Dr. Steffen Jost
Dr. Ulrich Schöpp

Ludwig-Maximilians-Universität München
Institut für Informatik
6. Mai 2019

Programmierung und Modellierung

Blatt 2

A2-1 Pattern-Matching In Aufgabe A1-3 haben wir folgende Funktionen betrachtet.

```
implies :: Bool -> Bool -> Bool
implies x y = if not x then True else y

foo :: Bool -> Bool -> Bool
foo x y = implies (not x) y && implies y (not x)
```

a) Implementieren Sie die Funktion `implies :: Bool -> Bool -> Bool` noch einmal mit Pattern-Matching, d.h. ohne `if-then-else` oder Funktionen wie `not`, `&&` oder `||` aus der Standardbibliothek zu verwenden.

b) Ändern Sie den Typ von `implies` so, dass die beiden Argumente in einem Tupel übergeben werden:

```
implies :: (Bool, Bool) -> Bool
```

Passen Sie Ihre Implementierung aus a) sowie `foo` entsprechend an.

A2-2 Pattern-Matching Implementieren Sie folgende Funktionen mit Pattern-Matching.

a) `third :: [Int] -> Int` soll das dritte Element einer Liste zurückgeben; wenn es kein drittes Element gibt soll 0 zurückgegeben werden.

Beispiele: `third [1,2,3,4] == 3` und `third [1,2] == 0`.

b) `last :: [Int] -> Int` soll das letzte Element einer Liste zurückgeben, bzw. 0 wenn die Liste leer ist. Beispiel: `last [1,2,3,4,5] == 5`

Verwenden Sie zur Implementierung die Funktion `reverse :: [Int] -> [Int]` aus der Standardbibliothek, die eine Liste herumdreht. Das letzte Element der Liste ist das erste Element der umgedrehten Liste.

c) `firstAndLast :: [Int] -> (Int, Int)` soll das erste und das letzte Element einer Liste zurückgeben. Wenn es solche Elemente nicht gibt, soll wieder 0 zurückgegeben werden. Beispiel: `firstAndLast [1,2,3,4,5] == (1, 5)`

Verwenden Sie zur Implementierung wieder `reverse`.

A2-3 Funktionen als Werte In Aufgabe H1-2 haben wir die Wertetafeln Boolescher Funktionen verglichen. Solche Aufgaben möchte man natürlich möglichst automatisieren.[1]

a) Definieren Sie Funktionen zum Vergleich ein- und zweistelliger Boolescher Funktionen.

```
same1 :: (Bool -> Bool) -> (Bool -> Bool) -> Bool
same2 :: (Bool -> Bool -> Bool) -> (Bool -> Bool -> Bool) -> Bool
```

Es ist eine gute Übung, die beiden Funktionen ohne List-Comprehensions zu definieren.

b) Eine Funktion `f` heiße kommutativ falls `f x y == f y x` für alle `x` und `y` gilt. Schreiben Sie eine Funktion, die eine zweistellige Boolesche Funktion auf Kommutativität prüft.

```
commutative :: (Bool -> Bool -> Bool) -> Bool
```

A2-4 Funktionen auf Strings Gegeben sei folgendes Programm.

```
import Data.Char
  -- importiere Funktionen für den Typ Char
  -- Beispiel: toLower 'A' == 'a'

f :: String -> Char -> Bool
f s x = [ y | y <- s, y == x] /= []

g :: (Char -> Bool) -> String -> String
g f s = [ x | x <- s,  not (f x)]
```

Was berechnen die Funktionen `f` und `g`? Was ist das Ergebnis folgender Beispiele?

```
g (f " aeiou") "Was soll der Unsinn?"
g (\x -> f " aeiou" (toLower x)) "Was soll der Unsinn?"
```

Überprüfen Sie mit GHCi.

H2-1 Pattern Matching (3 Punkte; Datei `H2-1.hs` als Lösung abgeben)
Implementieren Sie die folgenden Sortierfunktionen direkt mit Pattern-Matching und ohne die Verwendung von Bibliotheksfunktionen.

```
sort2 :: (Int, Int) -> (Int, Int)
sort3 :: (Int, Int, Int) -> (Int, Int, Int)
sortFirst3 :: [Int] -> [Int]
```

Die Funktionen `sort2` und `sort3` sollen Tupel von zwei bzw. drei Zahlen aufsteigend sortieren. Die Funktion `sortFirst3` soll die *ersten drei* Elemente einer gegebenen Liste aufsteigend sortieren. Der Rest der Liste soll unverändert bleiben. Wenn es weniger als drei Elemente gibt, sollen nur die vorhandenen sortiert werden. Beispiele:

```
sort2 (3, 1) == (1, 3)
sort3 (2, 3, 1) == (1, 2, 3)
sortFirst3 [3, 1] == [1, 3]
sortFirst3 [2, 3, 1] == [1, 2, 3]
sortFirst3 [2, 4, 3, 1, 5, 2] == [2, 3, 4, 1, 5, 2]
```

Hinweis: Schreiben Sie `{-# OPTIONS_GHC -Wincomplete-patterns #-}` in die erste Zeile Ihrer Haskell-Datei. Dann warnt der Compiler bei evtl. vergessenen Fällen im Pattern-Matching.

[1] Der Gleichheittest `==` ist für Funktionen nicht verfügbar.

Suchen Sie sich eine der beiden nachfolgenden Aufgaben aus!

Die volle Punktzahl von 6 Punkten für dieses Blatt kann also **entweder** durch korrekte Bearbeitung von H2-1 und H2-2 **oder** von H2-1 und H2-3 erreicht werden.

H2-2 Funktionen auf Strings (3 Punkte; Datei `H2-2.hs` als Lösung abgeben)
Um das ungewollte Spoilern von Filmen oder Serien in Textnachrichten zu vermeiden, kann man bestimmte Abschnitte des Texts vor versehentlichem Lesen schützen. Zum Beispiel kann man dort jeden Buchstaben durch den jeweils übernächsten Buchstaben des Alphabets ersetzen. Statt `"Franz jagt im komplett verwahrlosten Taxi quer durch Bayern"` würde man `"Htcpb lciv ko mqorngvv xgtycjtnquvgp Vczk swgt fwtej Dcagtp"` schreiben. Damit kann man den Text nicht mehr aus Versehen lesen, aber man kann die Ersetzung der Buchstaben leicht rückgängig machen, wenn man den Text doch sehen will.

Implementieren Sie diese Übersetzung und ihre Umkehrung in folgenden zwei Funktionen.

```
garble :: String -> String
ungarble :: String -> String
```

Die Funktion `garble` soll den Text unlesbar machen, indem sie jeden Buchstaben (a-z, A-Z, keine Umlaute oder Sonderzeichen) durch den übernächsten Buchstaben im Alphabet ersetzt. Kleinbuchstaben bleiben Kleinbuchstaben, Großbuchstaben bleiben Großbuchstaben. Alle anderen Zeichen bleiben unverändert. Die Funktion `ungarble` soll den ursprünglichen Text wieder herstellen. Beispiel:

```
garble "John Snow is Anakins's father." == "Lqp Upqy ku Cpcmkp'u hcvjgt."
ungarble "Lqp Upqy ku Cpcmkp'u hcvjgt." == "John Snow is Anakins's father."
```

Hinweise: Sie können neben den beiden Funktionen nach Bedarf auch eigene Hilfsfunktionen definieren. Sie können die Funktionen aus der Bibliothek `Data.Char` benutzen. Schreiben Sie dazu `import Data.Char` an den Anfang Ihres Programms. Insbesondere die Funktionen `ord`, `chr`, `isAsciiLower` (wahr für a-z, keine Umlaute), `isAsciiUpper` (wahr für A-Z, keine Umlaute) sollten nützlich sein.

H2-3 Funktionen auf Strings (3 Punkte; Datei `H2-3.hs` als Lösung abgeben)
Implementieren Sie folgende Funktionen auf Strings. Sie sollten keine weiteren Funktionen aus der Standardbibliothek verwenden.

a) `count :: Char -> String -> Int` soll zählen, wie oft ein Zeichen in einem String vorkommt. Beispiel `count 'n' "Unsinn" == 3`.

b) `freq :: String -> [(Char, Int)]` soll eine Liste zurückgeben, die für jeden Kleinbuchstaben (`'a'` bis `'z'`) seine Häufigkeit im gegebenen String angibt. Die Effizienz der Lösung spielt dabei in dieser Aufgabe noch keine Rolle.

   Beispiel: `freq "banana" == [('a',3),('b',1),('n',2)]`

Abgabe: Lösungen zu den Hausaufgaben können bis Montag, den 13. Mai über uni2work abgegeben werden.