Higher-order abstractions

Stephan Boyer
@stepchowfun

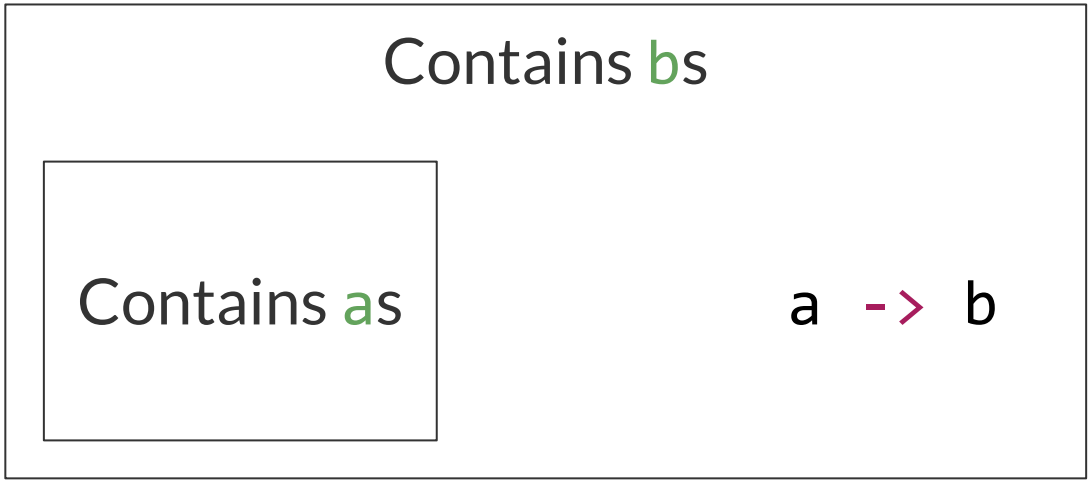
Review: Functor

Functor

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

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

Functor



Functor

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

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

```haskell
instance Functor [] where
  fmap = map
```

```haskell
instance Functor Maybe where
  fmap f (Just x) = Just (f x)
  fmap f Nothing = Nothing
```

```haskell
instance Functor ((->) r) where
  fmap = (.)
```

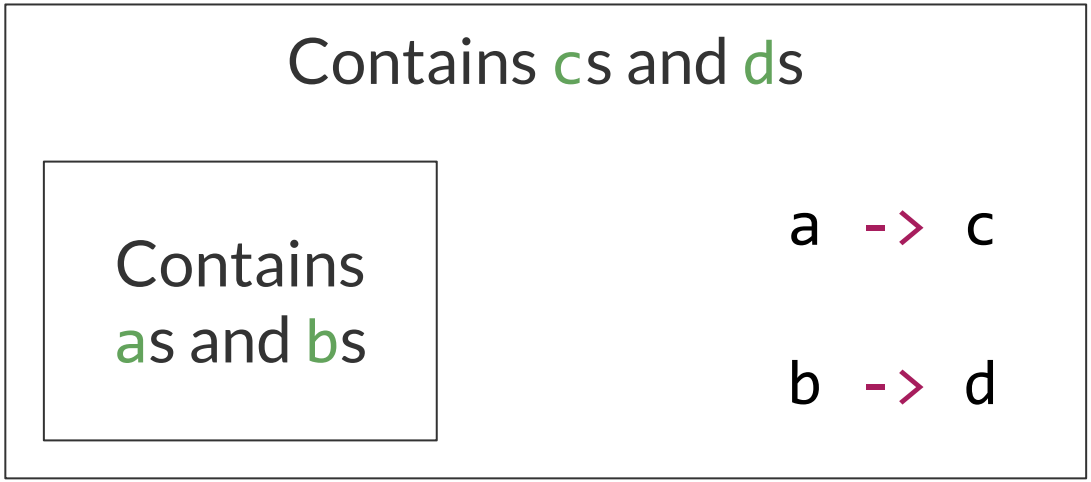
Bifunctor

Bifunctor

```haskell
class Bifunctor p where
  bimap :: (a -> b) -> (c -> d) -> p a c -> p b d
```

```haskell
bimap id id = id
bimap (f . g) (h . i) = bimap f h . bimap g i
```

Bifunctor



Bifunctor

```haskell
class Bifunctor p where
  bimap :: (a -> b) -> (c -> d) ->
           p a c -> p b d
```

```haskell
bimap id id = id
bimap (f . g) (h . i) =
  bimap f h . bimap g i
```

```haskell
instance Bifunctor (,) where
  bimap f g (x, y) = (f x, g y)
```

```haskell
instance Bifunctor Either where
  bimap f g (Left x) = Left (f x)
  bimap f g (Right y) = Right (g y)
```

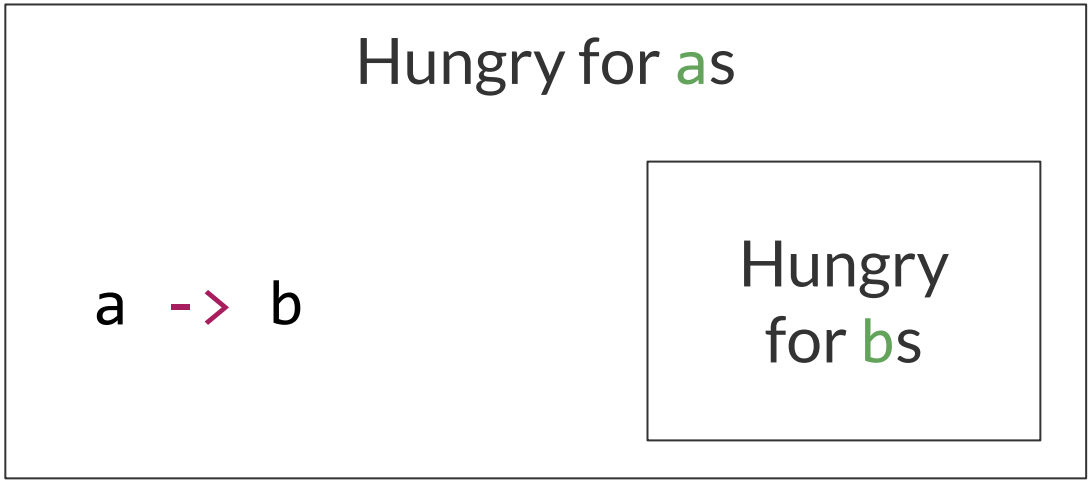
Contravariant

Contravariant

```haskell
class Contravariant f where
  contramap :: (a -> b) -> f b -> f a
```

```haskell
contramap id = id
contramap f . contramap g = contramap (g . f)
```

Contravariant



Contravariant

```
class Contravariant f where
  contramap :: (a -> b) -> f b -> f a
```

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

```
newtype Predicate a = Predicate {
  getPredicate :: a -> Bool
}
```

```
instance Contravariant Predicate where
  contramap f (Predicate p) =
    Predicate (p . f)
```

When is a type parameter contravariant?

---

1. `Int -> a`
2. `a -> Int`
3. `(Int -> a) -> Int`
4. `(a -> Int) -> Int`
5. `((a -> Int) -> Int) -> Int`
6. `a -> a -> Int`
7. `a -> (a -> Int) -> Int`
8. `data Phantom a = Phantom`

1. Covariant
2. Contravariant
3. Contravariant
4. Covariant
5. Contravariant
6. Contravariant
7. Invariant
8. Bivariant

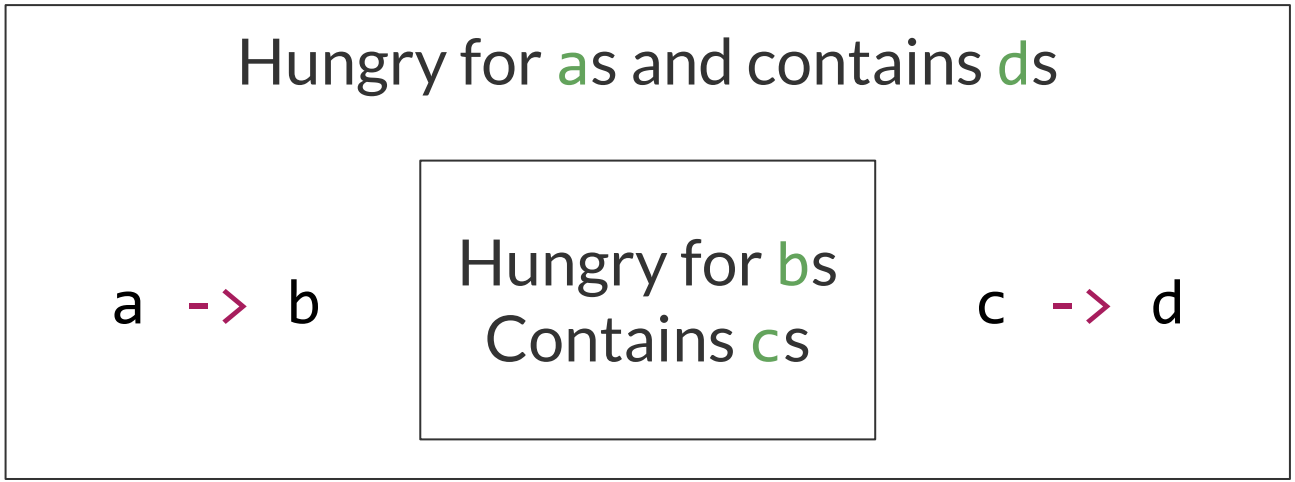
Profunctor

Profunctor

```haskell
class Profunctor p where
  dimap :: (a -> b) -> (c -> d) -> p b c -> p a d
```

```haskell
dimap id id = id
dimap (f . g) (h . i) = dimap g h . dimap f i
```

Profunctor



Profunctor

```haskell
class Profunctor p where
  dimap :: (a -> b) -> (c -> d) ->
           p b c -> p a d
```

```haskell
instance Profunctor (->) where
  dimap h f g = f . g . h
```

Also: Lens, Prism, ...

```haskell
dimap id id = id
dimap (f . g) (h . i) =
  dimap g h . dimap f i
```

Thanks!

@stepchowfun

References:

1. Edward Kmett: https://github.com/ekmett
2. The Extended Functor Family: https://www.youtube.com/watch?v=JZPXzJ5tp9w
3. Covariance and Contravariance: https://www.fpcomplete.com/blog/2016/11/covariance-contravariance
4. Functorality: https://bartoszmilewski.com/2015/02/03/functoriality/
5. Profunctors in Haskell: http://blog.sigfpe.com/2011/07/profunctors-in-haskell.html