

# Теория типов

Михайлов Максим

11 декабря 2021 г.

# Оглавление

Лекция 1	7 сентября	4
1	Лямбда-исчисление . . . . .	4
1.1	Определение . . . . .	4
1.2	Булево исчисление . . . . .	4
1.3	Числа . . . . .	5
1.4	Типизированное лямбда-исчисление . . . . .	6
1.5	Y-комбинатор и противоречивость нетипизированного $\lambda$ -исчисления .	6
Лекция 2	14 сентября	8
2	Формализация $\lambda$ -исчисления . . . . .	8
Лекция 3	21 сентября	12
3	Просто-типизированное $\lambda$ -исчисление . . . . .	12
3.1	Исчисление по Карри . . . . .	13
3.2	Исчисление по Чёрчу . . . . .	14
Лекция 4	28 сентября	15
3.3	Противоречивость нетипизированного $\lambda$ -исчисления . . . . .	15
4	Изоморфизм Карри-Ховарда . . . . .	16
4.1	Импликационный фрагмент ИИВ . . . . .	16
Лекция 5	5 октября	19
5	Алгебраические термы . . . . .	19
5.1	Эквивалентность уравнений и систем . . . . .	20
5.2	Алгоритм унификации . . . . .	21
5.3	Вывод типов в $\lambda_{\rightarrow}$ . . . . .	21
5.3.1	Построение уравнений . . . . .	22
5.3.2	Разрешение системы . . . . .	22
6	Исчисление предикатов второго порядка . . . . .	22
Лекция 6	12 октября	24
7	Абстрактные типы данных . . . . .	24
Лекция 7	19 октября	25
8	Типовая система Хиндли-Милнера . . . . .	25
8.1	Алгоритм $W$ . . . . .	27
Лекция 8	26 октября	29
9	$\lambda$ -куб . . . . .	29
9.1	Обобщенные типовые системы . . . . .	29
Лекция 9	2 ноября	32

---

10 Гомотопическая теория типов . . . . .	32
Лекция 10 9 ноября	35
11 Равенство . . . . .	35
Лекция 11 15 ноября	38
12 Классы . . . . .	39
Лекция 12 23 ноября	41
13 Язык . . . . .	41
13.1 Проблема . . . . .	41
13.2 Естественность проблемы . . . . .	41
13.3 Наивное решение проблемы . . . . .	42
13.4 Предикативность . . . . .	42
14 Аксиома выбора . . . . .	44

# Лекция 1

## 7 сентября

### 1 Лямбда-исчисление

То, чем мы будем заниматься, можно назвать прикладной матлогикой.

В рамках курса матлогики мы рукомахательно рассмотрели изоморфизм Карри-Ховарда, в этом курсе мы его формализуем. Мы затронем систему типов Хиндли-Милнера (*Haskell*) и язык Arend, основанный на гомотопической теории типов.

#### 1.1 Определение

В 20-30х годах XX века Алонзо Чёрчем была создана альтернатива теории множеств как основе математики — лямбда-исчисление. Основная идея — выбросить из языка все, кроме вызова функций.

В лямбда исчислении есть три конструкции:

- Функция (*абстракция*):  $(\lambda x. A)$
- Применение функции (*аппликация*):  $(A B)$
- Переменная (*атом*):  $x$

Большими буквами начала латинского алфавита мы будем обозначать термы, малыми буквам конца — переменные.  $\lambda$  жадная, как  $\forall$  и  $\exists$  в исчислении предикатов. Аппликация идёт слева направо, т.е.  $\lambda p. p F T = \lambda p. ((p F) T)$

Вычисление происходит с помощью  $\beta$ -редукции, его мы определим позже, общее понимание у нас есть из вводной лекции функционального программирования.

#### 1.2 Булево исчисление

Определим булево исчисление в  $\lambda$ -исчислении:

- $T := \lambda x. \lambda y. x$  — истина
- $F := \lambda x. \lambda y. y$  — ложь
- $\text{Not} := \lambda p. p \ F \ T$

$$\begin{aligned} \text{Not } F &\rightarrow_{\beta} \\ ((\lambda x. \lambda y. y) \ F) \ T &\rightarrow_{\beta} \\ (\lambda y. y) \ T &\rightarrow_{\beta} T \end{aligned}$$

- $\text{And} := \lambda a. \lambda b. a \ b \ F$

And берёт свой второй аргумент, если первый аргумент истина и ложь иначе.

And использует идею **карринга** — функция от 2 аргументов есть функция от первого аргумента, возвращающая другую функцию от второго аргумента.<sup>1</sup> Например, в выражении “((+) 2) 3” ((+) 2) это функция, которая прибавляет к своему аргументу 2.

### 1.3 Числа

Числа в лямбда-исчислении кодируются **нумералами Чёрча**. Это только один из способов кодировки, есть и другие. Общая идея — число  $n$  применяет данную функцию к данному аргументу  $n$  раз.

- $0 = \lambda f. \lambda x. x$
- $1 = \lambda f. \lambda x. f \ x$
- $3 = \lambda f. \lambda x. f \ (f \ (f \ x))$
- $\overline{n+1} = \lambda f. \lambda x. f \ (\overline{n} \ f \ x)$
- $(+1) = \lambda n. \lambda f. \lambda x. n \ f \ (f \ x)$  — функция инкремента.
- $(+) = \lambda a. \lambda b. b \ ((+) \ \overline{1}) \ a$ :  $b$  раз прибавляет единицу к  $a$ .
- $(\cdot) = \lambda a. \lambda b. a \ ((+) \ b) \ \overline{0}$ :  $a$  раз прибавляет  $b$  к 0.

Ходят легенды, что Клини изобрел декремент у зубного врача под действием наркоза. Существует много способов определить декремент различных степеней упоротости.

Рассмотрим декремент, основанный на следующей идее: пусть есть упорядоченная пара  $\langle a, b \rangle$  и функция  $(*) : \langle a, b \rangle \mapsto \langle b, b+1 \rangle$ . Тогда применив  $(*)$   $n$  раз к  $\langle 0, 0 \rangle$  и взяв первый элемент, возьмём первый элемент пары.

<sup>1</sup> Аналогично для  $n$  аргументов.

Упорядоченная пара определяется следующим способом:

$$\text{MkPair} = \lambda a. \lambda b. (\lambda p. p \ a \ b)$$

Можно потрогать эмулятор лямбда-исчисления lci, будет полезно для домашних заданий.

## 1.4 Типизированное лямбда-исчисление

Лямбда-исчисление для нас будет просто языком программирования. Для начала мы его типизируем, потому что нетипизированное лямбда-исчисление противоречиво.

Пусть у каждого выражения  $A$  есть тип  $\tau$ , что обозначается  $A : \tau$ . Также используется некоторый контекст с переменными и их типами, обозначаемый  $M$ . Все вместе это записывается как  $M \vdash A : \tau$ , что напоминает исчисление предикатов.

## 1.5 $Y$ -комбинатор и противоречивость нетипизированного $\lambda$ -исчисления

Мы хотим, чтобы  $\rightarrow_\beta$  сохраняло значения, т.к. иначе мы вообще не можем говорить о равенстве термов.

**Определение.**  $Y := \lambda f. (\lambda x. f \ (x \ x)) (\lambda x. f \ (x \ x))$  —  $Y$ -комбинатор, для него верно  $Y f \approx f(Y f)$ . Такое свойство называется “быть комбинатором неподвижной точки”, т.е. он находит неподвижную точку функции:  $A$  такое, что  $f(A) = A$ .

Пусть мы добавили бинарную операцию  $(\supset)$  — импликацию с некоторыми аксиомами. Оказывается, что доказуемо любое  $A$ . Мы это докажем на последующих лекциях.

$Y$ -комбинатор полезен тем, что позволяет реализовывать рекурсию.

*Пример.* Запишем факториал в неформальном виде:

$$\text{Fact} = \lambda n. \text{If} \ (\text{IsZero } n) \ \bar{1} \ (\text{Fact } (n - 1) \cdot n)$$

На самом деле  $\text{Fact}$  есть неподвижная точка функции

$$\lambda f. \lambda n. \text{If} \ (\text{IsZero } n) \ \bar{1} \ (f \ (n - 1) \cdot n)$$

по определению неподвижной точки функции. Тогда  $\text{Fact}$  это

$$Y(\lambda f. \lambda n. \text{If} \ (\text{IsZero } n) \ \bar{1} \ (f \ (n - 1) \cdot n))$$

У нас появляется проблема: есть выражения, которым мы не можем приписать значение, например

$$Y(\lambda f. \lambda x. f \ (\text{Not } x))$$

Эта проблема происходит из-за того, что наш язык слишком мощный — мы написали решатель любых уравнений, даже тех, у которых нет решения. Логичный выход из этой ситуации — запретить то, из-за чего у нас возникают проблемы. Как запретить  $Y$ ? Оказывается, это позволяют сделать типы — они будут делить выражения на добропорядочные и недобропорядочные.

# Лекция 2

## 14 сентября

### 2 Формализация $\lambda$ -исчисления

**Определение.** Пред- $\lambda$ -терм определяется индуктивно как одно из:

1.  $x$  — переменная
2.  $(L\ L)$  — применение
3.  $(\lambda x.L)$  — абстракция

Почему пред- $\lambda$ -терм? Мы не хотим различать  $\lambda x.x$  и  $\lambda y.y$ .

**Определение.**  $\alpha$ -эквивалентность — обозначается  $A =_\alpha B$  и выполняется, если<sup>1</sup>:

1.  $A \equiv x, B \equiv x$  — одна и та же переменная
2.  $A \equiv P\ Q, B \equiv R\ S, P =_\alpha R, Q =_\alpha S$
3.  $A \equiv \lambda x.P, B \equiv \lambda y.Q$  и существует  $t$  — новая переменная, такая что  $P[x := t] =_\alpha Q[y := t]$

**Определение.** Свобода для подстановки:  $A[x := B]$ , никакое свободное вхождение переменной в  $B$  не станет связанным.

**Определение** ( $\lambda$ -терм). Множество всех  $\lambda$ -термов это  $\Lambda / =_\alpha$

**Определение** ( $\beta$ -редекс). Выражение вида  $(\lambda x.A)\ B$

**Определение** ( $\beta$ -редукция). Обозначается  $A \rightarrow_\beta B$  и выполняется, если выполняется одно из:

1.  $A \equiv P\ Q, B \equiv R\ S$  и либо  $P \rightarrow_\beta R$  и  $Q =_\alpha S$ , либо  $P =_\alpha R$  и  $Q \rightarrow_\beta S$ .

---

<sup>1</sup> И только если.



2.  $A \equiv \lambda x.P, B \equiv \lambda x.Q$  и  $P \rightarrow_\beta Q$

3.  $A \equiv (\lambda x.P) Q, B \equiv P[x := Q]$  и  $Q$  свободно для подстановки.

**Определение.** Придуман Моисеем Шейнфинкелем.

$I := \lambda x.x$  — Identität<sup>2</sup>

**Определение.**

- $K = \lambda x.\lambda y.x$
- $\Omega = \omega \omega$
- $\omega = \lambda x.x x$

*Пример.*



**Определение.**  $R$  обладает ромбовидным свойством (*diamond*), если для любых  $a, b, c$ , таких что:

1.  $a R b, a R c$
2.  $b \neq c$

существует  $d$ :  $b R d$  и  $c R d$ .



*Пример.*  $>$  на  $\mathbb{Z}$  не ромбовидно: для  $a = 3, b = 2, c = 1$  выполнено условие, но  $\nexists d$ .

$>$  на  $\mathbb{R}$  ромбовидно.

---

<sup>2</sup> Тождество (с немецкого)

**Определение** ( $\beta$ -редуцируемость). Рефлексивное, транзитивное замыкание отношения  $\rightarrow_\beta$ , обозначается  $\rightarrow_\beta^*$ .

**Теорема 1** (Чёрча-Россера).  $\beta$ -редуцируемость обладает ромбовидным свойством.

**Определение.**  $\Rightarrow_\beta$  — параллельная  $\beta$ -редукция, выполняется если:

0.  $A =_\alpha B$
1.  $A \equiv P Q, B \equiv R S$  и  $P \Rightarrow_\beta R$  и  $Q \Rightarrow_\beta S$ .
2. Аналогично  $\beta$ -редукции.
3. Аналогично  $\beta$ -редукции.

**Лемма 1.**  $(\Rightarrow_\beta)$  обладает ромбовидным свойством.

**Лемма 2.** Если  $R$  обладает ромбовидным свойством, то  $R^*$  обладает ромбовидным свойством.

*Доказательство.* Две индукции. □

**Лемма 3.**  $(\Rightarrow_\beta) \subseteq (\rightarrow_\beta)$

*Доказательство теоремы Чёрча-Россера.* Заметим, что:

1.  $(\Rightarrow_\beta)^* \subseteq (\rightarrow_\beta)$  — из леммы
  2.  $(\rightarrow_\beta) \subseteq (\Rightarrow_\beta)^*$  — из определения
  3. Т.к.  $(\Rightarrow_\beta)^*$  обладает р.с., то и  $(\rightarrow_\beta)$  обладает р.с.
- 

**Следствие 1.1.** У  $\lambda$ -выражения существует не более одной нормальной формы.

*Доказательство.* Пусть  $A$  имеет две нормальные формы:  $A \rightarrow_\beta B, A \rightarrow_\beta C$  и  $B \neq_\alpha C$ . Тогда есть  $D: B \rightarrow_\beta D$  и  $C \rightarrow_\beta D$ . Противоречие. □

**Определение.** Нормальный порядок редукции — редуцируем самый левый редекс.

**Теорема 2.** Если нормальная форма существует, она может быть получена нормальным порядком редукции.

*Примечание.* Нижеследующее объяснение — с практики.

Рассмотрим  $Y f =_\beta f (Y f) =_\beta f (f (Y f)) =_\beta \dots$ . Можно считать, что у  $f$  сколько угодно аргументов, первый аргумент можно считать указателем на свой рекурсивный вызов.

*Пример.* Числа Фибоначчи:

```
fib a b n =  
  if n = 0 then a  
  else fib b (a + b) (n - 1)
```

Здесь решение уравнения заматано под ковер, в  $\lambda$ -исчислении оно видно:

$$\text{Fib} = \lambda f. \lambda a. \lambda b. \lambda n. (\text{IsZero } n) \ a \ (f \ f \ a \ (a + b) \ (n - 1))$$

Здесь  $f$  передается само себе, чтобы иметь ссылку на себя для рекурсивного вызова.

Для работы Fib нужно дать его самому себе: Fib Fib 1 1 10.

# Лекция 3

## 21 сентября

В  $\lambda$ -исчислении можно сделать:

1. Целые числа, где  $\langle a, b \rangle \leftrightarrow a - b$
2. Рациональные числа в виде дробей
3. Матлогику?

Попытки сделать матлогику всегда приводили к парадоксам.

Оказывается, нельзя относиться к любому выражению как к логическому.

*Обозначение.*  $\supset$  — импликация

*Пример.* Рассмотрим комбинатор  $\Phi_A =_{\beta} A \supset \Phi_A$ . Это  $Y (\lambda f. \lambda a. a \supset f a)$ .

Добавим аксиому  $(A \supset (A \supset B)) \supset (A \supset B)$ . Если такой аксиомы нет, то теория грустная.

Мы также хотим, чтобы если  $X =_{\beta} Y$ , то  $X \supset Y$ .

Каким-то образом мы получим парадокс.

## 3 Просто-типизированное $\lambda$ -исчисление

**Определение** (типовые переменные).

- $\alpha, \beta, \gamma$  — атомарные
- $\tau, \sigma$  — составные

2 традиции:

1. Исчисление по Чёрчу

## 2. Исчисление по Карри

Мы сначала рассмотрим исчисление по Карри.

### 3.1 Исчисление по Карри

Типизация:  $\Gamma \vdash A : \tau, \Gamma = \{x_1 : \tau_1, x_2 : \tau_2 \dots\}$

Правила:

1.  $\frac{}{\Gamma, x_1 : \tau_1 \vdash x_1 : \tau_1} \text{Ax.}$
2.  $\frac{\Gamma \vdash A : \sigma \rightarrow \tau \quad \Gamma \vdash B : \sigma}{\Gamma \vdash A B : \tau}$
3.  $\frac{\Gamma, x : \tau \vdash A : \sigma}{\Gamma \vdash \lambda x. A : \tau \rightarrow \sigma}$

Пример.

$$\lambda f^{\alpha \rightarrow \alpha}. \lambda x^{\alpha}. f (f x) : (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha$$

Подгоним доказательство под результат:

$$\frac{\frac{\frac{f : \alpha \rightarrow \alpha \vdash f : \alpha \rightarrow \alpha}{f : \alpha \rightarrow \alpha, x : \alpha \vdash f (f x) : \alpha}}{f : \alpha \rightarrow \alpha \vdash \lambda x. f (f x) : \alpha \rightarrow \alpha}}{\lambda f. \lambda x. f (f x) : (\alpha \rightarrow \alpha) \rightarrow (\alpha \rightarrow \alpha)}$$

**Теорема 3.** Если  $\Gamma \vdash A : \tau$ , то любое подвыражение имеет тип.

*Доказательство.* По индукции по длине.

База. Это правило 1.

Переход. Пусть любое выражение длиной  $< n$  символов обладает искомым свойством. Покажем искомое для  $A : |A| = n$ . Рассмотрим варианты того, по какому правилу доказана типизируемость  $A$ :

1. Второе правило:  $B$  и  $C$  короче  $A$ , следовательно для них искомое верно.
2. Третье правило: аналогично для  $x, B$

□

**Теорема 4** (Subject reduction, о редукции). Если  $\Gamma \vdash A : \sigma$  и  $A \rightarrow_{\beta} B$ , то  $\Gamma \vdash B : \sigma$

Доказательство. Скучно.

Самая интересная часть: рассмотрим  $A \rightarrow_\beta B$ . Случаи:

1.  $\lambda x.A \rightarrow \lambda x.B$  — индукция
2.  $A B$  — индукция
3.  $(\lambda x.A) B \rightarrow A[x := B]$

По теореме о типизации подвыражений,  $(\lambda x^{\tau \rightarrow \sigma}.A^\sigma) B^\tau : \sigma$ . Кроме того, доказывается  $(A[x := B]) : \sigma$ .

□

**Лемма 4.** Если  $\Gamma, x : \tau \vdash A : \sigma, \Gamma \vdash B : \tau$ , то  $\Gamma \vdash A[x := B] : \sigma$

**Теорема 5 (Чёрча-Россера).** Если  $\Gamma \vdash M : \sigma$  и существуют  $N, P : M \twoheadrightarrow_\beta N, M \twoheadrightarrow_\beta P$ , то найдется такой  $S$ , что  $\Gamma \vdash S : \sigma$  и  $N \twoheadrightarrow_\beta S$  и  $P \twoheadrightarrow_\beta S$

## 3.2 Исчисление по Чёрчу

Язык:

- $x$  — переменная
- $A B$  — аппликация
- $\lambda x^\tau.P$  — абстракция

**Обозначение.** Когда нужно различить исчисления, будем писать  $\vdash_{\text{ч}}$  или  $\vdash_{\text{к}}$

**Теорема 6.** Если контекст  $\Gamma$  и выражение  $P$  типизируется, то  $\Gamma \vdash_{\text{ч}} P : \sigma$

*Пример.*

$$\vdash_{\text{к}} \lambda x.x : \alpha \rightarrow \alpha$$

$$\vdash_{\text{к}} \lambda x.x : \beta \rightarrow \beta$$

$$\vdash_{\text{ч}} \lambda x^\sigma.x : \sigma \rightarrow \sigma$$

# Лекция 4

## 28 сентября

### 3.3 Противоречивость нетипизированного $\lambda$ -исчисления

???

1. Логические выражения
2. Запрещенные выражения

$Y$  явно нехорошее выражение.  $\Phi_A =_\beta \Phi_A \supset A$

Добавим очевидные аксиомы:

1.  $A =_\beta B$ , то  $\vdash A \supset B, \vdash B \supset A$ . Почему? Потому что мы хотим, чтобы  $\sin 0 = 0$ , а не только  $\sin 0 \rightarrow 0$
2.  $(A \supset A \supset B) \supset (A \supset B)$
3.  $A, A \supset B$ , тогда  $B$

Тогда заметим, что при любом  $A, \vdash A$ :

$$\begin{array}{l}
 \Phi_A \supset \Phi_A \\
 \Phi_A \supset (\Phi_A \supset A) \\
 (\Phi_A \supset (\Phi_A \supset A)) \supset (\Phi_A \supset \Phi_A) \\
 \Phi_A \supset A \\
 \Phi_A \\
 A
 \end{array}$$

## 4 Изоморфизм Карри-Ховарда

$$\begin{array}{c}
 \frac{}{\Gamma, x : \tau \vdash x : \tau} \quad x \notin \Gamma \\
 \frac{}{\Gamma, x : \tau \vdash x : \tau} \quad x \notin \Gamma \\
 \frac{}{\Gamma, \tau \vdash \tau} \\
 \frac{\Gamma \vdash A : \sigma \rightarrow \tau \quad \Gamma \vdash B : \sigma}{\Gamma \vdash AB : \tau} \\
 \frac{\Gamma \vdash \sigma \rightarrow \tau \quad \Gamma \vdash \sigma}{\Gamma \vdash \tau} \\
 \frac{\Gamma, x : \sigma \vdash A : \tau}{\Gamma \vdash \lambda x. A : \sigma \rightarrow \tau} \quad x \notin \Gamma \\
 \frac{\Gamma, \sigma \vdash \tau}{\Gamma \vdash \sigma \rightarrow \tau}
 \end{array}$$

**Теорема 7** (об изоморфизме Карри-Ховарда).

1.  $\Gamma \vdash_{\lambda \rightarrow} A : \tau$ , то  $|\Gamma| \vdash_{\rightarrow} \tau$
2. Если  $\Delta \vdash_{\rightarrow} \tau$ , то найдутся  $\Gamma, A : |\Gamma| = \Delta, \Gamma \vdash A : \tau$

**Определение.**

$$|\Gamma| := \{\tau \mid x : \tau \in \Gamma\}$$

### 4.1 Импликационный фрагмент ИИВ

У нас есть только импликация.

Есть три правила:  $I_{\rightarrow}, E_{\rightarrow}, \text{Ax}$ . Будет ли у нас всё работать? Утверждается, что да.

*Обозначение.* Доказуемость в И.Ф. ИИВ будем обозначать  $\Gamma \vdash_{\rightarrow} \tau$

**Определение** (язык И.Ф. ИИВ).

$$\tau ::= \alpha \mid (\tau \rightarrow \tau)$$

**Теорема 8.** Импликационный фрагмент ИИВ замкнут относительно доказуемости: Если  $\Gamma \vdash \tau$  и  $\tau$  содержит только пропозиционные переменные и импликацию, то  $\Gamma \vdash_{\rightarrow} \tau$ . Обратное очевидно верно.

*Доказательство.* Рассмотрим  $\Gamma^*$  — множество формул, замкнутых по доказуемости.

$$\tau \in \Gamma^* \Leftrightarrow \Gamma^* \vdash \tau$$



*Обозначение.*  $\Gamma$  — множество формул, тогда  $\Gamma^*$  — замыкание этого множества по доказуемости, а  $\Gamma^{\rightarrow}$  — замыкание по доказуемости в ИФИИВ.

Рассмотрим множество миров:  $\Gamma^{\rightarrow} \preceq \Delta^{\rightarrow}$ , если  $\Gamma^{\rightarrow} \subseteq \Delta^{\rightarrow}$ ,  $\Delta^{\rightarrow}$  — замкн.,  $\Gamma^* \Vdash \tau$ , если  $\tau \in \Gamma^*$

*Утверждение.*  $\Gamma^*$  образует модель Крипке.

**Определение** (модель Крипке).

1. Множество миров, упорядоченных отношением  $\preceq$
2.  $\Vdash$  такое, что если  $\Gamma \Vdash \alpha$ , то  $\Gamma \preceq \Delta$ , то  $\Delta \Vdash \alpha$ .

Тогда  $\Gamma \Vdash \tau \rightarrow \sigma$  тогда и только тогда, когда в любом  $\Gamma \preceq \Delta$  из  $\Delta \Vdash \tau$ , следует  $\Delta \Vdash \sigma$ .

*Утверждение.*  $\tau \in \Gamma^{\rightarrow}$  тогда и только тогда, когда  $\Gamma^{\rightarrow} \vdash_{\text{и}} \tau$

*Доказательство.* Индукция по структуре  $\tau$ .

База.  $\tau \equiv \alpha$

$$\Rightarrow \alpha \in \Gamma^{\rightarrow}, \text{ то } \alpha \vdash_{\text{и}} \alpha$$

$$\Leftarrow \alpha \vdash_{\text{и}} \alpha, \text{ тогда очевидно } \alpha \in \Gamma^{\rightarrow}$$

Переход.  $\tau \equiv \delta \rightarrow \pi$

$$\Rightarrow \sigma \rightarrow \pi \in \Gamma^{\rightarrow}, \text{ то } \Gamma^{\rightarrow} \vdash_{\rightarrow} \sigma \rightarrow \pi$$

$$\Leftarrow \Gamma^{\rightarrow} \vdash_{\text{и}} (\sigma \rightarrow \pi). \text{ Значит, } \Gamma^{\rightarrow} \Vdash \sigma \rightarrow \pi.$$

□

Рассмотрим  $\Gamma^{\rightarrow} \preceq \Delta$  :  $\Delta \Vdash \sigma$ , то  $\Delta \Vdash \pi$ . Значит,  $\Delta \Vdash \sigma$ . Значит,  $\sigma \in \Delta$ , т.е.  $\Delta \vdash_{\rightarrow} \sigma$ . Значит,  $\Delta \vdash_{\rightarrow} \pi$  по М.Р., т.к.  $\Gamma^{\rightarrow} \Vdash \sigma \rightarrow \pi \Rightarrow \Delta \Vdash \sigma \rightarrow \pi \Rightarrow \Delta \vdash_{\rightarrow} \sigma \rightarrow \pi$

*Утверждение.*  $\Gamma \Vdash \tau \Leftrightarrow \Gamma|_{\rightarrow} \tau$

*Доказательство.*

$$\Rightarrow \Gamma \Vdash \tau.$$

$$1. \Gamma \Vdash \alpha, \text{ т.е. } \alpha \in \Gamma, \text{ т.е. } \alpha \vdash_{\rightarrow} \alpha$$

$$2. \Gamma \Vdash \sigma \rightarrow \pi.$$

Рассмотрим  $\Gamma \preceq \Delta$ , причём  $\Delta \Vdash \sigma$ , тогда  $\Delta \Vdash \pi$ . Т.е. по индукционному предположению  $\Delta \vdash_{\rightarrow} \sigma$ . Пусть  $\Delta = \{\Gamma, \sigma\}^*$ . Тогда  $\Gamma, \sigma \vdash_{\rightarrow} \pi$ .

Тогда  $\Gamma, \sigma \Vdash \pi$  по индукционному предположению и определению  $\Vdash$ . Тогда  $\Gamma, \sigma \vdash_{\rightarrow} \pi$ , т.е.  $\Gamma \vdash_{\rightarrow} \sigma \rightarrow \pi$

$\Leftarrow \Gamma \vdash_{\rightarrow} \tau$ , тогда  $\Gamma \Vdash \tau$

1.  $\tau \equiv \alpha$  — очевидно.
2.  $\tau \equiv \sigma \rightarrow \pi$ . Дано, что  $\Gamma_{\rightarrow} \vdash \sigma \rightarrow \pi$ .

Пусть  $\Delta \Vdash \sigma$ .  $\Gamma \preceq \Delta$ . Тогда  $\Delta \vdash_{\rightarrow} \sigma$  по индукционному предположению.  $\Gamma \vdash_{\rightarrow} \sigma \rightarrow \pi$ , т.е.  $\Delta \vdash_{\rightarrow} \sigma \rightarrow \pi$ . По М.Р.  $\Delta \vdash_{\rightarrow} \pi$ . По индукционному предположению  $\Delta \Vdash \pi$ . Т.е.  $\Gamma \Vdash \sigma \rightarrow \pi$ . **В лекции было  $\models$ .**

□

Схема доказательства:

1.  $\tau \in \Gamma^*$ , если  $\Gamma^* \vdash_{\text{и}} \tau$
2.  $\Gamma^* \Vdash \tau$
3.  $\Gamma^* \Vdash \tau$  тогда и только тогда, когда  $\Gamma^* \vdash_{\rightarrow} \tau$

□

Обозначение.  $\lambda_{\rightarrow}$  — типизированное  $\lambda$ -исчисление.

1. Обитаемость:  $\overset{?}{\Gamma} \vdash ? : \tau$  — по изоморфизму Карри-Ховарда и теореме об эквивалентности  $\Gamma \vdash \tau$
2. Вывод (реконструкция):  $\Gamma \vdash A : ?$
3. Проверка:  $\Gamma \vdash A : \tau$

Пункты 2 и 3 это одно и то же.

# Лекция 5

## 5 октября

### 5 Алгебраические термы

Определение (алгебраические термы).

$$T ::= \underbrace{a}_{\text{переменная}} \mid \underbrace{f}_{\substack{\text{функциональный} \\ \text{символ}}} T_1 \dots T_n$$

Функциональные символы  $\in F$ , переменные  $\in T$

Пример.  $f (f_2 a b) c$

**Определение.** Подстановка переменных — отображение  $S_0 : V \rightarrow T$ , являющееся тождественным почти всюду<sup>1</sup>, то есть  $\exists$  фиксированные  $a_1 \dots a_n$ , для которых  $S_0$  не тождественна:  $S_0(a_i) = T_i$ , а для  $b \notin \{a_i\}$   $S_0(b) = b$ .

Тогда можно определить определить подстановку  $S : T \rightarrow T$ :

$$S(f T_1 \dots T_n) = f (S(T_1)) (S(T_2)) \dots (S(T_n))$$

$$S(a) = S_0(a)$$

**Определение.** Рассмотрим уравнение  $T_1 = T_2$ . Его **решение** — такая подстановка  $S$ , что  $S(T_1) \equiv S(T_2)$ , где  $\equiv$  обозначается равенство строк.

Пример.

$$\begin{aligned} f a (g b) &= f (g c) d \\ S_0(a) &= g c \quad S_0(d) = g b \\ S(f a (g b)) &= f (g c) (g b) \end{aligned}$$

<sup>1</sup> Кроме конечного количества.

**Определение** (система уравнений).

$$\begin{cases} T_1 = P_1 \\ T_2 = P_2 \\ \vdots \\ T_n = P_n \end{cases}$$

### 5.1 Эквивалентность уравнений и систем

**Определение.** Две системы  $E_1 : \begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$  и  $E_2 : \begin{cases} T'_1 = P'_1 \\ \vdots \\ T'_n = P'_n \end{cases}$  называются эквивалентными, если любое решение системы  $E_1$  подходит к  $E_2$  и наоборот.

*Утверждение.* Для любой системы существует эквивалентное уравнение.

*Доказательство.* Выберем новый  $n$ -местный функциональный символ  $h$ , построим уравнение  $h T_1 \dots T_n = h P_1 \dots P_n$ .  $\square$

**Определение.**

$$(U \circ T)(P) = U(T(P))$$

**Определение.** Определим порядок на подстановках.  $S \preceq T$ , если  $S$  — частный случай  $T$ , т.е.  $\exists U: S = U \circ T$

**Определение.** Наиболее общим решением уравнения  $T = P$  назовём подстановку  $S$ , такую что  $S(T) = S(P)$  и для любой  $S_1: S_1(T) \equiv S_1(P)$  выполнено  $S_1 \preceq S$

**Теорема 9.** У уравнения в алгебраических термах  $T = P$  всегда есть наиболее общее решение, если есть хоть какое-то.

**Определение.** Несовместная система — система с уравнениями вида  $f T_1 \dots T_n = g P_1 \dots P_k$ , где  $f \not\equiv g$ , либо  $x = \dots x \dots$

В `OCaml` и `Haskell` это называется “occurs check”.

**Определение.** Система в разрешённой форме — система вида  $\begin{cases} a_1 = T_1 \\ \vdots \\ a_n = T_n \end{cases}$ , где:

1. Все  $a_i$  различны
2.  $T_i$  не содержит  $a_j$  для  $i \neq j$

Альтернативное определение — каждый  $a_i$  входит по одному разу.

## 5.2 Алгоритм унификации

Рассмотрим систему  $\begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$

Применяем одно из следующих:

1.  $x = x$  — отбрасываем.
2.  $T = x$ , где  $T \neq x$ , тогда заменяем на  $x = T$

$$3. \begin{cases} x = P \\ \vdots \\ T_2 = P_2 \\ \vdots \\ T_n = P_n \end{cases} \Rightarrow \begin{cases} T_2[x := P] = P_2[x := P] \\ \vdots \\ T_n[x := P] = P_n[x := P] \\ x = P \end{cases}$$

$$4. f T_1 \dots T_n = f P_1 \dots P_n \Rightarrow \begin{cases} T_1 = P_1 \\ \vdots \\ T_n = P_n \end{cases}$$

**Теорема 10.** Применяя шаги алгоритма унификации, за конечное время можно получить систему либо в разрешенной форме, либо несовместную.

*Доказательство.* Рассмотрим тройку  $\left\langle \begin{smallmatrix} \text{количество} \\ \text{неразрешенных} \\ \text{переменных} \end{smallmatrix}, \begin{smallmatrix} \text{максимальная} \\ \text{сложность} \\ \text{слева} \end{smallmatrix}, \begin{smallmatrix} \text{количество} \\ \text{уравнений} \\ \text{максимальной} \\ \text{сложности} \\ \text{слева} \end{smallmatrix} \right\rangle$ . Сложность — вложенность.

1. Применения правил уменьшают тройку.
2.  $\langle 0, 0, t \rangle$  — система в разрешенной форме.
3. Количество применений правил конечно, т.к. каждая из троек  $\in \omega^3$  и любая последовательность убывающих ординалов конечна.

□

## 5.3 Вывод типов в $\lambda_{\rightarrow}$

$(\rightarrow)$  — 2-местный функциональный символ.

Проведём индукцию по структуре  $\lambda$ -выражения. Результатом разбора будет пара  $\langle \text{система}, \text{тип} \rangle$

### 5.3.1 Построение уравнений

Структура	Комментарий	Система	Тип
$x$	Введём тип $\alpha_x$	$\emptyset$	$\alpha_x$
$A B$	Рекурсивный вызов на $A$ и $B$ даст $\langle E_A, \tau_A \rangle, \langle E_B, \tau_B \rangle$ . Вводим $\beta$ — новый тип.	$E_A, E_B, \tau_B \rightarrow \beta = \tau_A$	$\beta$
$\lambda x.A$	Рекурсивный вызов на $A$ даст $\langle E_A, \tau_A \rangle$ . Берём тип для $x : \alpha_x$ .	$E_A$	$\alpha_x \rightarrow \tau_A$

Несложно заметить, что эти правила соответствуют правилом вывода в типизированном  $\lambda$ -исчислении.

### 5.3.2 Разрешение системы

Это унификация.

*Пример.* Разберём  $B = \lambda x. \overbrace{x}^A$ .

- $E_A = \emptyset, \tau_A = \alpha_x$
- $E_B = \emptyset, \tau_B = \alpha_x \rightarrow \alpha_x$

Разрешим систему уравнений  $\tau_A, \tau_B$ . Оказывается, эта система уже в разрешенной форме. Таким образом,  $\vdash \lambda x.x : \alpha \rightarrow \alpha$ . Контекст пустой, т.к. свободных переменных нет ( $E_A, E_B = \emptyset$ ).

**Определение.** Терм называется **слабо-нормализуемым**, если существует последовательность  $\beta$ -редукций, приводящих его в нормальную форму.

**Определение.** Терм называется **сильно-нормализуемым**, если не существует бесконечной последовательности  $\beta$ -редукций, нигде не приводящая его в нормальную форму.

*Пример.*  $K I \Omega$  — слабо нормализуемый, но не сильно нормализуемый

**Теорема 11.**  $\lambda_{\rightarrow}$  сильно нормализуемо.

*Примечание.* Это сильно ограничивает выразительность  $\lambda_{\rightarrow}$ .

## 6 Исчисление предикатов второго порядка

Второй порядок — это когда переменные есть предикаты.

**Определение (предикат).**

$$\Phi_{\Pi} ::= p \mid \Phi_{\Pi} \cup \Phi_{\Pi} \mid \Phi_{\Pi} \& \Phi_{\Pi} \mid \Phi_{\Pi} \rightarrow \Phi_{\Pi} \mid \forall p. \Phi_{\Pi} \mid \exists p. \Phi_{\Pi}$$

*Утверждение.* Можно выразить:

$$a \& b ::= \forall p. (a \rightarrow b \rightarrow p) \rightarrow p$$

$$a \vee b ::= \forall p. (a \rightarrow p) \rightarrow (b \rightarrow p) \rightarrow p$$

$$\perp ::= \forall p. p$$

$$\exists p. A ::= \forall x. (\forall p. p \rightarrow x) \rightarrow x$$

Это исчисление также называется “Система F”, оно же  $L_2$ .

$$L_2 ::= x \mid \lambda x^\alpha. A \mid P Q \mid P \tau \mid \lambda \alpha. A$$

# Лекция 6

## 12 октября

### 7 Абстрактные типы данных

ООП = АДТ + наследование.

Пример (стек).

$\text{push} : \alpha \rightarrow \alpha \text{ stack} \rightarrow \alpha \text{ stack}$

$\text{pop} : \alpha \text{ stack} \rightarrow (\alpha \cdot \alpha \text{ stack})$

$\text{empty} : \alpha \text{ stack}$

Что мы понимаем под  $\exists \alpha. \varphi$ ?  $\varphi$  — интерфейс и существует где-то в природе тип, который этому интерфейсу соответствует.

Для стека:

$$\exists \alpha. \underbrace{(\eta \rightarrow \alpha \rightarrow \alpha)}_{\text{push}} \& \underbrace{(\alpha \rightarrow \alpha \& \eta)}_{\text{pop}} \& \underbrace{\eta}_{\text{empty}}$$

Правила вывода:

$$\frac{\Gamma \vdash \varphi[x := \theta]}{\exists x. \varphi}$$

$$\frac{\Gamma, \psi \vdash \varphi \quad \Gamma \vdash \exists x. \psi}{\Gamma \vdash \varphi} \quad x \notin \text{FV}(\Gamma)$$

$$\frac{\Gamma \vdash M : \sigma[\alpha := \tau]}{\Gamma \vdash \text{pack } \tau, M \text{ to } \exists \alpha. \sigma : \exists \alpha. \sigma}$$

TBD.



# Лекция 7

## 19 октября

### 8 Типовая система Хиндли-Милнера

Мы рассмотрели две системы типов:

1. Просто типизированное лямбда исчисление: недостаточно выразительно
2. Система F: местами выразительна, местами недостаточно. Кроме того, потеряна разрешимость.

Ограничим излишнюю свободу системы F.

**Определение (ранг типа).** Пусть  $\sigma$  — тип без кванторов.  $R \subset \text{тип} \times \mathbb{N}_0$ , такое что:

1.  $R(\sigma, 0)$
2. Если  $R(\tau, k)$ , то  $R(\forall \alpha. \tau, \max(k, 1))$
3. Если  $R(\tau_0, k)$  и  $R(\tau_1, k + 1)$ , то  $R(\tau_0 \rightarrow \tau_1, k + 1)$ .

*Пример.*

$$R(\alpha, 0) \Rightarrow R(\alpha, 5) \Rightarrow R(\forall \alpha. \alpha, 5)$$

$$R(\alpha \rightarrow \alpha, 0) \Rightarrow R(\forall \alpha. \alpha \rightarrow \alpha, 1)$$

**Определение (Типовая система Хиндли-Милнера).** Рассмотрим  $\lambda$ -исчисление 2 порядка по Карри.

Типы:

1. Типы без кванторов:  $\tau = \alpha \mid (\tau \rightarrow \tau)$
2. Типовые схемы:  $\sigma = \forall \alpha. \sigma \mid \tau$

**Определение** (Отношение “быть частным случаем” (специализация)).  $\sigma_1 \sqsubseteq \sigma_2$  ( $\sigma_2$  — частный случай  $\sigma_1$ ), если  $\sigma_1 \equiv \forall \alpha_1 \dots \forall \alpha_n. \tau_1$ ,  $\sigma_2 \equiv \forall \beta_1 \dots \forall \beta_k. \tau_2[\alpha_1 := \theta_1 \dots \alpha_n := \theta_n]$  и новые  $\beta_1 \dots \beta_k$  не входят свободно в  $\sigma_1$ .

*Пример.*  $\tau \sqsubseteq \text{string}$



*Пример.*

$$\forall \alpha. \alpha \rightarrow \alpha \sqsubseteq \forall \beta. (\beta \rightarrow \beta) \rightarrow (\beta \rightarrow \beta)$$

Правила вывода:

$$\begin{array}{c}
 \frac{}{\Gamma, x : \sigma \vdash x : \sigma} \quad \alpha \notin \text{FV}(\Gamma) \\
 \frac{\Gamma \vdash A : \tau \rightarrow \tau' \quad \Gamma \vdash B : \tau}{\Gamma \vdash A B : \tau'} \\
 \frac{\Gamma, x : \tau \vdash A : \tau'}{\Gamma : \lambda x. A : \tau \rightarrow \tau'} \\
 \frac{\Gamma \vdash A : \sigma \quad \Gamma, x : \sigma \vdash B : \tau}{\Gamma \vdash \text{let } x = A \text{ in } B : \tau} \\
 \frac{\Gamma \vdash A : \sigma' \quad \sigma' \sqsubseteq \sigma}{\Gamma \vdash A : \sigma} \\
 \frac{\Gamma \vdash A : \sigma \quad \alpha \notin \text{FV}(\Gamma)}{\Gamma \vdash A : \forall \alpha. \sigma} \\
 \text{let } x = A \text{ in } B \rightarrow_{\beta} B[x := A]
 \end{array}$$

Казалось бы, `let` похож на  $(\lambda x. B) A$ . Однако, мы разрешаем кванторы в  $A$ .

*Пример.*

$$\begin{array}{l}
 I \equiv \lambda x. x : \forall \alpha. \alpha \rightarrow \alpha \\
 \triangleleft (I \ 1, I \ \text{"a"}) \quad I \ 1 : \text{int}, I \ \text{"a"} : \text{string}
 \end{array}$$

1.

$$\text{let } \underbrace{I = \lambda x. x}_{I : \forall \alpha. \alpha \rightarrow \alpha} \text{ in } (I \ 1, I \ \text{"a"})$$

2. То же самое, но без let:

$$(\lambda i.(i\ 1, i\ "a")) (\lambda x.x)$$

В этом варианте тип внутри лямбды без кванторов. В силу этого операцию  $(i\ 1, i\ "a")$  сложно выполнить — нужно угадать, какой тип подставлять.

Эта система очевидно уже, чем System F. Мы её сузили, чтобы получить разрешимость.

## 8.1 Алгоритм $W$

Мы хотим решить  $? \vdash A : ?$ , т.е. найти контекст и тип выражения, притом наиболее общие.

$W(\Gamma, E) \Rightarrow (\tau, S)$  — по контексту и выражению получаем тип и подстановку.

1.  $E \equiv x, x \in \Gamma, x : \sigma_x = \forall \alpha_1 \dots \alpha_n. \tau$ . Тогда введём новые переменные  $\beta_1 \dots \beta_n$  и результатом будет:

$$W(\Gamma, E) = (\forall \beta_1 \dots \beta_n. \tau, \emptyset)$$

2.  $E \equiv \lambda x. P$ . Пусть  $W(\Gamma \cup \{x : \gamma\}, P) = (\tau_P, S_1)$ .

$$W(\Gamma, E) = (S_1(\gamma \rightarrow \tau_P), S_1)$$

3.  $E \equiv P\ Q$ . Введём новый тип  $\gamma$ . Пусть  $\mathcal{U}$  — вызов алгоритма унификации и:

$$W(\Gamma, P) = (\tau_P, S_1) \quad W(S_1\Gamma, Q) = (\tau_Q, S_2) \quad \mathcal{U}(S_2\tau_P, \tau_Q \rightarrow \gamma) = S_3$$

Тогда:

$$W(\Gamma, E) = (S_3\gamma, S_3 \circ S_2 \circ S_1)$$

4.  $E \equiv \text{let } x = P \text{ in } Q$ . Пусть:

$$W(\Gamma, P) = (\tau_P, S_1) \quad W(S_1\Gamma \cup \{x : \forall^1. \tau_f\}, Q) = (S_2, \tau_Q)$$

$$W(\Gamma, E) = (\tau_Q, S_2 \circ S_1)$$

*Пример.*

$$\text{let } I = \lambda x.x \text{ in } (I\ 1, I\ " ")$$

Согласно 4 пункту алгоритма:

$$I : \forall \alpha. \alpha \rightarrow \alpha \vdash (I\ 1, I\ " ")$$

Мы теряем полноту по Тьюрингу, т.к. это частный случай системы F. Мы такого не хотим, поэтому добавим чего-нибудь, что её нам даст.

<sup>1</sup> Кванторы по всем свободным в  $\tau_f$  переменным.

1. Правило для  $Y$ :

$$Y : \forall \alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha$$

Теория становится противоречивой — вместо  $\alpha \rightarrow \alpha$  всегда можно подставить  $\text{id}$  и получить любое  $\alpha$ .

2.  $\text{IntList} = (\text{int} \ \& \ \text{IntList}) \vee \text{Nil}$ . Это какое-то уравнение. Как его решить?

Введём  $\mu$ -оператор:  $\mu \eta. (\text{int} \ \& \ \eta) \vee \text{Nil}$  или в общем случае  $\mu \eta. \tau$  — тип, решающий уравнение  $\eta = \tau$

Есть две традиции решения таких уравнений:

1. Экви-рекурсивные:  $\mu$  существует как тип (Java).
2. Изо-рекурсивные: ищем  $\mu \eta. \tau(\eta)$  вводятся две операции

(a)  $\text{Roll}: \tau(\eta) \rightarrow \eta$

(b)  $\text{Unroll}: \eta \rightarrow \tau(\eta)$

Итого мы взяли систему  $F$  и:

1. Запретили типы с неповерхностными кванторами
2. Добавили  $\text{let}$ -полиморфизм
3. Добавили противоречивость через  $Y$ -комбинатор и решение уравнений на типах.

# Лекция 8

## 26 октября

Это последняя лекция, посвященная части “Матлогика в языках программирования”.  
Вторая часть — “Языки программирования в матлогике и математике”.

### 9 λ-куб

Мы упустили теорию первого порядка.

#### 9.1 Обобщенные типовые системы

$$\mathcal{F} ::= x \mid \underbrace{\mathcal{F} \mathcal{F}}_{\text{применение}} \mid \underbrace{\lambda x : \mathcal{F}. \mathcal{F}}_{\lambda\text{-абстракция}} \mid \Pi x : \mathcal{F}. \mathcal{F} \mid \underbrace{C}_{\text{константа}}$$

$$C ::= * \mid \square$$

Мы решили, что типы и выражения должны сосуществовать, например в C++ можно написать `Array<int, 23+7>`.

Обозначение.  $s :=$  множество  $(*, \square)$

$$\frac{}{\vdash * : \square} \text{ аксиома}$$

$$\frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A} \text{ начальное правило, } x \notin FV(\Gamma)$$

$$\frac{\Gamma \vdash \varphi : (\Pi x^A. B) : s \quad \Gamma \vdash a : A}{\Gamma \vdash (\varphi a) : (B[x := A])} \text{ применение}$$

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s \quad B =_{\beta} B'}{\Gamma \vdash A : B'} \text{ преобразование}$$

$$\frac{\Gamma \vdash B : C \quad \Gamma \vdash A : s}{\Gamma, x : A \vdash B : C} \text{ ослабление, } x \notin FV(\Gamma)$$

$*$  — тип,  $\square$  — тип типа.

*Пример.* `array [a..b] of T`. Можно рассматривать `array [a..b] of` как оператор над типами. Его тип  $* \rightarrow *$ . Это также называется не тип, а род.

*Примечание.*

$$\begin{array}{cccc} 7 & : & int & : & * & : & \square \\ \text{знач.} & & \text{тип} & & \text{род} & & \text{сорт} \\ \text{value} & & \text{type} & & \text{kind} & & \text{sort} \end{array}$$

Пусть  $S \subseteq C \times C$  параметризует типовую систему. Здесь и далее  $(s_1, s_2)$  пробегает все пары  $\in S$ .

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x^A. B) : s_2} \text{ П-правило}$$

$$\frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\lambda x^A. b) : (\Pi x^A. B)} \text{ λ-правило}$$

*Обозначение.* Будем писать  $\Pi x^\varphi. \pi$  вместо  $\varphi \rightarrow \pi$ , если  $x \notin FV(\pi)$ .

*Обозначение.*

$$x : y : z \Rightarrow x : y, y : z$$

*Примечание.* Пусть  $x \notin FV(B)$ . Тогда рассмотрим следующее правило:

$$\frac{\Gamma \vdash A : * \quad \Gamma, x : A \vdash b : B \quad \Gamma, x : A^1 \vdash B : *}{\Gamma \vdash (\lambda x^A. b) : A \rightarrow B}$$

А что если  $x \in FV(B)$ ? Тогда мы получаем **зависимый тип**.

*Пример.* `printf("%d", "a")` — так нельзя.

`printf : (x : string) → F(x)` — так мы не пишем, не сложилось по традиции. Мы будем писать  $\Pi x^{\text{string}}. F(x)$

`printf "%s" : string → string`

Рассмотрим  $S$  из определения.

---

<sup>1</sup> Можно убрать, т.к.  $x \notin FV(B)$

$S$	Название системы типов	Характерный представитель
$(*, *)$	$\lambda_{\rightarrow}$	$\lambda x.x$
$(*, *), (\square, *)$	$\lambda_{\rightarrow}$	$\Lambda \alpha. \lambda x^{\alpha}. x$
$(*, *), (\square, *)$	$\lambda \ \omega$ слабая	<code>Int[]</code>
$(*, *), (\square, *), (\square, \square)$	$\lambda \ \omega$	
$(*, *), (*, \square)$		<code>int[n]</code>
$(*, *), (*, \square), (\square, *), (\square, \square)$	$\lambda C$ : исчисление конструкций <sup>2</sup>	

Объектно-ориентированное программирование не описывается через  $S$ .

Дальше в лекции были примеры, которые не записаны.

---

<sup>2</sup> Языки доказательств

# Лекция 9

## 2 ноября

### 10 Гомотопическая теория типов

Первые полчаса лекции пропущены.

**Определение.** Путь между  $a$  и  $b$  в пространстве  $X$  — функция  $f : [0, 1] \rightarrow X$ ,  $f(0) = a$ ,  $f(1) = b$  и  $f$  непрерывно. Если есть путь из  $a$  в  $b$ , то мы считаем  $a$  и  $b$  равными.

**Определение.** Интервальный тип:  $I = [left, right]$

Почему не  $I = \{left, right\}$ ?

*Пример.* Докажем, что  $2 + 1 = 1 + 2$ . Рассмотрим  $f$  — потенциальный путь  $f(left) = 1 + 2$ ,  $f(right) = 2 + 1$

**Определение.** Отображение непрерывно, если<sup>1</sup> прообраз открытого множества открыт.

*Пример.*  $f : \{0, 1\} \rightarrow \{2, 3\}$ , дискретная топология. Непрерывна ли  $fx = x + 2$ ?

Открытые в  $\{2, 3\} : \{\}, \{2\}, \{3\}, \{2, 3\}$

$$f^{-1}(\{\}) = \{\}, f^{-1}(\{2\}) = \{0\}, f^{-1}(\{3\}) = \{1\}, f^{-1}(\{2, 3\}) = \{0, 1\}$$

Таким образом  $f$  непрерывна.

*Пример.*  $\lceil f : \mathbb{R} \rightarrow \mathbb{R}, f(x) = \lfloor x \rfloor$

$f^{-1}((0.1, 0.2)) = \emptyset$ , что не является открытым множеством.

Кроме того,  $f^{-1}([0, 5]) = \{0, 1, 2, 3, 4, 5\}$

---

<sup>1</sup> И только если.



**Определение.** Пространство  $X$  не связно, если существует  $P, Q : X = P \cup Q, P \cap Q = \emptyset, P \neq X, Q \neq X, P, Q$  открыты.

Множество связно, если является связным пространством под индуцированной топологией (или при сужении топологии на него, то же самое).

*Пример.*  $(0, 1) \cup (2, 3)$  — не связно.

*Пример.*  $(0, 1) \cup (2, 3]$  — тоже не связно, т.к.  $(2, 3]$  открыто в  $(0, 1) \cup (2, 3]$

*Пример.*  $(0, 1) \cup (2, 3)$  в дискретной топологии — не связно.

*Упражнение 1.* Предложите топологию, в которой пространство  $(0, 1) + (2, 3)$  связно. Тривиальная топология.

**Определение.** Пространство линейно связно, если существует путь из любой точки в любую.

Первое определение гласит, что мы не можем провести границу, а второе — что не можем ???.

**Определение.** Равенство — тип пути из  $a : X$  в  $b : X$ . Равенство обитаемо, если такой путь существует.

Такое определение сложно уместить в язык программирования, т.к.  $[0, 1]$  не влезает в компьютер, поэтому придуман интервальный тип  $I = [left, right]$ .

*Пример.*  $\leq 1 : \text{Nat}, 2 : \text{Nat}$ , в  $\text{Nat}$  дискретная топология. Нужно построить  $f : f(0) = 1, f(1) = 2$ .

Рассмотрим `Path` в `Arend`:

```
\data Path (A : I → \Type) (a : A left) (a' : A right)
  | path (\Pi (i : I) → A i)
```

$I$  — интервальный тип,  $A$  — отображение интервал  $\rightarrow$  тип. `path` — единственный конструктор, принимает функцию, сопоставляющую точки интервала значение  $A$  в этой точке.

Докажем в `Arend`, что  $1 = 1$ :

```
\func oneone : (1 = 1) ⇒ idp
```

Если мы не знаем, что сделать, мы можешь написать `?`, то выражение условно принимается. Иногда работают рефакторинги, которые заменяют `?` на доказательство.

Фигурные скобки вокруг аргумента обозначают, что аргумент неявный. Запишем в явном виде:

```
\func oneone : (1 = 1) ⇒ idp {Nat} {1}
```

`oneone` — значение зависимого типа равенства.

```
\func arar : ((1 Nat.+ 2) = (2 Nat.+ 1)) => idp
```

idp это refl из Lean.

У нас интенциональное равенство.

Построим функцию с типовым параметром:

```
\func second (t : \Type) (ttt : \Pi (x : t) -> t) (y : t) : t => ttt y
```

Здесь ttt — функция  $t \rightarrow t$ , мы её применяем к  $y : t$  и получаем  $ttt\ y : t$

```
\func third (t : \Type) (ttt : \Pi (x : t) -> t) (y : t) => ttt y = ttt y
```

# Лекция 10

## 9 ноября

### 11 Равенство

Напоминание:  $a = b$  по определению выполнено тогда и только тогда когда существует путь  $a \rightsquigarrow b$ . Путь можно определить так:

**Определение.** Пусть  $I$  — интервальный тип. Если существует непрерывная функция  $f : I \rightarrow A$ , такая что  $f \text{ left} =_{\beta} a$ ,  $f \text{ right} =_{\beta} b$ , то  $a \rightsquigarrow b$ .

То же самое в рамках Agda:

```
\data Path
  path (f : I → A) : f left = f right
```

Когда мы говорим о любом типе данных, у нас есть две конструкции:

1. Построение
2. Удаление

Для Path есть построение — конструктор. Мы хотим получить ещё и элиминатор.

*Пример* (элиминатор для  $\vee$ ).

```
case (f : L → θ) (g : R → θ) (v : L ∨ R) : θ
```

*Пример* (элиминатор для  $I$ ).

```
\func coe (P : I → Type)
  (a : P left)
  (i : I) : P i \elim i
| left ⇒ a
```

Т.к. все значения на пути равны, то мы возвращаем значение на left.

`\elim` это паттерн матчинг. Т.к.  $I$  — особый тип, нам не нужно расписывать все случаи.

Это все весьма странно, но это единственная конструкция подобного рода.

```
\func transport {A : \Type} (B : A → \Type) {a a' : A} (p : a = a') (b : B a)
  : B a'
⇒ coe (\lam i ⇒ B (p @ i)) b right
```

Обозначение. Если  $p : a = a'$ , то:

1.  $p @ \text{left}$  это  $a$
2.  $p @ \text{right}$  это  $a'$

*Примечание.* Фигурные скобки означает типы, которые компилятор сам выведет из контекста. Рассчитывать на него нельзя, т.к. выводение типов неразрешимо в общем случае.

*Пример* (доказательство равенства).

```
\func inv (A : \Type) {a a' : A} (p : a = a') : a' = a
⇒ transport (\lam x ⇒ x) p idp
```

```
\func idp {A : \Type} {a : A} : a = a
⇒ path (\lam _ ⇒ a)
```

*Пример* (конгруэнтность).

```
\func pmap (A B : \Type) (f : A → B) (a a' : A) {p : a = a'} : f a = f a'
⇒ transport (\lam x ⇒ f a = f x) p idp
```

*Пример* (натуральные числа).

```
\data Nat
| zero
| suc (k : Nat)
```

```
\data Empty
```

```
Not (A : \Type) : A → Empty
```

```
Not (zero = suc zero)
```

```
\func proof_ne (a : Nat) : \Type \elim a
| zero ⇒ 0 = 0
| suc x ⇒ Empty
```

```
\func zne (x : 0 = 1) : Empty
⇒ transport proof_ne {0} {1} x idp
```

*Примечание.* В стандартной библиотеке это доказано немного по-другому, вместо  $\emptyset = \emptyset$  используется `\Sigma` тип кортежа из нуля элементов, то есть `()` из Haskell.

# Лекция 11

## 15 ноября

Все, что мы доказывали, как-то не очень интересно — это тождества. Что если мы хотим доказать например  $a \leq b$ ? Для этого для начала надо уметь это записать.

**Определение** (меньше или равно).  $a \leq b$  это  $\exists x. a + b = b$

Несложно догадаться, что у нас экзистенциальный тип. В Arend все экзистенциальные типы это пары вида  $(x : A, a + x = b : A \rightarrow \text{\textbackslash Type})$ . Таким образом, наш тип это  $\text{\textbackslash Sigma } (x : \text{Nat}) (a + x = b)$ .

*Пример.* Хотим доказать  $5 \leq 12$ , тогда доказательство это  $(7, \text{idp}) : \text{\textbackslash Sigma } (x : \text{Nat}) (5 + x = 12)$

Определим наш тип “ $\leq$ ” индуктивно:

```
\data loe (a b : Nat) \with
  | zero, _ => base
  | suc a', suc b' => next (p : loe a' b')
```

*Примечание.* base и next — конструкторы типа, а не магическая вещь для индукции.

*Пример.*

- base: loe 0 15
- next (next (base)): loe 1 16

Эти два определения эквивалентны. Докажем это.

Из индуктивного в экзистенциальный тип:

```
\func f1 {a b : Nat} (p1 : loe a b) : loe' a b
  | {0}, {b}, base => (b, idp)
  | {suc a}, {suc b}, next (pr1) =>
    \let (pb, ppr) => f1 pr1 \in (pb, pmap suc ppr)
```

В обратную сторону:

```
\func f2 {a b : Nat} (p1 : loe' a b) : loe a b
| {0}, _ => base
| {suc a}, {0}, (x, p) => absurd (transport (\lam t => \case t \with {
| 0 => Empty
| suc n => \Sigma}) p ())
```

Это не удобно писать, поэтому напомним `contradiction`<sup>1</sup>. Эта конструкция умеет доказывать противоречия за 1–2 шага. Таким образом:

```
\func f2 {a b : Nat} (p1 : loe' a b) : loe a b
| {0}, _ => base
| {suc a}, {0}, (x, p) => contradiction
| {suc a}, {suc b}, (x, p) => next (f2 (x, pmap minus1 p))
```

Докажем часть домашнего задания:

```
\func plus-assoc {a b c : Nat} : (a + b) + c = a + (b + c) \elim c
| 0 => idp
| suc c => {?}
```

Можно вместо `{?}` написать `pmap suc (plus-assoc)`, но это скучно. Можем написать `rewrite plus-assoc idp`, который докажет нам `suc ((a + b) + c) = suc (a + (b + c))`, переписав `(a + b) + c` на `a + (b + c)`, т.к. есть доказательство `(a + b) + c = a + (b + c)`. Это звучит как магия, но на самом деле делается так:

```
transport (\lam x => (a + b) + suc x = suc x) plus-assoc idp
```

## 12 Классы

*Пример.* Группа:  $\langle R, +, e, {}^{-1} \rangle$ , такие что:

- $e + x = x$
- $x + e = x$
- $x + x^{-1} = e$
- $x^{-1} + x = e$

Попробуем описать что-то подобное в Arend.

```
\instance OrdNat : Preorder Nat
| ≤ (a b : Nat) => TruncP (\Sigma (r : Nat) (r + a = b))
| ≤-reflexive => inP (0, idp)
| ≤-transitive {x} {y} {z} =>
```

<sup>1</sup> Для этого надо добавить `dependencies: [arend-lib]` в `arend.yaml`

```

\lam (t1 : TruncP (\Sigma (r : Nat) (r + x = y)))
      (t2 : TruncP (\Sigma (r : Nat) (r + y = z))) =>
  \case t1, t2 \with {
    | inP (d1, p1), inP (d2, p2) => inP (d2 + d1, plus-assoc *> (rewrite p1 p2))
  }

```

У `TruncP` есть математическое объяснение, и есть программистское. У нас есть различные доказательства и мы их все объявляем равными.

План дальнейших лекций:

1. Set/Prop
2. Теорема Диаконеску: теория множеств + ИИВ + аксиома выбора это классическая логика.

Дальше есть несколько вариантов:

1. Гомотопическая теория типов (математически)
2. Другие языки: возможно Idris, Coq
3. Другие исчисления, возможно  $F_\Omega$ , линейные типы.



# Лекция 12

## 23 ноября

### 13 Язык

Какой язык мы обсуждали? Это примерно  $\lambda_C$ , но это не совсем так.

#### 13.1 Проблема

*Пример.*

```
\func id (x : *) : * = x
\func idid  $\Rightarrow$  id id
```

Написать `idid` нельзя, потому что вместо `*` мы подставили `*  $\rightarrow$  *`. Это можно исправить:

```
\func id2 (x : \Type) : (x  $\rightarrow$  x)  $\Rightarrow$  \lam a  $\Rightarrow$  a
\func idid2  $\Rightarrow$  id2 (\Type  $\rightarrow$  \Type) (id2 \Type)
```

Тогда получается, что  $\backslash\text{Type} \rightarrow \backslash\text{Type} : \backslash\text{Type}$ . Таким образом, выразительная сила этого языка выше, чем  $\lambda_C$ .

#### 13.2 Естественность проблемы

Такие конструкции не искусственны и встречаются в природе:

*Пример.*

```
\func Church  $\Rightarrow$  (x : \Type)  $\rightarrow$  (x  $\rightarrow$  x)  $\rightarrow$  (x  $\rightarrow$  x)
\func inc (n : Church) : Church  $\Rightarrow$  {?}
\func add m n  $\Rightarrow$  m Church inc n
```

Таким образом, несложно догадаться, что у нас будут встречаться функции высших родов. Таким образом, тип это в том числе и род. Как это типизировать?

### 13.3 Наивное решение проблемы

Очевидная идея:  $\backslash\text{Type}$  это  $\backslash\text{Type}$ . Это невозможно в силу парадокса Жирара, который будет рассмотрен на отдельной лекции.

### 13.4 Предикативность

Пусть типы организуют не один универсум, а множество вложенных универсумов.

У  $\backslash\text{Type}$  теперь есть натуральный аргумент  $n$ , называемый **предикативностью**:

- $\backslash\text{Type } 0$  — базовые типы.
- $\backslash\text{Type } 1$  — все, включая  $\backslash\text{Type } 0$ .
- $\vdots$
- $\backslash\text{Type } (k + 1)$  — все, включая  $\backslash\text{Type } k$ .

При этом если некоторая функция принимает аргумент  $x : \backslash\text{Type } n$  и возвращает  $x \rightarrow x$ , то она имеет предикативность  $n + 1$ .

*Примечание.* В большинстве случаев компилятор выводит предикативность сам, но иногда он не справляется и ему нужно помочь.

*Пример.*

```
\func Church => \Pi (x : \Type) -> (x -> x) -> (x -> x)
\func inc (n : Church) => \lam t f x => n t f (f x)
\func add (m : Church \levels (\suc \lp) \lh)
          (n : Church \levels \lp \lh)
          : \Church \levels \lp \lh
=> m Church inc n
```

Здесь мы указываем, что уровень предикативности  $m$  это  $\backslash\text{lp} + 1$ , где  $\backslash\text{lp}$  — специальная внутренняя переменная компилятора, которая следит за тем, какой уровень предикативности выражения, которое мы сейчас компилируем. Таким образом, у нас предикативность  $m$  есть предикативность  $n+1$ .

$$m \text{ Church} : \overbrace{(\underbrace{\text{Church}}_{\backslash\text{lp}} \rightarrow \text{Church}) \rightarrow (C \rightarrow C)}^{\backslash\text{lp}+1}$$

Число  $\backslash\text{lh}$  пока что не рассматриваем.

Пусть  $\backslash\text{Prop}$  — вселенная пропозиций, т.е. “чистых утверждений”.

**Определение.**  $x : \backslash\text{Type} \rightarrow x : \backslash\text{Prop}$ , если все элементы  $x$  равны.

Грубо говоря, это тип доказательств, которые не зависят от выбора представителей.

Пример.

1. Доказательство равенства `Nat` это `Prop`.
2. `a : \Prop, b : \Prop`, тогда `(a, b) : \Prop`
3. `Either a b` — не `\Prop`, потому что можно доказывать через `inLeft a`, либо через `inRight b` и эти доказательства не равны.
4. `Σ` — не `Prop`.

**Определение.** `\Set` — тип, в котором равенство это `\Prop`.

Гомотопический уровень типа  $- + 1$  от уровня равенства на нём.

Пропозициональное урезание: `||x|| : \Prop`.

Пример.

- `||Either a b|| ⇒ a || b`
- `||Sigma (x : N) (T(x))|| ⇒ ∃ x^N. T(x)`

В `Arend` `||||` обозначается как `TruncP : \Type → \Prop`.

Пример.

```
\data Int'
  | pos' Nat
  | neg' Nat
```

В этом типе есть два нуля — положительный и отрицательный. Так нельзя. Нужно объявить, что эти два нуля равны:

```
\data Int
  | pos Nat
  | neg (n : Nat) \elim n {
    | 0 ⇒ pos 0
  }
```

Также можно писать `\truncated \data ...` и тогда не нужно будет писать `TruncP`, все элементы типа уже будут равны между собой.

```
\truncated \data Quotient
  (A : \Type) (R : A → A → \Type) : \Set
  | inR A
  | eq (a a' : A) (r : R a a') (i : I)
    \elim i
    | left ⇒ inR a
    | right ⇒ inR a'
```

## 14 Аксиома выбора

- $P$  — семейство множеств
- $p : A \rightarrow B$

Тогда

$$(\forall x \in A. \exists t \in p(x)) \rightarrow \exists f : \forall x \in A. f(x) \in p(x)$$

То же самое, но в Arend:

$$(A : \text{Set}) (P : A \rightarrow \text{Set}) \rightarrow (\text{Pi } (x : A) \rightarrow \|P(x)\|) \rightarrow \|\text{Pi } (x : A) \rightarrow P(x)\|$$

Это влечёт `Decide (Either a b)` — закон исключенного третьего, где `Decide` это:

```
\data Decide (x : \Type)
| yes (a : x)
| no  (b : Not x)
```

, то есть по типу можно алгоритмически определить его обитаемость.