

# Лемма Йонеды

Косарев Дмитрий a.k.a. Kakadu

матмех СПбГУ

22 мая 2019 г.

1 Предварительные знания о теории категорий

2 Лемма Йонеды

3 Применения

# Категории

Категория  $\mathbb{C}$  состоит из:

- набора объектов  $|\mathbb{C}|$ ;
- множества стрелок  $\mathbb{C}(A, B)$  из объекта  $A$  в объект  $B$  (индексированное парами объектов  $A, B \in |\mathbb{C}|$ );
- тождественная (identity) стрелка  $\text{id}_A \in \mathbb{C}(A, A)$  для каждого объекта  $A \in |\mathbb{C}|$ ;
- композиций  $g \circ f \in \mathbb{C}(A, C)$  для каждой пары “соединяемых” стрелок  $f \in \mathbb{C}(A, B)$  and  $g \in \mathbb{C}(B, C)$ ,

и такая, что

- композиция ассоциативна;
- соответствующие тождественные стрелки служат нейтральными элементами для композиции.

Стрелки ещё называют *морфизмами*.

# Локально малые (locally small) категории

## Определение

*Локально малая* категория – у которой набор стрелок между двумя произвольными объектами образует множество.

## Определение

*Малая* категория – это локально малая категория, у которой набор *объектов* образует множество.

Наборы стрелок в категории  $\mathbb{C}$  из объекта  $A$  в объект  $B$  обозначаются  $\mathbb{C}(A, B)$ .  
Множества  $\mathbb{C}(A, B)$  называются homsets.

Категория  $\mathbf{Set}$  локально малая

- Объекты – множества
- Стрелки из  $\mathbf{Set}(A, B)$  – всюду определенные отображения из  $A$  в  $B$

На типы и программы можно смотреть как на категорию  $\mathbf{Hask}$ .  
Тогда объекты будут типами, а стрелки – программами.

# Примеры

Любой предпорядок на множестве  $(A, \leq)$  порождает категорию  $\mathbb{P}re(A, \leq)$

- Объекты – элементы множества  $A$
- Множества  $\text{homset } \mathbb{P}re(A, \leq)(a, b)$ 
  - либо состоят из одного элемента (когда  $a \leq b$ )
  - либо пустые

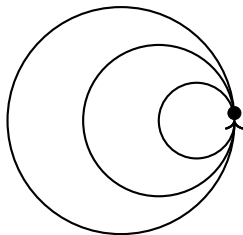
Рефлексивность  $a \leq a$  соответствует  $\text{id}_a$ .

Транзитивность – это композиция стрелок.

# Примеры

Всякий моноид  $(M, \oplus, e)$  порождает категорию  $\text{Mon}(M, \oplus, e)$

- С единственным объектом  $*$
- С единственным homset  $\text{Mon}(M, \oplus, e)(*, *)$ 
  - нейтральный элемент отображается в identity стрелку
  - остальные элементы – в остальные стрелки



## Дуальная (двойственная, opposite) категория

У каждой категории  $\mathbb{C}$  есть *дуальная* категория  $\mathbb{C}^{\text{op}}$  с теми же объектами и развернутыми стрелками

- объекты  $\mathbb{C}^{\text{op}}$  – это объекты  $\mathbb{C}$ , и наоборот;
- стрелки  $\mathbb{C}^{\text{op}}(A, B)$  из  $A$  в  $B$  категории  $\mathbb{C}^{\text{op}}$  являются стрелками  $\mathbb{C}(B, A)$  из  $B$  в  $A$  категории  $\mathbb{C}$ ;
- композиция направлена в обратную сторону

TODO: рассказать про дуальность.



# TODO: про декартову замкнутость

И на примере Haskell тоже

# Функторы

Функторы – это отображения между категориями, сохраняющие структуру.

Формально, функтор  $F : \mathbb{C} \rightarrow \mathbb{D}$  – это отображение объектов и стрелок из  $\mathbb{C}$  в объекты и стрелки из  $\mathbb{D}$ , такое, что

- объект  $F(A) \in |\mathbb{D}|$  для каждого объекта  $A \in |\mathbb{C}|$ ;
- стрелка  $F(f) \in \mathbb{D}(F(A), F(B))$  для каждой стрелки  $f \in \mathbb{C}(A, B)$ ,
- $F(\text{id}_A) = \text{id}_{F(A)}$  для каждого объекта  $A \in |\mathbb{C}|$ ;
- $F(g \circ f) = F(g) \circ F(f)$  для каждой пары  $f, g$  “соединяемых” стрелок.

TODO: сказать про категорию (малых) категорий, что там стрелки – это функторы, а объекты – малые категории. И свойства программмерского функтора вытекают из свойств стрелок.

# Homfunctor

Дана  $\mathbb{C}$  и объекты  $A, B \in |\mathbb{C}|$ ,  $\text{homset } \mathbb{C}(A, B)$  – это множество стрелок;

Введем  $\mathbb{C}(A, -)$  – это отображение из  $|\mathbb{C}|$  в  $|\text{Set}|$ , переводящее объекты  $B \mapsto \mathbb{C}(A, B)$  (т.е. объекты в набор морфизмов).

Его можно расширить до стрелок  $f \in \mathbb{C}(B, C)$ :  
 $f \mapsto (f \circ) \in \text{Set}(\mathbb{C}(A, B), \mathbb{C}(A, C))$

Функтор  $\mathbb{C}(A, -)$  из  $\mathbb{C}$  в  $\text{Set}$  будем называть *homfunctor*.

Аналогично,  $\mathbb{C}(-, B)$  функтор преобразующий  $f \mapsto (\circ f)$ ; но контравариантный, т.е. homfunctor из категории  $\mathbb{C}^{\text{op}}$  в  $\text{Set}$

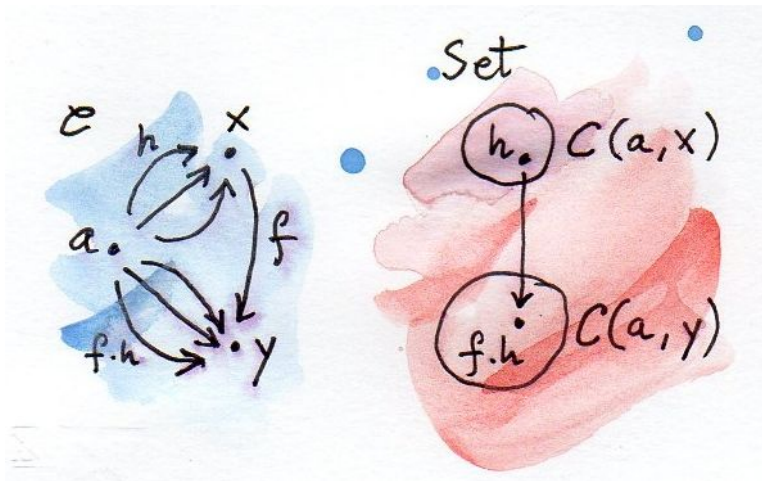


Рис.: Ковариантный Hom функтор от Бартоша Милевски

# В терминах Haskell

Ковариантный – композиция после

```
type Reader a x = a -> x
```

```
instance Functor (Reader a) where  
    fmap f h = f . h
```

Контравариантный – композиция до

```
type Op a x = x -> a
```

```
instance Contravariant (Op a) where  
    contramap f h = h . f
```

# Примеры функторов

Функторы  $\mathbb{Pre}(A, \leq) \rightarrow \mathbb{Pre}(B, \sqsubseteq)$  – это монотонные функции

Функторы  $\mathbb{Mon}(M, \oplus, e) \rightarrow \mathbb{Mon}(N, \otimes, e')$  – гомоморфизмы моноидов

# Естественные преобразования (natural transformations)

По сути: отображение между функторами, сохраняющее структуру.

Формально, даны функторы  $F, G : \mathbb{C} \rightarrow \mathbb{D}$ . *Естественным преобразованием*  $\phi : F \rightarrow G$  из  $F$  в  $G$  будет семейство стрелок в  $\mathbb{D}$  индексированное объектами из  $\mathbb{C}$ , такое что

- стрелка  $\phi_A \in \mathbb{D}(F(A), G(A))$  для каждого объекта  $A \in |\mathbb{C}|$ ;
- *условие натуральности*: для произвольной стрелки  $f \in \mathbb{C}(A, B)$  верно  $\phi_B \circ F(f) = G(f) \circ \phi_A$

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \phi_A \downarrow & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$





## Примеры из программирования (1/2)

Отображение  $\text{reverse} : \text{List} \rightarrow \text{List}$  – естественное из функтора  $\text{List}$  в себя.

$$\begin{array}{ccc} F(A) & \xrightarrow{F(f)} & F(B) \\ \phi_A \downarrow & & \downarrow \phi_B \\ G(A) & \xrightarrow{G(f)} & G(B) \end{array}$$

(a) Натуральность

$$\begin{array}{ccc} [a] & \xrightarrow{\text{map}(f)} & [b] \\ \text{reverse}_a \downarrow & & \downarrow \text{reverse}_b \\ [a] & \xrightarrow{\text{map}(f)} & [b] \end{array}$$

(b) Специализация для  $\text{List}$

Здесь  $\phi \equiv \text{reverse}$ ;  $F, G \equiv \text{списки}$

## Примеры из программирования (2/2)

Для функтора  $\text{Tree} : \mathbb{H}\text{ask} \longrightarrow \mathbb{H}\text{ask}$ , обход дерева будет естественным преобразованием  $\text{Tree} \longrightarrow \text{List}$ .

Условие естественности:

$$\text{inorder}_B \circ \text{Tree}(f) = \text{List}(f) \circ \text{inorder}_A$$

Комментарий про естественность

1 Предварительные знания о теории категорий

2 Лемма Йонеды

3 Применения

Для произвольных: малой категории  $\mathbb{C}$ , объекта  $A \in \mathbb{C}$  и функтора  $F : \mathbb{C} \rightarrow \mathbf{Set}$

*Лемма (Йонеды для ковариантного hom-функтора)*

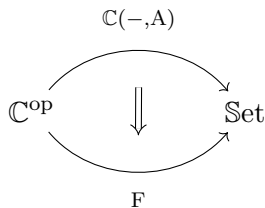
$$[-]^{A,F} : [\mathbb{C}, \mathbf{Set}](\mathbb{C}(A, -), F) \simeq F(A) \quad : [-]^{A,F}.$$

*Лемма (Йонеды для контравариантного hom-функтора)*

$$[-]^{A,F} : [\mathbb{C}^{\text{op}}, \mathbf{Set}](\mathbb{C}(-, A), F) \simeq F(A) \quad : [-]^{A,F}.$$

$$[-]^{A,F} : [\mathbb{C}^{\text{op}}, \text{Set}](\mathbb{C}(-, A), F) \simeq F(A) : [-]^{A,F}.$$

- $\mathbb{C}$  – категория
- $\mathbb{C}^{\text{op}}$  – тоже категория; те же объекты, стрелки развернуты
- $\text{Set}$  – тоже категория; объекты  $\equiv$  множества, стрелки  $\equiv$  функции
- $[A, B]$  – категория; объекты – функторы из  $A$  в  $B$ ; стрелки – натуральные трансформации
- для  $\forall$  объекта  $A$  существует функтор  $\mathbb{C}(-, A)$  – hom функтор.
- Для категории  $D$  множество стрелок из  $a$  в  $b$  обозначается как  $D(a, b)$ .
- $F$  – функтор  $\mathbb{C}^{\text{op}} \rightarrow \text{Set}$
- В левой части множество морфизмов в  $[\mathbb{C}^{\text{op}}, \text{Set}]$  из  $\mathbb{C}(-, A)$  в  $F$
- Слева и справа множества. Изоморфизм  $\equiv$  биекция



## Доказательство (1/2)

$$\text{id}_C \in \mathbb{C}(A, A) \quad \eta : \mathbb{C}(-, A) \longrightarrow F$$

$$\begin{array}{ccc} \mathbb{C}(A, A) & \xrightarrow{\mathbb{C}(f, A)} & \mathbb{C}(B, A) \\ \eta_A \downarrow & & \downarrow \eta_B \\ F(A) & \xrightarrow{F(f)} & F(B) \end{array}$$

$$\begin{array}{ccc} \text{id}_A & \xrightarrow{\quad} & f \\ \downarrow & & \downarrow \\ u := \eta_A(\text{Id}_A) & \xrightarrow{F(f)} & \eta_B(f) \end{array}$$

$\eta$  определяется полностью единственным значением  $u := \eta_A(\text{Id}_A) \in F(A)$ , так как  $\forall B \in \mathbb{C} \ \eta_B : \mathbb{C}(B, A) \rightarrow F(B)$  она должна преобразовывать  $f \in \mathbb{C}(B, A)$  (т.е. морфизм  $f : B \rightarrow A$ ) согласно коммутативности.

**Ключевая идея:** условий натуральности для произвольной  $\eta : \mathbb{C}(-, A) \longrightarrow F$  достаточно, чтобы постановить, что  $\eta$  однозначно определилась своим значением  $\eta_A(\text{id}_A) \in F(A)$  за счет компоненты  $\eta_A : \mathbb{C}(A, A) \rightarrow F(A)$  на тождественном морфизме  $\text{id}_A$ .

## Доказательство (2/2)

Слева направо

$$[\mathbb{C}^{\text{op}}, \text{Set}](\mathbb{C}(-, A), F) \xrightarrow{\text{comp}_A} \text{Set}(\mathbb{C}(A, A), F(A)) \xrightarrow{\text{ev}_{\text{Id}_A}} F(A)$$

“application to identity arrow”

Справа налево: надо перегнуть элемент  $x$  множества  $F(A)$  в натуральную трансформацию  $\mathbb{C}(-, A) \rightarrow F$ . Конструируется покомпонентно.  $\eta_B$  должна быть стрелкой в  $\text{Set}$ , т.е. функцией  $\mathbb{C}(B, A) \rightarrow F(B)$

$$[x]^{A, F}(f) = F(f)(x), \quad \text{где } f \text{ — стрелка в } \mathbb{C}$$

“using functorial action”

homfunctor  $\mathbb{C}(A, -) : \mathbb{C} \rightarrow \text{Set}$  получается фиксацией *начала* и варьированием *конца* стрелки

homfunctor  $\mathbb{C}(-, B) : \mathbb{C}^{\text{op}} \rightarrow \text{Set}$  получается фиксацией *конца* и варьированием *начала* стрелки

Можно построить  $\mathbb{C}(-, -) : \mathbb{C}^{\text{op}} \times \mathbb{C} \rightarrow \text{Set}$   
и получить путём каррирования  $H^\bullet : \mathbb{C}^{\text{op}} \rightarrow [\mathbb{C}, \text{Set}]$



## Лемма

Функтор  $H^\bullet : C^{\text{op}} \rightarrow [C, \text{Set}]$  полный (full), строгий (faithful) и инъективен на объектах

Функтор полный, если он сюръективен на каждом homset

Функтор строгий, если инъективен на каждом homset.

TODO: какойнить пример

Подставим  $\mathbb{C}(B, A)$  вместо  $F$  в лемме Йонеды

$[\mathbb{C}^{\text{op}}, \text{Set}]$

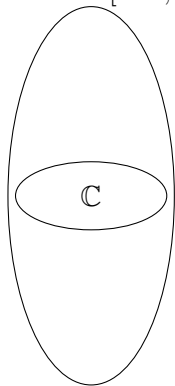
$$[\mathbb{C}, \text{Set}](\mathbb{C}(A, -), \mathbb{C}(B, -)) \simeq \mathbb{C}(B, A)$$

Отображение справа налево переводит стрелку  $f \in \mathbb{C}(B, A)$  в  $\mathbb{C}(f, -) = (\text{of})$

Инъективность на объектах получается автоматически, так как разные  $\text{homset}$  не пересекаются

Faithful (строгий) – мотивирует название embedding

Full (полный) означает, что встраивание "хорошее": отображение  $\mathbb{C}(A, -) \rightarrow \mathbb{C}(B, -)$  в  $[\mathbb{C}, \text{Set}]$  – это то же самое отображение, что и  $B \rightarrow A$  в  $\mathbb{C}$ .



# Представление универсального элемента

$\mathbb{C}$  – малая категория и функтор  $F : \mathbb{C} \rightarrow \mathbf{Set}$

Тогда представление функтора  $F$  состоит из объекта  $A \in |\mathbb{C}|$  вместе с элементом  $u \in F(A)$  таким, что для произвольного  $B \in |\mathbb{C}|$  и  $x \in F(B)$  существует уникальное отображение  $f : A \rightarrow B$ , такое что  $F(f)(u) = x$

1 Предварительные знания о теории категорий

2 Лемма Йонеды

3 Применения

# Indirect inequality

Для произвольного предпорядка  $(A, \leq)$

$$(b \leq a) \Leftrightarrow (\forall c. (a \leq c) \Rightarrow (b \leq c))$$

Доказательство:

$\Rightarrow$  транзитивность

$\Leftarrow$  рефлексивность  $\leq$

# Indirect inequality

Категория  $\mathbb{Pre}(A, \leq)$ .

$\text{homset } \mathbb{Pre}(A, \leq)(b, a)$  – “худое” множество (одноэлементное, если  $b \leq a$ , иначе пустое).

$\mathbb{Pre}(A, \leq)(a, -)$  – “худая” функция: переводит  $c \in A$  в одноэлементное множество, если  $a \leq c$ , иначе в пустое.

Нат. тр-я  $\phi : \mathbb{Pre}(A, \leq)(a, -) \rightarrow \mathbb{Pre}(A, \leq)(b, -)$  – это семейство функций, очень просто устроенных

Семейство  $\phi$  функций – свидетель того, что для каждого  $c$ , если  $\mathbb{Pre}(A, \leq)(a, c)$  не пусто, то  $\mathbb{Pre}(A, \leq)(b, c)$  тоже, а следовательно  $(a \leq c)$  влечет  $(b \leq c)$

# Indirect equality

$$(b \simeq a) \Leftrightarrow (\forall c . (a \leq c) \Leftrightarrow (b \leq c))$$

$$F(f) \circ F(g) = \text{id} \\ \Leftrightarrow F \text{ is a functor}$$

$$F(f \circ g) = F(\text{id}) \\ \Leftarrow \text{Leibniz}$$

$$f \circ g = \text{id}$$

- Если  $F$  строгий (faithful), то последний шаг – это  $\Leftrightarrow$
- В этом случае, если  $F(f)$  и  $F(g)$  образуют изоморфизм, то  $f$  и  $g$  тоже
- Если  $F$  full (сюръективен по стрелкам), то если  $F(f)$  имеет обратное  $h$ , то  $h$  достигим путём  $F$ , т.е.  $\exists g : h = F(g)$
- Итого, для full&faithful функтора  $F$ , стрелка  $f$  образует изоморфизм  $\Leftrightarrow F(f)$  образует тоже
- Yoneda embedding предоставляет full&faithful функтор

$$(\mathbb{C}(B, -) \simeq \mathbb{C}(A, -)) \Leftrightarrow (A \simeq B) \Leftrightarrow (\mathbb{C}(-, A) \simeq \mathbb{C}(-, B))$$

# Воплощение в Haskell

```
data Yo f a = Yo { unYo :: forall r . (a -> r) -> f r }
```

Лемма гласит, что  $Yo\ f\ a \simeq f\ a$ , если  $f$  – это функтор.

Надо предъявить изоморфизм

```
fromYo :: Yo f a -> f a
```

```
fromYo y = unYo y id {- application to identity -}
```

```
toYo :: Functor f => f a -> Yo f a
```

```
toYo x = Yo (\h -> fmap h x) {- use functorial action -}
```

Неформально:  $Yo\ f\ a$  берет произвольную функцию  $a \rightarrow r$  для произвольного  $r$  и возвращает значение типа  $f\ r$ . В некотором смысле она должна иметь  $fa$  сохраненным внутри себя.



## Частный случай: $f = \text{Id}$

$$Y \circ \text{Id } a \simeq \text{Id } a \simeq a$$

$\Downarrow$

$$b \rightarrow Y \circ \text{Id } a \simeq b \rightarrow \text{Id } a \simeq b \rightarrow a$$

$$Y \circ \text{Id } a = \forall r. (a \rightarrow r) \rightarrow r$$

Что сильно напоминает преобразование

$$b \rightarrow a \rightsquigarrow \forall r. b \rightarrow (a \rightarrow r) \rightarrow r$$

*Теорема*

*CPS преобразование корректно*

**Доказательство.**

Лемма Йонеды



## Воплощение CoYoneda в Haskell (1/2)

$$\forall a . f\ a \rightarrow (\forall r . (a \rightarrow r) \rightarrow g\ r)$$

$\simeq$  Универсальное свойство квантора всеобщности

$$\forall a . \forall r . (f\ a \rightarrow (a \rightarrow r) \rightarrow g\ r)$$

$\simeq$  uncurrying

$$\forall a . \forall r . (f\ a \times (a \rightarrow r) \rightarrow g\ r)$$

$\simeq$  swap кванторов

$$\forall r . \forall a . (f\ a \times (a \rightarrow r) \rightarrow g\ r)$$

$\simeq$  Универсальное свойство квантора существования

$$\forall r . (\exists a . (f\ a \times (a \rightarrow r))) \rightarrow g\ r)$$

## Воплощение CoYoneda в Haskell (2/2)

```
data CoYo f r = exists a . CoYo { unCoYo :: (f a, a -> r) }
```

```
fromCoYo :: Functor f => CoYo f b -> f b  
fromCoYo (CoYo (x, h)) = fmap h x
```

```
toCoYo :: f b -> CoYo f b  
toCoYo y = CoYo (y, id)
```

```
instance Functor (CoYo f ) where  
    fmap g (CoYo (x, h)) = CoYo (x, g . h)
```

Использоваться, чтобы для GADT, где типы используются только как индесы, породить функтор

## Представимый (representable) функтор

Выбрав в категории  $\mathbb{C}$  объект  $a$  мы автоматически получаем функтор  $\text{Hom}(a, -)$ , который представляет нашу категорию в категории  $\text{Set}$ . Мы представляем объекты и морфизмы  $\mathbb{C}$  как множества и функции в  $\text{Set}$ .

Функтор  $\text{Hom}(a, -)$  для некоторого  $a$  иногда называют представимым.

Вообще, любой функтор, который для некоторого  $a$  изоморфен  $\text{hom}$  функтору, называется *представимым*.

Чтобы функтор был представимым, нужно предъявить  $\alpha$  и  $\beta$  с двумя условиями

- $\alpha \circ \beta = \text{id} = \beta \circ \alpha$
- Условие натуральности:  $Ff \circ \alpha_x = \alpha_y \circ \mathbb{C}(a, f)$

```
alpha :: forall x. (a -> x) -> F x
fmap f . alpha = alpha . fmap f
```

Выше правый `fmap` – от функтора `Reader`. Можно чуть-чуть упростить

```
fmap f (alpha h) = alpha (f . h)
```

Аналогично для второй функции  $\beta$

```
beta :: forall x. F x -> (a -> x)
```

# Непредставимость функтора List

Число 42 – с потолка.

```
alpha :: forall x. (Int -> x) -> [x]  
alpha h = map h [42]
```

Условие натуральности выполняется

```
map f (map h [12]) = map (f . h) [12]
```

А что на счет обратной части изоморфизма  $\beta$ ?

```
beta :: forall x. [x] -> (Int -> x)
```

Всё будет плохо на пустом списке

## Представимый функтор в Haskell

```
class Functor f => Naperian f where
  type Log f
  lookup    :: f a -> (Log f -> a)  — each other's...
  tabulate  :: (Log f -> a) -> f a  — ... inverses
```

```
data Stream x = Cons x (Stream x)
instance Naperian Stream where
  type Rep Stream = Integer
```

```
lookup (Cons b bs) n =
  if n == 0 then b
  else index bs (n - 1)
tabulate f = Cons (f 0) (tabulate (f . (+1)))
```

## Представимый функтор в Haskell

```
class Functor f => Naperian f where
  type Log f
  lookup    :: f a -> (Log f -> a)  — each other's...
  tabulate  :: (Log f -> a) -> f a  — ... inverses
```

```
data Stream x = Cons x (Stream x)
instance Naperian Stream where
  type Rep Stream = Integer
```

```
lookup (Cons b bs) n =
  if n == 0 then b
  else index bs (n - 1)
tabulate f = Cons (f 0) (tabulate (f . (+1)))
```

Вопрос: можно ли задать представимый функтор по-другому, другими функциями, потому что эти другие функции проще описывать?








```
class Functor f => Naperian f where
  type Log f
  lookup    :: f a -> (Log f -> a)  — each other's...
  tabulate  :: (Log f -> a) -> f a  — ... inverses

  positions :: f (Log f)
  tabulate h = fmap h positions
  positions = tabulate id
```

Если функтор – это пара,  
то `positions = (True, False)`  
и `tabulate f = (f True, f False)`

# Конец

-  [Functors from GADTs via coYoneda](#)  
*Gabriel Gonzalez*  
[ссылка](#)
-  [The Yoneda Lemma: What's It All About?](#)  
*Tom Leinster*  
[ссылка](#)
-  [From the Yoneda lemma to categorical physics](#)  
*John Baez*  
[ссылка](#)
-  [What You Needa Know about Yoneda](#)  
*Guillaume Boisseasu & Jeremy Gibbons*  
[ссылка](#)
-  [Univeral element in nCat lab](#)