Composición de functores

```
newtype Compose f g a = Compose {getCompose :: f (g a)}


-- Operador de composicion de functores
infixr :.:
type f :.: g = Compose f g


-- La composicion de functores es un functor
instance (Functor f, Functor g) => Functor (f :.: g) where
   fmap h (Compose x) = Compose (fmap (fmap h) x)
```

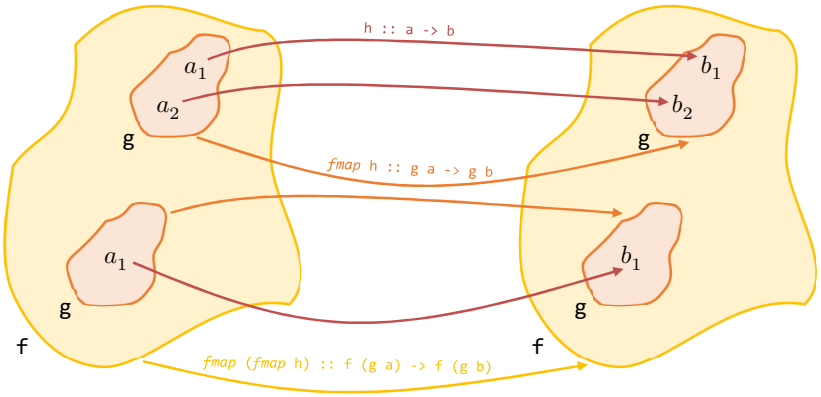
El tipo de `fmap` es

```
fmap :: Functor f => (a -> b) -> f a -> f b
```

con lo cual, el `fmap` más externo tiene tipo

```
fmap :: (g a -> g b) -> f (g a) -> f (g b)
```

```
fmap h (Compose x) = Compose (fmap (fmap h) x)
```




```
-- La composicion de functores aplicativos es un functor aplicativo
instance (Applicative f, Applicative g) => Applicative (f :.: g) where
   pure x                  = Compose (pure (pure x))
   Compose h <*> Compose x = Compose ((<*>) <$> h <*> x)
```

Recordar los tipos de `<*>` y de `pure`:

```
(<*>) :: Applicative f => f (a -> b) -> f a -> f b
pure  :: Applicative f => a -> f a
```

En este caso,

```
pure (<*>) :: (Applicative f, Applicative g) => f (g (a -> b) -> g a -> g b)
```

y

```
(<*>) :: Applicative g => g (a -> b) -> g a -> g b
```

Además, se tiene que `h <$> t = fmap h t ≡ pure h <*> t`. Entonces,

```
Compose h <*> Compose x = Compose (pure (<*>) <*> h <*> x)
```

La composición de dos mónadas puede no ser una mónada, pero si es un functor aplicativo.