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

WS 2012/13
20.11.2012
Abgabe: 27.11.2012, 15:30

Einführung in die Informatik 2
6. Übung

Aufgabe G6.1 Funktionen als Zahlen

Die grundlegenden mathematischen Operationen wie z.B. `(+)` und `(*)` sind in Haskell in der Typklasse `Num` definiert. Funktionen lassen sich auch als Zahlen behandeln, indem man die entsprechenden Operationen punktweise definiert. Das heißt, für Funktionen f und g würde man $f + g$ definieren durch $(f + g)(x) = f(x) + g(x)$. Ähnlich geht man bei den anderen Funktionen vor.

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

Implementieren Sie eine `Num`-Instanz für den Typ `a -> b`. Überlegen Sie zunächst: Welche der beiden Typen `a`, `b` müssen bereits Instanzen von `Num` sein?

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

Gegeben seien die folgenden Fuktionen:

```
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)

ffoldl = foldl . foldl

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

1. Finden Sie den allgemeinsten Typ von `foldl`, `foldr`, `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 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 G6.4 Induktion

Beweisen Sie die Gleichung

```
reverse = foldl (\xs x -> x : xs) []
```

per Extensionalität mit anschließender Induktion. Benutzen Sie das aus der Vorlesung bekannte “induction template” (Folie 120). Geben Sie zusätzlich die Gleichung, die Sie als Induktionshypothese (IH) benutzen, explizit an.

Die Funktionen `foldl :: (a -> b -> a) -> a -> [b] -> a`, `reverse :: [a] -> [a]` und `(++) :: [a] -> [a] -> [a]` sind wie folgt definiert:

```
foldl f y [] = y                              (foldl_Nil)
foldl f y (x : xs) = foldl f (f y x) xs       (foldl_Cons)

reverse [] = []                               (reverse_Nil)
reverse (x : xs) = reverse xs ++ [x]          (reverse_Cons)

[] ++ ys = ys                                 (append_Nil)
(x : xs) ++ ys = x : (xs ++ ys)               (append_Cons)
```

Zusätzlich dürfen Sie die folgenden, aus der Vorlesung bereits bekannten Gleichungen benutzen:

```
(xs ++ ys) ++ zs = xs ++ (ys ++ zs)           (append_assoc)
xs ++ [] = xs                                 (append_Nil2)
```

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

Hinweis: Wenn Sie Schwierigkeiten haben die Induktionshypothese anzuwenden, sollten Sie versuchen, eine verallgemeinerte Gleichung beweisen.

Aufgabe G6.5 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))
```

Wichtig: Bei bisherigen Hausaufgaben gab es immer wieder eingereichte Lösungen die zwar inkorrekt waren aber dennoch alle unsere Tests bestanden haben. Das ist nichts Ungewöhnliches, da die Tests die Korrektheit nicht garantieren. Allerdings sind wir daran interessiert unsere Tests zu verbessern, indem eventuelle Lücken geschlossen werden.

Dabei zählen wir auf Ihre Unterstützung, die wir gerne mit zusätzlichen Hausaufgabenpunkten entlohnen werden. Falls Sie also eine inkorrekte Implementierung haben, die unsere Tests besteht, können Sie zusätzliche Tests in Ihren hochgeladenen Lösungsvorschlag einschließen. Benutzen Sie bitte die Tags `{-QC-}` ... `{-CQ-}` um die neuen Tests zu kennzeichnen.

Wir werden die sinnvollen Tests in unsere Testsuite einpflegen. Wenn Ihre Tests Fehler unter den eingereichten Lösungen offenbaren, die wir nicht gefunden haben, bekommen Sie einen zusätzlichen Punkt gutgeschrieben (es gilt First-Come-First-Served).

Aufgabe H6.1 Speichereffiziente Mengendarstellung (12 Punkte)

Auf dem zweiten Übungsblatt haben wir Listen zur Darstellung von Mengen verwendet. Eine andere Möglichkeit sind *Paarlisten*, bei denen Mengen als Listen von Paaren dargestellt werden. Der Haskell-Typ einer Paarliste ist

```
type PairList = [(Int,Int)]
```

Zum Beispiel können wir die Menge $\{1, 2, 3, 8, 9, 10\}$ darstellen als $[(1, 3), (8, 10)]$. Das Paar $(1, 3)$ bedeutet, dass 1, 2 und 3 Elemente der Menge sind, das Paar $(8, 10)$ dagegen, dass 8, 9 und 10 Elemente der Menge sind. Eine Paarliste soll folgende Bedingungen erfüllen:

1. Für ein Paar (x, y) muss gelten $x \leq y$. Die Paare $(3, 6)$ und $(2, 2)$ sind also in Ordnung, $(4, 1)$ jedoch nicht.
2. Zwei Paare dürfen sich weder überlappen noch berühren. Zum Beispiel dürfen die Paare $(2, 6)$ und $(3, 9)$ nicht in der gleichen Paarliste vorkommen, sie sollten durch $(2, 9)$ ersetzt werden. Ähnliches gilt für $(3, 4)$ und $(5, 8)$ – stattdessen sollte $(3, 8)$ in der Liste stehen.
3. Die Paare müssen aufsteigend sortiert sein, d.h. $[(1, 3), (6, 8), (10, 13)]$ ist in Ordnung, $[(6, 8), (1, 3), (10, 13)]$ aber nicht.

Implementieren Sie als erstes eine Funktion `wellformed :: PairList -> Bool`, die überprüft, ob eine gegebene Paarliste alle diese Bedingungen erfüllt.

Danach implementieren Sie die folgenden Funktionen:

```
empty  :: PairList
member :: Int -> PairList -> Bool
insert :: Int -> PairList -> PairList
union  :: PairList -> PairList -> PairList
delete :: Int -> PairList -> PairList
```

`empty` liefert eine leere Menge zurück, `member` überprüft, ob eine Zahl in der von der Paarliste reprästentierten Menge enthalten ist, `insert` und `delete` dienen dem Einfügen und Löschen eines einzelnen Elementes und `union` der Vereinigung zweier Paarlisten. Falls die Eingabe wohlgeformt ist, sollen `insert`, `delete` und `union` immer eine wohlgeformte Paarliste zurückgeben. Wird ein bereits enthaltenes Element neu eingefügt oder ein nicht vorhandenes Element gelöscht, so soll einfach die ursprüngliche Paarliste zurückgegeben werden.

Die Laufzeit der Funktionen soll linear von der Länge der eingegebenen Paarliste abhängen (nicht von der Größe der dargestellten Menge). Das heißt insbesondere, dass Sie keine der obigen Funktionen implementieren sollen, indem Sie eine Paarliste zunächst in die Liste ihrer Elemente umwandeln.

Aufgabe H6.2 Anonymisieren von E-Mail-Adressen (8 Punkte, Wettbewerbsaufgabe)

Gesucht ist eine Funktion `anonymize :: String -> String`, die alle gültigen E-Mail-Adressen im Eingabetext durch anonymisierte Versionen ersetzt und den restlichen Text unverändert lässt.

Beispiele:

```
anonymize "pelle@foretag.se" == "p____@f______.s_"
anonymize "Hello anna.svensson@univ.com, j.cash@music.org!" ==
  "Hello a___.s_______@u___.c__, j.c___@m____.o__!"
anonymize "My e-mail addresses are karin@karin.com, \
          \anna@anna.com, and annakarin@hej.se." ==
  "My e-mail addresses are k____@k____.c__, a___@a___.c__, \
  \and a________@h__.s_."
```

Die offizielle Syntax von E-Mail-Adressen ist sehr komplex. Für die Hausaufgabe können Sie sich auf die folgende, vereinfachte Definition beschränken:

> Eine *E-Mail-Adresse* ist eine möglichst lange Teilzeichenkette bestehend aus Buchstaben (`a`–`z`, `A`–`Z`), Zahlen (`0`–`9`), Punkten (`.`), Unterstrichen (`_`) und mindestens einem Klammeraffen (`@`). Enthält diese Zeichenkette genau einen Klammeraffen, so handelt es sich um eine *gültige*, andernfalls um eine *ungültige* E-Mail-Adresse.

Ungültige E-Mail-Adressen müssen unverändert bleiben. Die Anonymisierung einer E-Mail-Adresse besteht darin, jeden Textteil zwischen Punkten (`.`) und Klammeraffen (`@`) in einen gleich langen Textteil zu übersetzen, wobei alle Zeichen außer dem Ersten durch Unterstriche (`_`) ersetzt werden.

Weitere Beispiele:

```
anonymize "apa@bepa@cepa.se" == "apa@bepa@cepa.se"
anonymize "apa_bepa@cepa.se" == "a_______@c___.s_"
```

Für den Wettbewerb sucht der Master of Competition Lösungen, die der offiziellen Definition von E-Mail-Adressen (RFC 5321 und 5322) entsprechen. Je näher Ihre Lösung an der offiziellen Definition liegt, desto besser. Bitte kennzeichnen Sie in Ihrer Lösung explizit, wenn Sie sich daran versuchen. Laut Wikipedia sind folgende Adressen gültig:

```
user@[IPv6:2001:db8:1ff::a0b:dbd0]
"much.more unusual"@example.com
"very.unusual.@.unusual.com"@example.com
```

Die Folgenden sind jedoch ungültig:

```
Abc..123@example.com
a"b(c)d,e:f;g<h>i[j\k]l@example.com
just"not"right@example.com
```

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.