Westfälische Hochschule

Fachbereich Informatik und Kommunikation
Prof. Dr. Marcel Luis

17. Juni 2016

Aufgabenblatt 9 zu Funktionale Programmierung

Aufgabe 9.1 (Praktikum)

Realisieren Sie eine Funktion $conc :: [String] \to (String, Map)$, wobei Map ein Typsynonym für $[(String, Integer)]$ ist.

$conc$ soll für eine Liste von Zeichenketten die Verkettung dieser Zeichenketten (vgl. $concat$) sowie die Statistik, wie oft jede Zeichenkette vorkommt, liefern. Die Statistik ist eine Liste von Paaren, wobei die ersten Komponenten dieser Paare die Wörter und die zweiten die jeweiligen Häufigkeiten enthalten.

Diese Aufgabe soll eine Übung für (Zustands-) Monaden sein. So wie im Beispiel aus der Vorlesung bzw. dem Bird-Buch die Monade $State$ verwendet wird, um bei der Auswertung eines Terms „nebenbei“ die vorkommenden Div zu zählen, können Sie eine spezielle Monade definieren, um bei der Verkettung der Zeichenketten „nebenbei“ die Wörter-Statistik zu führen.

Aufgabe 9.2 (Praktikum)

Diese Aufgabe bezieht sich auf das Thema der Tautologien, das wir vor einigen Wochen in der Vorlesung behandelt haben.

Definieren Sie eine Grammatik für die Sprache L_{Prop} der Propositionen. Die Sprache soll alle Textdarstellungen von Termen des Datentyps $Prop$ enthalten, in denen so weit wie möglich auf Klammern verzichtet wird, d. h.

$$L_{Prop} = \{show2\ p \mid p \in Prop\}$$

Realisieren Sie einen Parser `parseProp :: Parser Prop` für diese Sprache L_{Prop}.

Hinweis: Damit Sie für die Lösung dieser Aufgabe nicht sämtliche Definitionen zum Parser in Ihr Skript kopieren müssen, habe ich diese in einem Modul `ModulParser` zusammengefasst. Um es zu nutzen, müssen Sie es von der Moodle-Seite herunterladen und in Ihrem Skript Folgendes einbauen:

```
import Data.Char
import ModulParser (Parser(MkP),
                    apply, applyParser,
                    orelse,
                    ident, int, token, symbol)
```

Aufgabe 9.3 (Praktikum)

Realisieren Sie in dem Skript `10_lambda-kalkuel.hs` (s. Code-Beispiele zur Vorlesung) eine Funktion, die die Reduktion eines λ-Terms nach der Strategie *leftmost-innermost* durchführt. (Die vorhandene Funktion $normalOrderEval$ führt die Reduktion nach der Strategie *leftmost-outermost* durch.)

Aufgabe 9.4 (Übung)

Gegeben sei die folgende Monade:

```
data St2 a = MkSt2 (Int -> Int, a)

instance Monad St2 where
  -- return :: a -> St2 a
  return x = MkSt2 (id, x)
  -- (>>=) :: St2 a -> (a -> St2 b) -> St2 b
  (MkSt2 (f, x)) >>= q = MkSt2 (g . f, y)
                         where MkSt2 (g, y) = q x
```

Beweisen Sie, dass für diese Monade die folgenden beiden Monadengesetze gelten.

$$\begin{aligned} p >>= return &= p \\ return\ e >>= q &= q\ e \end{aligned}$$