Hochschule Flensburg
Fachbereich 3
Angewandte Informatik

Jan Christiansen

Hochschule Flensburg
University of Applied Sciences

6. Übung zur Vorlesung „Fortgeschrittene funktionale Programmierung“

Typklassen

Labor am Freitag, 4. November 2016, 12:15 Uhr

Aufgabe 1 - Typklasse Appendable

Implementieren Sie eine Typklasse mit dem Namen *Appendable*. Diese Typklasse soll für eine Instanz τ einen binären Operator $(<>) :: \tau \to \tau \to \tau$ und eine Konstante $empty :: \tau$ zur Verfügung stellen. Jede Instanz der Typklase *Appendable* sollte dabei die folgenden Gesetze erfüllen. Überprüfen Sie, wenn Sie eine Instanz definieren, ob die Gesetze erfüllt sind.

- Für alle $m :: \tau$ gilt $empty <> m = m = m <> empty$
- Für alle $m, n, o :: \tau$ gilt $m <> (n <> o) = (m <> n) <> o$

Implementieren Sie eine Instanz der Typklasse *Appendable* für den Typen $[a]$.

Schreiben Sie eine Funktion

$$foldMapList :: Appendable\ m \Rightarrow (a \to m) \to [a] \to m,$$

die jedes Element einer Liste mit Hilfe einer Funktion in Werte der Typklasse *Appendable* umwandelt und anschließend alle Werte zu einem einzigen zusammenfasst.

Definieren Sie mit Hilfe der Funktion *foldMapList* eine Funktion

$$mcombine :: Appendable\ m \Rightarrow [m] \to m.$$

Definieren Sie mit Hilfe der Funktion *mcombine* eine Funktion $concat :: [[a]] \to [a]$.

Aufgabe 2 - Appendable-Instanzen

Definieren Sie zwei Datentypen *Any* und *All*, die jeweils einen Konstruktor mit einem booleschen Wert zur Verfügung stellen. Machen Sie beide Datentypen zu einer Instanz der Typklasse *Appendable*. Dabei soll der Operator $<>$ auf dem Typ *Any* einem booleschen Oder entsprechen und der Operator $<>$ auf dem Typ *All* einem booleschen Und. Implementieren Sie mit Hilfe dieser Instanzen und der Funktion *foldMapList* die Funktionen

$$any :: (a \to Bool) \to [a] \to Bool$$

$$all :: (a \to Bool) \to [a] \to Bool$$

Wir betrachten die folgenden beiden Datentypen.

data *Sum a* = *Sum a*
data *Product a* = *Product a*

Machen Sie beide Datentypen zu einer Instanz der Typklasee *Appendable*. Dabei soll — wie der Name schon sagt — die Appendable-Struktur des Datentyp *Sum* einer Summe und die des Datentyp *Product* einem Produkt entsprechen. Überlegen Sie sich, wie sie die Argumente der Typkonstruktoren wählen können, um eine möglichst allgemeine Instanz zu definieren. Implementieren Sie die folgende beiden Funktionen.

$$sum :: [Int] \to Int$$

$$product :: [Int] \to Int$$

Können Sie die Typen dieser Funktionen verallgemeinern?

Definieren Sie eine Instanz der Typklasse *Appendable* für ein Paar von Werten, die beide eine Instanz der Typklasse *Appendable* sind. Definieren Sie eine Funktion $sumProduct :: Num\ a \Rightarrow [a] \to (a, a)$, die die Summe und das Produkt einer Liste in einem Durchlauf berechnet.

Aufgabe 3 - Appendable für Bäume

Wir betrachten den folgenden Datentypen für Bäume.

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

Implementieren Sie eine Funktion $foldMapTree :: Appendable\ m \Rightarrow (a \to m) \to Tree\ a \to m$. Implementieren Sie mit Hilfe der Funktion *foldMapTree* eine $toList :: Tree\ a \to [a]$.

Aufgabe 4 - Suchen mit Appendable (Optional)

Wir betrachten den folgenden Datentyp.

```
data First a = First {getFirst :: Maybe a}
```

Wir betrachten die folgende Funktionsdefinition.

```
findFirst :: (a → Bool) → [a] → Maybe a
findFirst p = getFirst ∘ foldMapList (First ∘ select)
where
  select x = if p x then Just x else Nothing
```

Geben Sie eine Instanz der Typklasse *Appendable* für den Datentyp *First* an, so dass, die Funktion *findFirst* das erste Element aus der Liste liefert, das das gegebene Prädikat erfüllt. Wie können Sie eine Funktion *findLast* definieren?

Hinweis: Bitte geben Sie die Lösungen der Aufgaben per Mail oder auf Papier bis zum 21.10. ab. Die Papierlösungen können im ersten Stock von Haus A in den Schrank in das entsprechende Fach geworfen werden.