Falten statt Schleifen

Funktionale Programmierung

Prof. Dr. Oliver Braun

Letzte Änderung: 09.10.2018 06:54

Folds

- Faltung (*folding*) ist ein Konzept um Listen (und auch andere faltbare Datentypen) zu einem Wert **zusammenzufalten**
- Folds sind *Catamorphismen*
    - Catamorphismen dekonstruieren Daten

Falten

```
foldr ::                (a → b → b) → b → [a] → b
```

seit GHC 7.10 verallgemeinert:

```
foldr :: Foldable t ⇒ (a → b → b) → b → t a → b
```

- Beispiel:

  ```
  foldr (+) 0 [1..3]
  ```

  wird zunächst (in mehreren Schritten) ersetzt durch

  ```
  1 + (2 + (3 + 0))
  ```

  und das wird dann von rechts zu einem Wert zusammengefaltet

Definition von **`foldr`**

- `foldr` ist (auf Listen beschränkt) wie folgt definiert:

```
foldr :: (a → b → b) → b → [a] → b
foldr f z [] = z
foldr f z (x:xs) = f x (foldr f z xs)
```

Definition von **`foldl`** — fold left

- `foldl` ist (auf Listen beschränkt) wie folgt definiert:

```
foldl :: (b → a → b) → b → [a] → b
foldl f acc [] = acc
foldl f acc (x:xs) = foldl f (f acc x) xs
```

- dieses Mal wird von links gefaltet:

```
foldl (+) 0 [1..3]
```

  wird zunächst schrittweise evaluiert zu

```
((0 + 1) + 2) + 3
```

- Beispiel

```
foldl (flip (:)) [] [1..3] - == reverse [1..3]
```

Zwischenergebnisse mit **scanr** und **scanl**

```
Prelude> foldr (+) 0 [1..5]
15
Prelude> scanr (+) 0 [1..5]
[15,14,12,9,5,0]

Prelude> foldl (+) 0 [1..5]
15
Prelude> scanl (+) 0 [1..5]
[0,1,3,6,10,15]
```

Beispiele (1/3)

- `Data.Map`
- qualifiziert importieren, da es gleichnamige Funktionen für die Map gibt, wie in der Prelude, z.B. `filter`

```haskell
import qualified Data.Map as M
```

- auch eine Map ist in Haskell unveränderbar

```haskell
myMap :: M.Map Integer String
myMap = M.empty

myMap' = M.insert 3 "Helga" myMap
myMap'' = M.insert 5 "Hugo" myMap'
```

Beispiele (2/3)

- viele Werte Einfügen z. B. mit einem `foldr`

```
foldr (uncurry M.insert) M.empty
  $ zip [1..] ["Maria", "Joseph", "Erni", "Bert"]
```

- anderes Beispiel: Liste aller Fiboncccci-Zahlen

```
fibs = 1 : scanl (+) 1 fibs
```

  - ein Zaubertrick der Lazy Evaluation

- n-te Fiboncccci-Zahl

```
fibsN x = fibs !! x
```

Beispiele (3/3)

- Maximum aller Elemente einer Liste

```
maximum' = foldr1 (\x y → max x y)
```