Technische Universität München
Institut für Informatik
Prof. Tobias Nipkow, Ph.D.
L. Noschinski, L. Hupel, Dr. J. Blanchette, M. Eberl

WS 2014/15
17.10.2014
Abgabe: 24.10.2014

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 `allSums :: [Integer] -> [Integer] -> [Integer]`. Das Ergebnis von `allSums 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 `nLists :: [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

   ```
   allEvenSumLists :: [Integer] -> [Integer] -> [[Integer]]
   ```

   die für Eingaben `xs` und `ys` die Ausgabe `nLists (evens (allSums 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 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

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

   die entscheidet, ob zwei Brüche gleich sind.

2. Prüfen Sie mittels QuickCheck-Tests, ob sich die `eqFrac`-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 Indizieren

Implementieren Sie eine Funktion `index :: [Char] -> Char -> Int`, die den Index eines Elements in einer Liste ermittelt. Falls das übergebene Zeichen nicht in der Liste vorkommt, soll die Länge der Liste zurückgegeben werden. Falls das übergebene Zeichen mehrfach vorkommt, soll der Index des ersten Vorkommens ermittelt werden.

```
index "Abrakadabra" 'a' == 3
index "Abrakadabra" 'x' == 11
```

Sie dürfen hierbei Basisfunktionen der `List`-Standardbibliothek verwenden, insbesondere auch `!!` (Indexzugriff auf Listen) und `length` (Länge von Listen).

Hinweis: Beachten Sie bitte, dass in dieser Aufgabe statt `Integer` der Typ `Int` verwendet wird, denn aus historischen Gründen nutzen viele Listenfunktionen aus der Standardbibliothek letzteren.

Aufgabe H2.2 Es kann nur einen geben

Der *Master of Competiton* (MC) speichert die Punkte der Studenten als Assoziationsliste vom Typ `[(String,Integer)]`, d. h. jeder Listeneintrag ist Paar aus einem Namen und der dazugehörigen Punktezahl.

Der MC möchte nun gerne wissen, ob es momentan einen Gleichstand gibt, also ob es zwei oder mehr Studenten gibt, die laut der Liste die gleiche Punkteanzahl haben. Leider ist der MC jedoch so beschäftigt, dass er keine Zeit hat, diese Funktion selbst zu implementieren und delegiert diese Aufgabe daher an seine Studenten, also Sie.

Implementieren Sie eine Funktion

```
hasDraws :: [(String,Integer)] -> Bool
```

die zu einer gegebenen Assoziationsliste zurückgibt, ob eine Punktzahl mehrmals vorkommt. Sie dürfen davon ausgehen, dass die Eingabeliste keine Namen mehrfach enthält.

Beispiel:

```
hasDraws
  [ ("Edsger Dijkstra", 44),
    ("Ada Lovelace", 45),
    ("Donald Knuth", 42),
    ("Grace Hopper", 43),
    ("Alan Turing", 42) ]
 == True
```

```
hasDraws
  [ ("Bertrand Russell", 42),
    ("Kurt Gödel", 43),
    ("Alonzo Church", 44) ]
 == False
```

Hinweis: Die vordefinierte Funktion `nub :: [a] -> [a]`, die mehrfache Vorkommnisse in einer Liste löscht, könnte nützlich sein (das `a` in der Typsignatur bedeutet hierbei nur, dass sich `nub` auf beliebige Listen anwenden lässt).

Aufgabe H2.3 Zahlenkombinationen

Gesucht ist eine Funktion `intLists :: (Integer, Integer) -> Integer -> [[Integer]]`, die als Parameter ein Paar (m, n) und eine Zahl k bekommt und als Ergebnis alle Listen der Länge k zurückgibt, die folgende Eigenschaft erfüllen: Für jedes Element i der Liste gilt $m \leq i \leq n$.

Das Ergebnis soll keine Liste mehrfach enthalten. Zum Beispiel:

```
intLists (2,3) 2 ==
    [[2,2],[2,3],[3,2],[3,3]]
intLists (0,5) 1 ==
    [[0],[1],[2],[3],[4],[5]]
intLists (2,3) (-1) == []
intLists (3,2) 2 == []
```

Aufgabe H2.4 Bits auspacken (Wettbewerbsaufgabe)

Echte Programmierer wissen, wie man die Bits richtig anpackt – und auspackt! Darum geht es in dieser Aufgabe. Es geht aber auch um Magie: Aus zwei Zahlen wird eine, und umgekehrt! Programmierer, die gleichzeitig Zauberer sind, sind in der Wirtschaft sehr begehrt ($$$), und sollten Sie sich für eine akademische Laufbahn entscheiden ($ↄ), werden Sie mit diesen Künsten auch dort gut zurechtkommen.

Um es konkret zu machen: Gegeben seien zwei natürliche Zahlen, x und y. Sie wollen diese beiden Zahlen in einer Internet-Datenbank speichern, irgendwo in der Cloud, aber es kostet doppelt so viel, zwei Zahlen zu speichern als nur eine (10 $ pro Monat statt 5 $). Wichtig dabei zu wissen: Sie sind im Auftrag einer Stuttgarter Firma.

Zum Glück sind Sie auf einen cleveren Trick gekommen, um Geld zu sparen. Die Zahlen x und y lassen sich binär als endliche Sequenzen von Bits (0 und 1) darstellen:

$$\begin{aligned} x &= x_m\, x_{m-1} \ldots x_2\, x_1 \\ y &= y_n\, y_{n-1} \ldots y_2\, y_1 \end{aligned}$$

Falls die beiden Zahlen nicht die gleiche Länge haben, kann man die kürzere Zahl vorne mit Nullen auffüllen. Nehmen wir also an, dass $m = n$. Die clevere Kodierung von x und y ist wie folgt:

$$y_m\, x_m\, y_{m-1}\, x_{m-1} \ldots y_2\, x_2\, y_1\, x_1$$

Dementsprechend ist die binäre Darstellung von $x = 12$ und $y = 35$

$$\begin{aligned} x &= (\quad 1100)_2 \\ y &= (100011)_2 \end{aligned}$$

und ihre Kodierung

$$(100001011010)_2 = 2138$$

Und das geht auch rückwärts! Wie? Das gilt es herauszufinden.

Implementieren Sie eine Funktion `decodeIntPairs :: [Integer] -> [(Integer, Integer)]`, die eine Liste von (wie oben beschrieben) kodierten Zahlen als Argument nimmt und eine Liste von Paaren von dekodierten Zahlen zurückgibt. Negative Zahlen in der Eingabenliste sollen einfach ignoriert werden.

Beispiele:

```
decodeIntPairs [] == []
decodeIntPairs [0] == [(0, 0)]
decodeIntPairs [2138] == [(12, 35)]
decodeIntPairs [1, -1, 2] == [(1, 0), (0, 1)]
decodeIntPairs [3, 4, 5, 6, 7] ==
  [(1, 1), (2, 0), (3, 0), (2, 1), (3, 1)]
decodeIntPairs [15, -255, 255, 65535] ==
  [(3, 3), (15, 15), (255, 255)]
```

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) muss innerhalb von Kommentaren {`-WETT-`} und {`-TTEW-`} stehen. Zum Beispiel:

```
import Data.List
import Test.QuickCheck

{-WETT-}
decodeIntPair :: Integer -> (Integer, Integer)
decodeIntPair z = ...

decodeIntPairs :: [Integer] -> [(Integer, Integer)]
decodeIntPairs zs = ...
{-TTEW-}
```

Hinweis: Implementieren Sie zuerst eine Funktion `decodeIntPair :: Integer -> (Integer, Integer)`, die eine nicht negative Zahl als Argument nimmt und sie dekodiert. Diese können Sie dann für die Implementierung von `decodeIntPairs` verwenden.

Hinweis: Sie dürfen den oberen Hinweis ignorieren, falls Sie ernsthaft am Wettbewerb teilnehmen wollen.

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.