Анаморфизмы и гилеморфизмы

Анаморфизмы

В билетах 20-21 было описано что такое алгебраические типы и катаморфизм, для этого билета полезно ознакомиться с двумя предыдущими.

Анаморфизм - вещь противоположная катаморфизму. На чём идейно был основан катаморфизм: тип запаковывается в нерекурсивную версию с помощью оператора неподвижной точки fix. Затем мы реализуем с помощью специально вводимой фунции phi для этого типа аналог свертки для списка. А что делает анаморфизм? Для нерекурсивного типа реализует аналог unfoldr для списка.

Соответственно, для этого вводится ряд обратных операций:

```
-- операция обратная In - запаковке. Пара In и out описывает
-- изоморфизм между типами Fix f и f (Fix f) (f-изоморфизм), т.е. типы
-- однозначно друг в друга переводятся

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

-- преобразование типа самого в себя
copy' :: Functor f => Fix f -> Fix f
copy' x = In $ fmap copy' $ out x
```

Соответственно, аналогично phi, для анаморфизма вводится psi, которая описывает действия в ходе развертки, и сам анаморфизм, который эту развертку осуществляет.

```
psi :: a -> f a

ana :: Functor f => (a -> f a) -> a -> Fix f
ana psi x = In $ fmap (ana psi) (psi x)

type Coalgebra f a = a -> f a
ana :: Functor f => Coalgebra f a -> a -> Fix f
```

Аналогично тому, что phi называлось f-алгеброй, psi называется f-коалгеброй, что логично, так как она является по сути дополнением psi.

Также, было введено понятие терминальной коалгебры.

```
psiOut :: Coalgebra f (Fix f)
psiOut = out
```

Собственно, она просто ничего не делает. Это изоморфизм из коалгебры в коалгебру.

Гилеморфизмы

Гилеморфизм – последовательное применение катаморфизма, а затем анаморфизма.

немного кода:

```haskell
-- определение
hylo :: Functor f => Algebra f a -> Coalgebra f b -> b -> a
hylo phi psi = cata phi . ana psi

-- жуткий код для фиббоначи через гилеморфизм

phiLProd :: Algebra (L Integer) Integer
phiLProd Nil = 1
phiLProd (Cons e es) = e * es

psiLToZero :: Coalgebra (L Integer) Integer
psiLToZero 0 = Nil
psiLToZero n = Cons n (n-1)

factorial :: Integer -> Integer
factorial = hylo phiLProd psiLToZero

-- а еще мы умеем выражать катаморфизм и анаморфизм через гилеморфизм

cata' :: Functor f => Algebra f a -> Fix f -> a
cata' phi = hylo phi out

ana' :: Functor f => Coalgebra f a -> a -> Fix f
ana' psi = hylo In psi
```

А зачем гилеморфизм нам нужен в принципе? Здесь лекции помогли мало, так что ссылаюсь вот сюда. С помощью катаморфизма мы можем сворачивать значение данного типа в значения любого другого типа, а с помощью анаморфизма мы можем разворачивать значения данного типа из значений любого другого типа. А вот уже гилеоморфизм нужен, чтобы их комбинировать и соответственно реализовать различного рода рекурсивные функции.

Например:

```haskell
-- Вычисление суммы от 0 до числа n
sumInt :: Int -> Int
sumInt = hylo range sum

sum x = case x of
    Nil      -> 0
    Cons a b -> a + b
```

```haskell
range n
    | n == 0    = Nil
    | otherwise = Cons n (n-1)

-- факториал
fact :: Int -> Int
fact = hylo range prod

prod x = case x of
    Nil      -> 1
    Cons a b -> a * b
```