Hochschule Flensburg
Fachbereich 3
Angewandte Informatik

Jan Christiansen

Hochschule Flensburg
University of Applied Sciences

7. Übung zur Vorlesung „Fortgeschrittene funktionale Programmierung“

Funktoren und IO-Monade
Labor am Freitag, 11. November 2016, 12:15 Uhr

Aufgabe 1 - Funktoren

In dieser Aufgabe sollen Sie die Typklasse *Functor* definieren und eine Reihe von Instanzen dieser Typklasse definieren. Fügen Sie am Beginn Ihres Programms die folgende **import**-Anweisung hinzu, um die vordefinierte Typklasse *Functor* zu verstecken.

```
import Prelude hiding (Functor (..))
```

Definieren Sie die Typklasse *Functor*, wie Sie sie in der Vorlesung kennengelernt haben. Definieren Sie nun die folgenden Instanzen dieser Typklasse.

- Definieren Sie eine Instanz der Typklasse *Functor* für den Typkonstruktor *Maybe*.
- Definieren Sie eine Instanz der Typklasse *Functor* für den folgenden Datentyp.

  ```
  data Tree a = Empty | Node (Tree a) a (Tree a)
  ```

- Definieren Sie eine Instanz der Typklasse *Functor* für die partielle Typanwendung *Either e*. Dabei ist *Either e* die partielle Anwendung des zweistelligen Typkonstruktors *Either* auf die Typvariable *e*.

optional Definieren Sie eine Instanz der Typklasse *Functor* für die partielle Typanwendung $(,)\ e$. Dabei ist $(,)\ e$ die partielle Anwendung des zweistelligen Typkonstruktors für Paare auf eine Typvariable.

optional Definieren Sie eine Instanz der Typklasse *Functor* für den Typ $(\rightarrow)\ r$. Dabei ist $(\rightarrow)\ r$ die partielle Anwendung des zweistelligen Typkonstruktors $(\rightarrow)$ auf die Typvariable r. Überlegen Sie sich zuerst, welchen Typ die Funktion *fmap* in diesem Fall haben muss.

Aufgabe 2 - Either

Wir betrachten die folgende Funktion.

```
readInt :: String → Either String Int
readInt str =
  case reads str of
    [(n, "")] → Right n
    _         → Left err
  where
    err = "Die Zeichenkette " ++ str ++ " ist keine Zahl"
```

Diese Funktion liest aus einer Zeichenkette eine ganze Zahl und liefert eine Fehlermeldung, falls die Umwandlung fehlgeschlagen ist.

Schreiben Sie nun eine Funktion *rangeCheck* :: *Int* → *Int* → *Int* → *Either String Int*, die überprüft, ob eine Zahl in einem bestimmten Bereich liegt. Die beiden Grenzen des Bereiches werden der Funktion ebenfalls als Argumente übergeben.

Schreiben Sie mit Hilfe der beiden definierten Funktionen eine Funktion *readIntRange* :: *Int* → *Int* → *String* → *Either String Int*. Diese Funtion erhält eine untere und eine obere Grenze und eine Zeichenkette, wandelt die Zeichenkette in eine Zahl um und prüft, ob die Zahl in dem entsprechenden Bereich liegt. Nutzen Sie für die Definition der Funktion *readIntRange* den Operator (>> −), den Sie in einer der frühereren Übungen definiert haben.

Aufgabe 3 - Interaktives Menü

In dieser Aufgabe soll es um die Implementierung von Funktionen in der *IO*-Monade gehen.

Implementieren Sie eine Funktion *menu* :: [*String*] → *IO Int*, die ein Menü anzeigt. Die Funktion zeigt die übergebene Liste als Optionen an, die der Nutzer wählen kann. Dabei werden die Einträge durchnummeriert. Anschließend fragt die Funktion den Benutzer nach einer Zahl, mit der eine der Optionen gewählt werden kann. Falls die eingegebene Zahl einem validen Index entspricht, liefert *menu* diese Zahl als Ergebnis zurück. Liegt die Zahl außerhalb des gültigen Bereichs, wird das Menü ein weiteres Mal angezeigt und eine weitere Eingabe gefordert. Dieser Prozess wird wiederholt, bis eine valide Eingabe getätigt wurde. Verwenden Sie für die Eingabe der Zahl die Funktion *readIntRange* aus der vorherigen Aufgabe.

Schreiben Sie auf Basis von *menu* eine Funktion *menuSelect* :: [(*String*, *IO a*)] → *IO a*, die eine Liste von Optionen und IO-Aktionen erhält, dem Benutzer die verschiedenen Optionen zur Wahl stellt und nach der Wahl die Aktion an der gewählten Stelle ausführt.

Schreiben Sie möglichst viele pure Funktionen, das heißt, verwenden Sie die *IO*-Monade nur, wo dies unbedingt notwendig ist. Sie dürfen alle Funktinonen nutzen, die Haskell zur Verfügung stellt.

Aufgabe 4 - Word-Count

Definieren Sie eine Funktion *wordCount* :: *FilePath* → *IO* (), die die Anzahl der Zeichen, Wörter und Zeilen in einer angegebenen Datei zählt. Im Folgenden sehen Sie zum Beispiel einen Aufruf der Funktion *wordCount*.

```
*WordCount> wordCount "pg100.txt"
1. Count chars in file
2. Count words in file
3. Count lines in file
4. Quit
1
5589887
```

Mit Hilfe der Funktion *readFile* :: *FilePath* → *IO String* können Sie den Inhalt einer Datei einlesen. Dabei ist *FilePath* ein Typsynonym für den Typ *String*. Sie können außerdem die Funktionen *lines* :: *String* → [*String*] und *words* :: *String* → [*String*] verwenden.

Nutzen Sie die gesammelten Werke von Shakespeare unter http://www.gutenberg.org/ebooks/100, (Plain Text UTF-8) um Ihr Programm zu testen.

Hinweis: Bitte geben Sie die Lösungen der Aufgaben per Mail oder auf Papier bis zum 18.11. ab. Die Papierlösungen können im ersten Stock von Haus A in den Schrank in das entsprechende Fach geworfen werden.