Applikative Funktoren

Funktionale Programmierung

Prof. Dr. Oliver Braun

Letzte Änderung: 09.10.2018 06:54

Ein Funktor

- Erinnerung:

```
class Functor (f :: * → *) where
  fmap :: (a → b) → f a → f b
```

- Problem: kann nicht mit Funktionen mit mehreren Argumenten verwendet werden

- Beispiel:

```
Prelude> :t fmap (+) (Just 1)
fmap (+) (Just 1) :: Num a ⇒ Maybe (a → a)
```

- also nicht **applikativ**

Ein applikativer Funktor

- applikativer Funktor

```
class Functor f ⇒ Applicative f where
  pure :: a → f a
  (<*>) :: f (a → b) → f a → f b
```

- Beispiel:

```
Prelude> pure (+) <*> Just 1 <*> Just 3
Just 4
```

- um das noch schöner schreiben zu können, ist definiert

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

- damit:

```
(+) <$> Just 1 <*> Just 3
```

Beispiel: **Maybe** als applikativer Funktor

```haskell
instance Applicative Maybe where
  pure = Just
  Nothing <*> _ = Nothing
  _ <*> Nothing = Nothing
  Just f <*> Just a = Just (f a)
```

Beispiel: Liste als applikativer Funktor

- als Funktor: Anwendung **einer** Funktion auf alle Elemente einer Liste

```
Prelude> fmap (2^) [1, 2, 3]
[2,4,8]
Prelude> fmap (^2) [1, 2, 3]
[1,4,9]
```

- als applikativer Funktor: Anwendung **aller** Funktionen auf **alle** Elemente einer Liste

```
Prelude> [(+1), (*2)] <*> [1..3]
[2,3,4,2,4,6]
Prelude> [(+),(*)] <*> [1..3] <*> [5,6]
[6,7,7,8,8,9,5,6,10,12,15,18]
Prelude> concat [[x+y,x*y] | x<-[1..3], y<-[5,6]]
[6,5,7,6,7,10,8,12,8,15,9,18]
```

Gesetze für applikative Funktoren

- Identität

```
pure id <*> v == v
```

- Komposition

```
pure (.) <*> u <*> v <*> w ==
    u <*> (v <*> w)
```

- Homomorphismus

```
pure f <*> pure x == pure (f x)
```

- (spezielle Art von) Kommutativität

```
u <*> pure y == pure ($ y) <*> u
```