Institut für Informatik **Ludwig-Maximilians-Universität München**
Prof. François Bry, Thomas Prokosch
Kilian Rückschloß

Programmierung und Modellierung, SoSe 21
Übungsblatt 9

Abgabe: bis Mo 21.06.2021 08:00 Uhr

Aufgabe 9-1 Monoide

Monoide werden in der Haskell-Dokumentation als "types with an associative binary operation that have an identity" beschrieben; um einen Datentyp als Monoid zu implementieren, müssen also zwei Voraussetzungen erfüllen sein: Erstens muss eine irgendwie-geartete assoziative Operation für zwei Argumente diesen Typs implementiert werden, zweitens muss zu dieser Operation ein neutrales Element bzw. Neutrum definiert sein. So ist die Operation `+` über Integer mit dem Neutrum 0 ein Monoide, die Operation `/` wegen nicht gegebener Assoziativität jedoch nicht.

Implementieren Sie die folgenden Beschreibungen von Datentypen als Monoid.

a) Implementieren Sie einen Datentyp `ComplexNumber`, der eine komplexe Zahl darstellt. Implementieren Sie dann die Addition zweier komplexer Zahlen als Monoid. Implementieren Sie zusätzlich für `ComplexNumber` die Typklasse `Show` welche die komplexe Zahl in der Form $a + bi$ (wobei a der Realteil und b der Imaginärteil der Zahl ist) aus.

b) Im RGB-Farbraum wird eine Farbe als Tripel `(Red, Green, Blue)` von drei ganzen Zahlen im Bereich von 0 bis 255 dargestellt. Jede Komponente des Tripels repräsentiert dabei den Anteil der jeweiligen Farbe an der Gesamtfarbe.

   Für das Mischen von Farben gibt es verschiedene Modelle. In dieser Aufgabe werden Farben additiv gemischt, d.h. die entsprechenden Farbwerte werden einfach addiert. Zu beachten gibt es hier nur, dass der Wert einer Farbe nicht über 255 steigen darf. **Implementieren Sie einen Datentyp für RGB-Werte und additive Farbmischung als Monoid.**

   Handelt es sich bei der subtraktiven Farbmischung (wie additive Farbmischung, nur mit Substraktion anstatt Addition) **auch um einen Monoiden?** Wenn ja, dann implementieren Sie diese Art der Farbmischung für Ihren Datentyp. Wenn nicht, begründen Sie *kurz*, wieso es sich hierbei um keinen Monoiden handelt.

Aufgabe 9-2 Monoide und ihre Eigenschaften

In dieser Aufgabe betrachten wir Zahlenraum, der nur die Zahlen 0 und 1 enthält und einen Operator $\oslash$. Der Operator ist wie folgt für die Werte aus dem Zahlenraum definiert:

- $0 \oslash 0 = 0$
- $0 \oslash 1 = 1$
- $1 \oslash 0 = 1$
- $1 \oslash 1 = 0$

a) Implementieren Sie den Operator $\oslash$ in Form einer Funktion `op :: Integer -> Integer -> Integer`.

b) Zeigen Sie, dass der gegebenen Zahlenraum bezüglich des Operators $\oslash$ die Bedingungen für einen Monoid erfüllt.

c) Implementieren Sie für den Datentyp `Integer` die Typklasse `Monoid`, so dass `Integer` ein Monoid bezüglich $\oslash$ ist.

Aufgabe 9-3 (applikative) Funktoren

a) Die Typklasse `Functor` verallgemeinert die Listen mit `map`. Die Funktion `fmap` eines Functors überträgt eine normale „Funktion“ auf Functor-Werte. Nutzen Sie dies aus, um einen eigenen Datentyp `List` zu implementieren, der keine oder beliebig viele Elemente enthalten kann.

   Hinweis: Eine ganz ähnliche Aufgabe gab es bereits auf einem der vergangenen Übungsblätter.

b) Realisieren Sie eine Funktion `scale :: (Functor f, Num b) => b -> f b -> f b`, die alle Elemente einer Liste vom Typ `List` mit einem Faktor vom Typ `Num` multipliziert. Implementieren Sie dazu zuvor die Typklasse `Functor` für Ihren Listentyp und benutzen Sie `fmap` für Ihre Lösung.

c) Die Funktion `<*>` eines Applicative Functors ermöglicht die Anwendung einer Functor-Funktion innerhalb des Functor. Implementieren Sie dazu die Typklasse `Applicative` für Ihren Listentyp und definieren Sie unter Benutzung der Funktion `<*>` die Funktion `lzipWith` für Ihren Listentyp analog zur Haskell Funktion `zipWith`.

Aufgabe 9-4 Funktoren und Applikative

a) Dreidimensionale Punkte werden häufig als Tripel (Vektoren) dargestellt. Implementieren Sie einen Datentyp `Triple` mit einer eigenen String Repräsentation durch die Typklasse `Show`.

b) Um auf die einzelnen Komponenten des Tripels zugreifen zu können, werden, analog zu den Standard Haskell Tupel Funktionen `fst` und `snd`, Äquivalente für den neuen Datentyp `Triple` benötigt; implementieren Sie diese.

c) Konvertierungsmethoden aus oder auf bestehende Datentypen sind sehr praktisch, um einen neuen Datentyp verwenden zu können. Implementieren Sie daher eine Funktion, die einen Tripel aus einer Liste und eine Funktion, die aus einer Liste ein Tripel erzeugt.

d) Als nächstes sollen Sie das Kreuzprodukt zweier Tripel implementieren. Im dreidimensionalen kartesischen Koordinatensystem lässt sich das Kreuzprodukt wie folgt berechnen:

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} \times \begin{pmatrix} d \\ e \\ f \end{pmatrix} = \begin{pmatrix} bf - ce \\ cd - af \\ ae - bd \end{pmatrix}$$

e) Implementieren Sie auch noch eine Funktion, um ein Tripel mit einem Skalar zu multiplizieren können. Definieren Sie hierfür den Funktor `Triple` (die Funktion `fmap` wird für die Implementierung nützlich sein; implementieren Sie dafür die Typklasse `Functor` für `Triple`). Die Skalarmultiplikation ist wie folgt definiert:

$$s \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} sa \\ sb \\ sd \end{pmatrix}$$

f) Nützlich ist auch noch das Skalarprodukt und die Vektor-Addition/-Subtraktion. Um diese zu realisieren, definieren Sie ein Applicative `Triple`. Implementieren Sie dann die Funktionen nach den folgenden Definitionen.

Implementieren Sie dafür zuerst die `Applicative`-Typklasse für `Triple` und machen Sie sich deren Funktionen zunutze.

$$\begin{pmatrix} a \\ b \\ c \end{pmatrix} * \begin{pmatrix} d \\ e \\ f \end{pmatrix} = ad + be + cf \qquad \begin{pmatrix} a \\ b \\ c \end{pmatrix} \pm \begin{pmatrix} d \\ e \\ f \end{pmatrix} = \begin{pmatrix} a \pm d \\ b \pm e \\ c \pm f \end{pmatrix}$$

g) Zum Schluss muss noch die Länge eines Tripels berechnet werden. Implementieren Sie diese wie folgt:

$$\left| \begin{pmatrix} a \\ b \\ c \end{pmatrix} \right| = \sqrt{a^2 + b^2 + c^2}$$