

   b) jeder Bürger im Ergebnis auch mit der passenden Schuhgröße in der Eingabe enthalten ist.

   Zur Erinnerung: Wenn ihre Properties Parameter haben, generiert QuickCheck beim Testen automatisch Eingabewerte für diese. Nutzen Sie dies!

   Hinweis: Verwenden Sie `quickCheckWith (stdArgs { maxSize=4 })` um ihre Properties zu testen. Dies konfiguriert QuickCheck, nur kleine Parameter zu generieren. Warum ist dies hier sinnvoll? (Sie können `verboseCheckWith` verwenden, um sich alle generierten Parameter anzuschauen).

Aufgabe H2.3 Strings der Länge n (5 Punkte)

Gesucht ist eine Funktion `wordsOfLength :: [Char] -> Integer -> [[Char]]`, die als Parameter eine Liste von Buchstaben (das Alphabet) und eine nicht-negative Zahl n erhält und alle Strings der Länge n, die aus Buchstaben des Alphabets bestehen, in einer Liste zurückgibt. Ihre Ausgabeliste soll keine Duplikate enthalten. Zum Beispiel:

```
wordsOfLength "abc" 2 ==
    ["aa","ab","ac","ba","bb","bc","ca","cb","cc"]
wordsOfLength "ab" 3 ==
    ["aaa","aab","aba","abb","baa","bab","bba","bbb"]
wordsOfLength "aab" 3 ==
    ["aaa","aab","aba","abb","baa","bab","bba","bbb"]
wordsOfLength "ab" 4 ==
    ["aaaa","aaab","aaba","aabb","abaa","abab","abba","abbb",
     "baaa","baab","baba","babb","bbaa","bbab","bbba","bbbb"]
```

Sie können die Funktion `nub :: [a] -> [a]` verwenden, um gegebenenfalls Duplikate aus der Eingabeliste zu entfernen (das `a` in der Typsignatur bedeutet, dass die Funktion auf Listen beliebigen Typs anwendbar ist).

Aufgabe H2.4 Palindromemordnilap (5 Punkte, Wettbewerbsaufgabe)

Gesucht ist eine Funktion `palindromesOfRadius :: [Char] -> Integer -> [[Char]]`, die die Liste der Palindrome über ein Alphabet erzeugt. Damit diese Liste endlich ist, gibt das zweite Argument eine obere Schranke auf den Radius der Palindrome an.

Ein *Palindrom* ist ein Word `w`, für das `w == reverse w` gilt. Zum Beispiel sind `ada`, `otto`, `hanah` und `habibah` Palindrome. Der *Radius* eines Palindroms ist die Länge des sich wiederholenden Wortteils. Folglich haben `otto` und `hanah` einen Radius von 2. Der Radius eines Palindroms der Länge n beträgt $\lfloor n/2 \rfloor$.

Die Liste der Palindrome darf keine Duplikate enthalten, auch wenn das gegebene Alphabet nicht duplikatfrei ist. Die Liste muss auch der Länge nach sortiert sein, d.h. kurze Palindrome sollen vor längeren stehen.

Beispiele:

```
palindromesOfRadius "" 0 == [""]
palindromesOfRadius "" 10 == [""]
palindromesOfRadius "a" 0 == ["","a"]
palindromesOfRadius "ab" 0 == ["","a","b"]
palindromesOfRadius "a" 1 == ["","a","aa","aaa"]
palindromesOfRadius "ab" 1 ==
  ["","a","b","aa","bb","aaa","aba","bab","bbb"]
palindromesOfRadius "otto" 2 == ["","o","t","oo","tt","ooo",
  "oto","tot","ttt","oooo","otto","toot","tttt","ooooo",
  "ootoo","ototo","ottto","tooot","totot","ttott","ttttt"]
palindromesOfRadius "xxxxxxxxxxxxxxxxx" 3 ==
  ["","x","xx","xxx","xxxx","xxxxx","xxxxxx","xxxxxxx"]
elem "saippuakauppias" (palindromesOfRadius "psukia" 7)
```

Für gleichlange Palindrome ist die relative Reihenfolge in der Liste nicht spezifiziert. Somit darf der Aufruf `palindromesOfRadius "ab" 0` sowohl `["","a","b"]` als auch `["","b","a"]` zurückliefern.

Sie dürfen für Ihre Implementierung die folgenden Funktionen aus der `Data.List`-Standardbibliothek benutzen:

```
elem :: a -> [a] -> Bool    -- elem 2 [1,3,2] == True
nub :: [a] -> [a]           -- nub [1,3,1,2,2] == [1,3,2]
reverse :: [a] -> [a]       -- reverse [1,3,2] == [2,3,1]
sort :: [a] -> [a]          -- sort [1,3,2] == [1,2,3]
```

Der Buchstabe `a` stellt einen beliebigen Typ (z.B. `Integer`, `Char` oder `[Char]`) dar. Andere Bibliothekfunktionen sowie Funktionen, die für H2.1–H2.3 definiert wurden (z.B. `count`), sind ebenfalls zulässig.

Für den Wettbewerb zählt die Tokenanzahl (je kleiner, desto besser; siehe Aufgabenblatt 1). Lösungen, die gleichwertig bezüglich der Tokenanzahl sind, werden im Hinblick auf ihre Effizienz verglichen. Die vollständige Lösung (außer `import`-Direktiven und bereits vorhanden Funktionen wie `count`) muss innerhalb von Kommentaren {`-WETT-`} und {`-TTEW-`} stehen. Zum Beispiel:

```
import Data.List
import Test.QuickCheck

{-WETT-}
palindromesOfRadius :: [Char] -> Integer -> [[Char]]
palindromesOfRadius alphabet radius = ...
{-TTEW-}
```

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.