Prof. Martin Hofmann, PhD
Dr. Steffen Jost

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

5. Übung zur Vorlesung

Programmierung und Modellierung

Hinweise:

- Ab sofort dürfen alle Funktionen des Moduls `Prelude` verwendet werden, so fern in einer Aufgabe nichts anderes angegeben wurde.
- Die Bearbeitungszeit der Hausübungen für dieses Blatt beträgt 14 Tage.
- Übungsblatt 6 wird aufgrund der Feiertage am Do 28.5., Fr 29.5., Di 2.6. und Mi 3.6. behandelt werden. Es entfallen außerplanmäßig die Übungen am Mi 27.5. und Fr 5.6.

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.

A5-2 *Funktionen höherer Ordnung*

a) Implementieren Sie folgende Funktionen:

```
curry3   :: ((a, b, c) -> d) -> a -> b -> c -> d
uncurry3 :: (a -> b -> c -> d) -> (a, b, c) -> d
```

Hinweis: Die Typsignaturen lassen bei der Implementation einer totalen Funktion hier schon keine Wahl mehr zu, wenn man auf Schummeleien wie `undefined`, `error` oder endlose Rekursion verzichtet.

b) Diskutieren Sie anhand des Typs den Unterschied zwischen folgenden drei Funktionen:

i) `uncurry3 foldr`
ii) `uncurry (uncurry foldr)`
iii) `(uncurry . uncurry) foldr`

A5-3 ***Modul*** *`Data.Map`* Dr. Jost möchte zur Erbauung der Moral Bonuspunkte für Hausübungen verteilen. Zur Verwaltung dieser Bonuspunkte erstellt er folgendes Modul:

```
module Bonus (eintragPunkte, eintragAbschreiber, punkteAuslesen, leer) where

  import qualified Data.Map as Map

  data Bonus    = Punkte [Int] | Abschreiber
  type Student  = String -- zur Vereinfachung, besser data verwenden
  type Register = Map.Map Student Bonus

  leer :: Register
  leer = undefined --TODO

  punkteAuslesen :: Student -> Register -> Int
  punkteAuslesen = undefined --TODO

  eintragAbschreiber :: Student -> Register -> Register
  eintragAbschreiber = undefined --TODO

  eintragPunkte :: Student -> Int -> Register -> Register
  eintragPunkte = undefined --TODO
```

Ein `Register` ordnet jedem `Student` einen `Bonus` zu. Ein `Bonus` ist entweder eine Liste von erzielten Hausaufgabenpunkten oder der Vermerk, dass der Student mindestens einmal abgeschrieben hat.

a) Vervollständigen Sie die Implementation durch Bearbeitung der vier mit –TODO— markierten Stellen. Die Funktion `punkteAuslesen` soll die Summe der erzielten Hausaufgabenpunkte für den abgefragten Studenten liefern; ist der Student unbekannt oder ein bekannter Abschreiber, so soll 0 geliefert werden. `eintragAbschreiber` soll den Eintrag des Studenten auf `Abschreiber` setzen; alle eventuell bis dahin erzielten Bonuspunkte verfallen. `eintragPunkte` fügt einem Eintrag eines Studenten die angegebenen Punkte hinzu; dabei sollen zur Protokollierung die Punkte nicht sofort aufaddiert werden, sondern die Liste mit Punkten einfach um einen Eintrag erweitert werden; unbekannte Studenten sollen neu angelegt werden; Abschreiber dürfen keine Punkte mehr sammeln und bleiben Abschreiber. *Beispiele:*

```
> let r1 = eintragPunkte      "Martin"  8 (eintragPunkte "Steffen" 4 leer)
> let r2 = eintragAbschreiber "Martin"    (eintragPunkte "Steffen" 6 r1)
> let r3 = eintragPunkte      "Martin" 12 (eintragPunkte "Steffen" 2 r2)
> punkteAuslesen "Steffen" r3
12
> punkteAuslesen "Martin" r3
0
```

b) Warum werden in der ersten Zeile des oben vorgegebenen Codes vier Funktionsnamen in runden Klammern gelistet? Was wäre der Vorteil/Nachteil, wenn dort nur `module Bonus where` stünde?

H5-1 ***Falten*** (0 Punkte; Datei `H5-1.hs` als Lösung abgeben)
Schnell und ohne dabei in die Folien zu schauen: Definieren Sie die Funktionen `length` und `reverse` ohne direkte Rekursion, sondern nur unter Verwendung von `foldl`.
Zur Erinnerung: `foldl :: (b -> a -> b) -> b -> [a] -> b`

H5-2 ***Module*** (6 Punkte; Datei `Warteschlange.hs` abgegeben)
In der Vorlesung wurde das Modul `Data.Map` behandelt, welche eine abstrakte Datenstruktur für endliche Abbildungen bereitstellt.

Schreiben Sie ganz ähnlich dazu ein Modul `Warteschlange`, welches ausschließlich folgende Funktionen exportiert:

```
leer       :: Warteschlange a
einstellen :: a -> Warteschlange a -> Warteschlange a
abholen    :: Warteschlange a -> (Maybe a, Warteschlange a)
fanwenden  :: (a -> b) -> Warteschlange a -> Warteschlange b
```

a) Funktion `leer` liefert eine leere Warteschlange. Bei einer Warteschlange können wir mit `einstellen` Werte in die Warteschlange einfügen; und mit `abholen` das zuerst eingestellte Element wieder herausholen (First-In-First-Out).

Es ist Ihnen überlassen, wie Sie die Warteschlange innerhalb des Moduls tatsächlich implementieren, so lange alle vorgegebenen Typsignaturen eingehalten werden. Wichtig ist, dass der Datentyp abstrakt bleibt, d.h. es werden keine Konstruktoren, sondern nur die oben genannten Funktionen exportiert.

Das Beispiel in der rechten Spalte verrät auch schon eine Möglichkeit, eine Warteschlange zu implementieren: Eine Warteschlange besteht aus zwei Listen; neue Elemente fügen wir immer in die erste Liste ein; das nächste auszuliefernde Element ist der Kopf der zweiten Liste. Falls die zweite Liste leer ist, dann befüllen wir diese durch Umdrehen aller Elemente der ersten Liste. Sind beide Listen leer, dann liefert `abholen` einfach `(Nothing,leer)` zurück.

Sie dürfen hier alle Funktionen des Moduls `Data.Map` (und `Prelude`) verwenden; die auf den Folien 05-40ff. behandelten Funktion reichen aber bereits aus.

Beispiel:

```
> einstellen 'a' leer
WS("a","")
> einstellen 'b' it
WS("ba","")
> einstellen 'c' it
WS("cba","")
> abholen it
(Just 'a',WS("","bc"))
> abholen (snd it)
(Just 'b',WS("","c"))
> einstellen 'd' (snd it)
WS("d","c")
> abholen it
(Just 'c',WS("d",""))
> abholen (snd it)
(Just 'd',WS("",""))
> abholen (snd it)
(Nothing,WS("",""))
```

b) Implementieren Sie die Funktion `fanwenden`, welche eine Funktion auf alle Elemente in der Warteschlange anwendet. *Beispiele:*

```
> abholen (fanwenden (*2) (einstellen 5 (einstellen 3 (einstellen 1 leer))))
(Just 2,WS([],[6,10]))

> fanwenden fromEnum (einstellen 'B' (einstellen 'A' leer))
WS([66,65],[])
```

H5-3 ***Abstiegsfunktion III*** (3 Punkte; Abgabeformat: Text oder PDF)
Beweisen Sie, dass folgende Funktion für alle nicht-leeren Listen terminiert, welche ausschließlich gerade Zahlen enthalten.

```
baz :: [Int] -> Int
baz (x1:x2:xs)
  | even x1, even x2  = baz (2*x1 : xs) + baz (xs ++ [2*x2])
  | otherwise         = baz (x1+x2 `div` 2 : 77 : xs)
baz [x1] = x1
baz []   = 42 `div` 0
```

Hinweis: Auf) und Def) sind in dieser Aufgabe besonders wichtig! Sie dürfen zur Vereinfachung annehmen, dass das l_1`++`l_2 nur Elemente enthält, welche in dem Argument l_1 und/oder l_2 vorkommen; und dass $|l_1$`++`$l_2| = |l_1| + |l_2|$ gilt.

Abgabe: Lösungen zu den Hausaufgaben können bis Dienstag, den **2.06.2015**, 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.