Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
12. Mai 2015

4. Übung zur Vorlesung

Programmierung und Modellierung

A4-1 ***Instanzen von Typklassen definieren*** Gegeben sind folgende Deklarationen:

```
data List a = Leer | Element a (List a)  deriving Show

class Messbar a where
  messen :: a -> Double
```

a) Machen Sie den Datentyp `List a` zu einer Instanz der Klasse `Messbar`. Das Maß eines Wertes des Typs `List a` soll einfach die Länge der Liste sein.

b) Machen Sie den Datentyp `List a` zu einer Instanz der Klasse `Eq`, falls auch `Eq a` gilt. Siehe dazu auch Folie 05-15. (Die Aufgabe einfach nur durch Hinzufügen von `deriving Eq` zu lösen ist hier natürlich nicht im Sinne der Aufgabenstellung.)

c) Vergleichen Sie die Funktion `compare :: Ord a => a -> a -> Ordering` von Folie 05-26 mit folgender Funktionsdefinition und finden Sie einen grundlegenden Unterschied:

```
vergleiche :: (Messbar a, Messbar b) => a -> b -> Ordering
vergleiche x y = messen x `compare` messen y
```

A4-2 ***Induktion*** Gegeben ist folgende rekursive Funktionsdefinition:

```
foo :: (Integer,Integer) -> Integer
foo (n,m)  |  n == 0    = m+1
           |  n >  0    = foo (n-1, foo (n-1,m))
           |  otherwise = error "1. argument for foo must not be negative."
```

Beweisen Sie mit Induktion, dass für beliebige $(n, m) \in \mathbb{N} \times \mathbb{N}$ die Gleichung $\texttt{foo}(n, m) = 2^n + m$ gilt. Überlegen Sie sich zuerst worüber die Induktion geführt werden soll, also z.B. ob über n oder m oder $n + m$ oder $n \cdot m$ oder $\max(n, m)$ oder ...

A4-3 ***Induktion mit Listen I*** Wir bezeichnen mit $|l| \in \mathbb{N}$ die Länge einer Liste l. Für eine beliebige nicht-leere Liste $(x : xs)$ gilt $|(x : xs)| = 1 + |xs|$.

a) Beweisen Sie mit Induktion über die Länge der Liste, dass für alle ganzen Zahlen $z \in \mathbb{Z}$, für alle natürlichen Zahlen $n \in \mathbb{N}$, und alle Listen von ganzen Zahlen zs mit $|zs| = n$ die Gleichung $|\texttt{insert z zs}| = 1 + |zs|$ gilt. Die Funktion `insert` ist definiert wie auf Folie 3-25:

```
insert :: Int -> [Int] -> [Int]
insert x   [] = [x]
insert x (y:ys) | x <= y    = x : y : ys
                | otherwise = y : insert x ys
```

Da Haskell eine rein funktionale Sprache ist, dürfen wir die definierenden Gleichung einer Funktion beliebig von links-nach-rechts oder auch von rechts-nach-links einsetzen. Lediglich bei überlappenden Mustervegleichen und/oder Wächtern müssen wir die entsprechenden Seitenbedingungen beachten. So dürfen wir zum Beispiel in unseren Rechnungen den Ausdruck $y : \texttt{insert}\ x\ ys$ nur dann durch `insert x (y : ys)` ersetzen, wenn $x > y$ angenommen werden darf. Das ist eigentlich klar, denn ansonsten würde ja bei der Ausführung der andere Zweig ausgewählt.

b) Beweisen Sie, dass für eine beliebige Liste l von ganzen Zahlen gilt

$$|l| = |\texttt{sort}\ l|$$

Dabei dürfen Sie die Aussage von Teilaufgabe a) ohne weiteres direkt verwenden. Die Funktion `sort` ist definiert wie auf Folie 3-25.

```
sort :: [Int] -> [Int]
sort    []  = []
sort (x:xs) = insert x (sort xs)
```

A4-4 ***Fehlerhafte Induktion*** Durch vollständige Induktion wollen wir zeigen:

Alle Smartphones benutzen das gleiche Betriebssystem.

Um Induktion einsetzen zu können, verallgemeinern die Aussage zu *"In einer Menge von n Smartphones benutzen alle das gleiche Betriebssystem."* Da die Anzahl aller Smartphones in der Welt eine natürliche Zahl ist, reicht dies für die ursprüngliche Aussage.

Der Induktionsanfang ist klar: Ein Smartphone benutzt das gleiche Betriebssystem wie es selbst.

Angenommen, wir hätten eine Menge von $n+1$ Smartphones. Wählen wir irgendein Smartphone davon aus und bezeichnen es mit s. Die übrigen Smartphones bilden eine Menge von n Smartphones. Nach Induktionsvoraussetzung benutzen diese n Smartphones alle das gleiche Betriebssystem. Nun entfernt man von diesen n Smartphones mit gleichem Betriebssystem eines und fügt s wieder dazu. Damit haben wir wieder eine Menge mit n Smartphones, welche nach Induktionsvoraussetzung wieder alle das gleiche Betriebssystem besitzen – insbesondere also auch s. Damit benutzen aber offenbar alle $n+1$ Smartphones das gleiche Betriebssystem und die Behauptung ist bewiesen.

Wo liegt der Fehler? Begründen Sie Ihre Antwort!

H4-1 ***Wiederholung: Listen, Rekursion*** (0 Punkte; Datei `H4-1.hs` als Lösung abgeben) Implementieren Sie schnell die Funktion `concat :: [[a]] -> [a]` aus der Standardbibliothek, welche die Elemente einer Liste von Listen miteinander verkettet. Verwenden Sie dazu Pattern-Matching auf Listen und Rekursion. Zur Vermeidung von Namenskonflikten soll die Funktion `myConcat` heissen! Sie dürfen die Infix-Funktion `(++) :: [a] -> [a] -> [a]` verwenden. *Beispiele:*

```
> myConcat [[1,2,3],[4,5],[],[6],[7,8]]   |   > myConcat ["Hi ", "there", "!"]
[1,2,3,4,5,6,7,8]                         |   "Hi there!"
```

H4-2 ***Induktion mit Listen II*** (4 Punkte; Abgabeformat: Text oder PDF)
Es sei vs und ws zwei beliebige Listen, weiterhin sei $n = |vs|$. Beweisen Sie mit Induktion über die Länge $n \in \mathbb{N}$ von vs, dass $|\texttt{zip}\ vs\ ws| = \min\big(|vs|, |ws|\big)$ gilt, wobei `zip` definiert ist wie auf Folie 03-26. *Hinweis:* Eine Induktion über n reicht aus. Eine Induktion über die Länge von ws ist nicht notwendig!

```
zip :: [a] -> [b] -> [(a,b)]
zip    []     _   = []
zip    _      []  = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

H4-3 ***Instanzen II*** (4 Punkte; Datei `H4-3.hs` als Lösung abgeben)
Gegeben sind folgende Deklarationen:

```
data Brotzeit = Leberkas | Weisswurscht Int | Breze Brotzeit
  deriving (Show, Eq)

class Messbar a where
  messen :: a -> Double
```

a) Machen Sie den Typ `Brotzeit` zur Instanz der Klasse `Messbar`. Jedes Lebensmittel zählt einmal; also der Leberkas hat das Maß 1, n Weisswürste haben das Maß $\frac{n}{2}$, und eine Breze mit Brotzeit hat das Maß 1 plus der beinhalteten Brotzeit.

b) Machen Sie den Typ `Brotzeit` zu einer Instanz der Klasse `Ord`. Eine Brotzeit umso "größer", je mehr Brezen diese enthält. Eine Leberkas ist genausogut wie 2 Weisswürschte, aber 3 Weisswürschte schlagen jeden Leberkas. Siehe dazu auch Folie 05-26. *Beispiele:*

```
> compare (Breze Leberkas) (Breze (Weisswurscht 1))
GT
> compare (Breze (Weisswurscht 7)) (Breze (Breze Leberkas))
LT
> compare (Weisswurscht 3) Leberkas
GT
```

Hinweis: Die 5. Übung findet nächste Woche regulär statt. Übungsblatt 6 wird aufgrund der Feiertage am Do 28.5., Fr 29.5., Di 2.6. und Mi 3.6. behandelt werden. Es entfallen lediglich die Übungen am Mi 27.5. und Fr 5.6. außerplanmäßig.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den 19.05.2015, 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.