

# Типы в функциональном программировании

Косарев Дмитрий а.к.а. Kakadu

матмех СПбГУ

12 сентября 2018 г.

Вопрос к залу: что такое тип?

## Вопрос к залу: что такое тип?

Ну, если  $x$  принадлежит типу  $T$ , то  $T$  определяет, какие значения может принимать  $x$ .

Тип  $T$  у  $x$  – это множество совокупность значений, которые могут быть у  $x$ .

Если  $x :: T$ , то тип  $T$  *населен*  $x$ ком.

Если  $\nexists x$ , таких что  $x :: T$ , то тип  $T$  *не населен*.

# Синтаксис типов в Haskell (1/2)

*Тип функции*, действующей из аргумента типа  $A$  и возвращающей результат  $B$ , обозначается как  $A \rightarrow B$ .

# Синтаксис типов в Haskell (1/2)

*Тип функции*, действующей из аргумента типа  $A$  и возвращающей результат  $B$ , обозначается как  $A \rightarrow B$ .

Функции, у которых  $n$  аргументов ( $n > 1$ ) моделируются как функции возвращающие функцию от  $n - 1$  аргументов. Например,  $A \rightarrow (B \rightarrow C)$ .

# Синтаксис типов в Haskell (1/2)

*Тип функции*, действующей из аргумента типа  $A$  и возвращающей результат  $B$ , обозначается как  $A \rightarrow B$ .

Функции, у которых  $n$  аргументов ( $n > 1$ ) моделируются как функции возвращающие функцию от  $n - 1$  аргументов. Например,  $A \rightarrow (B \rightarrow C)$ .

Ассоциативность правая:

$$A \rightarrow (B \rightarrow C)$$

то же самое, что и

$$A \rightarrow B \rightarrow C$$

## Синтаксис типов в Haskell (2/2)

*Параметрический полиморфизм* – когда один и тот же код, работает для разных типов (generics в Java/C#).

## Синтаксис типов в Haskell (2/2)

*Параметрический полиморфизм* – когда один и тот же код, работает для разных типов (generics в Java/C#).

Ещё бывает *ad-hoc полиморфизм* – работает разный код для разных типов (overloading в C++). Но это потом.



## Синтаксис типов в Haskell (2/2)

*Параметрический полиморфизм* – когда один и тот же код, работает для разных типов (generics в Java/C#).

Ещё бывает *ad-hoc полиморфизм* – работает разный код для разных типов (overloading в C++). Но это потом.

*Типовые переменные* в типах полиморфных функций пишутся с маленькой буквы. Например,  $a \rightarrow b$  или  $a \rightarrow b \rightarrow c$  или  $a \rightarrow (a \rightarrow b) \rightarrow b$

## Синтаксис типов в Haskell (2/2)

*Параметрический полиморфизм* – когда один и тот же код, работает для разных типов (generics в Java/C#).

Ещё бывает *ad-hoc полиморфизм* – работает разный код для разных типов (overloading в C++). Но это потом.

*Типовые переменные* в типах полиморфных функций пишутся с маленькой буквы. Например,  $a \rightarrow b$  или  $a \rightarrow b \rightarrow c$  или  $a \rightarrow (a \rightarrow b) \rightarrow b$

*Имена типов* пишутся с заглавной буквы. Например, Int, String, Float.

# Синтаксис типов в Haskell (2/2)

*Параметрический полиморфизм* – когда один и тот же код, работает для разных типов (generics в Java/C#).

Ещё бывает *ad-hoc полиморфизм* – работает разный код для разных типов (overloading в C++). Но это потом.

*Типовые переменные* в типах полиморфных функций пишутся с маленькой буквы. Например,  $a \rightarrow b$  или  $a \rightarrow b \rightarrow c$  или  $a \rightarrow (a \rightarrow b) \rightarrow b$

*Имена типов* пишутся с заглавной буквы. Например, Int, String, Float.

Когда какое-то имя *типизируется как (имеет тип) T*, то это записывают так:  $x :: T$ .

# Применение типов

*Применение типов (type application):* `List Int`, `List String`, `List a`, и т.д.

(Тут должна быть картинка)

# Чем функции в программировании отличаются от математических?

# Чем функции в программировании отличаются от математических?

- Аварийное завершение
- Отсутствие завершения

# Примеры параметрического полиморфизма

```
Prelude> let id x = x  
Prelude> :t id  
id :: p -> p
```

# Примеры параметрического полиморфизма

```
Prelude> let id x = x
```

```
Prelude> :t id
```

```
id :: p -> p
```

```
Prelude> :t (id :: String -> String)
```

```
(id :: String -> String) :: String -> String
```



# Примеры параметрического полиморфизма

```
Prelude> let id x = x
```

```
Prelude> :t id
```

```
id :: p -> p
```

```
Prelude> :t (id :: String -> String)
```

```
(id :: String -> String) :: String -> String
```

```
Prelude> :t (id :: Int -> Int)
```

```
(id :: Int -> Int) :: Int -> Int
```

# Примеры параметрического полиморфизма

```
Prelude> let id x = x
```

```
Prelude> :t id
```

```
id :: p -> p
```

```
Prelude> :t (id :: String -> String)
```

```
(id :: String -> String) :: String -> String
```

```
Prelude> :t (id :: Int -> Int)
```

```
(id :: Int -> Int) :: Int -> Int
```

**?** Каков *наиболее общий тип* для `id`?

# Примеры типов

Тип, который не населен

```
Prelude> :i Data.Void.Void  
data Data.Void.Void -- Defined in 'Data.Void'
```

Тип, у которого только один житель

```
Prelude> :i ()  
data () = () -- Defined in 'GHC.Tuple'  
Prelude> :t ()  
() :: ()
```

# Как ведет себя функция с типом

? `Void -> Int`

? `() -> Int`

? `a -> ()`

## Как ведет себя функция с типом

? Void → Int

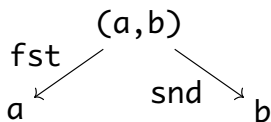
? () → Int

? a → ()

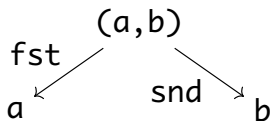
Ответ:

```
unit x = ()  
{- a лучше -}  
unit _ = ()
```

# Тип пары (Декартово произведение)



## Тип пары (Декартово произведение)



Определим проекции:

```
Prelude> let fst (x,_) = x
```

```
Prelude> :t fst
```

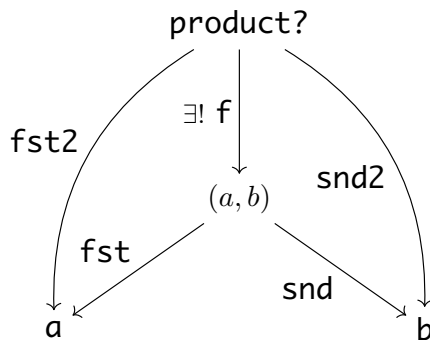
```
fst :: (a, b) -> a
```

```
Prelude> let snd (_,y) = y
```

```
Prelude> :t snd
```

```
snd :: (a, b) -> b
```

## Произведение (product) – обобщение пары

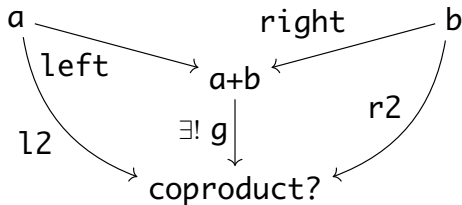


$$\text{fst2} = \text{fst} \cdot f$$

$$\text{snd2} = \text{snd} \cdot f$$



# Coproduct или Either или тип-сумма



$l2 = g \cdot \text{left}$   
 $r2 = g \cdot \text{right}$

```
Prelude> :i Either
data Either a b = Left a | Right b
-- Defined in 'Data.Either'
```

# Упражнения про произведение и сумму

- Придумайте (науглите) какой-нибудь тип, который похож на произведение? Реализуйте для него функцию  $f$ .
- Придумайте (науглите) какой-нибудь тип, который похож на копроизведение? Реализуйте для него функцию  $g$ .
- Кто населяет тип  $(\text{Void}, a)$  для произвольного типа  $a$ ?
- Кто населяет тип  $((), a)$  для произвольного типа  $a$ ?
- Кто населяет тип  $\text{Either Void } a$  для произвольного типа  $a$ ?
- Кто населяет тип  $=\text{Either } () a=$  для произвольного типа  $a$ ?
- В некоторых упражнениях выше можно строить изоморфизм между двумя типами. Сообразите между какими и постройте там, где возможно.

# Алгебраические типы данных (1/2)

## Синтаксис

```
data TypeName arg1 arg2 ... argk =  
  | C1 t11 t12 ... t1n1  
  | C2 t21 t22 ... t2n2  
  | ...  
  | Cm tm1 tm2 ... tm nm
```

## Примеры

- `data Bool = True | False`
- `data Status1 = On | Off`
- `data Maybe a = Just a | Nothing`
- `data Either a b = Left a | Right b`

## Алгебраические типы данных (2/2)

? Что такое *связный список*?

## Алгебраические типы данных (2/2)

- ? Что такое *связный список*?
- ? А что такое *список* вообще?

## Алгебраические типы данных (2/2)

? Что такое *связный список*?

? А что такое *список* вообще?

Ещё примеры:

- `data [] a = [] | a : [a]`
- `data ListG a r = Nil | Cons a r`
- `data Fix f = Fix (f (Fix f))`

Упражнение: из типов `ListG` и `Fix` можно соорудить что-то похожее на стандартный список Haskell. Предъявите изоморфизм.

# Формальные аксиомы

Distinctness:  $C_j(x) \neq C_i(y)$ , если  $j \neq i$

Injectivity:  $C_{ij}(x_1, \dots, x_{n_{ij}}) = C_{ij}(y_1, \dots, y_{n_{ij}}) \Rightarrow x_k = y_k$

Exhaustiveness:  $x :: \text{ADT} \Rightarrow \exists i, n: x = C_i(y_1, \dots, y_n)$

Selection:  $s_{ij}^k(C_{ij}(x_1, \dots, x_{n_{ij}})) = x$

Упражнение. Для типов, указанных выше, можно построить изоморфные к ним, используя только типы `()`, `Void`, `Either a b`, `(a,b)` (ну и ещё либо рекурсию, либо `Fix`). Попробуйте описать эти типы, предъявляя изоморфизм.



# Как может работать функция со следующим типом?

? `[a] -> [a]`

? `[a] -> Bool`

? `(a -> b) -> [a] -> [b]`

? `(a -> a -> Bool) -> [a] -> [a]`

? `(a -> a -> Ordering) -> [a] -> [a]`, если

```
Prelude> :i Ordering
```

```
data Ordering = LT | EQ | GT
```

? `(a -> b -> a) -> a -> [b] -> a`

? `Maybe a -> Maybe b -> (a -> b -> c) -> Maybe c`