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

WS 2013/14
19.11.2013
Abgabe: 26.11.2013

Einführung in die Informatik 2
6. Übung

Aufgabe G6.1 Der Unterschied

Was ist der Unterschied zwischen

1. `(div 5)` und `(`‘`div`‘ `5)`?
2. `(+ 7)` und `((+) 7)`?
3. `map (: [])` und `map ([] :)`?
4. `flip . flip` und `id`?
5. `[x, y, z]` und `x : y : z`?

Aufgabe G6.2 „Wer bin ich – und wenn ja, was ist mein Typ?“

Gegeben seien die folgenden Funktionen:

```
foldl f z [] = z
foldl f z (x : xs) = foldl f (f z x) xs

foldr f z [] = z
foldr f z (x : xs) = f x (foldr f z xs)

isCons = not . null

ffoldl = foldl . foldl

oo = (.).(.)
```

1. Finden Sie den allgemeinsten Typ von `foldl`, `foldr`, `isCons` und `ffoldl` und `oo` ohne Hilfe von GHCi heraus. Vergleichen Sie Ihre Hypothesen mit der Ausgabe von GHCi.
2. In welchem Kontext könnten sich `ffoldl` und `oo` als nützlich erweisen?

Aufgabe G6.3 Induktion

Beweisen Sie die Gleichung

```
sum . filter (/= 0) = sum
```

per Extensionalität mit anschließender Induktion. Benutzen Sie das aus der Vorlesung bekannte “induction template” (Folie 124). Geben Sie zusätzlich die Gleichung, die Sie als Induktionshypothese (IH) benutzen, explizit an.

Folgende Definitionen sind gegeben:

```
filter f [] = []                                    (filter_Nil)
filter f (x : xs) =                                 (filter_Cons)
   if f x then x : filter f xs else filter f xs

sum [] = 0                                          (sum_Nil)
sum (x : xs) = x + sum xs                           (sum_Cons)

(f . g) x = f (g x)                                 (compose_def)
```

Verwenden Sie in jedem Schritt *nur eine* dieser Gleichungen oder die Induktionshypothese und vergessen Sie nicht, den Namen der angewendeten Gleichung anzugeben.

Hinweis: Sie werden eine Fallunterscheidung durchführen müssen.

Aufgabe G6.4 Teile und Falte (optional)

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 (\x -> True) xs == (xs, [])
partition (\x -> False) xs == ([], xs)
partition even [4, 4, 1, 2] == ([4, 4, 2], [1])
```

In Blatt 5 haben Sie `partition` bereits implementiert. Implementieren Sie nun `partition` mittels `foldr` ohne die Verwendung weiterer rekursiver Funktionen.

Aufgabe H6.1 (Inter)valle, valle, manche Strecke

Auf dem zweiten Übungsblatt haben wir Listen zur Darstellung von Mengen verwendet. Dabei entsprach jedes Element der Liste einem Element der Menge. Eine andere Darstellung (für bestimmte Typen) sind Intervalllisten.

Der Haskell-Typ einer Intervallliste für rationale Zahlen ist

```
type IList = [(Rational,Rational)]
```

Dabei stellt das Paar `(x,y)` ein geschlossenes Intervall von x bis y dar. Der Typ `Rational` ist in `Data.Ratio` definiert und verfügt über die üblichen arithmetischen Operationen (insbesondere Vergleiche).

Zum Beispiel können wir die Menge $\left\{x \mid \frac{2}{3} \leq x \leq 5 \vee \frac{11}{2} \leq x \leq 11\right\}$ darstellen als `[(2 % 3, 5), (11 % 2, 11)]`. Dabei konstruiert `%` einen Bruch (der automatisch gekürzt wird).

Eine Intervallliste soll folgende Bedingungen erfüllen:

- Für ein Intervall `(x,y)` muss `x` $\leq$ `y` gelten. Das Intervall `(3 % 2, 6)` ist also in Ordnung, `(2,2 % 3)` jedoch nicht.
- Zwei Intervalle dürfen sich weder überlappen noch berühren. Zum Beispiel dürfen die Intervalle `(2 % 1, 5)` und `(4,8)` nicht in der gleichen Intervallliste vorkommen, sie müssen durch `(4 % 2, 8)` ersetzt werden.
- Die Intervalle müssen aufsteigend sortiert sein, d.h. `[(1,3),(6,8),(10,13)]` ist in Ordnung, `[(6,8), (1,3), (10,13)]` aber nicht.

Aufgaben:

1. Implementieren Sie als erstes eine Funktion `wellformed :: IList -> Bool`, die überprüft, ob eine gegebene Intervallliste alle diese Bedingungen erfüllt.
2. Implementieren Sie die folgenden Funktionen:

   ```
   empty  :: IList
   member :: Rational -> IList -> Bool
   insert :: (Rational,Rational) -> IList -> IList
   ```

   `empty` liefert eine leere Menge zurück, `member` überprüft, ob eine Zahl in der von der Intervallliste reprästentierten Menge enthalten ist, `insert` dient dem Einfügen eines Intervalls. Falls die Eingabe wohlgeformt ist, soll `insert` immer eine wohlgeformte Intervallliste zurückgeben. Wird ein bereits enthaltenes Intervall neu eingefügt (bzw. ein Teilintervall eines vorhandenen Intervalls), so soll einfach die ursprüngliche Intervallliste zurückgegeben werden.
3. Versichern Sie sich der Fehlerfreiheit Ihrer Funktionen, indem Sie zwei QuickCheck-Tests schreiben, die die Funktionen `member` und `insert` testen.

   Hinweis: In der Vorlage finden Sie Tipps zur besseren Verwendung von QuickCheck.

Aufgabe H6.2 Weder Map noch Fold (8 Punkte)

Die Funktionen `map` und `foldl`/`foldr` sind unentbehrliche Bestandteile im Repertoire eines Haskell-Programmierers. Es gibt Anwendungen, in denen auch eine Kombination der beiden Ansätze erforderlich ist. Implementieren Sie deshalb die Funktion

```
foldlMap :: (a -> c -> (b, c)) -> [a] -> c -> ([b], c)
```

Gegeben sei der Aufruf `foldlMap` f $[a_1, a_2, \ldots, a_n]$ c_0.

- Die Funktion f nimmt einen Eingabenwert (vom Typ `a`) und einen Zustand (vom Typ `c`) und gibt einen Ausgabenwert (vom Typ `b`) und einen neuen Zustand (vom Typ `c`) zurück.
- Die Liste $[a_1, a_2, \ldots, a_n]$ gibt die Eingabenwerte an.
- c_0 ist der Startzustand.

Das Ergebnis besteht aus einer Liste von Ausgabenwerten $[b_1, b_2, \ldots, b_n]$ und dem Endzustand c_n. Im Allgemeinen gilt die Gleichung

$$\texttt{foldlMap}\ f\ [a_1, a_2, \ldots, a_n]\ c_0 = ([b_1, b_2, \ldots, b_n], c_n)$$

mit

$$(b_1, c_1) = f\ a_1\ c_0 \qquad (b_2, c_2) = f\ a_2\ c_1 \qquad \ldots \qquad (b_n, c_n) = f\ a_n\ c_{n-1}$$

Als konkrete Anwendung für diese etwas abstrakte Funktion dient die Generierung von „frischen“ Namen. Gegeben sei eine Funktion

```
fresh :: String -> [String] -> (String, [String])
fresh s names | elem s names = fresh (s ++ "_") names
fresh s names = (s, s : names)
```

die einen Namensvorschlag s und eine Liste der schon vergebenen Namen als Argumente nimmt und einen frischen (d.h. nicht bereits vergegebenen) Namen, zusammen mit der aktualisierten Liste der vergebenen Namen, zurückgibt. Falls s schon vergeben ist, wird rekursiv ein Unterstrich (_) angehängt, bis ein frischer Name erreicht wird. Zum Beispiel:

```
fresh "a" [] == ("a", ["a"])
fresh "a" ["a"] == ("a_", ["a_", "a"])
fresh "a" ["a", "a_", "b"] == ("a__", ["a__", "a", "a_", "b"])
```

Die Funktion `foldlMap` lässt sich verwenden, um eine Liste mit Namensvorschlägen in eine Liste mit eindeutigen Namen umzuwandeln:

```
freshes :: [String] -> [String] -> ([String], [String])
freshes = foldlMap fresh

fst (freshes ["a", "a", "a", "b", "a", "b"] [])
  == ["a", "a_", "a__", "b", "a___", "b_"]
```

Testen Sie Ihre Implementierung von `foldlMap` anhand dieser Anwendung.

Aufgabe H6.3 Geschachtelte Indexierung (2 Punkte, Wettbewerbsaufgabe)

Der Master of Competition hat vor, seine reichhaltigen literarischen Ergüsse zu indexieren. Er hat damit angefangen, eine Funktion

```
index1 :: [a] -> Int -> ([Int], Int)
```

zu schreiben, die eine beliebige Liste $[a_1, a_2, \ldots, a_n]$ sowie einen Startzähler j als Argumente nimmt und eine Liste $[j,\ j{+}1,\ \ldots,\ j{+}n{-}1]$ zurückgibt. Damit kann er einfache Listen indexieren:

```
fst (index1 ["pelle", "foretag", "se"] 10) == [10, 11, 12]
```

Danach hat er eine Funktion implementiert, die mit Listen von Listen umgeht:

```
index2 :: [[a]] -> Int -> ([[Int]], Int)
```

Die Ausgabenliste hat die gleiche geschachtelte Struktur wie die Eingabenliste, besteht aber aus numerischen Indizes:

```
fst (index2 [["e", "mail"], [], ["address"]] 20) == [[20, 21], [], [22]]
```

Die gesamten literarischen Werke des MCs gehen aber noch viel tiefer. Deshalb sucht er für das größtmögliche n eine Funktion

$$\texttt{index}n\ \texttt{::}\ \overbrace{\texttt{[[}\cdots\texttt{[[}}^{n\text{-mal}}\texttt{a}\overbrace{\texttt{]]}\cdots\texttt{]]}}^{n\text{-mal}}\ \texttt{->}\ \texttt{Int}\ \texttt{->}\ \texttt{(}\overbrace{\texttt{[[}\cdots\texttt{[[}}^{n\text{-mal}}\texttt{Int}\overbrace{\texttt{]]}\cdots\texttt{]]}}^{n\text{-mal}}\texttt{, Int)}$$

Ihm ist auch wichtig, dass die besten Lösungen bequem in eine Twitter-Nachricht passen. Deshalb ist die Länge der Lösung auf maximal 60 Token beschränkt.

Alle Bibliotheksfunktionen sowie Funktionen, die für H6.1 und H6.2 definiert wurden, sind zulässig. Die vollständige Lösung (außer `import`-Direktiven und bereits vorhandenen Funktionen) muss innerhalb von Kommentaren `{-WETT-}` und `{-TTEW-}` stehen. Zum Beispiel:

```
{-WETT-}
index2 :: [[a]] -> Int -> ([[Int]], Int)
index2 ass j = ...
{-TTEW-}
```

oder

```
{-WETT-}
index432 :: [[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[[
index432 asssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
{-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 Hausaufgabe abgeben möchten, lassen Sie bitte die `{-WETT-}` ... `{-TTEW-}` Kommentare weg. Bei der Bewertung Ihrer Hausaufgabe entsteht Ihnen hierdurch kein Nachteil.