Катаморфизмы

Катаморфизм — это обобщение свертки списков на любые алгебраические типы, если говорить понятным языком. Еще понятнее: хотим уметь делать те же операции, что и foldr/foldl, например суммировать все элементы, но не над списком, а над какой-нибудь ещё структурой, например деревом. По сути мы задаем две вещи:

```
1) phi :: f a -> a
2) Functor instance для нашей структуры
```

и потом утвержаем, что катаморфизм – это

```
cata :: Functor f => (f a -> a) -> Fix f -> a
cata phi (In x) = phi $ fmap (cata phi) x
```

По факту In – запаковка типа x в свой нерекурсивный аналог.

```
In :: f (Fix f) -> Fix f
```

Функция phi называется f-алгебра. А тип a, к которому применяли функцию, называем носителем. Пример катаморфизма и функции, где этот катаморфизм нужен, для нерекурсивного списка:

```
phiL :: L a [a] -> [a]
phiL Nil = []
phiL (Cons e es) = e : es
listify :: List a -> [a]
listify = cata phiL

GHCi> listify $ In Nil
[]
GHCi> listify $ In $ Cons 'i' $ In Nil
"i"
GHCi> listify $ In $ Cons 'h' $ In $ Cons 'i' $ In Nil
"hi"
```

Здесь просто реализвано добавление элемента в начало нерекурсивного спика. Можно посмотреть еще примеры для списка в презентации.

Конструктор In, который переводил тип в нерекурсивный, сам по себе является алгеброй. Такая алгебра называется инициальная алгебра, а её катаморфизм - id.

\+ код из дз на рекурсивные типы, где можно увидеть пример преобразования типа в нерекурсивный и катаморфизм для него

```
{-# LANGUAGE StandaloneDeriving, FlexibleContexts, UndecidableInstances #-}
newtype Fix f = In (f (Fix f))
```

```haskell
deriving instance Show (f (Fix f)) => Show (Fix f)
deriving instance Eq (f (Fix f)) => Eq (Fix f)

out :: Fix f -> f (Fix f)
out (In x) = x

type Algebra f a = f a -> a

cata :: Functor f => Algebra f a -> Fix f -> a
cata phi (In x) = phi $ fmap (cata phi) x


data B b = Empty | Zero b | One b deriving (Eq,Show)

type Bin = Fix B

instance Functor B where
  fmap _ Empty = Empty
  fmap f (Zero a) = Zero (f a)
  fmap f (One a) = One (f a)

phiB :: B Int -> Int
phiB Empty = 0
phiB (Zero x) = 2*x
phiB (One x) = 2*x + 1

bin2int :: Bin -> Int
bin2int = cata phiB
```

Здесь бинарное число задано нерекурсивно, задан катаморфизм, позволяющий перевести бинарное число в обычное.