Typklassen

Natascha Widder

19.11.2007

Motivation

Typklassen

- fassen Typen mit ähnlichen Operatoren zusammen
- ermöglichen überladenen Funktionen

Definition Typklassen

Deklarationsschema

```
class Name Platzhalter where
    Liste von Signaturen
```

Beispiel

```
class Mischmal m where
   mischen :: m -> m -> m
   ismischbar :: m -> m -> Bool
   analyse :: m -> [m]
   anzeigen :: m -> String
```

Verwendung Typklassen

Instanz

```
instance Klasse Datentyp where
  Implementierung für Signaturen
```

Verwendung Typklassen

Beispiel: Instanz Mischmal

```
data Farbe =  Gelb | Blau | Grün

instance Mischmal Farbe where
   mischen Blau Gelb = Grün
   mischen Gelb Blau = Grün
   ismischbar Blau Gelb = True
   ismischbar Gelb Blau = True
   ismischbar a b = False
   analyse Grün = [Gelb, Blau]
   analyse a = []
   anzeigen Gelb = "gelb"
   anzeigen Blau = "blau"
   anzeigen Grün = "grün"
```

Verwendung Typklassen

```
Main> analyse Grün
```

Verwendung Typklassen

```
Main> analyse Grün

[Gelb,Blau]
```

MischGetränke

```
data Getränke = Fanta | Cola | Bananensaft | Spezi | Bäh deriving (Eq, Show)

instance Mischmal Getränke where
   mischen Fanta Cola = Spezi
   mischen Cola Fanta = Spezi
   mischen a b = Bäh

   ismischbar a b | mischen a b == Bäh = False
                  | otherwise = True

   analyse Spezi = [Cola, Fanta]
   analyse Bäh = []
   analyse a = [a]
   anzeigen Spezi = "Cola-Fanta"
   anzeigen Cola = "Cola"
   anzeigen Fanta = "Fanta"
   anzeigen Bananensaft = "Bananensaft"
   anzeigen Bäh = "nicht zu empfehlen"
   anzeigen a = show a
```

Funktion Savemisch

Funktion Savemisch

```
savemisch:: Mischmal m => m -> m -> [m]
savemisch a b | ismischbar a b = [mischen a b]
              | otherwise = [a,b]
```

Eingabe/Ausgabe

```
Main> savemisch Cola Fanta
[Spezi]
Main> savemisch Blau Gelb
[Grün]
```

vordefinierte Klasse –Eq–

Eq definiert den Gleichheitsoperator

Eq

```
class Eq a where
    (==)        :: a -> a -> Bool
    (/=)        :: a -> a -> Bool
```

Default

Eq

```
class  Eq a  where
    (==), (/=)  ::  a -> a -> Bool

     x /= y  = not (x == y)
     x == y  = not (x /= y)
```

vordefinierte Klasse –Eq–

Instanz Eq Bool

```
instance Eq Bool where
  x == y = if x then y else not y
```

Instanz Eq Int

```
Instance Eq Int where
 (==)  :: eqInt
```

In beiden Fällen braucht man nur den Gleichheitsoperator zu defnieren, die Default-Implementierung für /= ergänzt die Deklaration.

vordefinierte Klasse -show-

- show wandelt einen Typen in einen String um

show Signatur

```
show :: Show a => a -> String
```

Instanz show

```
data Farbe =  Gelb | Blau | Grün

instance show Farbe where
   show a = anzeigen a
```

vordefinierte Klasse -show-

show angewendet auf Mischen

```
data Farbe = Gelb | Blau | Grün deriving(Show)
```

deriving leitet Funktionen automatisch ab

Main> anzeigen Blau

vordefinierte Klasse -show-

show angewendet auf Mischen

```
data Farbe = Gelb | Blau | Grün deriving(Show)
```

deriving leitet Funktionen automatisch ab

Main> anzeigen Blau
"blau"

vordefinierte Klasse -read-

Read Abfrage auf Mischmal

```
Main> anzeigen (read "Gelb" :: Farbe)
"gelb"
```

read ohne Typangabe

```
Main> anzeigen (read"Gelb")
ERROR - Unresolved overloading
*** Type       : (Read a, Mischmal a) => [Char]
*** Expression : anzeigen (read "Gelb")
```

Mehrdeutigkeit

Die Fehlermeldung ist auf Ambiguität(Mehrdeutigkeit) zurückzuführen. Die Semantik ist für read in Verbindung mit show nicht eindeutig.

- Show = Werte als Zeichenketten
- Read= Zeichenkette wird als Wert eines anderen Typs interpretiert
- read liest Zeichenkette s ein und s wird mit show als Zeichenkette dargestellt.
- Gesamtausdruck = String
- Teilergebnis= Typ a

Mehrdeutigkeit

Def. read, show

```
read :: Read a => String -> a
show :: Show a => a -> String
```

read liest Zeichenkette s ein und s wird mit show als Zeichenkette dargestellt.

show(read s)

```
show(read s) :: (Read a, Show a) => String
```

vordefinierte Klasse -num-

Signatur num

```
class Num a where
  (-x), (+), (-), (*) :: a -> a -> a
```

Beispiel an square

```
square :: Num a => a -> a
square x = x*x
```

Wörterbuch

- Zu jeder Instanz einer Typklasse gehört ein Wörterbuch, das die Implementierung der Methoden für die Instanz enthält und die passende Operation extrahiert.
- Wörterbücher werden vom Compiler verwendet.

Wörterbuch

Eq

```
class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool
```

Wörterbuch

EqDict

```
data EqDict a = EqDict (a -> a -> Bool) (a -> a -> Bool)

class Eq a where
 (==) :: a -> a -> Bool
 (/=) :: a -> a -> Bool

eq, neq           :: EqDict a -> a ->a -> Bool
eq (EqDict e _)    = e
neq (EqDict _ ne) = ne

eqDefault, neqDefault  :: EqDict a -> a -> a -> Bool
eqDefault eqDictA x y  = not(neq eqDictA x y)
neqDefault eqDictA x y = not(eq eqDictA x y)
```

Wörterbuch

Verwendung durch elem auf Eq

```
elem :: Eq a => a -> [a] -> Bool
elem x [] = False
elem x (y : ys) = x==y || (elem x ys)
```

Verwendung durch elem auf EqDict

```
elem :: EqDict a => a -> [a] -> Bool
elem dict x [] = False
elem dict x (y : ys) = (eqDict) x y || elem dict x ys
```

Ableitung

- Teilmengen einer Basisklasse werden dabei zu abgeleiteten Typklassen zusammengefasst.
- Teilmengen erweitern die Eigenschaften ihrer Basisklasse um neue.
- Operationen, die auf der Basisklasse definiert sind, sind auch auf allen abgeleiteten Klassen definiert

Ableitung

Ableitung Eq nach Ord

```
class (Eq a) => Ord a where
   (<), (<=), (>=), (>) :: a -> a -> Bool
   max, min             :: a -> a -> a

   x <= y               = not ( x > y)
   x < y                = x <= y && x/= y
   x >= y               = not (x < y)
   x > y                = x >= y && x /=y

   max x y | x <= y = y
           | otherwise = x
   min x y | x <= y = x
           | otherwise = y
```

Zusammenfassung

- Typklasen fassen Typen mit ähnlichen Operatoren zusammen
- Deklaration einer Typklasse : Schlüsselwort class, Namen der Typklasse, Platzhalter (z.B. a) für einen Typ, Schlüsselwort where,Liste von Signaturen.
- konkrete Verwendung von Klassen durch Instanzen (Schlüsselwort „instance“,Klasse, Datentyp, Schlüsselwort“ where“, Implementierung für Signaturen)
- Zu jeder Instanz einer Typklasse gehört ein Wörterbuch, das die Implementierung der Methoden für die Instanz enthält und die passende Operation extrahiert.

Zusammenfassung

- Wörterbücher werden vom Compiler verwendet.
- Abgeleitete Klasse erbt alle Eigenschaften ihrer Basisklasse
- Ambiguität liegt vor, wenn die Semantik nicht eindeutig ist