Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
3. Juni 2014

7. Übung zur Vorlesung

Programmierung und Modellierung

Hinweise: Am 12.6. um 17h findet in B101 eine Aufholübung statt. Näheres dazu findet sich auf der Vorlesungshomepage.

Die Übungen am 10.6., 11.6. und 20.6.(!) entfallen. Übungen finden statt am:

Übung	**Dienstag**	**Mittwoch**	**Freitag**
7.	3.6.	4.6.	6.6.
8.	17.6.	18.6.	**13.6.**
9.	24.6.	25.6.	27.6.

Bitte verwenden Sie zur Bearbeitung die auf der Vorlesungshomepage bereitgestellten Dateivorlagen. Bearbeiten Sie die mit `-- TODO` markierten Stellen. Sie dürfen auch neue Hilfsfunktion definieren, und alle vorhandenen Funktion nutzen; Sie dürfen aber nicht die importierten Module am Anfang der Datei abändern.

A7-1 *Data.Map* Das Modul `Data.Map` der Standardbibliothek bietet die Verwendung von endlichen Abbildungen über eine Implementierung von balancierten Suchbäumen bereit. In dieser Aufgabe wollen wir lernen, wie wir dieses Modul benutzen können.

Importieren Sie das Modul mit `import qualified Data.Map as Map`, um Namenskonflikte zu vermeiden (siehe dazu auch Folie 6-37). Wichtige Funktionen des Moduls sind:

```
empty  :: Map k a
insert :: Ord k => k -> a -> Map k a -> Map k a
delete :: Ord k => k -> Map k a -> Map k a
lookup :: Ord k => k -> Map k a -> Maybe a
```

weitere nützliche Funktionen finden Sie in der Dokumentation des Untermoduls `Data.Map.Lazy` in Standardbibliothek. Die Unterteilung von `Data.Map` in Untermodule können Sie ignorieren.

a) Dr. Jost möchte zur Erbauung der Moral ein Programm zur Verwaltung der ProMo-Studenten schreiben. Dazu hat er bereits folgendes implementiert:

```
import qualified Data.Map as Map

data Bemerkung = Störer | Abschreiber deriving (Eq, Show)
type Student = String -- zur Vereinfachung, besser newtype/data verwenden!
type Register  = Map.Map Student [Bemerkung]

eintragStörer :: Student -> Register -> Register
eintragStörer stud sreg = undefined

hatGestört :: Student -> Register -> Bool
hatGestört stud sreg = undefined
```

Vervollständigen Sie die Definitionen für `eintragStörer` und `hatGestört`, so dass folgendes funktioniert:

```
> :m + Data.Map
> let reg = eintragStörer "Eustachius" $ eintragStörer "Wally" empty
> hatGestört "Xaver" reg
False
> hatGestört "Wally" reg
True
```

b) Eustachius hat eine Umkehrfunktion zu `vervielfache` aus Aufgabe A6-2c geschrieben:

```
import Data.List (nub)
count xs = [(x, length [ 1 | x0 <- xs, x0 == x] ) | x <- nub xs]

> count "abcdabcabaa"
[('a',5),('b',3),('c',2),('d',1)]
```

Diese Implementierung ist korrekt, aber leider ineffizient, wie sie leicht testen können:

```
> :set +s
> let testString n = concat (replicate n "Floki fährt furchtlos mit dem
                                          vollbesetzten Flos durch den fulminanten Fjord. ")
> count $ testString 10000
[('F',30000),('l',60000),('o',50000),('k',10000),('i',30000),(' ',110000),('f',30000),('\228',10000),('h'
(4.56 secs, 1149487632 bytes)
```

Reimplementieren Sie `count` durch die Verwendung von `Data.Map` effizienter!

A7-2 ***Suchbäume*** Ein *Suchbaum* ist ein Baum mit Beschriftungen aus einer geordneten Menge, bei dem für jeden Knoten k gilt, dass die Beschriftung von k größer-gleich ist als alle Beschriftungen des linken Teilbaums von k und kleiner-gleich als alle Beschriftungen des rechten Teilbaums. Ein binärer Suchbaum ist balanciert, wenn für jeden Knoten k gilt, sich die Höhe der beiden Teilbäume von k höchstens um 1 unterscheidet.

Die Funktion `isBalanced` entscheidet, ob ein Binärbaum balanciert ist (die Suchbaumeigenschaft selbst wird hier zur Vereinfachung nicht geprüft):

```
data Tree a = Empty | Node { label :: a, left,right :: Tree a }
deriving (Show, Eq)

tiefe :: Tree a -> Int
tiefe Empty = 0
tiefe Node {left=l,right=r} = 1 + max (tiefe l) (tiefe r)

isBalanced :: Tree a -> Bool
isBalanced Empty = True
isBalanced Node {left=l,right=r} =
  let hdiff = (tiefe l) - (tiefe r)
  in isBalanced l && isBalanced r && (abs hdiff <= 1)
```

Die Funktion `abs : int -> int` liefert den absoluten Betrag des Arguments, also gilt z.B. `abs 5 = 5` als auch `abs (-5) = 5`.

a) Geben Sie die worst-case-Laufzeitkomplexität von `isBalanced` an.

b) Geben Sie eine alternative Definition von `isBalanced` an, welche lineare Zeitkomplexität besitzt, d.h. jeden Knoten nur einmal betrachtet. Z.B. betrachtet die Funktion `foldTree` jeden Knoten eines Baumes nur einmal. (Die Suchbaumeigenschaft muss weiterhin nicht geprüft werden, sondern nur die Eigenschaft, balanciert zu sein.)

A7-3 *Typregel* Überlegen Sie sich analog zu den in der Vorlesung behandelten Typregeln eine sinnvolle Typregel für den Haskell Ausdruck `let ... = ... in ...`

a) Die Konklusion der Typregel ist bereits vollständig vorgegeben. Ergänzen Sie lediglich die fehlende(n) Prämisse(n):

$$\frac{}{\Gamma \vdash \texttt{let}\ x\ =\ e_1\ \texttt{in}\ e_2\ ::\ C} \qquad \text{(Let)}$$

Hinweis: Aus der vorgegeben Konklusion der Typregel folgt auch, dass Ihre Typregel kein volles Pattern-Matching wie in Haskell erlauben muss. Es reicht uns hier zur Vereinfachung, wenn der Unterausdruck e_1 mit einer Variablen x gematched wird.

b) Beachten Sie, dass Haskell rekursive let-Definitionen wie z.B.
`let ones = 1 : ones in take 111 ones` erlaubt. Ihre Typregel soll dies ebenfalls erlauben, ändern Sie Ihre Typregel aus der vorherigen Teilaufgabe entsprechend ab, falls Typregel dies noch nicht erlaubt.

H7-1 *Data.Map II* (3 Punkte) (`.hs`-Datei als Lösung abgeben)
Fortsetzung von Aufgabe A7-1. Sie dürfen alle Funktion des Moduls `Data.Map` und dessen Untermodule nutzen, so wie alle Funktionen der `Prelude`, wenn Sie möchten. Achten Sie auf eine effiziente Implementierung!

a) Implementieren Sie die Funktion `eintragAbschreiber :: [Student] -> Register -> Register` welche einer Liste von Studenten den Eintrag `Abschreiber` in das Register hinzufügt.

Beispiel:

```
> eintragAbschreiber ["Eustachius","Ignaz"] $
  eintragAbschreiber ["Ignaz","Edeltraud"]  $ Map.empty
fromList [("Edeltraud",[Abschreiber]),
          ("Eustachius",[Abschreiber]),
          ("Ignaz",[Abschreiber,Abschreiber])]
```

b) Implementieren Sie die Funktion `gesamtRegister :: [Register] -> Register` welche die Eintragungen der Register mehrerer Vorlesungen zu einem einzigen Register kombiniert.

H7-2 *Suchbäume II* (2 Punkte) (`.hs`-Datei als Lösung abgeben)
Ein *AVL-Baum* ist ein binärer Suchbaum, in dem für jeden Knoten k gilt, dass sich die Höhe der beiden Teilbäume von k höchstens um 1 unterscheidet. Um die Balance effizient zu halten, trägt jeder Knoten zusätzlich eine Zahl, welche die Höhendifferenz seiner beiden Unterbäume angibt Wir vereinbaren dabei die Konvention: ein positiver Wert steht dafür, dass der rechte Unterbaum größer ist; ein negativer Wert steht dafür, dass der linke Unterbaum größer ist.

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

AVL-Bäume wurden 1962 von **A**delson-**V**elsky und **L**andis als erste Datenstruktur für balancierte Suchbäume vorgeschlagen.

Schreiben Sie eine effiziente Funktion `isAVL :: AVL a -> Bool` welche entscheidet, ob ein Binärbaum ein AVL-Baum ist, d.h. wie in Aufgabe A7-2 muss geprüft werden, ob der Baum ausreichend balanciert ist. Zusätzlich ist jedoch zu prüfen, ob die Balance Annotation korrekt ist. Sie müssen jedoch nicht überprüfen, ob die Knoten-Label der Suchbaumeigenschaft genügen.

H7-3 *Substitution* (3 Punkte) (`.hs`-Datei als Lösung abgeben)
Gegeben ist folgende Datentypdeklaration zur Repräsentation von Termen:

```
data Term = Var Char | Const Int | App Term Term | Abs Char Term
```

a) Schreiben Sie eine Funktion `freeVars :: Term -> [Char]` welche die freien Variablen eines Terms effizient berechnet. *Hinweis:* Die Aufgabe ist sehr einfach, wenn Sie den Unterschied zwischen freien und gebundenen Variablen verstanden haben, siehe dazu auch Folien 10-9 und 10-18.

b) Implementieren Sie die Substitution von Folie 10-8 effizient als Funktion `subst :: (Char, Term) -> Term -> Term`. Dabei steht `subst ('x', t1) t2` für den Term, den man erhält, wenn alle freien Vorkommen von `Var 'x'` in `t2` durch `t1` ersetzt werden.

Hinweise: Bitte beachten Sie, dass Folie 10-8 leider einen Tippfehler enthielt, welcher ab 3.6.14 korrigiert wurde. Laden Sie die Folien ggf. frisch von der Vorlesungshomepage herunter.

Verwenden Sie zur Vereinfachung folgende partielle Funktion zur Erzeugung frischer Variablen:

```
genFreshV :: [Char] -> Char
genFreshV vs = head $ filter (\c -> not $ c `elem` vs) ['a'..'z']
```

Abgabe: Lösungen zu den Hausaufgaben können bis Donnerstag, den 12.06.2014, 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.