Hochschule Flensburg
Fachbereich 3
Angewandte Informatik

Jan Christiansen

Hochschule Flensburg
University of Applied Sciences

8. Übung zur Vorlesung „Fortgeschrittene funktionale Programmierung“

Monaden

Labor am Freitag, 18. November 2016, 12:15 Uhr

Aufgabe 1 - *Monad*-Instanzen

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

```
import Prelude hiding (Monad (..))
```

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

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

  ```
  data Identity a = Identity {runIdentity :: a}
  ```

  Wofür können Sie diese Instanz der Typklasse *Monad* nutzen?

optional Definieren Sie eine Instanz der Typklasse *Monad* für die partielle Typanwendung *Either e*.

optional Definieren Sie eine Instanz für $(,)\ m$ für den Fall, dass m eine Instanz der Typklasse *Monoid* ist.

optional Definieren Sie eine Instanz der Typklasse *Monad* für den Typ $(\rightarrow)\ r$.

Aufgabe 2 - Listen-Monade

Implementieren Sie eine Instanz der Typklasse *Monad* für Listen. Überlegen Sie sich, ob Ihre Implementierung den Gesetzen einer Monade genügt. Definieren Sie mit Hilfe dieser Instanz eine Funktion $cross :: [a] \rightarrow [b] \rightarrow [(a, b)]$, die das Kreuzprodukt von zwei Listen berechnet.

Aufgabe 3 - *State*-Monade

Wir betrachten den folgenden Baum-Datentyp.

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

Schreiben Sie eine Funktion $numberTree' :: Tree\ a \rightarrow Int \rightarrow (Tree\ Int, Int)$. Diese Funktion erhält einen Baum und einen Startindex und nummeriert die Einträge im Baum, startend mit dem Startindex, durch. Die zweite Komponente des Paares liefert nach einem Aufruf den nächsten Index, der verwendet werden könnte. Nutzen Sie zur Definition dieser Funktion einen **let**-Ausdruck, um auf die Komponenten des Paares im rekursiven Ergebnis

zuzugreifen. Definieren Sie mit Hilfe der Funktion *numberTree′* eine Funktion *numberTree* :: *Tree a* → *Tree Int*, die die Knoten eines Baumes mit 1 startend durchnummeriert.

Das Durchschleifen der aktuellen Nummer in der Funktion *numberTree′* ist recht umständlich und fehleranfällig. Alternativ kann man das Durchschleifen eines Zustandes mit Hilfe einer Monade modellieren. Definieren Sie den folgenden Datentyp.

```
data State s a = State { runState :: s → (a, s) }
```

Definieren Sie zuerst eine Instanz der Typklasse *Functor* für *State s*. Für die Definition ist ein **let**-Ausdruck sehr hilfreich.

Definieren Sie eine Instanz der Typklasse *Monad* für *State s*. Das heißt, die monadische Aktion führt die Veränderung des Zustandes aus und liefert einen Wert vom Typ *a* zurück. Auch hier ist ein **let**-Ausdruck sehr hilfreich.

In Haskell muss eine Instanz der Typklasse *Monad* auch eine Instanz der Typklasse *Applicative* sein. Verwenden Sie die folgende Instanz-Definition. Die Funktion *ap* stammt dabei aus dem Modul *Control.Monad*.

```
instance Applicative (State s) where
  pure = return
  (<*>) = ap
```

Definieren Sie außerdem die folgenden Funktionen.

- *evalState* :: *State s a* → *s* → *a*
- *get* :: *State s s*
- *put* :: *s* → *State s* ()
- *modify* :: (*s* → *s*) → *State s* ()

Implementieren Sie die Funktion *numberTreeS′* :: *Tree a* → *State Int* (*Tree Int*) mit Hilfe der *State*-Monade. Implementieren Sie nun eine Funktion *numberTreeS* :: *Tree a* → *Tree Int* mit Hilfe der Funktion *numberTreeS′*.

Hinweis: Bitte geben Sie die Lösungen der Aufgaben per Mail oder auf Papier bis zum 25.11. ab. Die Papierlösungen können im ersten Stock von Haus A in den Schrank in das entsprechende Fach geworfen werden.