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

WS 2013/14
26.11.2012
Abgabe: 03.12.2012, 15:30

Einführung in die Informatik 2
7. Übung

Aufgabe G7.1 Brüche sind auch Zahlen

Im letzten Blatt haben wir Haskells vordefinierten Typ für Brüche `Rational` verwendet. Nun werden wir unsere eigene Variante implementieren.

Definieren Sie dazu einen Typ `Fraction` mit einem Konstruktor `Over :: Integer -> Integer -> Fraction`.

1. Damit sich die Brüche nicht benachteiligt fühlen, sollen die üblichen mathematischen Operatoren auch mit Brüchen funktionieren. Definieren Sie daher `Fraction` als eine Instanz der Typklasse `Num`.

```
class Num a where
  (+), (-), (*)       :: a -> a -> a
  negate              :: a -> a
  fromInteger         :: Integer -> a
  -- The functions 'abs' and 'signum' should
  -- satisfy the law:
  --
  -- > abs x * signum x == x
  abs                 :: a -> a
  signum              :: a -> a
```

2. Bei der Definition eines Datentypes kann Haskell diesen automatisch zu einer Instanz von `Eq` machen (mittels `deriving Eq`). Ist diese automatisch abgeleitete Instanz hier sinnvoll? Wenn nein, wie sollte sie stattdessen aussehen?
3. Schreiben Sie QuickCheck-Tests, die das in der Definition angegebene Gesetz sowie weitere wünschenswerte Eigenschaften prüfen, z.B. dass sich `(-)` konsistent zu `negate` und `(+)` verhält.

Weiterführende Hinweise:

- Die Typklasse `Num` enthält keinen Divisionsoperator, dieser ist in der Typklasse `Fractional` implementiert. In einem echten Programm würde man `Fraction` daher zu darüber hinaus zu einer Instanz von `Fractional` machen.
- Die Funktion `fromInteger` wird von Haskell benutzt, um ein Integer-Literal in den gesuchten Typ umzuwandeln. Der Ausdruck `3 :: Fraction` ist also der Bruch `(fromInteger 3) :: Fraction`. Die Funktion `fromRational` aus der Typklasse `Fractional` macht das gleiche für Dezimalliterale (z.B. `3.14`).

Aufgabe G7.2 Vereinfachen Sie die Definition!

Geben Sie alternative Definitionen mit möglichst kleiner Anzahl von Parametern (links vom Gleichheitszeichen) für die folgenden Funktionen an. Schreiben Sie dabei alle bestehenden λ-Ausdrücke um und führen Sie keine neuen λ-Ausdrücke ein (λ = lambda = \).

```
f1 xs = map (\x -> x + 1) xs
f2 xs = map (\x -> 2 * x) (map (\x -> x + 1) xs)
f3 xs = filter (\x -> x > 1) (map (\x -> x + 1) xs)
f4 f g x = f (g x)
f5 f g x y = f (g x y)
f6 f g x y z = f (g x y z)
f7 f g h x = g (h (f x))
```

Aufgabe G7.3 Figurativ

Wir wollen einfache geometrische Figuren in Haskell als Datentyp modellieren.

1. Definieren Sie einen Datentyp `Shape`. Dieser soll zwei Konstruktoren haben: `Circle` mit einem `Integer` als Radius und `Rectangle` mit zwei `Integer`n für Länge und Breite.

   Hinweis: Zum Ausprobieren auf der Konsole bietet es sich an, für den Datentyp eine `Show`- und eine `Eq`-Instanz zu definieren. Automatisch geht das mittels

   ```
   data Shape = ... deriving (Eq, Show)
   ```

2. Schreiben Sie eine Funktion `isValid :: Shape -> Bool`, die genau dann `True` zurückgibt, wenn die Form keine negative Länge, Breite oder Radius hat.
3. Implementieren Sie eine Funktion, die eine Figur bezüglich ihres Umfangs um einen Faktor r skaliert.

   Beispiel:

   ```
   scale 2 (Rectangle 1 3) = Rectangle 2 6
   scale 5 (Circle 4) = Circle 20
   ```

4. Fügen Sie dem Datentyp einen weiteren Konstruktor für Dreiecke hinzu. Durch wie viele `Integer`s ist ein Dreieck vollständig bestimmt (ohne Berücksichtigung von Position oder Drehung im Koordinatensystem)?

   Passen Sie `isValid` und und `scale` so an, dass sie auch mit Dreiecken zurecht kommen. Welche zusätzliche Eigenschaft von Dreiecken sollte in `isValid` geprüft werden?

Aufgabe H7.1 Figurativ II (9 Punkte)

> **Wichtig:** In dieser Hausaufgabe verwenden Sie bitte die ursprüngliche Definition von `Shape` als Grundlage, d.h. mit `Circle` und `Rectangle`. Dreiecke müssen in dieser Aufgabe nicht behandelt werden.
>
> Beachten Sie weiterhin, dass wir Ihnen für den zweiten Teil der Aufgabe keine automatisierten Tests anbieten können, weil Sie dort Ihren eigenen Datentyp definieren.

In der Haskell-Standardbibliothek ist der Datentyp `Maybe a` wie folgt definiert:

```
data Maybe a = Nothing | Just a
```

Dieser Typ wird unter anderem dafür verwendet, die Möglichkeit eines Fehlerzustands im Typ auszudrücken. Eine Funktionssignatur wie `a -> Maybe b` kann man so interpretieren, dass für einige Werte vom Typ `a` ein gültiges Ergebnis existiert (`Just b`), und für alle anderen nicht (`Nothing`).

1. Da unsere Figuren über `Integer`s definiert sind, können wir zwar die Fläche eines Rechtecks präzise ausrechnen, aber nicht die eines Kreises. Da ein Kreis natürlich nicht die Fläche 0 hat, müssen wir als Typ für die Fläche einer Figur folglich `Maybe Integer` wählen.

   Gesucht ist nun eine Funktion `area :: Shape -> Maybe Integer`, die falls möglich die Fläche einer Figur berechnet.

2. Bestimmen Sie, gegeben eine Liste von Figuren, die Summe der Flächen aller Rechtecke in dieser Liste. In der Liste vorhandene Kreise sollen ignoriert werden.

   Beispiel:

   ```
   rectArea [Rectangle 10 20, Rectangle 5 5] = 225
   rectArea [Rectangle 1 1, Circle 10] = 1
   ```

   Hinweis: Nutzen Sie die eben definierte Hilfsfunktion. Die Funktion `mapMaybe` aus dem `Data.Maybe`-Modul könnte hilfreich sein.

Geometrische Figuren stehen oft nicht allein im Raum, sondern sind relativ zu einem Koordinatensystem positioniert. Für diese Aufgabe wählen wir ein zweidimensionales kartesisches Koordinatensystem.

3. Definieren Sie einen neuen Datentyp `PosShape`, der zusätzlich zu einem `Shape` noch die x- und y-Koordinaten als jeweils ein `Integer` speichert. Bei einem Rechteck soll (x, y) die untere linke Ecke darstellen, bei einem Kreis dagegen den Mittelpunkt.

   Hinweis: Ein Konstruktor ist ausreichend.

4. Implementieren Sie wie in den Gruppenaufgaben eine Skalierungsfunktionen. Die Typsignatur ist bis auf `PosShape` statt `Shape` gleich. Beim Skalieren sollen sich die (x, y)-Koordinaten nicht ändern.

5. Schreiben Sie eine Funktion `move :: (Integer, Integer) -> PosShape -> PosShape`, wobei ein Aufruf `move (a, b) s` die x-Koordinate von s um a und die y-Koordinate um b verschieben soll.

Aufgabe H7.2 In Farbe und bunt (4 Punkte)

> **Wichtig:** Beachten Sie bitte, dass wir Ihnen für diese Aufgabe keine automatisierten Tests anbieten können, weil Sie hier Ihren eigenen Datentyp definieren.

Die Übungsleiter sind mit derzeitiger Zeichensoftware nicht zufrieden, weshalb es ihr erklärtes Ziel ist, für diesen Zweck eine funktionierende Software in Haskell zu schreiben. Für das Zeichnen von einfachen geometrischen Figuren in *HaskellPaint* haben Sie bereits in den vorangegangenen Aufgaben einen Grundstein gelegt. Allerdings sind die Zeichnungen noch sehr monochrom, weshalb Sie in dieser Aufgabe etwas Farbe ins Spiel bringen sollen.

Leider gibt es viele verschiedene Systeme für Farben, darunter auch RGB und YUV.

1. Definieren Sie einen Datentyp `Colour`, der die Konstruktoren `RGB` und `YUV` anbietet.

   Eine Farbe im RGB-Farbraum soll mit drei `Float`-Zahlen (r, g, b) mit Wertebereich $[0, 1]$ dargestellt werden. Für den YUV-Farbraum soll ebenfalls ein Tripel von `Floats` (y, u, v) verwendet werden, allerdings mit $y \in [0, 1]$, $u \in [-0.436, 0.436]$ und $v \in [-0.615, 0.615]$.

   Schreiben Sie weiterhin eine Funktion `isColourValid :: Colour -> Bool`, die prüft, ob die Werte für eine gegebene Farbe im gültigen Bereich liegen.

2. Für die Darstellung auf einem Monitor müssen Farben in den RGB-Raum konvertiert werden. Schreiben Sie daher drei Funktionen

   ```
   redComponent :: Colour -> Float
   greenComponent :: Colour -> Float
   blueComponent :: Colour -> Float
   ```

   die den Rot-, Grün- und Blauanteil einer Farbe liefern. Handelt es sich bei der Eingabe um eine Farbe im YUV-Farbraum, so muss diese natürlich erst konvertiert werden. Sie können dafür die nachstehende Konvertierungsvorschrift[1] nutzen (die Sie auch vorher vereinfachen können):

$$
\begin{aligned}
W_R &= 0.299 \\
W_G &= 0.587 \\
W_B &= 0.114 \\
U_{Max} &= 0.436 \\
V_{Max} &= 0.615
\end{aligned}
\qquad
\begin{aligned}
r &= y + v \cdot \frac{1 - W_R}{V_{Max}} \\
g &= y - u \cdot \frac{W_B \cdot (1 - W_B)}{U_{Max} \cdot W_G} - v \cdot \frac{W_R \cdot (1 - W_R)}{V_{Max} \cdot W_G} \\
b &= y + u \cdot \frac{1 - W_B}{U_{Max}}
\end{aligned}
$$

[1] Quelle: `http://en.wikipedia.org/w/index.php?title=YUV&oldid=578238215`

Bei der Konvertierung sollen – falls die übergebene Farbe gültig ist – jeweils Werte im Bereich von $[0, 1]$ berechnet werden. Gegebenfalls müssen Sie kleinere oder größere Werte entsprechend „runden“.

Aufgabe H7.3 Korrektuz (7 Punkte, Wettbewerbsaufgabe)

Gesucht wird eine Funktion

```
fixTypos :: [String] -> [String] -> [String]
```

die eine Vokabelliste *vocabs* und eine Liste von teilweise falsch geschriebenen Wörtern *text* nimmt. Der Aufruf `fixTypos` *vocabs text* gibt eine Liste korrigierter Wörter (den *korrigierten Text*) zurück.

Definitionen: Ein *Wort* ist eine Liste von `Char`s aus den 26 kleinen angelsächsischen Buchstaben `a` bis `z`. Ein Wort ist *falsch geschrieben*, wenn es nicht in der Vokabelliste *vocabs* steht. Zwei Wörter sind *quasi-identisch*, wenn sie die gleiche Länge haben und das eine Wort aus dem anderen hergeleitet werden kann, indem genau ein Buchstabe durch einen anderen ersetzt wird. Beispielsweise sind `gelegenheed` und `gelegenheid` quasi-identisch, da der zehnte Buchstabe von `gelegenheed` (also `e`) durch `i` ersetzt werden kann, um das (auf Niederländisch korrekt geschriebene) Wort `gelegenheid` herzuleiten.

Zur Korrektur eines falsch geschriebenen Wortes wird dieses durch ein in *vocabs* vorhandenes quasi-identisches Wort ersetzt. Sind keine oder mehrere unterschiedliche quasi-identische Wörter vorhanden, soll das falsche Wort unverändert bleiben.

Beispiele:

```
vocabs = ["i", "more", "more", "now", "want", "won", "wow"]

fixTypos vocabs ["i", "want", "mose"] ==
  ["i", "want", "more"]
fixTypos vocabs ["and", "j", "want", "it", "naw"] ==
  ["and", "i", "want", "it", "now"]
fixTypos vocabs ["won", "", "now", "wow", "wob", "nob"] ==
  ["won", "", "now", "wow", "wob", "now"]
fixTypos vocabs ["rome", "boban", "wat", "ant", "watn"] ==
  ["rome", "boban", "wat", "ant", "watn"]
```

Alle Bibliotheksfunktionen sind erlaubt.

Schreiben Sie zudem mindestens zwei sinnvolle QuickCheck-Tests. Welche Eigenschaften sollte der korrigierte Text haben?

Für den Wettbewerb zählt die Effizienz, vor allem die Skalierbarkeit. Ihre Implementierung sollte mit langen (bzw. kurzen) Vokabellisten, großen (bzw. kleinen) Texten, langen (bzw. kurzen) Wörtern und absurd vielen (bzw. keinen) Duplikaten möglichst effizient vorgehen. Der wichtigste Fall ist erfahrungsgemäß der, in dem die Vokabelliste im 10.000-Wort-Bereich und der Text

im 100.000-Wort-Bereich liegt – denn der Master of Competition hat vor, seine reichhaltigen literarischen Ergüsse korrigieren zu lassen.

Die Lösung des MC benötigt weniger als vier Sekunden für das Beispiel

```
Exercise_7> import Data.List
Exercise_7> import Data.Hashable
Exercise_7> let vocabs = take 10000 (permutations "abcdefghi")
Exercise_7> let text = take 100000 (permutations "abcxefgih")
Exercise_7> :set +s
Exercise_7> hash (fixTypos vocabs text)
3652546476494351287
(3.65 secs, 1029321328 bytes)
```

in GHCi. Die Funktion `hash` wird hier verwendet, um die Ausgabe drastisch zu reduzieren.[2]

Lösungen, die sich effizienztechnisch ähnlich verhalten, werden bezüglich ihrer Lesbarkeit verglichen, wobei Formatierung und Namensgebung besonders gewichtet werden. Um teilzunehmen, müssen Sie Ihre Lösung innerhalb der beiden 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.

[2] Sofern Sie die Haskell Platform in der Version $\geq$ 2013.2.0.0 verwenden, so ist das dafür notwendige Paket bereits installiert. Andernfalls können Sie es mit `cabal install hashable` nachinstallieren. Auf `fp.in.tum.de` stehen dieses und alle weiteren Pakete der Haskell Platform zur Verfügung.