Курс: Функциональное программирование
Практика 13. Рекурсивные типы

Оператор неподвижной точки для типов

Оператор неподвижной точки для типов

```
data Mu f = In (f (Mu f))
```

```
> :k Mu
Mu :: (* -> *) -> *
> :t In
In :: f (Mu f) -> Mu f
```

Для описания рекурсивного типа, эквивалентного `data Nat = Z | S Nat` задаём нерекурсивный тип

```
data N x = Z | S x
```

```
instance Functor N where
  fmap g  Z    = Z
  fmap g (S x) = S (g x)
```

и вводим рекурсивный через неподвижную точку функтора на уровне типов

```
type Nat = Mu N
```

Катаморфизм и анаморфизм

Катоморфизм

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

Функцию `phi :: f a -> a` называют f-алгеброй.

```
phiN :: N Int -> Int
phiN Z     = 0
phiN (S n) = succ n

natToInt :: Nat -> Int
natToInt = cata phiN
```

Анаморфизм

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

Функцию `psi :: a -> f a` называют f-коалгеброй.

```
psiN :: Int -> N Int
psiN 0 = Z
psiN n = S (n-1)

intToNat :: Int -> Nat
intToNat = ana psiN
```

Тип для двоичных чисел

Рассмотрим рекурсивный тип данных, представляющих числа в двоичной форме:

```
data Bin = Empty | Zero Bin | One Bin
```

При таком определении двоичные числа читаются справа налево

```
              Empty   == 0
           One Empty   == 1
      Zero (One Empty)  == 2
      One  (One Empty)  == 3
Zero (Zero (One Empty)) == 4
```

▶ Разработайте нерекурсивный функтор `B`, описывающий структуру этого типа, и определите `Bin` как неподвижную точку этого функтора.

▶ Определите B-алгебру `phiB :: B Int -> Int`, позволяющую задать катаморфизм

```
bin2int :: Bin -> Int        -- синоним  Mu B -> Int
bin2int = cata phiB

> let four  = In $ Zero $ In $ Zero $ In $ One $ In Empty
> bin2int four
4
```

▶ Определите B-коалгебру `psiB :: Int -> B Int`, позволяющую задать анаморфизм

```
int2bin :: Int -> Bin
int2bin = ana psiB

> int2bin 31
In (One (In (One (In (One (In (One (In (One (In Empty))))))))))
```

Тип для вычислителя выражений

Рассмотрим рекурсивный тип данных, представляющих арифметическое выражение:

```
Expr = Num Int
     | Add Expr Expr
     | Mult Expr Expr
```

▶ Разработайте нерекурсивный функтор `E`, описывающий структуру этого типа, и определите `Expr` как неподвижную точку этого функтора.

Для тестов ниже используются следующие выражения

```
en = In . Num
e3     = en 3
ep35   = In (Add e3 (en 5))
emp357 = In (Mult ep35 (en 7))
em7p35 = In (Mult (en 7) ep35)
```

▶ Определите E-алгебру `phiE :: E Int -> Int`, позволяющую задать катаморфизм, вычисляющий выражение

```
eval :: Expr -> Int
eval = cata phiE

> eval ep35
8
> eval emp357
56
```

▶ Определите E-алгебру `phiEShow :: E String -> String`, позволяющую задать катаморфизм, конструирующий строковое представление выражения

```
> cata phiEShow e3
"3"
> cata phiEShow ep35
"(3+5)"
> cata phiEShow emp357
"((3+5)*7)"
```

▶ Определите E-алгебру `phiEShow' :: E String -> String`, позволяющую задать катаморфизм, конструирующий строковое представление выражения в префиксной бесскобочной записи (польской нотации)

```
> cata phiEShow' ep35
"+ 3 5"
> cata phiEShow' emp357
"* + 3 5 7"
> cata phiEShow' em7p35
"* 7 + 3 5"
```

▶ Реализуйте вычислитель на стеке, используя следующие сервисные функции над стеком, реализованным через список:

```
push a as = a : as
add  (a : b : cs) = (b + a) : cs
mult (a : b : cs) = (b * a) : cs
```

Для этого определите E-алгебру `phiE' :: E ([Int] -> [Int]) -> [Int] -> [Int]`, позволяющую задать катаморфизм, осуществляющий вычисление

```
eval' :: Expr -> [Int] -> [Int]
eval' = cata phiE'

> eval' emp357 []
[56]
```