

Monoide

Funktionale Programmierung

Prof. Dr. Oliver Braun

Letzte Änderung: 03.12.2018 07:53

- gleichartige Strukturen sollten abstrahiert werden
- Beispiel: Collections
- in Haskell geht das darüber hinaus
- **und**: viele Strukturen sind mathematisch fundiert und haben Gesetze

- eine Halbgruppe ist ein Datentyp mit einer binären Operation
- Haskell Typklasse

```
class Semigroup a where  
  ( $\diamond$ ) :: a  $\rightarrow$  a  $\rightarrow$  a
```

- ein **Monoid** ist eine
 - Datentyp
 - binäre, assoziative Operation
 - mit einer Identität
- Haskell Typklasse

```
class Semigroup a  $\Rightarrow$  Monoid a where
```

```
empty :: a
```

```
mappend :: a  $\rightarrow$  a  $\rightarrow$  a
```

```
mappend = ( $\diamond$ )
```

```
mconcat :: [a]  $\rightarrow$  a
```

```
mconcat = foldr mappend empty
```

```
{-# MINIMAL empty #-}
```

- Operation: (`++`), Identität: `[]`
- Nutzung z.B.

```
Prelude> [1, 2, 3] <math>\diamond</math> [4, 5, 6]
```

```
[1,2,3,4,5,6]
```

```
Prelude> mconcat [[1..3], [4..6]]
```

```
[1,2,3,4,5,6]
```

```
Prelude> foldr mappend mempty [[1..3], [4..6]]
```

```
[1,2,3,4,5,6]
```

- für Listen umständlicher, aber allgemein gültig

- auch `Maybe a` ist Instanz von `Monoid`, wenn der Typ `a` darin “eingepackt” ist, ein `Monoid` ist
- Nutzung z.B.

```
Prelude> Just "Hallo" <#> Just " "
                <#> Just "Welt"

Just "Hallo Welt"
Prelude> mconcat [ Just "Hallo", Just " "
                  , Just "Welt", Just "!" ]

Just "Hallo Welt!"
Prelude> mconcat [ Just "Hallo", Nothing
                  , Just "Welt", Just "!" ]

Just "HalloWelt!"
```

Auch für eigene Datentypen

- Studenten

```
newtype Students = Students Int
```

- Semigroup-Instanz

```
instance Semigroup Students where  
  Students x  $\diamond$  Students y = Students $ x + y
```

- Monoid-Instanz

```
instance Monoid Students where  
  mempty = Students 0
```

```
*Prelude> (Students 12)  $\diamond$  (Students 23)  
Students 35
```

- Studentengruppen

```
data StudentGroups = StudentGroups
  { maxMembersPerGroup :: Int
  , groups :: [Int]
  }
```


- Studenten in Gruppen aufteilen

```
mkStudentGroups :: Int → Students → StudentGroups
mkStudentGroups size (Students i) =
  let first = i `mod` size
      rest  = replicate (i `div` size) size
  in StudentGroups size $
      if first /= 0 then first : rest else rest
```

- Beispiel

```
*Prelude> group1 = mkStudentGroups 20 (Students 77)
```

```
*Prelude> group1
```

```
StudentGroups { maxMembersPerGroup = 20  
                , groups = [17,20,20,20]}
```

```
*Prelude> group2 = mkStudentGroups 10 (Students 26)
```

```
*Prelude> group2
```

```
StudentGroups { maxMembersPerGroup = 10  
                , groups = [6,10,10]}
```

- Beispiel für Semigroup- und Monoid-Instanz

```
instance Semigroup StudentGroups where  
  (StudentGroups sizeX xs) <◇  
    (StudentGroups sizeY ys) =  
    mkStudentGroups (max sizeX sizeY)  
      (Students $ sum xs + sum ys)
```

```
instance Monoid StudentGroups where  
  mempty = StudentGroups 0 []
```

- Beispiel

```
*Prelude> g1 ◊ g2
```

```
StudentGroups { maxMembersPerGroup = 20  
                , groups = [3,20,20,20,20,20]}
```

```
*Prelude> mempty ◊
```

```
StudentGroups 22 [12, 13, 17, 8, 12]  
StudentGroups { maxMembersPerGroup = 22  
                , groups = [18,22,22]}
```

- liegt im Wesentlichen daran, dass die binäre Operation $+$ oder $*$ sein könnte
- es gibt aber

```
*Prelude> import Data.Monoid  
*Prelude Data.Monoid> :info Sum  
newtype Sum a = Sum {getSum :: a}
```

```
*Prelude Data.Monoid> :info Product  
newtype Product a = Product {getProduct :: a}
```

- Linksidentität

$$\text{mempty} \diamond x \equiv x$$

- Rechtsidentität

$$x \diamond \text{mempty} \equiv x$$

- Assoziativität

$$x \diamond (y \diamond z) \equiv (x \diamond y) \diamond z$$

- Tipp: Gesetze von QuickCheck testen lassen

Monoid-Instanz, aber kein Monoid! (1/2)

- zwar in der Typklasse `Monoid`, aber kein Monoid

```
instance Monoid StudentGroups where  
  mempty = StudentGroups 1 []  
  (StudentGroups sizeX xs) `mappend`  
    (StudentGroups sizeY ys) =  
    mkStudentGroup sizeX  
      (Students (sum xs + sum ys))
```

- es wird immer die Gruppengröße des ersten Operanden genommen

Monoid-Instanz, aber kein Monoid! (2/2)

```
*Prelude> g3
```

```
StudentGroups { maxMembersPerGroup = 12  
                , groups = [7,12]}
```

```
*Prelude> g3 <◇ mempty
```

```
StudentGroups { maxMembersPerGroup = 12  
                , groups = [7,12]}
```

```
*Prelude> mempty <◇ g3
```

```
StudentGroups { maxMembersPerGroup = 1  
                , groups = [0,1,1,1,1,1,1,1,1,1,  
                            ,1,1,1,1,1,1,1,1,1,1]}
```

keine Linksidentität