

# Введение в Теорию Типов

## Конспект лекций

Штукенберг Д. Г.  
Университет ИТМО

10 ноября 2018 г.

## 1 Введение

Эти лекции были рассказаны студентам групп М3336–М3339 в 2018 году в Университете ИТМО, на Кафедре компьютерных технологий Факультета информационных технологий и программирования.

Конспект подготовили студенты Кафедры: Егор Галкин (лекции 1 и 2), Илья Кокорин (лекции 3 и 4), Никита Дугинец (лекции 5 и 6), Степан Прудников (лекции 7 и 8).  
(возможно, история сложнее)

## 2 Лекция 3

### 2.1 Y-комбинатор

**Определение 2.1.** Комбинатором называется  $\lambda$ -выражение, не имеющее свободных переменных

**Определение 2.2.** ( $Y$ -комбинатор)

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

Очевидно,  $Y$ -комбинатор является комбинатором.

**Теорема 2.1.**  $Yf =_{\beta} f(Yf)$

*Доказательство.*  $\beta$ -редуцируем выражение  $Yf$

$$\begin{aligned} &=_{\beta} (\lambda f.(\lambda x.f(xx))(\lambda x.f(xx)))f \\ &=_{\beta} (\lambda x.f(xx))(\lambda x.f(xx)) \\ &=_{\beta} f((\lambda x.f(xx))(\lambda x.f(xx))) \\ &=_{\beta} f(Yf) \end{aligned}$$

Так как при второй редукции мы получили, что  $Yf =_{\beta} (\lambda x.f(xx))(\lambda x.f(xx))$  □

Следствием этого утверждения является теорема о неподвижной точке для бестипового лямбда-исчисления

**Теорема 2.2.** В лямбда-исчислении каждый терм  $f$  имеет неподвижную точку, то есть такое  $p$ , что  $f p =_{\beta} p$

*Доказательство.* Возьмём в качестве  $p$  терм  $Yf$ . По предыдущей теореме,  $f(Yf) =_{\beta} Yf$ , то есть  $Yf$  является неподвижной точкой для  $f$ . Для любого терма  $f$  существует терм  $Yf$ , значит, у любого терма есть неподвижная точка.  $\square$

## 2.2 Рекурсия

С помощью  $Y$ -комбинатора можно определять рекурсивные функции, например, функцию, вычисляющую факториал Чёрчевского нумерала. Для этого определим вспомогательную функцию

$$fact' = \lambda f. \lambda n. isZero\ n\ \bar{1}(mul\ n\ f((-1)n))$$

Тогда  $fact = Yfact'$

Заметим, что  $fact\ \bar{n} =_{\beta} fact'\ (Y\ fact')\ \bar{n} =_{\beta} fact'\ fact\ \bar{n}$ , то есть в тело функции  $fact'$  вместо функции  $f$  будет подставлена  $fact$  (заметим, что это значит, что именно функция  $fact$  будет применена к  $\bar{n} - \bar{1}$ , то есть это соответствует нашим представлениям о рекурсии.)

Для понимания того, как это работает, посчитаем  $fact\ \bar{2}$

$$\begin{aligned} & fact\ \bar{2} \\ &=_{\beta} Y\ fact'\ \bar{2} \\ &=_{\beta} fact'\ (Y\ fact'\ \bar{2}) \\ &=_{\beta} (\lambda f. \lambda n. isZero\ n\ \bar{1}(mul\ n\ f((-1)n)))(Y\ fact')\ \bar{2} \\ &=_{\beta} isZero\ \bar{2}\ \bar{1}(mul\ \bar{2}\ ((Y\ fact')((-1)\bar{2}))) \\ &=_{\beta} mul\ \bar{2}\ ((Y\ fact')((-1)\bar{2})) \\ &=_{\beta} mul\ \bar{2}\ (Y\ fact'\ \bar{1}) \\ &=_{\beta} mul\ \bar{2}\ (fact'\ (Y\ fact'\ \bar{1})) \end{aligned}$$

Раскрывая  $fact'\ (Y\ fact'\ \bar{1})$  так же, как мы раскрывали  $fact'\ (Y\ fact'\ \bar{2})$ , получаем

$$=_{\beta} mul\ \bar{2}\ (mul\ \bar{1}\ (Y\ fact'\ \bar{0}))$$

Посчитаем  $(Y\ fact'\ \bar{0})$ .

$$\begin{aligned} & (Y\ fact'\ \bar{0}) \\ &=_{\beta} fact'\ (Y\ fact'\ \bar{0}) \\ &=_{\beta} (\lambda f. \lambda n. isZero\ n\ \bar{1}(mul\ n\ f((-1)n)))(Y\ fact')\ \bar{0} \\ &=_{\beta} isZero\ \bar{0}\ \bar{1}(mul\ \bar{0}\ ((Y\ fact')((-1)\bar{0}))) =_{\beta} \bar{1} \end{aligned}$$

Таким образом,

$$\begin{aligned} & fact\ \bar{2} \\ &=_{\beta} mul\ \bar{2}\ (mul\ \bar{1}\ (Y\ fact'\ \bar{0})) \\ &=_{\beta} mul\ \bar{2}\ (mul\ \bar{1}\ \bar{1}) =_{\beta} mul\ \bar{2}\ \bar{1} =_{\beta} \bar{2} \end{aligned}$$

## 2.3 Парадокс Карри

Попробуем построить логику на основе  $\lambda$ -исчисления. Введём логический символ  $\rightarrow$ .

Будем требовать от этого исчисления наличия следующих схем аксиом:

1.  $\vdash A \rightarrow A$
2.  $\vdash (A \rightarrow (A \rightarrow B)) \rightarrow (A \rightarrow B)$
3.  $\vdash A =_\beta B$ , тогда  $A \rightarrow B$

А так же правила вывода МР:

$$\frac{\vdash A \rightarrow B, \vdash A}{\vdash B}$$

Не вводя дополнительные правила вывода и схемы аксиом, покажем, что данная логика является противоречивой. Для чего введём следующие условные обозначения:

$$F_\alpha = \lambda x. (x x) \rightarrow \alpha$$

$$\Phi_\alpha = F_\alpha F_\alpha = (\lambda x. (x x) \rightarrow \alpha) (\lambda x. (x x) \rightarrow \alpha)$$

Редуцируя  $\Phi_\alpha$ , получаем

$$\begin{aligned} & \Phi_\alpha \\ &=_\beta (\lambda x. (x x) \rightarrow \alpha) (\lambda x. (x x) \rightarrow \alpha) \\ &=_\beta (\lambda x. (x x) \rightarrow \alpha) (\lambda x. (x x) \rightarrow \alpha) \rightarrow \alpha \\ &=_\beta \Phi_\alpha \rightarrow \alpha \end{aligned}$$

Таким образом, для доказательства  $\alpha$  нужно всего лишь доказать  $\Phi_\alpha$  и применить правило МР.

- |   |  |
|---|--|
| 1) $\vdash \Phi_\alpha \rightarrow \Phi_\alpha \rightarrow \alpha$  | Так как $\Phi_\alpha =_\beta \Phi_\alpha \rightarrow \alpha$                     |
| 2) $\vdash (\Phi_\alpha \rightarrow \Phi_\alpha \rightarrow \alpha) \rightarrow (\Phi_\alpha \rightarrow \alpha)$ | Так как $\vdash (A \rightarrow (A \rightarrow B)) \rightarrow (A \rightarrow B)$ |
| 3) $\vdash \Phi_\alpha \rightarrow \alpha$  | МР 2, 3  |
| 4) $\vdash (\Phi_\alpha \rightarrow \alpha) \rightarrow \Phi_\alpha$  | Так как $\vdash \Phi_\alpha \rightarrow \alpha =_\beta \Phi_\alpha$              |
| 5) $\vdash \Phi_\alpha$   | МР 3, 4  |
| 6) $\vdash \alpha$  | МР 3, 5  |

Таким образом, введённая логика оказывается противоречивой.

## 2.4 Импликационный фрагмент интуиционистского исчисления высказываний

Рассмотрим подмножество ИИВ, со следующей грамматикой:

$$\Phi ::= x \mid \Phi \rightarrow \Phi \mid (\Phi)$$

То есть состоящее только из мереманных и импликаций.

Добавим в него одну схему аксиом

$$\Gamma, \varphi \vdash \varphi$$

И два правила вывода

1. Правило введения импликации:

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}$$

2. Правило удаления импликации:

$$\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

**Пример.** Докажем  $\vdash \varphi \rightarrow \psi \rightarrow \varphi$

$$\frac{\frac{\varphi, \psi \vdash \varphi}{\varphi \vdash \psi \rightarrow \varphi} \text{ (Введение импликации)}}{\vdash \varphi \rightarrow (\psi \rightarrow \varphi)} \text{ (Введение импликации)}$$

**Пример.** Докажем  $\alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \gamma$

$$\frac{\frac{\alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \alpha \rightarrow \beta \rightarrow \gamma \quad \alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \alpha}{\alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \beta \rightarrow \gamma} \quad \alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \beta}{\alpha \rightarrow \beta \rightarrow \gamma, \alpha, \beta \vdash \gamma}$$

## 2.5 Просто типизированное по Карри лямбда-исчисление

**Определение 2.3.** Тип в просто типизированном лямбда-исчислении по Карри это либо маленькая греческая буква  $(\alpha, \phi, \theta, \dots)$ , либо импликация  $(\theta_1 \rightarrow \theta_2)$

Таким образом,  $\Theta ::= \theta_i | \Theta \rightarrow \Theta | (\Theta)$

Импликация при этом считается правоассоциативной операцией.

**Определение 2.4.** Язык просто типизированного лямбда-исчисления это язык бестипового лямбда-исчисления.

**Определение 2.5.** Контекст  $\Gamma$  это список выражений вида  $A : \theta$ , где  $A$  - лямбда-терм, а  $\theta$  - тип

**Определение 2.6.** Просто типизированное лямбда-исчисление по Карри.

Рассмотрим исчисление с единственной схемой аксиом:

$$\Gamma, x : \theta \vdash x : \theta, \text{ если } x \text{ не входит в } \Gamma$$

И следующими правилами вывода

1. Правило типизации абстракции

$$\frac{\Gamma, x : \varphi \vdash P : \psi}{\Gamma \vdash (\lambda x. P) : \varphi \rightarrow \psi}$$

2. Правило типизации импликации:

$$\frac{\Gamma \vdash P : \varphi \rightarrow \psi \quad \Gamma \vdash Q : \varphi}{\Gamma \vdash PQ : \psi}$$

Если  $\lambda$ -выражение типизируется с использованием этих двух правил и одной схемы аксиом, то будем говорить, что оно типизируется по Карри.

**Пример.** Докажем  $\vdash \lambda x. \lambda y. x : \alpha \rightarrow \beta \rightarrow \alpha$

$$\frac{\frac{x : \alpha, y : \beta \vdash x : \alpha}{x : \alpha \vdash \lambda y. x : \beta \rightarrow \alpha} \text{ (Правило типизации импликации)}}{\vdash \lambda x. \lambda y. x : \alpha \rightarrow \beta \rightarrow \alpha} \text{ (Правило типизации импликации)}$$

**Пример.** Докажем  $\vdash \lambda x. \lambda y. x y : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta$

$$\frac{\frac{\frac{x : \alpha \rightarrow \beta, y : \alpha \vdash x : \alpha \rightarrow \beta \quad x : \alpha \rightarrow \beta, y : \alpha \vdash y : \alpha}{x : \alpha \rightarrow \beta, y : \alpha \vdash x y : \beta}}{x : \alpha \rightarrow \beta \vdash \lambda y. x y : \alpha \rightarrow \beta}}{\vdash \lambda x. \lambda y. x y : (\alpha \rightarrow \beta) \rightarrow \alpha \rightarrow \beta}$$

## 2.6 Отсутствие типа у Y-комбинатора

**Теорема 2.3.** Y-комбинатор не типизируется в просто типизированном по Карри лямбда исчислении

**Неформальное доказательство**  $Y f =_{\beta} f (Y f)$ , поэтому  $Y f$  и  $f (Y f)$  должны иметь одинаковые типы.

Пусть  $Y f : \alpha$

Тогда  $Y : \beta \rightarrow \alpha, f : \beta$

Из  $f (Y f) : \alpha$  получаем  $f : \alpha \rightarrow \alpha$  (так как  $Y f : \alpha$ )

Тогда  $\beta = \alpha \rightarrow \alpha$ , из этого получаем  $Y : (\alpha \rightarrow \alpha) \rightarrow \alpha$

Можно доказать, что  $\lambda x. x : \alpha \rightarrow \alpha$ . Тогда  $Y \lambda x. x : \alpha$ , то есть любой тип является обитаемым. Так как это невозможно, Y-комбинатор не может иметь типа, так как тогда он сделает нашу логику противоречивой.

**Формальное доказательство** Докажем от противного. Пусть Y-комбинатор типизируем. Тогда в выводе его типа есть вывод типа выражения  $x x$ . Так как  $x x$  - абстракция, то и типизированна она может быть только по правилу абстракции. Значит, в выводе типа Y-комбинатора есть такой вывод:

$$\frac{\Gamma \vdash x : \varphi \rightarrow \psi \quad \Gamma \vdash x : \varphi}{\Gamma \vdash x x : \psi}$$

Рассмотрим типизацию  $\Gamma \vdash x : \varphi \rightarrow \psi$  и  $\Gamma \vdash x : \varphi$ .  $x$  это атомарная переменная, значит, она могла быть типизирована только по единственной схеме аксиом.

Следовательно,  $x$  типизируется следующим образом.

$$\frac{\Gamma', x : \varphi \rightarrow \psi, x : \varphi \vdash x : \varphi \rightarrow \psi \quad \Gamma', x : \varphi \rightarrow \psi, x : \varphi \vdash x : \varphi}{\Gamma', x : \varphi \rightarrow \psi, x : \varphi \vdash x x : \psi}$$

Следовательно, в контексте  $\Gamma$  переменная  $x$  встречается два раза, что невозможно по схеме аксиом.

## 2.7 Изоморфизм Карри-Ховарда

Заметим, что аксиомы и правила вывода импликационного фрагмента ИИВ и просто типизированного по Карри лямбда-исчисления точно соответствуют друг другу.

Просто типизированное $\lambda$ -исчисление	Импликативный фрагмент ИИВ
$\Gamma, x : \theta \vdash x : \theta$	$\Gamma, \varphi \vdash \varphi$
$\frac{\Gamma, x : \varphi \vdash P : \psi}{\Gamma \vdash (\lambda x. P) : \varphi \rightarrow \psi}$	$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}$
$\frac{\Gamma \vdash P : \varphi \rightarrow \psi \quad \Gamma \vdash Q : \varphi}{\Gamma \vdash P Q : \psi}$	$\frac{\Gamma \vdash \varphi \rightarrow \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$

Установим соответствие и между прочими сущностями ИИВ и просто типизированного по Карри лямбда-исчисления.

Просто типизированное $\lambda$ -исчисление	Импликативный фрагмент ИИВ
Тип	Высказывание
Терм	Доказательство высказывания
Проверка того, что терм имеет заданный тип	Проверка доказательства на корректность
Обитаемый тип	Доказуемое высказывание
Проверка того, что существует терм, имеющий заданный тип	Проверка того, что заданное высказывание имеет доказательство

## 3 Лекция 4

### 3.1 Расширение просто типизированного $\lambda$ -исчисления до изоморфного ИИВ

Заметим, что между просто типизированным по Карри  $\lambda$ -исчислением и импликационным фрагментом ИИВ существует изоморфизм, но при этом в просто типизированном  $\lambda$ -исчислении нет аналогов лжи, а также связок  $\vee$  и  $\&$ .

Для установления полного изоморфизма между ИИВ и просто типизированным  $\lambda$ -исчислением введём три необходимые для установления этого изоморфизма сущности:

1. Тип "Ложь" ( $\perp$ )
2. Тип упорядоченной пары  $A \& B$ , соответствующий логическому "И"
3. Алгебраический тип  $A | B$ , соответствующий логическому "ИЛИ"

**Тип  $\perp$**  Введём тип  $\perp$ , соответствующий лжи в ИИВ. Поскольку из лжи может следовать что угодно, добавим в исчисление новое правило вывода

$$\frac{\Gamma \vdash A : \perp}{\Gamma \vdash A : \tau}$$

То есть выражение, типизированное как  $\perp$ , может быть типизировано так же любым другим типом.

В программировании аналогом этого типа может являться тип `Nothing`, который является подтипом любого другого типа.

Тип `Nothing` является необитаемым, им типизируется выражение, никогда не возвращающее свой результат (например, `throw new Error() : Nothing`).

Тот факт, что выражение, типизированное как `Nothing`, может быть типизировано любым другим типом, позволяет писать следующие функции:

```
def assertStringNotEmpty(s: String): String = {
  if (s.length != 0) {
    s
  } else {
    throw new Error("Empty string")
  }
}
```

так как `throw new Error("Empty string"): Nothing`, то

`throw new Error("Empty string"): String`, поэтому функция может иметь тип `String`.

Теперь, имея тип  $\perp$ , можно ввести связку "Отрицание". Обозначим  $\neg A = A \rightarrow \perp$ , то есть в программировании это будет соответствовать функции

```
def throwError(a: A) = throw new Error()
```

**Упорядоченные пары** Введём возможность запаковывать значения в пары. Функция `makePair` будет выглядеть следующим образом:

$$makePair = \lambda first. \lambda second. \lambda f. f \ first \ second$$

Тогда

$$< first, second > = makePair \ first \ second$$

Надо также написать функции, которые будут доставать из пары упакованные в неё значения. Назовём их  $\Pi_1$  и  $\Pi_2$ .

Пусть

$$\Pi_1 = \lambda Pair. Pair \ (\lambda a. \lambda b. a)$$

$$\Pi_2 = \lambda Pair. Pair \ (\lambda a. \lambda b. b)$$

Заметим, что

$$\begin{aligned} & \Pi_1 < A, B > \\ &=_{\beta} (\lambda Pair. Pair \ (\lambda a. \lambda b. a))(makePair \ A \ B) \\ &=_{\beta} (\lambda Pair. Pair \ (\lambda a. \lambda b. a))((\lambda first. \lambda second. \lambda f. f \ first \ second) \ A \ B) \\ &=_{\beta} (\lambda Pair. Pair \ (\lambda a. \lambda b. a))((\lambda second. \lambda f. f \ A \ second) \ B) \\ &=_{\beta} (\lambda Pair. Pair \ (\lambda a. \lambda b. a))(\lambda f. f \ A \ B) \\ &=_{\beta} (\lambda f. f \ A \ B) (\lambda a. \lambda b. a) \\ &=_{\beta} (\lambda a. \lambda b. a) \ A \ B \\ &=_{\beta} (\lambda b. A) \ B \\ &=_{\beta} A \end{aligned}$$

Аналогично,  $\Pi_2 < A, B > =_{\beta} B$

Таким образом, мы умеем запаковывать элементы в пары и доставать элементы из пар. Теперь, добавим к просто типизированному  $\lambda$ -исчислению правила вывода, позволяющие типизировать такие конструкции.

Добавим три новых правила вывода:

1. Правило типизации пары

$$\frac{\Gamma \vdash A : \varphi \quad \Gamma \vdash B : \psi}{\Gamma \vdash < A, B > : \varphi \&\psi}$$

2. Правило типизации первого проектора:

$$\frac{\Gamma \vdash < A, B > : \varphi \&\psi}{\Gamma \vdash \Pi_1 < M, N > : \varphi}$$

3. Правило типизации второго проектора:

$$\frac{\Gamma \vdash < A, B > : \varphi \&\psi}{\Gamma \vdash \Pi_2 < M, N > : \psi}$$

**Алгебраические типы** Добавим тип, который является аналогом `union` в C++, или алгебраического типа в любом функциональном языке. Это тип, который может содержать одну из двух альтернатив.

Например, тип `OptionInt = None | Some of Int` может содержать либо `None`, либо `Some of Int`, но не обе альтернативы разом, причём в каждый момент времени известно, какую альтернативу он содержит.

Заметим, что определение алгебраического типа похоже на определение дизъюнкции в ИИВ (в ИИВ если выполнено  $\vdash a \vee b$ , известно, что из  $\vdash a$  и  $\vdash b$  выполнено).

Для реализации алгебраических типов в  $\lambda$ -исчислении напомним три функции:

1.  $in_1$ , создающее экземпляр алгебраического типа из первой альтернативы, то есть запаковывающее первую альтернативу в алгебраический тип
2.  $in_2$ , выполняющее аналогичные действия, но со второй альтернативой.
3.  $case$ , принимающую три параметра: экземпляр алгебраического типа, функцию, определяющую, что делать, если этот экземпляр был создан из первой альтернативы (то есть с использованием  $in_1$ ), и функцию, определяющую, что делать, если этот экземпляр был создан из второй альтернативы (то есть с использованием  $in_2$ )

Аналогом  $case$  в программировании является конструкция, известная как `pattern-matching`, или сопоставление с образцом.

```
let isEmptyList list = match list with
  | Nil -> true
  | Cons(_, _) -> false
;;
```

Функция  $in_1$  будет выглядеть следующим образом:

$$in_1 = \lambda x. \lambda f. \lambda g. f x$$

А  $in_2$  - следующим:

$$in_2 = \lambda x. \lambda f. \lambda g. g x$$

То есть  $in_1$  принимает две функции, и применяет первую к  $x$ , а  $in_2$  применяет вторую. Тогда  $case$  будет выглядеть следующим образом:

$$case = \lambda algebraic. \lambda f. \lambda g. algebraic f g$$

Заметим, что

$$\begin{aligned}
& case (in_1 A) F G \\
&=_{\beta} (\lambda algebraic. \lambda f. \lambda g. algebraic f g) ((\lambda x. \lambda h. \lambda s. h x) A) F G \\
&=_{\beta} (\lambda algebraic. \lambda f. \lambda g. algebraic f g) (\lambda h. \lambda s. h A) F G \\
&=_{\beta} (\lambda f. \lambda g. (\lambda h. \lambda s. h A) f g) F G \\
&=_{\beta} (\lambda g. (\lambda h. \lambda s. h A) F g) G \\
&=_{\beta} (\lambda h. \lambda s. h A) F G \\
&=_{\beta} (\lambda s. F A) G \\
&=_{\beta} F A
\end{aligned}$$



Аналогично,  $case (in_2 B) F G =_\beta G B$ .

То есть  $case$ ,  $in_1$  и  $in_2$  умеют применять нужную функцию к запакованной в экземпляр алгебраического типа одной из альтернатив.

Теперь добавим к просто типизированному  $\lambda$ -исчислению правила вывода, позволяющие типизировать эти конструкции.

Добавим три новых правила вывода:

1. Правило типизации левой инъекции

$$\frac{\Gamma \vdash A : \varphi}{\Gamma \vdash in_1 A : \varphi \vee \psi}$$

2. Правило типизации правой инъекции:

$$\frac{\Gamma \vdash B : \psi}{\Gamma \vdash in_2 B : \varphi \vee \psi}$$

3. Правило типизации  $case$ :

$$\frac{\Gamma \vdash L : \varphi \vee \psi, \quad \Gamma \vdash f : \varphi \rightarrow \tau, \quad \Gamma \vdash g : \psi \rightarrow \tau}{case L f g : \tau}$$