Monoid

Monoid

Definition:

A monoid is an algebraic structure with an associative binary operation that has an identity element.

Data.Monoid

```
class Monoid a where

    -- an identity element
    mempty  :: a

    - an associative operation
    mappend :: a -> a -> a
```

Rules of monoid

Rule 1: identity element

```
mempty `mappend` n   ==  n
n `mappend` mempty   ==  n
```

Rule 2: associative operation

```
(a `mappend` b) `mappend` c
                        == a `mappend` (b `mappend` c)
```

Examples of monoid

Plus (+)

Product (*)

Examples of monoide

binary operator : (++)

identity element : [ ]

[ ] ++ n == n

n ++ [ ] == n

associative : (a ++ b) ++ c

== a ++ (b ++ c)

Question

We already have individual functions like (++), why do we use mappend instead?

Monoid

```
newtype Sum a = Sum { getSum :: a }
  deriving (Eq, Ord, Read, Show, Bounded)

instance Num a => Monoid (Sum a) where
  mempty = Sum 0
  Sum x `mappend` Sum y = Sum (x + y)
```

Monoid

```
newtype Product a = Product { getProduct :: a }
  deriving (Eq, Ord, Read, Show, Bounded)


instance Num a => Monoid (Product a) where
  mempty = Product 1
  Product x `mappend` Product y = Product (x * y)
```

Foldable

- Type class : Data.Foldable
- abstract foldl and foldr from list
- applicable to arbitrary structures

Composition

- Tuples are already instances of monoids.

- Tuples of monoids

Extend monoid

define type class Aggregation:

```
class (Monoid a) => Aggregation a where
    type AggResult a :: *
    aggResult :: a -> AggResult a
```

Summary

- Definition
- Rules
- Usage

Questions?