Universität Bremen

Praktische Informatik 3 WS 14/15

Christoph Lüth
Sandor Herms
Daniel Müller
Jan Radtke
Henrik Reichmann
Sören Schulze
Felix Thielke

5. Übungsblatt

Ausgabe: 18.11.14
Abgabe: 28.11.14

5.1 *Listenkombinatoren* *5 Punkte*

Schon für die Bearbeitung des 4. Übungsblatt stellten sich Listenkombinatoren wie map, concatMap, filter , foldr und weitere als sehr nützlich heraus. Da diese in der Programmierung mit Haskell häufig zur Vereinfachung und besseren Lesbarkeit von Code einsetzbar sind, haben wir hier noch einige Fingerübungen für Sie.

Definieren Sie die folgenden Funktionen unter Verwendung von Listenkombinatoren:

```
unzip :: [(α, β)] → ([α], [β])

count :: Eq α ⇒ α → [α] → Int

maximum :: Ord α ⇒ [α] → α

takeWhile :: (α → Bool) → [α] → [α]

group :: Eq α ⇒ [α] → [[α]]
```

Die Funktion count gibt dabei die Anzahl der Vorkommen des ersten Parameters in der als zweiten Parameter übergebenen Liste zurück, die Definitionen der anderen Funktionen finden Sie im Haskell-Modul Data.List. Versuchen Sie, in Ihrer Implementierung ohne Hilfsfunktionen oder lokale Definitionen mit **where** oder **let** auszukommen, und benutzen Sie η-Kontraktion, wo immer dies möglich ist.

5.2 *Datenbanken* *15 Punkte*

Genug der Übung, nun gilt es, das Gelernte auch anzuwenden. Wenn man mit großen Datenmengen arbeitet, möchte man sich oft gern Teilmengen dieser Daten oder aggregierte Werte wie Summe oder Anzahl solcher Teilmengen anzeigen lassen. In relationalen Datenbanken nutzt man dafür in der Regel Ausdrücke in der Sprache SQL, mit welcher man explizit einzelne Datenfelder einer Tabelle selektieren kann, welche bestimmte Bedingungen erfüllen. Auch Aggregationen wie Summe, Durchschnitt oder Anzahl sind möglich.

Wir wollen nun eine solche Selektion in Haskell realisieren. Gegeben ist eine Datenbanktabelle in der Datenstruktur TableData, welche in einer Liste Daten über Länder speichert:

```
type TableData = [Row]
data Row = Row String String Integer Integer Integer deriving Show
```

Die in einer Zeile gespeicherten Daten sind dabei (in dieser Reihenfolge) der Name des Lands, die Region, in der es sich befindet, die Bevölkerungszahl, die Fläche in Quadratkilometer und das Bruttoinlandsprodukt.

Die in der Konstante countries gegebenen Daten bestehen lediglich aus einer Liste von Tupeln, die die Länderinformationen enthalten. Implementieren Sie zunächst eine Funktion fromList, welche diese in das oben beschriebene Tabellenformat umwandelt:

```
fromList :: [(String, String, Integer, Integer, Integer)] → TableData
```

Schreiben Sie nun eine Funktion selectTable, welche eine Selektion von Feldern einer Tabelle durchführt:

```
selectTable :: [Row → String] → (Row → Bool) → TableData → TableSelect
```

Die Funktion erhält als ersten Parameter eine Liste von Selektoren für die auszugebenden Feldwerte, als zweiten Parameter ein Prädikat, das Zeilen erfüllen müssen, um in der Ausgabe enthalten zu sein, und als dritten Parameter die Tabelle. Der Rückgabetyp TableSelect ist definiert als eine Liste von Stringlisten, welche für jeden im ersten Parameter übergebenen Selektor je eine Liste mit allen dazugehörigen Werten der selektierten Zeilen enthält:

```
type TableSelect = [[String]]
```

Nun können Sie Datensätze selektieren, aber es fehlt eine brauchbare Ausgabe. Schreiben Sie eine Funktion formatSelect, welche die Ausgabe von selectTable so formatiert, dass alle Strings eines selektierten Felds dieselbe Länge haben, wobei aber nie Feldwerte abgeschnitten werden, und jede ausgegebene Zeile auch in je einer Zeile der Ausgabe steht:

```
formatSelect :: TableSelect → String
```

Zwischen zwei Spalten der Ausgabe steht dabei mindestens ein Leerzeichen. Bei der Umsetzung ist es hilfreich, zunächst die Längen der Feldwerte aufzufüllen und anschließend die Tabelle zeilenweise auszugeben.

Mit diesen Funktionen schreiben Sie nun eine Funktion select, welche dieselben Parameter wie selectTable erhält und diese mit der in formatSelect realisierten Formatierung ausgibt. Damit können Sie SQL-ähnliche Selektionen durchführen, wie in diesem Beispiel alle Länder mit mehr als 100 Milionen Einwohnern:[1]

```
> putStrLn (select [name, region] ((10^8 <). population) (fromList countries))
Bangladesh                      Asien
Brasilien                       Suedamerika
China                           Asien
Indien                          Asien
Indonesien                      Suedostasien
Japan                           Asien
Mexiko                          Nordamerika
Nigeria                         Afrika
Pakistan                        Asien
Russland                        Asien
Vereinigte Staaten von Amerika Nordamerika
```

Jetzt fehlt nur noch die Möglichkeit von Aggregationen wie Anzahl, Summe oder Durchschnitt bestimmter Feldwerte. Schreiben Sie hierfür die Funktion selectAggr, welche analog zu select ist, aber statt einer Liste von Selektoren im ersten Parameter eine Liste von Tupeln mit je einer Aggregationsfunktion und einem Selektor für den Feldwert, der aggregiert werden soll, erhält:

```
selectAggr :: [([Integer] → Integer, Row → Integer)] → (Row → Bool) → TableData → String
```

Auch hierfür ist es sinnvoll, zunächst eine Hilfsfunktion zu schreiben, die eine Selektion durchführt und den Typ TableSelect zurückgibt. Die Formatierung des Ergebnisses ist dieselbe wie bei select.

Hiermit können Sie nun SQL-ähnliche Aggregationen (aber keine Gruppierungen) durchführen, wie in folgendem Beispiel:[2]

```
>putStrLn (selectAggr [(sum, population), (maximum, gdp)] (("Europa" ==). region) (fromList countries))
584426608 1864000000000
```

[1] Analog in SQL: `SELECT name, region FROM countries WHERE 100000000 < population`

[2] Analog in SQL: `SELECT SUM(population), MAX(gdp) FROM countries WHERE 'Europa' = region`

Führen Sie nun einige Selektionen durch und dokumentieren Se Ihre dazu benutzten Befehle sowie die Ausgaben in Ihrer Dokumentation:

- Selektieren Sie Name und Region aller Länder mit mindestens zehn Millionen Einwohnern.
- Selektieren Sie Name, Population und Bruttoinlandsprodukt aller Länder, deren Population oder Bruttoinlandsprodukt unter 100 ist.
- Selektieren Sie Name und Region in der Form `Deutschland (Europa)` und Population aller Länder mit einer Fläche zwischen 50000 und 100000 km^2.
- Selektieren Sie die Anzahl der Länder in Europa.
- Selektieren Sie das durchschnittliche weltweite Bruttoinlandsprodukt. Hierfür kann es sinnvoll sein, zusätzlich eine Funktion `avg`, welche den Durchschnitt einer Liste von ganzen Zahlen liefert, zu schreiben.

? *Verständnisfragen*

1. `foldr` ist die „kanonische einfach rekursive Funktion“ (Vorlesung). Was bedeutet das, und warum ist das so? Für welche Datentypen gilt das?
2. Wann kann `foldr f a xs` auch für ein zyklisches Argument `xs` (bspw. eine zyklische Liste) terminieren?
3. Was ist die Grundidee hinter Parserkombinatoren, und funktioniert diese Idee nur für Parser oder auch für andere Problemstellungen?