Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
30. Juni 2015

10. Übung zur Vorlesung

Programmierung und Modellierung

Hinweise:

- Einmalige Raumänderung für die Übungen am Freitag, den 3.7.2015:
  10:00h→E216, 12:00h→E216, 14:00h→A014 im gleichen Gebäude.
- Donnerstag, 9.7., 17:00, B201: Freiwillige Fragestunde der ProMo-Tutoren!

A10-1 ***Wiederholung*** Rechnen Sie zügig mit Papier & Bleistift aus, zu welchem Wert der jeweilige Haskell Programmausdruck auswertet. Es reicht die Angabe des Endergebnis. Sie dürfen Ihnen unbekannte Funktionen dabei gerne in der Standardbibliothek nachschlagen!

a)
```
let bar x y z = if 1 < succ x then z else x+y
in  bar 0 2 (3 `div` 0)
```

b)
```
[(x,y,z)| x<-[minBound..maxBound], y<-[7,8], z<-[2,3], x==(even(y+z))]
```

c)
```
(\x y -> reverse $ ():x:():[()]) (()) ()
```

d)
```
let foo f = \x-> x : foo f (f x) in take 3 $ foo (*2) 2
```

e)
```
let m = Data.Map.fromList [('a',4),('b',7),('c',6),('d',9)]
in Control.Monad.mapM_ (Prelude.flip Data.Map.lookup m) ['b'..'c']
```

A10-2 ***Fizz buzz*** Im Kinderspiel "Fizz buzz" sitzen alle Teilnehmer in einem Kreis; ein Spieler beginnt und sagt "1", der nächste Spieler sagt dann schnell die nächsthöhere Zahl. Falls die Zahl jedoch durch 3 teilbar ist, so muss der Spieler "fizz" sagen. Falls die Zahl durch 5 teilbar ist, so muss der Spieler "buzz" sagen. Ist die Zahl sowohl durch 3 als auch durch 5 teilbar, so muss "fizz buzz" gesagt werden. Wer einen Fehler macht, scheidet aus!

Schreiben Sie fix ein Haskell Programm, welches dieses Spiel für die Zahlen 1 bis 111 ausführt. Dabei wird in jede Antwort in einer eignen Zeile wiedergegeben:

```
1
2
fizz
4
buzz
fizz
7
```

Versuchen Sie eine Version dieses Programmes zu erstellen, welche möglichst kurz und ohne direkte rekursive Aufrufe auskommt! Verwenden Sie also die in der Vorlesung behandelten Funktionen aus Modul `Control.Monad`

A10-3 ***Fehler-Monade*** Machen Sie den folgenden Datentyp `Entweder` zur Monade:

```
import Control.Applicative
import Control.Monad

data Entweder a b = Eines b | Anderes a deriving (Show, Eq, Ord)
```

Die Grundidee dieser Monade ist wie bei `Maybe`: eine erfolgreiche Berechnung liefert einen mit `Eines` verpackten Wert, währned ein Fehler durch die Rückgabe von `Anderes` signalisiert wird. Während die `Maybe`-Monade bei einem Fehler nur `Nothing` zurückliefert, könnte hier `Anderes` noch eine Fehlerbeschreibung zusätzlich liefern. Versuchen Sie zur Lösung dieser Aufgabe möglichst wenig in Vorlesungskapitel 09 nachzuschlagen!

Beispiel:

```
> (*) <$> (Eines 3) <*> (Eines 4)
Eines 12
> let foo x y = if y>0 then Eines $ x `div` y else Anderes "Div-by-Zero"
> foldM foo 100 [2,5,3]
Eines 3
> foldM foo 100 [2,5,0,3]
Anderes "Div-by-Zero"
```

Hinweise:

- Welchen Kind hat der Typkonstruktor `Entweder`? Welchen Kind benötigt die Instanzdeklaration für die Monade?
- Berücksichtigen Sie die Monaden-Gesetze!
- Der Wert `Anderes "foo"` des Typs `Entweder String Int` kann nicht einfach als Wert des Typs `Entweder String Double` aufgefasst werden! Hier muss umverpackt werden, d.h. den Konstruktor `Anderes` erst entfernen, danach wieder erneut davor setzen. Je nach Typ wird ja auch eine andere Menge an Speicherplatz reserviert. Fehlermeldungen wie `Couldn't match type 'a1' with 'b'...` oder `Could not deduce (b ~ a1)`... weisen auf dieses Problem hin.

H10-1 ***AVL Bäume*** (0 Punkte; Datei `H10-1.hs` als Lösung abgeben)
In der Vorlesung am 29.6.2015 wurden verschiedene Optimierungen von balancierten Suchbäumen besprochen. Implementieren Sie gemäß der Beschreibung in Kapitel 07 eine Funktion zum Einfügen eines Wertes in einen AVL-Baum

```
data AVL a = Empty | Node { label :: a, left,right :: Tree a, balance :: Int }

avl_insert :: Ord a => a -> AVL a -> AVL a
```

Hinweis: Diese Aufgabe mag etwas schwerer sein, ist aber durchaus auch lehrreich. Wer nicht weiter kommt findet eine Dateivorlage auf der Vorlesungshomepage, bei der schon viel vorgegeben wurde.

H10-2 ***Monadische Komposition*** (4 Punkte; Datei `H10-2.hs` als Lösung abgeben)

a) Schreiben Sie eine monadische Funktion `frage :: String -> IO String`, welche Ihr Argument auf den Bildschirm ausgibt und gleich danach den Benutzer nach einer Antwort fragt, und diese als Ergebnis zurück liefert. Verwenden Sie die DO-Notation!

b) Schreiben Sie folgende Infix-Funktion aus Modul `Control.Monad` selbst:

```
(<=<) :: Monad m => (b -> m c) -> (a -> m b) -> (a -> m c)
```

Diese Funktion erwartet zwei monadische Funktionen als Argument und verschmilzt diese zu einer einzigen. *Vorlage:*

```
-- import Control.Monad -- VERBOTEN
infixr 1 <=<
_ <=< _ = undefined -- TODO
```

Hinweise:

- Diese Teilaufgabe ist dank DO-Notation auch nicht schwerer als die Teilaufgabe a)!
- In **allen Teilaufgaben dieser Aufgabe** sind keinerlei Imports erlaubt! Die gesuchten Funktionen sind ja in Modul `Control.Monad` bereits enthalten, so dass nichts mehr zu tun wäre. (Für die späteren Teilaufgaben können `(>>=)` und `return` verwendet werden, da diese bereits durch Modul `Prelude` zur Verfügung stehen.)
- Verwenden Sie `infixr 1 <=<` zur Deklaration der Präzedenz des Infix-Operators.

c) Schreiben Sie eine Funktion `frageV2 :: String -> IO String`, welche funktioniert wie in Teilaufgabe a), aber vewenden Sie dieses Mal anstatt der DO-Notation Ihre neu definierte Infix-Funktion `(<=<)`!

Hinweise: Dabei gibt es das Problem, dass die Funktion `getLine` kein Argument erwartet, und daher vom Typ nicht als Argument für `(<=<)` geeignet ist. Dies kann man jedoch leicht mit einer anonymen Funktion lösen! Diese Aufgabe dient zur Übung, die Verwendung von `(<=<)` ist in diesem Fall vielleicht etwas unsinnig.

d) Lösen Sie Teilaufgabe b) erneut, ohne dabei die DO-Notation zu verwenden. Benutzen Sie stattdessen den Bind-Infix-Operator `(>>=)`. Benennen Sie zur Unterscheidung diese zweite Version `komp2 :: Monad m => (b -> m c) -> (a -> m b) -> (a -> m c)`

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 7.07.2015, 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.