Funktionale Programmierung

Funktoren

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

Letzte Änderung: 03.12.2018 08:03

Inhaltsverzeichnis

Ein Funktor

- wir haben eine Funktion `f` von `a` nach `b`, aber das Argument `x` ist vom Typ "verpacktes `a`",
  - z.B. `x :: [a]` oder `x :: Maybe a`
- dann können wir mit Hilfe eines Funktors die Funktion `f` auf `x` anwenden und bekommen einen Wert vom Typ "verpacktes `b`" zurück
- Beispiel:

```
map :: (a -> b) -> [a] -> [b]
```

- die Typklasse

```haskell
class Functor (f :: * -> *) where
  fmap :: (a -> b) -> f a -> f b
  (<$) :: a -> f b -> f a
  (<$) = fmap . const
  {-# MINIMAL fmap #-}
```

Der Listen-Funktor

- Instanz

```haskell
instance Functor [] where
  fmap = map
```

- Beispiele

```
Prelude> map (+1) [1..5]
[2,3,4,5,6]
Prelude> fmap (+1) [1..5]
[2,3,4,5,6]
```

Der Maybe-Funktor

- Instanz

```haskell
instance Functor Maybe where
  fmap _ Nothing = Nothing
  fmap f (Just x) = Just $ f x
```

- Beispiele

```
Prelude> fmap show $ Just 17
Just "17"
Prelude> fmap show Nothing
Nothing
```

Auch als Operator

- `fmap` als Operator

```haskell
(<$>) = fmap
```

- Beispiele

```
Prelude> show <$> Just 17
Just "17"
Prelude> show <$> Nothing
Nothing
Prelude> (+1) <$> [1..5]
[2,3,4,5,6]
```

Funktor-Gesetze

- Identität

  ```
  fmap id === id
  ```

- Komposition

  ```
  fmap (f . g) === fmap f . fmap g
  ```

Weitere Funktor-Instanzen

- Tupel

  ```
  Prelude> fmap (+2) (1,1)
  (1,3)
  Prelude> fmap show (2,3)
  (2,"3")
  ```

- für Funktionen

  ```
  Prelude> fmap (+1) negate $ 10
  -9
  Prelude> (+1) . negate $ 10
  -9
  Prelude> fmap negate (+1) $ 10
  -11
  ```

Stacked Functors

- Beispiel

  ```
  Prelude> fmap (fmap (+1)) $ fmap Just [1..3]
  [Just 2,Just 3,Just 4]
  ```

Und auch noch `IO`

- Instanz

```
instance Functor IO where
  fmap f action = do
    x <- action
    return $ f x
```

- Beispiele

```
Prelude> read <$> getLine :: IO Int
12
Prelude> negate <$> (+1) <$> read <$> getLine
-13
Prelude> negate . (+1) . read <$> getLine
-13
```

Und für eigene Datentypen

- Beispiel

```
data Tree a = Leaf
            | Tree a (Tree a) (Tree a)
```

- Funktor-Instanz

```
instance Functor Tree where
  fmap _ Leaf = Leaf
  fmap f (Tree x left right) =
    Tree (f x) (fmap f left) (fmap f right)
```

- Beispiel

```
*Main> tree = Tree 12 (Tree 8 Leaf Leaf)
                      (Tree 14 Leaf Leaf)
*Main> fmap (<10) tree
Tree False (Tree True Leaf Leaf)
           (Tree False Leaf Leaf)
*Main> fmap show tree
Tree "12" (Tree "8" Leaf Leaf)
          (Tree "14" Leaf Leaf)
*Main> fmap (("->"++) . (++"!") . show) tree
Tree "->12!" (Tree "->8!" Leaf Leaf)
             (Tree "->14!" Leaf Leaf)
```