

Декларативное программирование

Семинар №10, группа 22215

Завьялов А.А.

7 ноября 2022 г.

Кафедра систем информатики ФИТ НГУ

Можно ли перечислить пару перечислимых значений?

```
instance (Enum a, Enum b) => Enum (a, b) where
    toEnum n = (toEnum n, toEnum n)
    fromEnum (a, b) = ???
```

Ожидаемое поведение

```
GHCI> [(1,2)..(3,4)]
[(1,2),(2,3),(3,4)]
```

Монады. Введение

Знакомые нам виды вычислений

- Применение функции к аргументу (аргументам):
 - $f :: a \rightarrow b, x :: a \Rightarrow f \$ x :: b$
 - $g :: a \rightarrow b \rightarrow c, x :: a, y :: b \Rightarrow g \$ x y :: c$
- Применение функции к аргументу в контексте:
 - $f :: a \rightarrow b, x :: \boxed{a} \Rightarrow f <\$> x :: \boxed{b}$
 - $f :: a \rightarrow b \Rightarrow \text{fmap } f :: \boxed{a} \rightarrow \boxed{b}$
- Применение функции *в контексте* к аргументу в контексте:
 - $g :: \boxed{a \rightarrow b}, x :: \boxed{a} \Rightarrow g <*> x :: \boxed{b}$
- Применение бинарной операции к операндам в контексте:
 $h :: a \rightarrow b \rightarrow c, x :: \boxed{a}, y :: \boxed{b} \Rightarrow \text{liftA2 } h \ x \ y :: \boxed{c}$

Незнакомые нам виды вычислений

- $k :: a \rightarrow \boxed{b}, x :: \boxed{a} \Rightarrow k ??? x :: \boxed{b}$

Пример

```
safeHead :: [a] -> Maybe a
```

```
safeHead [] = Nothing
```

```
safeHead (x:_) = Just x
```

```
safeTail :: [a] -> Maybe [a]
```

```
safeTail [] = Nothing
```

```
safeTail (_,xs) = Just xs
```

```
safeSecond :: [a] -> Maybe [a]
```

```
safeSecond = ??? safeTail ??? safeHead
```

Класс типов Monad

```
class Applicative m => Monad m where
  {-# MINIMAL (>>=) #-}
  (>>=)      :: m a -> (a -> m b) -> m b
  (>>)       :: m a -> m b -> m b
  return     :: a -> m a
infixl 1 >>, >>=
```

```
>safeSecond xs = safeTail xs >>= safeHead
```

```
>safeSecond [1] = Nothing
```

```
>safeSecond [1,2] = Just 2
```

- Функции типа `a -> m a` называются *стрелками Клейсли*
- `(>>=)` (читается "bind") называется *оператором монадического связывания*

Функция `return :: a -> m a`

- Определяет тривиальную стрелку Клейсли
- `return = pure`

```
toKleisli :: Monad m => (a -> b) -> (a -> m b)
toKleisli f = \x -> return (f x)
```

```
>:t toKleisli (*2)
```

```
toKleisli (*2) :: (Monad m, Num a) => a -> m a
```

```
>(toKleisli (*2) 2) :: Maybe Int
```

```
Just 4
```

```
>(toKleisli (*2) 2) :: [Int]
```

```
[4]
```

```
>return 42 :: [Int]
```

```
[42]
```

Оператор (\gg) :: $m\ a \rightarrow m\ b \rightarrow m\ b$

- Определён по умолчанию:

$m \gg k = m \gg= \backslash_ \rightarrow k$

- Если m породило значение, отбрасывает m , возвращает k

Пример

```
>Just 1 >> Just 2
```

```
Just 2
```

```
>Nothing >> Just 2
```

```
Nothing
```

```
>[1] >> [2]
```

```
[2]
```

```
>[] >> [2]
```

```
[]
```


Другие полезные функции для работы с монадами

- `(>=>) :: Monad m => (a -> m b) -> (b -> m c) -> a -> m c`, композиция стрелок Клейсли
- `(<=<) :: Monad m => (b -> m c) -> (a -> m b) -> a -> m c`
- `(<=<) :: Monad m => (a -> m b) -> m a -> m b`
`(<=<) = flip (>=>)`

Пример

```
safeSecond = safeTail >=> safeHead
```

Для любого представителя Monad должны выполняться следующие свойства:

- $\text{return } a \gg= k = k \ a$

- $m \gg= \text{return} = m$

- **Ассоциативность**

$$m \gg= (\backslash x \rightarrow k \ x \gg= h) = (m \gg= k) \gg= h$$

¹англ. – monadic laws

Композиция Клейсли

```
infixr 1 >=>
(>=>) :: Monad m => (a -> m b) -> (b -> m c) -> (a -> m c)
f >=> g = \x -> f x >>= g
```

Выразим свойства через композицию стрелок:

- `return >=> h = h`
- `f >=> return = f`
- **Ассоциативность**
`(f >=> g) >=> h = f >= (g >= h)`

do-нотация

Синтаксический сахар для упрощенной записи цепочек
монадических вычислений

do-нотация

```
do { e1; e2 }
```

Обычный синтаксис

```
e1 >> e2
```

do-нотация

```
do { p <- e1; e2 }
```

Обычный синтаксис

```
e1 >>= \p -> e2
```

do-нотация

```
do { let v = e1; e2 }
```

Обычный синтаксис

```
let v = e1 in do e2
```

Пример использования do-нотации

```
safeSecond xs = do  
  x <- safeTail xs  
  y <- safeHead x  
  return y
```

≡

```
safeSecond xs =  
  safeTail xs >>= (\x ->  
    safeHead x >>= (\y ->  
      return y))
```

- Реализовать функции `fmap`, `pure`, `(<*>)` с помощью `return`, `(>>=)`

Q&A
