Монады для Барабанщиков

Антон Холомьёв
(github@anton-k)

Почему так трудно понять монады ?

- Необычное понятие
- Синтаксис языка ещё не ясен, но монады понять нужно
- Неудобное для новичка определение монады
- Из определения неясно, зачем нужны эти ваши монады

Зачем нужны монады в Haskell ?

```
def getTwo():
  a = getChar()
  b = getChar()
  return (a, b)
```

```
def getOne():
  a = getChar()
  return (a, a)
```

Как Haskell видит этот код ?

```
def getTwo():
   return
      ( getChar()
      , getChar() )
```

```
def getOne():
   return
      ( getChar()
      , getChar() )
```

Порядок вычисления Haskell

```
                          Py  Hs
def getTwo():
  a = getChar()           1   2
  b = getChar()           2   3
  return (a, b)           3   1
```

```
                          Py  Hs
def getOne():
  a = getChar()           1   2
  return (a, a)           2   1
```

Пример противоречий

```
                                        Py  Hs
def dialog():
    println('Как вас зовут?')          1
    name = getString()                  2
    println('Привет, ' + name)          3
    return ()                           4    1
```

Все функции чистые

```
def getTwo():
  a = getChar()
  b = getChar()
  return (a, b)
```

```
def getOne():
  a = getChar()
  return (a, a)
```

Почему это не работает в Haskell?

- Все функции **чистые**
  - функция строит результат только из того, что ей передали на входе
  - Результат функции полностью определяется аргументами
- Порядок вычисления следует из функциональных зависимостей (**ленивое вычисление**)

Ключевая идея !!!

Монады нужны для задания

порядка вычислений

Если у нас есть оператор порядка

```
(>>=) :: Expr -> Expr -> Expr

    ( a = getChar()  )
>>= ( return (a, a)  )
```

Если у нас есть оператор порядка

```
(>>=) :: ???

    (getChar()                )
>>= (\a -> return (a, a) )
```

IO = Значение + Контекст вычислений

```
(>>=) :: IO a -> (a -> IO b) -> IO b
```

```
    ( getChar()              )
>>= ( \a -> return (a, a) )
```

IO = Значение + Контекст вычислений

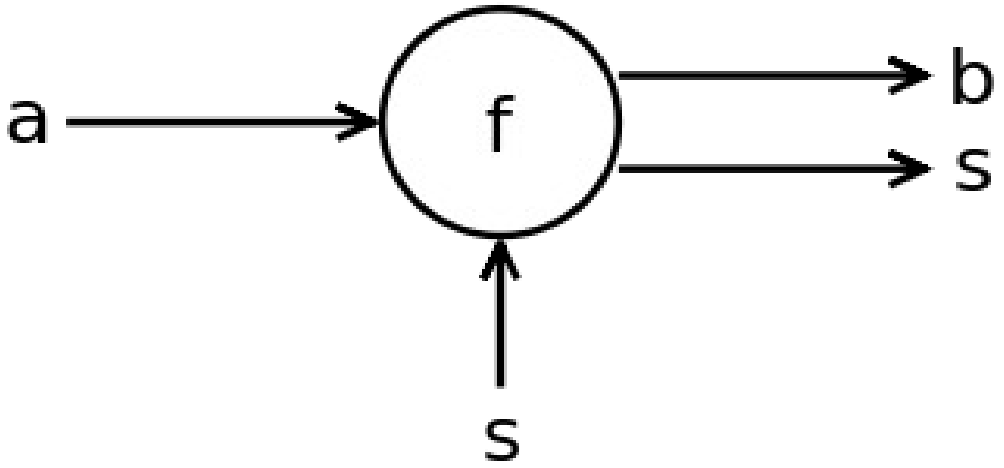
```
(>>=)  :: IO a -> (a -> IO b) -> IO b
return :: a -> IO a
```

```
    ( getChar()                )
>>= ( \a -> return (a, a) )
```

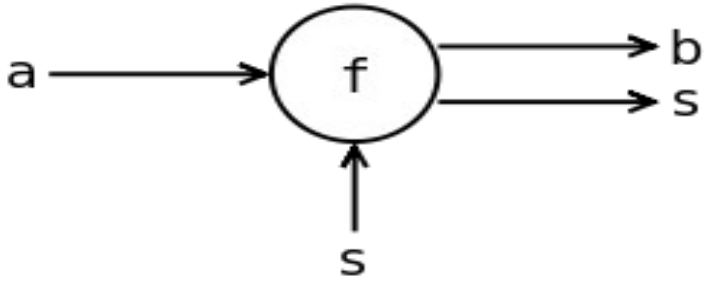
IO = Значение + Контекст вычислений

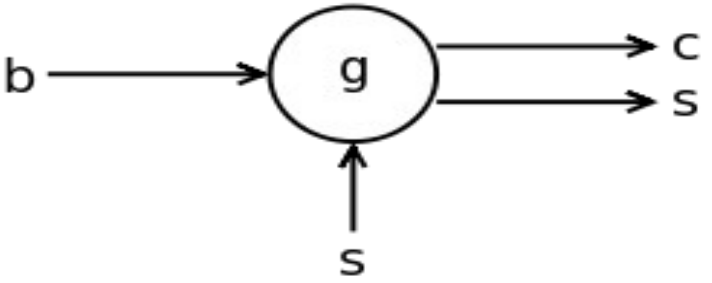
```
data IO a = IO (RealWorld -> (a, RealWorld))
```

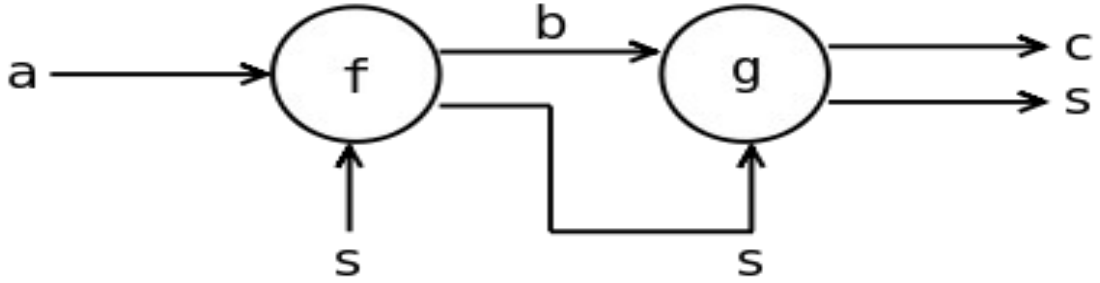
```
f :: a -> IO b
```

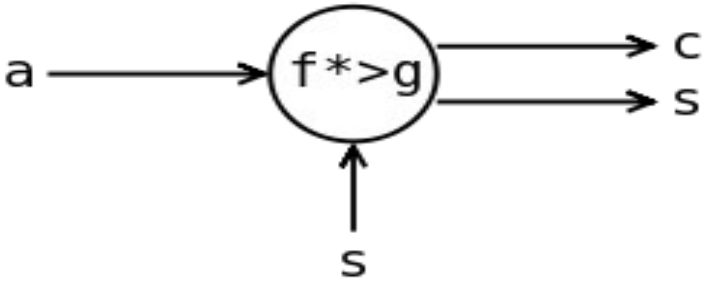


IO = Значение + Контекст вычислений









Пример с двумя запросами к пользователю

```
(>>=)  :: IO a -> (a -> IO b) -> IO b
return :: a -> IO a
```

```
    ( getChar()                              )
>>= ( \a ->      ( getChar()                 )
           >>= (\b -> return (a, b))  )
```

Зачем нужны монады в Haskell ?

```
getTwo () =
      getChar()
>>= (\a -> getChar()
>>= (\b -> return (a,b)))
```

```
getOne () =
      getChar()
>>= (\a -> return (a, a))
```

Do-нотация

```
      getChar ()
>>= (\a -> expr(a))
```

```
do
   a <- getChar()
   expr(a)
```

Do-нотация

```
getTwo = do
  a <- getChar
  b <- getChar
  return (a, b)
```

```
getOne = do
  a <- getChar
  return (a, a)
```

Пример противоречий

```
dialog = do
    print "Как вас зовут?"
    name <- getString
    print ("Привет, " ++ name)
    return ()
```

Do-нотация без аргумента

```
      print "Привет!"
>>= (\_ -> expr())
```

```
do
  print "Привет!"
  expr()
```

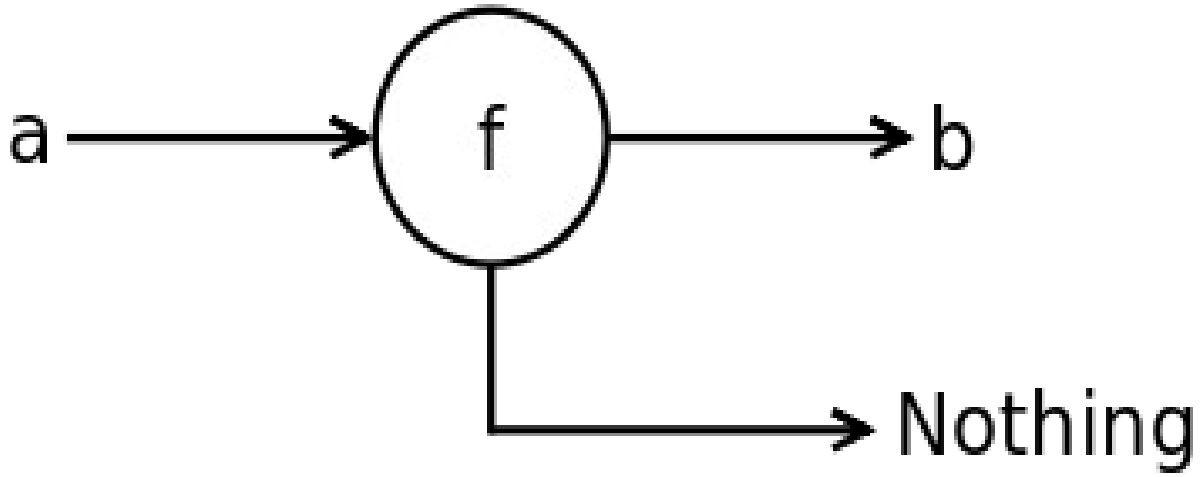
Класс Monad

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

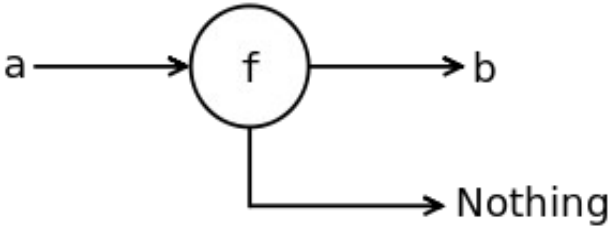
(>>=) читается как **bind**

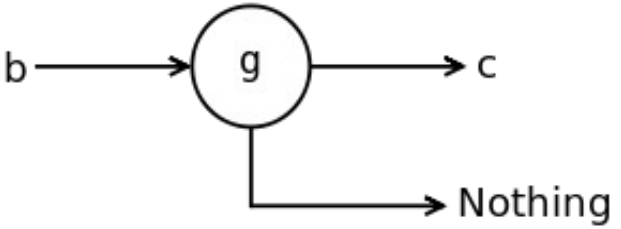
Монада Maybe

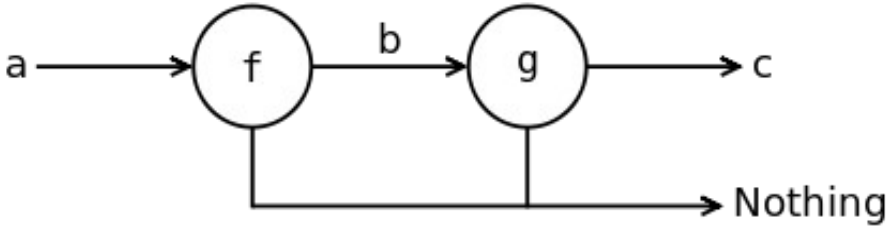
```
data Maybe a = Nothing
             | Just a
```

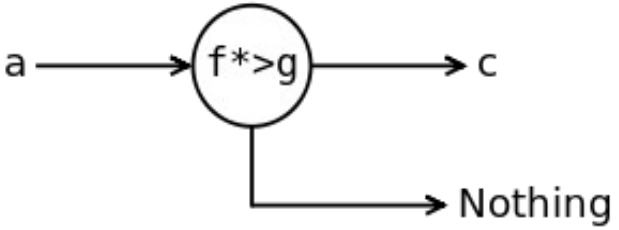


Композиция частично определённых функций







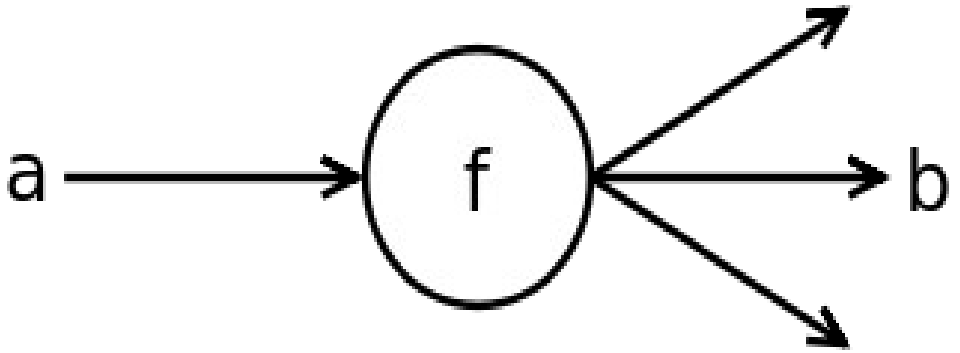


Монада для Maybe

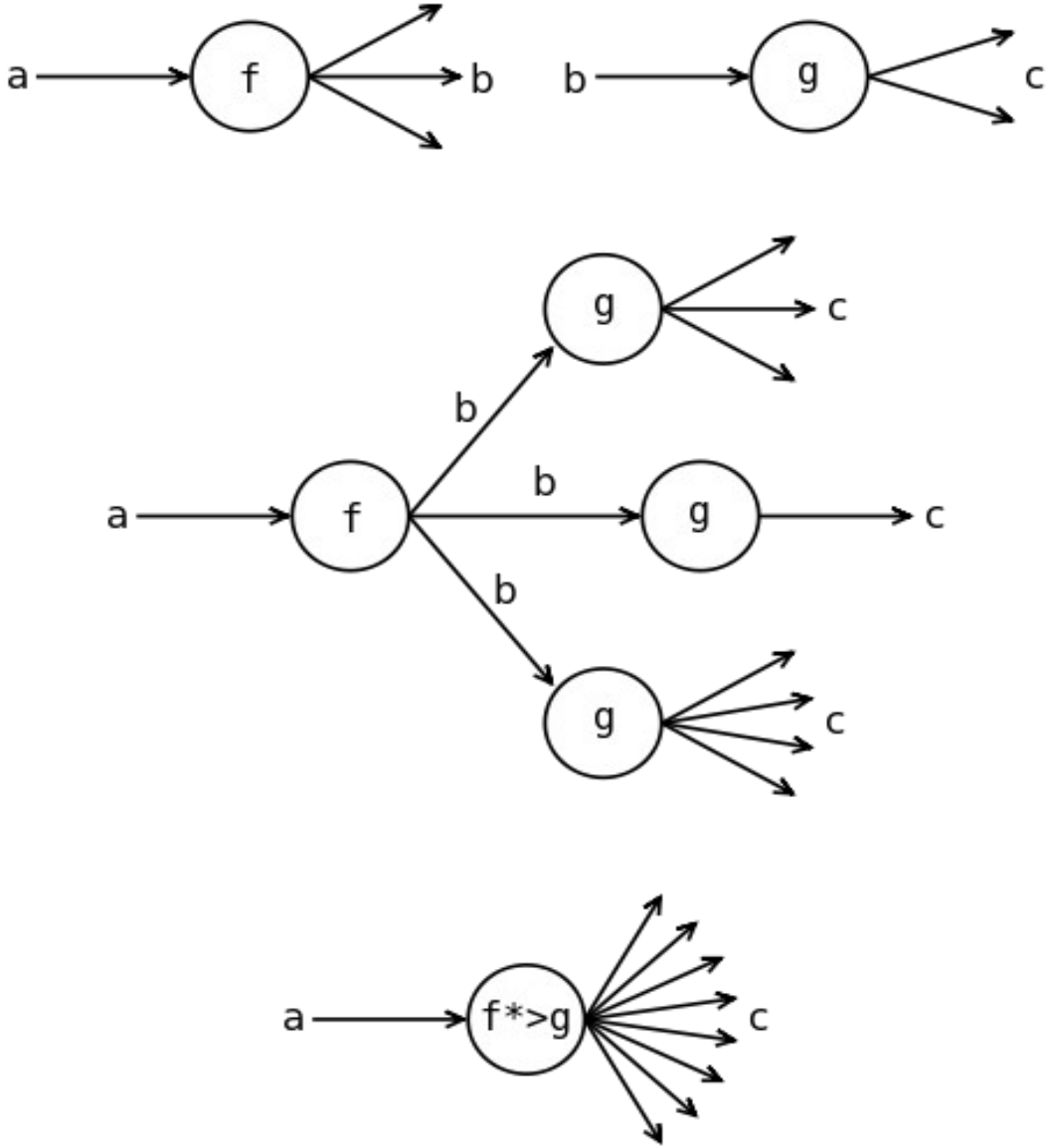
```
instance Monad Maybe where
   return a = Just a

   ma >>= f = case ma of
        Nothing -> Nothing
        Just a  -> f a
```

Монада List

```
data [a] = []
         | a :: [a]
```



Композиция многозначных функций



Монада для []

```
instance Monad [] where
   return a = [a]

   xs >>= f = concat (map f xs)
```

Что стоит за оператором >>=

```
(>>=)  :: m a -> (a -> m b) -> m b

(>>=)  :: a   -> (a -> b)   -> b

(=<<)  :: (a -> b) -> a -> b          :: ($)
```

(>>=) это ($) для значений в контексте m

с изменённым порядком аргументов

Ключевая идея!

Нет царского пути в геометрию

— Евклид

Узнать больше

- Monads for drummers

  (https://github.com/anton-k/monads-for-drummers)
- Dan Piponi о Монадах. Перевод на хабре

  (http://habrahabr.ru/post/96421/)
- Антон Холомьёв. Учебник по Haskell (главы 6, 7, 8)

  (http://anton-k.github.io/ru-haskell-book/book/toc.html)
- Philip Wadler. The essence of Functional Programming
- Simon Peyton Jones. Tackling the awkward squad

  (google search)

Благодарю за внимание!

- **github**: anton-k
- **email**: anton.kholomiov на gmail . com