Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
20. Mai 2014

5. Übung zur Vorlesung
Programmierung und Modellierung

A5-1 *Listenverarbeitung höherer Ordnung*

a) Ersetzen Sie die List-Comprehension in der folgenden Definition durch den Einsatz der Funktionen `map` und `filter` aus der Standardbibliothek:

```
foo1 f p xs = [f x | x <- xs, x >= 0, p x]
```

b) Die Funktion `dropWhile :: (a -> Bool) -> [a] -> [a]` aus der Standardbibliothek entfernt so lange Element vom Anfang einer Liste, so lange diese das gegebene Prädikat erfüllen. Implementieren Sie diese Funktion selbst mit Rekursion, ohne die Verwendung von Bibliotheksfunktionen.

Beispiel: `dropWhile (<4) [1,3,4,5,3,1] == [4,5,3,1]`

c) Die Funktion `all :: (a -> Bool) -> [a] -> Bool` aus der Standardbibliothek gibt nur dann `True` zurück, wenn alle Element der Liste das übergebene Prädikat erfüllen. Implementieren Sie die Funktion ohne direkte Rekursion, sondern unter Verwendung Funktionen höherer Ordnung aus der Standardbibliothek.

d) Die Funktion `concatMap :: (a -> [b]) -> [a] -> [b]` entspricht der Definition

```
> let concatMap1 f = concat . map f
concatMap1 :: (a -> [b]) -> [a] -> [b]
> concatMap1 (\x -> replicate x x) [1..5]
[1,2,2,3,3,3,4,4,4,4,5,5,5,5,5]
```

Implementieren Sie `concatMap` selbst, so wie es die Standardbibliothek tut: ohne Verwendung der Funktionen `concat` und `map`; dafür mit Verwendung der Funktionen `(++)`, `(.)` und `foldr`.

A5-2 ***Compose*** Alois Dimpfelmoser möchte eine Funktion programmieren, welche die Summe der Quadrate aller geraden Zahlen aus einer Liste berechnet. Weil Alois der pointfree-Stil so gut gefällt (komischerweise sogar besser als List-Comprehensions), hat er unter Verwendung von `compose` aus Folie 7-25 folgendes dazu implementiert:

```
geradequadratsumme = compose [sum, map (^2), filter even]
```

Leider mag GHC diese Definition nicht! Helfen Sie den armen Alois!

a) Wo liegen der/die Fehler?

b) Wie lautet die richtige Definition im pointfree-Stil?

A5-3 ***Falten*** Der Datentyp `Ziffer` sei wie folgt definiert:

```
data Ziffer = Null | Eins  | Zwei   | Drei | Vier
            | Fünf | Sechs | Sieben | Acht | Neun
  deriving (Show, Read, Eq, Ord, Enum, Bounded)
```

a) Schreiben Sie eine Funktion `ziffer2integer :: [Ziffer] -> Integer`, welche eine Liste von Ziffern in eine Zahl umwandelt: `ziffer2integer [Eins,Zwei,Neun] = 129` Verwendung Sie dazu mindestens einer der in der Vorlesung vorgestellten Faltungsfunktionen, also `foldl`, `foldr`, etc. Auch Funktionen der Klasse `Enum` könnten nützlich sein.

b) Schreiben Sie eine Funktion `integer2ziffer :: Integer -> [Ziffer]`, so dass `integer2ziffer . ziffer2integer` sich für positive Zahlen verhält wie die Funktion `id`. Verwenden Sie dazu folgende rekursive Funktion höherer Ordnung:

```
unfold p h t x | p x = []
               | otherwise = let construct = unfold p h t $ t x
                             in h x : construct
```

H5-1 ***Die Eule*** **(0 Punkte)** (`.hs`-Datei als Lösung abgeben)
Manche Leute sagen, der durch `((.)$(.))` definierte Infix-Operator ist *pointfree*; manche Leute sagen, er ist *pointless*; ich aber sage Euch: Implementiert diesen Operator in herkömmlicher Weise *mit Punkten*!

H5-2 ***Abstiegsfunktion IV*** **(2 Punkte)** (Abgabeformat: Text oder PDF)
Beweisen Sie, dass folgende Funktionen für alle Eingaben terminiert. Dabei dürfen Sie annehmen, dass die als Argument übergebene Funktion `f` ihrerseits für alle Argumente terminiert.

```
pam :: (a -> b) -> [b] -> [a] -> [b]
pam f xs []     = xs
pam f xs (y:ys) = pam f ((f y):xs) ys
```

Hinweis: Die Aufgabe ist einfacher, als sie auf den ersten Blick erscheinen mag.

H5-3 ***Polynome mit Funktionen modellieren*** **(6 Punkte)** (`.hs`-Datei abgeben)
Eine alternative Repräsentation von Polynomen ist gegeben durch folgende Datentypdeklaration:

```
data PolyFun = PolyFun { coeff :: Int -> Double, degree :: Int }
```

Ein Wert des Typs `PolyFun` besteht also aus zwei Komponenten: eine Funktion, welche den Koeffizienten für den angegebenen Exponenten berechnet und einem `Int`-wert ≥ -1, der den Grad des Polynoms angibt. Der Grad eines Polynoms ist der höchste vorkommende Exponent, d.h. für jedes `p :: PolyFun` soll gelten `(coeff p) (degree p) /= 0`. Das Nullpolynom 0 definieren wir mit `nullpoly = PolyFun {coeff= const 0, degree= -1}`

Beispiel: Das Polynom $1.1x^4 + 2.2x + 3.3$ wird modelliert mit

```
myp = PolyFun { coeff=myc, degree=4 }
   where myc :: Int -> Double
         myc 4 = 1.1
         myc 1 = 2.2
         myc 0 = 3.3
         myc _ = 0.0
```

a) Schreiben Sie eine Funktion `polyF2L :: PolyFun -> Polynom`, welche ein Polynom des Typs `PolyFun` in die Repräsentation aus Aufgabe H4-3 überführt!

b) Schreiben Sie eine Funktion `berechnePolyF :: Double -> PolyFun -> Double` welche den Wert eines Polynoms für einen gegebenen x-Wert berechnet:
`berechnenPolyF PolyFun { degree=2, coeff= \x->if x==2 then 3 else 0 } 5 == 75`

Hinweis: Für die volle Punktzahl auf dieser Teilaufgabe müssen Sie `foldl` oder `foldr` verwenden. Keine Punkte erhalten Sie, wenn Sie `polyF2L` verwenden.

c) Schreiben Sie eine Funktion `multPolyFun :: PolyFun -> PolyFun -> PolyFun`, welche zwei Polynome in der Repräsentation des Typs `PolyFun` miteinander multipliziert. Achten Sie auf Randfälle! Sie dürfen verwenden, was Sie möchten, aber es kann zu Punktabzug bei grob ineffizienten Programmen kommen.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 27.05.2014, 11:00 Uhr mit UniworX abgegeben werden.

Aufgrund des Klausurbonus müssen die Hausaufgaben von Ihnen alleine gelöst werden. Abschreiben bei den Hausaufgaben gilt als Betrug und kann zum Ausschluss von der Klausur zur Vorlesung führen.