Funktionale Programmierung, WS 2010/11, Blatt 2

Dr. Jan Schwinghammer, Prof. Dr. Gert Smolka
www.ps.uni-sb.de/courses/proseminar-ws10/

Typklassen und Überladen von Operatoren

Aufgabe 2.1 (Typklassen) Lesen Sie Kapitel 5 und 8 in *A Gentle Introduction to Haskell* (www.haskell.org/tutorial/) von Hudak, Peterson und Fasel. Experimentieren Sie mit einigen Beispielen aus dem Text.

Aufgabe 2.2 (Eq) Betrachten Sie den durch *data Set a = Set* $[a]$ definierten Typ zur Repräsentation von Mengen über a durch Listen. Geben Sie eine Instanzdeklaration *instance Eq a* ⇒ *Eq* (*Set a*) *where*..., so dass *Set*$[3,4]$ == *Set*$[4,3]$ den Wert *True* liefert. (Beachten Sie, dass $[3,4] == [4,3]$ den Wert *False* liefert.)

Aufgabe 2.3 (Komplexe Zahlen) Sei der Typ *data Comp a = Comp a a* gegeben, den wir als Datentyp von komplexen Zahlen auffassen wollen. Genauer soll *Comp Float* die komplexen Zahlen $\mathbb{C}$ repräsentieren, *Comp Int* die komplexen ganzen Zahlen $\mathbb{Z}[i]$ usw.

a) Geben Sie eine Instanzdeklaration *instance Eq a* ⇒ *Eq* (*Comp a*) *where*..., die Gleichheit auf den Elementen von *Comp a* definiert.

b) Geben Sie eine Instanzdeklaration *instance Show a* ⇒ *Show* (*Comp a*) *where*..., so dass *show* (*Comp 1 1*) den String `1+i1` liefert.

c) Geben Sie eine Instanzdeklaration *instance Num a* ⇒ *Num* (*Comp a*) *where*..., die die arithmetischen Operationen auf *Comp a* definiert. Beschränken Sie sich dabei auf die Operationen

$$\begin{aligned} (+),(-),(*) &:: \mathit{Comp}\ a \to \mathit{Comp}\ a \to \mathit{Comp}\ a \\ \mathit{negate} &:: \mathit{Comp}\ a \to \mathit{Comp}\ a \\ \mathit{fromInteger} &:: \mathit{Integer} \to \mathit{Comp}\ a \end{aligned}$$

Beispielsweise soll $i * i == -1$ für $i = \mathit{Comp}\ 0\ 1$ den Wert *True* liefern.

Aufgabe 2.4 (Konstruktorklassen) Wie in Kapitel 5 beschrieben, sind die Instanzen der Typklasse *Functor* Typkonstruktoren, die eine "map" Funktion besitzen. (Insbesondere ist *List* mit *fmap* = *map* eine Instanz von *Functor*.)

a) Geben Sie eine Instanzdeklaration für den Typkonstruktor *Set* aus Aufgabe 2.2 an, so dass beispielsweise *fmap* $(+1)$ (*Set*$[1,2,3]$) == *Set*$[2,3,4]$ gilt.

b) Betrachten Sie den Typ *data Tree a = Node a [Tree a]* von endlich verzweigenden Bäumen. Geben Sie eine Deklaration *instance Functor Tree where* ..., indem Sie eine geeignete Funktion *fmap* angeben.

c) Analog zu *Functor* lässt sich die Typklasse *BiFunctor* für Typkonstruktoren *f* mit *zwei* Typargumenten definieren:

```
class BiFunctor f where
    bifmap :: (a → a') → (b → b') → f a b → f a' b'
```

Geben Sie Instanzdeklarationen für Produkttypen (a, b) und Summentypen *Either a b* an.[1]

[1] *Either* ist in der Prelude durch die Deklaration *data Either a b = Left a | Right b* gegeben.