CS314. Функциональное программирование

Лекция 8. Монады (продолжение)

В. Н. Брагилевский

Направление «Фундаментальная информатика и информационные технологии»
Институт математики, механики и компьютерных наук имени И. И. Воровича
Южный федеральный университет

13 октября 2017 г.

Классы типов Functor, Applicative и Monad

```haskell
class Functor (f :: * -> *) where
  fmap :: (a -> b) -> f a -> f b

class Functor f => Applicative (f :: * -> *) where
  pure :: a -> f a
  (<*>) :: f (a -> b) -> f a -> f b

class Applicative m => Monad (m :: * -> *) where
  (>>=)  :: m a -> (a -> m b) -> m b
  (>>)   :: m a -> m b -> m b
  return :: a -> m a
```

Содержание

Список как монада

```haskell
instance Monad [] where
    -- return :: a -> [a]
    return x = [x]
    -- (>>=) :: [a] -> (a -> [b]) -> [b]
    xs >>= f = concat (map f xs)
```

- Недетерминированные вычисления — функция (возвращающая список) применяется к каждому элементу исходного списка.

```
ghci> [3,4,5] >>= \x -> [x,-x]
[3,-3,4,-4,5,-5]
ghci> [1,2] >>= \n -> ['a','b'] >>= \ch -> return (n,ch)
[(1,'a'),(1,'b'),(2,'a'),(2,'b')]
```

Три нотации для списков

```
ghci> [1,2] >>= \n -> ['a','b'] >>= \ch -> return (n,ch)
[(1,'a'),(1,'b'),(2,'a'),(2,'b')]
```

```
listOfTuples :: [(Int,Char)]
listOfTuples = do
    n <- [1,2]
    ch <- ['a','b']
    return (n,ch)
```

```
ghci> listOfTuples
[(1,'a'),(1,'b'),(2,'a'),(2,'b')]
```

```
ghci> [(n,ch) | n <- [1,2], ch <- ['a','b']]
[(1,'a'),(1,'b'),(2,'a'),(2,'b')]
```

Содержание

Законы монад

```
return a >>= k  =  k a
```

```
m >>= return    =  m
```

```
m >>= (\x -> k x >>= h)  =  (m >>= k) >>= h
```

Содержание

Монады и моноиды

Классы Alternative и MonadPlus

```
class Applicative f => Alternative f where
  empty :: f a
  (<|>) :: f a -> f a -> f a

class Alternative m, Monad m => MonadPlus m where
  mzero :: m a
  mplus :: m a -> m a -> m a
```

Функция guard (Control.Monad)

```
guard :: Alternative f => Bool -> f ()
guard True  = pure ()
guard False = empty
```

Чем guard отличается от when?

```
guard :: Alternative f => Bool -> f ()
guard True  = pure ()
guard False = empty
```

```
when :: Applicative f => Bool -> f () -> f ()
when True  s  = s
when False s  = pure ()
```

Примеры использования функции guard

```
ghci> guard True :: Maybe ()
Just ()
ghci> guard False :: Maybe ()
Nothing
ghci> guard True :: [()]
[()]
ghci> guard False :: [()]
[]
ghci> guard True >> return "good" :: [String]
["good"]
ghci> guard False >> return "good" :: [String]
[]
```

```
ghci> [1..50] >>= (\x -> guard ('7' `elem` show x) >> return x)
[7,17,27,37,47]
```

Реализация Alternative для Maybe и списков

```
instance Alternative Maybe where
  empty = Nothing

  Nothing <|> r = r
  Just x  <|> _ = Just x
```

```
instance Alternative [] where
  empty = []
  (<|>) = (++)
```

Три нотации для списков: guard

```
ghci> [1..50] >>= (\x -> guard ('7' `elem` show x) >> return x)
[7,17,27,37,47]
```

```
withSevensOnly :: [Int]
withSevensOnly = do
    x <- [1..50]
    guard ('7' `elem` show x)
    return x
```

```
ghci> withSevensOnly
[7,17,27,37,47]
```

```
ghci> [x | x <- [1..50], '7' `elem` show x]
[7,17,27,37,47]
```

Содержание

Задача о перемещении коня

Условие

Определить, может ли конь перейти из одной позиции в другую за заданное число ходов?

```
type KnightPos = (Int, Int)

moveKnight :: KnightPos -> [KnightPos]
moveKnight (c,r) = do
    (c',r') <- [(c+2,r-1),(c+2,r+1),(c-2,r-1),(c-2,r+1)
               ,(c+1,r-2),(c+1,r+2),(c-1,r-2),(c-1,r+2)]
    guard (c' `elem` [1..8] && r' `elem` [1..8])
    return (c',r')
```

Пример вычисления moveKnight

```
moveKnight (c,r) = do
    (c',r') <- [(c+2,r-1),(c+2,r+1),(c-2,r-1),(c-2,r+1)
               ,(c+1,r-2),(c+1,r+2),(c-1,r-2),(c-1,r+2)]
    guard (c' `elem` [1..8] && r' `elem` [1..8])
    return (c',r')
```

```
ghci> moveKnight (1,1)
```

```
[(3,0), (3,2),  (-1,0),  (-1,2),  (2,-1),  (2,3),  (0,-1)...
```

```
F       T       F        F        F        T       F...
[]      [()]    []       []       []       [()]    []...
```

```
[(3,0)] [(3,2)] [(-1,0)] [(-1,2)] [(2,-1)] [(2,3)] [(0,-1)]...
```

```
[]      [(3,2)] []       []       []       [(2,3)] []...
```

Упрощённая задача

Определить, может ли конь перейти из одной позиции в другую за три хода?

```
in3 start = do
   first <- moveKnight start
   second <- moveKnight first
   moveKnight second
```

```
in3 start = moveKnight start >>= moveKnight >>= moveKnight
```

- Здесь гнездо из трёх циклов!

Решение

```
canReachIn3 :: KnightPos -> KnightPos -> Bool
canReachIn3 start end = end `elem` in3 start
```

Решение задачи о коне

```
inMany :: Int -> KnightPos -> [KnightPos]
inMany n st = foldr (<=<) return (replicate n moveKnight) st

canReachIn :: Int -> KnightPos -> KnightPos -> Bool
canReachIn n start end = end `elem` inMany n start
```

- Здесь гнездо из n циклов!

Как это понять???

```
foldr :: Foldable t => (a -> b -> b) -> b -> t a -> b
(<=<) :: Monad m => (b -> m c) -> (a -> m b) -> a -> m c
return :: a -> m a
replicate :: Int -> a -> [a]
moveKnight :: KnightPos -> [KnightPos]
```