Monoids, Functors, Applicative Functors, Monads

Kelsey McKenna

Monoid

Definition

```
class Monoid m where
  mempty  :: m
  mappend :: m -> m -> m

  mconcat :: [m] -> m
  mconcat ms = foldr mappend mempty
```

Rules

```
mempty <> x   = x
x <> mempty   = x
(x <> y) <> z = x <> (y <> z)
```

Functor

Definition

```
class Functor m where
  fmap :: (a -> b) -> m a -> m b
```

Rules

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

Applicative Functor

Definition

```
class ApplicativeFunctor m where
  pure :: a -> m a
  <*>  :: m (a -> b) -> m a -> m b
```

Rules

The only 'interesting' rule is:

```
fmap f x = pure f <*> x
```

Monad

Definition

```
class Monad m where
  return :: a -> m a
  (>>=) :: m a -> (a -> m b) -> m b

  (>>) :: m a -> m b -> m b
  m a >> m b = m a >>= \_ -> m b
```

Rules

```
return a >>= f = f a
m >>= return = m
(m >>= f) >>= g =
    m >>= (\x -> f x >>= g)
```