Funktionale Programmierung, SS 2019, Blatt 2

Fabian Kunze, Kathrin Stark,
Prof. Dr. Gert Smolka
https://courses.ps.uni-saarland.de/fp_ss19/

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 (Gleichheit)

a) Deklarieren Sie Funktionen mit folgenden Typen, die Gleichheit berechnen, ohne auf Typklassen zurückzugreifen:

$$\begin{aligned} \mathit{eqBool} &:: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \\ \mathit{eqList} &:: (a \to a \to \mathit{Bool}) \to [a] \to [a] \to \mathit{Bool} \\ \mathit{eqPair} &:: (a \to a \to \mathit{Bool}) \to (b \to b \to \mathit{Bool}) \to (a,b) \to (a,b) \to \mathit{Bool} \end{aligned}$$

b) Deklarieren Sie zwei Konstanten *c1*, *c2* vom Typ $([\mathit{Bool}], [(\mathit{Bool}, \mathit{Bool})])$ und testen Sie diese mit denen von Ihnen geschriebenen Funktionen auf Gleichheit.

c) Nutzen Sie Typklassen, um *c1* und *c2* auf Gleichheit zu testen, und vergleichen Sie diese Variante mit ihrer Lösung für Teil (b).

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

a) Geben Sie eine Instanzdeklaration $\mathit{instance}\ \mathit{Eq}\ a \Rightarrow \mathit{Eq}\ (\mathit{Comp}\ a)\ \mathit{where} \ldots$, die Gleichheit auf den Elementen von $\mathit{Comp}\ a$ definiert.

b) Geben Sie eine Instanzdeklaration $\mathit{instance}\ \mathit{Show}\ a \Rightarrow \mathit{Show}\ (\mathit{Comp}\ a)\ \mathit{where} \ldots$, so dass $\mathit{show}\ (\mathit{Comp}\ 1\ 1)$ den String `1+1i` liefert.

c) Geben Sie eine Instanzdeklaration $\mathit{instance}\ \mathit{Num}\ a \Rightarrow \mathit{Num}\ (\mathit{Comp}\ a)\ \mathit{where} \ldots$, die die arithmetischen Operationen auf $\mathit{Comp}\ a$ 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 (Eq) Betrachten Sie den durch $data\ Set\ a = Set\ [a]$ definierten Typ zur Repräsentation von Mengen über a durch Listen.

a) Geben Sie eine Instanzdeklaration $instance\ Eq\ a \Rightarrow Eq\ (Set\ a)\ where \ldots$, so dass $Set[3,4] == Set[4,3]$ den Wert $True$ liefert. (Beachten Sie, dass $[3,4] == [4,3]$ den Wert $False$ liefert.)

b) Betrachten Sie den Ausdruck $Set[Set[3,2]] == Set[Set[2,3], Set[2,3,3]]$. Geben Sie die Instanzen der Typklasse Eq an, die inferiert werden müssen um diesen Ausdruck zu typisieren.

Aufgabe 2.5 (Bäume) Betrachten Sie den Typ $data\ Tree\ a = Node\ a\ [Tree\ a]$ von endlich verzweigenden Bäumen. Geben Sie eine Instanzdeklaration für die Typklasse Eq, so dass $(==)$ Bäume auf strukturelle Gleichheit testet.

Aufgabe 2.6 (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 2.3 an, so dass beispielsweise $fmap\ (+1)\ (Set[1,2,3]) == Set[2,3,4]$ gilt.

b) Geben Sie eine Instanzdeklaration für den Typkonstruktor *Tree* aus Aufgabe 2.4 an, indem Sie eine geeignete Funktion *fmap* angeben.

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

$$
\begin{aligned}
&class\ BiFunctor\ f\ where\\
&\quad bifmap\ ::(a \to a') \to (b \to b') \to f\ a\ b \to f\ a'\ b'
\end{aligned}
$$

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

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