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

WS 2013/14
12.11.2013
Abgabe: 19.11.2013

Einführung in die Informatik 2
5. Übung

Aufgabe G5.1 Mysteriöse Funktion

Beschreiben Sie in eigenen Worten das Verhalten des folgenden Ausdrucks:

```
head (tail (map gg (zip [xx, xx] [yy, yy])))
```

mit `gg`, `xx` und `yy` beliebig. Die Funktion `zip :: [a] -> [b] -> [(a, b)]` berechnet die Liste von Paaren der Elemente zweier paralleler Eingabelisten. Für `zip` gilt folgende Gleichung:

$$\texttt{zip}\ [x_1, \ldots, x_n]\ [y_1, \ldots, y_n] = [(x_1,\ y_1), \ldots, (x_n,\ y_n)]$$

Aufgabe G5.2 Iteration von Funktionsaufrufen

1. Schreiben Sie eine Funktion `iter :: Int -> (a -> a) -> a -> a`. Diese soll eine Zahl n, eine Funktion f und einen Wert x als Eingabe bekommen und f n-mal auf x anwenden. `iter` n f x berechnet dann $f^n(x)$. Negative n sollen wie $n = 0$ behandelt werden.

   Beispiel: `iter 3 sq 2 == 256`, wobei `sq x = x * x`

2. Nutzen Sie `iter`, um die folgenden Funktionen ohne Rekursion zu implementieren:

   a) Die Exponentialfunktion `pow :: Int -> Int -> Int` mit `pow` n $k = n^k$ (für alle $k \geq 0$).

   b) Die Funktion `drop :: Int -> [a] -> [a]` aus der Standardbibliothek nimmt eine Zahl k und eine Liste $[x_1, \ldots, x_n]$ und gibt die Liste $[x_{k+1}, \ldots, x_n]$ zurück.

   Implementieren Sie eine eigene Version dieser Funktion. Sie dürfen annehmen, dass $k \leq n$ gilt.

   c) Die Funktion `replicate :: Int -> a -> [a]` aus der Standardbibliothek nimmt eine Zahl $n \geq 0$ und einen Wert x und gibt die Liste $[\underbrace{x, \ldots, x}_{n\text{-mal}}]$ zurück.

   Implementieren Sie eine eigene Version dieser Funktion.

Aufgabe G5.3 Partitionierung

Die Funktion `partition :: (a -> Bool) -> [a] -> ([a], [a])` aus der Bibliothek `Data.List` teilt eine Liste in zwei Teillisten. Die eine Teilliste enthält die Elemente der Eingabeliste, die das gegebene Prädikat erfüllen. Die zweite Teilliste enthält die übrigen Elemente. Beide Teillisten sollen die Elemente in der gleichen Reihenfolge wie in der Eingabeliste enthalten.

Die folgenden Beispiele sind alle `True` (`xs` ist eine beliebige Liste):

```
partition alwaysTrue xs == (xs, [])
partition alwaysFalse xs == ([], xs)
partition even [4, 4, 1, 2] == ([4, 4, 2], [1])
```

wobei `alwaysTrue` und `alwaysFalse` – wie die Namen schon sagen – Funktionen sind, die stets `True` bzw. `False` liefern.

1. Implementieren Sie `partition` rekursiv.
2. Implementieren Sie `partition` mithilfe von `filter`.
3. Prüfen Sie mit QuickCheck, ob Ihre Funktionen mit `Data.List.partition` übereinstimmen.
4. Wie könnte ein sinnvoller QuickCheck-Test aussehen, der Aussagen über das Verhalten von Partition auf einer Liste xs ++ ys beschreibt?
5. Erörtern Sie die Vor- und Nachteile der zwei Implementierungen.

Aufgabe G5.4 Bewiesen. Oder nicht?

Gegeben seien die folgenden Definitionen:

```
zeros :: [Int]
zeros = 0 : zeros                          (zeros_def)

length :: [a] -> Int
length [] = 0                              (length_Nil)
length (_ : xs) = 1 + length xs            (length_Cons)
```

Aus diesen Gleichungen folgt

```
length zeros
  = length (0 : zeros)                     by zeros_def
  = 1 + length zeros                       by length_Cons
```

Nimmt man von beiden Seiten `length zeros` weg, bekommt man `0 = 1`, einen Widerspruch. Erklären Sie dies.

Aufgabe H5.1 Verrückte Entfernungstabellen (Wettbewerbsaufgabe, 10 Punkte)

Diese Wettbewerbsaufgabe ist als Teil dieses Blatts abzugeben. Sie bringt bis zu 60 Wettbewerbspunkte und bis zu 10 Hausaufgabenpunkte.

Entfernungstabellen geben die Entfernung zwischen einer Auswahl Städte an. In Haskell lassen sie sich als `[(String, Integer, String)]` darstellen. Zum Beispiel:

```
table =
  [("Cape Town",  1660, "Durban"),
   ("Cape Town",  1042, "East London"),
   ("Cape Town",  1402, "Johannesburg"),
```

```
("Durban",      667, "East London"),
("Durban",      598, "Johannesburg"),
("East London", 992, "Johannesburg")]
```

Laut dieser Tabelle beträgt die Distanz zwischen Kapstadt und Johannesburg 1402 km. Wir nehmen an, dass diese Entfernung für beide Fahrtrichtungen gilt und sparen uns somit die halbe Tabelle.

Für eine Dienstreise nach Südafrika sucht der MC diese Woche eine Haskell-Funktion

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

die überprüft, ob die gegebene Entfernungstabelle alle folgenden Eigenschaften erfüllt:

- **Nichtnegativität:** Keine Entfernungsangabe ist negativ.
- **Nichtreflexivität:** Die Tabelle enthält keine Einträge der Form $(x, \_, x)$.
- **Konsistenz:** Falls die Tabelle mehrere Tripel für die Städte x und y enthält, sind die Entfernungen gleich.[1]
- **Vollständigkeit:** Wenn x und y zwei ungleiche Städte sind, die in der Tabelle vorkommen, dann enthält die Tabelle ein Tripel der Form $(x, \_, y)$ oder $(y, \_, x)$.
- **Dreiecksregel:** Die Entfernung zwischen x und y plus die Entfernung zwischen y und z ist nicht kleiner als die Entfernung zwischen x und z.

Für den Wettbewerb zählt die Effizienz, vor allem die Skalierbarkeit. Ihre Implementierung sollte mit großen Tabellen, langen Wörtern und absurd vielen Duplikaten möglichst effizient vorgehen. Lösungen, die sich effizienztechnisch ähnlich verhalten, werden bezüglich ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden.

Die vollständige Lösung (außer Standardfunktionen und `import`-Direktiven) muss innerhalb der Kommentare {`-WETT-`} und {`-TTEW-`} abgegeben werden, um als Wettbewerbsbeitrag zu zählen. Zum Beispiel:

```
import Test.QuickCheck
import Data.List

{-WETT-}
isDistanceTableSane :: [(String, Integer, String)] -> Bool
isDistanceTableSane table = ...
{-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

[1] Dies schließt Symmetrie ein, d.h. der Abstand zwischen x und y muss gleich dem Abstand zwischen y und x sein.

Hausaufgabe abgeben möchten, lassen Sie bitte die {-WETT-} ... {-TTEW-} Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.

Aufgabe H5.2 Fixpunktiteration (4 Punkte)

Schreiben Sie eine Funktion `fixpoint :: (a -> a -> Bool) -> (a -> a) -> a -> a`, die als Parameter einen Gleichheitstest *eq*, eine Funktion f und einen Wert x bekommt und f solange wiederholt auf x anwendet, bis sich der Wert nicht mehr verändert (bezüglich des Gleichheitstests *eq*).

Falls dieser Prozess terminiert, gibt es also eine nicht-negative Zahl n, so dass zum einen `fixpoint` *eq* $f\ x = f^n(x) =$ `iter` $n\ f\ x$ und zum anderen $f^n(x)$ `` `eq` `` $f^{n+1}(x)$ gilt.

Gegeben folgende Hilfsfunktionen:

```
deleteA x = delete 'a' x
deleteL x = delete 'l' x

rotate [] = []
rotate (x : xs) = xs ++ [x]

firstEq (x:xs) (y:ys) = x == y
firstEq _ _ = False
```

Beispiele:

```
fixpoint (==) deleteA "Halli-Hallo" == "Hlli-Hllo"
fixpoint (==) deleteL "Halli-Hallo" == "Hai-Hao"
fixpoint firstEq rotate "Halli-Hallo" == "lli-HalloHa"
```

Wenn im letzten Beispiel `(==)` statt `firstEq` verwendet wird, dann terminiert die Berechnung nicht.

Aufgabe H5.3 Einhüllung (7 Punkte)

In dieser Aufgabe ist eine Funktion `trancl :: Ord a => [(a, a)] -> [(a, a)]` gesucht, die die *transitive Hülle* einer Relation R berechnet.

Mathematisch gesehen ist eine Relation R zwischen zwei Mengen A und B eine Teilmenge von $A \times B$. In Haskell ausgedrückt ist ein Relation zwischen zwei Typen a und b also eine Liste vom Typ `[(a, b)]`.

Die transitive Hülle einer Relation R ist nun die Liste aller Paare (u, v), so dass es einen nichtleeren Pfad von u nach v gibt. Das heißt, (u, v) ist genau dann in `trancl` R enthalten, wenn es eine Folge von Paaren

$$(u, w_1), (w_1, w_2), \ldots, (w_{n-1}, w_n), (w_n, v)$$

gibt, so dass jedes dieser Paare in R enthalten ist.

1. Schreiben Sie die Funktion `comp :: Eq b => [(a, b)] -> [(b, c)] -> [(a, c)]`, die die Komposition R von zwei Relationen R_1 und R_2 berechnet. Diese ist wie folgt definiert:

$$(a, c) \in R \iff \exists b.\, (a, b) \in R_1 \wedge (b, c) \in R_2$$

   Hinweis: Nutzen Sie eine Listenkomprehension. An dieser Stelle brauchen Duplikate noch nicht beachtet bzw. entfernt zu werden.

2. Implementieren Sie mittels `comp` die gesuchte Funktion `trancl` rekursiv. In der Ausgabe dürfen keine Duplikate vorhanden sein (auch wenn in der Eingabe Duplikate vorhanden waren). Die einzigen Funktionen höherer Ordnung, die Sie hierfür benutzen dürfen, sind `map` und `filter`.
3. Schreiben Sie eine neue Version von `trancl` ohne Rekursion. Sie dürfen zusätzlich die `fixpoint`-Funktion aus der vorangegangen Hausaufgabe nutzen.
4. Schreiben Sie zwei Quickcheck-Tests für Ihre Implementation. Prüfen Sie sinnvolle Eigenschaften, wie z.B. dass die Eingabe eine Teilmenge der Ausgabe sein muss.

Aufgabe H5.4 Mappen nach Zahlen (3 Punkte)

Implementieren Sie eine Funktion `mapIndex :: (Integer -> a -> b) -> [a] -> [b]`, die sich wie `map` verhält, aber der übergebenen Funktion zusätzlich auch noch den Index des Elements übergibt.

Beispiele:

```
mapIndex (+) [3, 4] == [3, 5]

mapIndex pair ["Ene", "mene", "mu"] ==
  [(0, "Ene"), (1, "mene"), (2, "mu")]
```

(für `pair x y = (x, y)`)