

# Applied Type Theory

## Лекция 1: введение в $\lambda$ -исчисление

Воронов Михаил Сергеевич

ВМК МГУ

Осень 2025

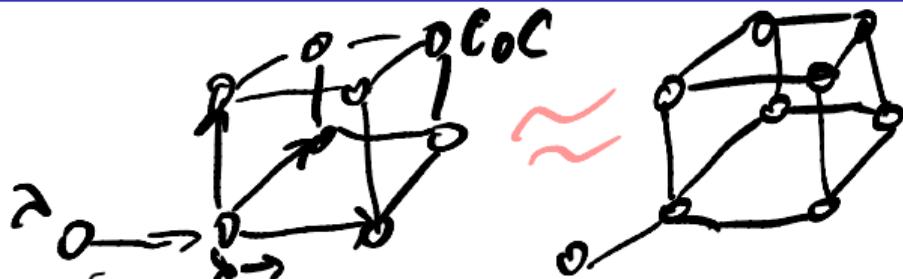
# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

# План лекции

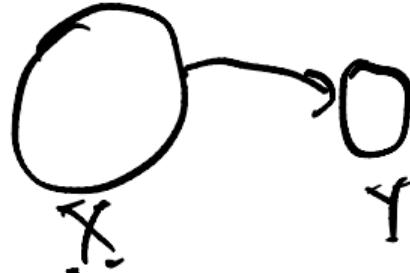
- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

# Теоретический план курса



- ➊ Простое нетипизированное  $\lambda \rightarrow \lambda \rightarrow \dots$  → алгебра типов.
- ➋ Классические логики и исчисления (NJ/LK).
- ➌  $\lambda 2$ ,  $\lambda \omega$ , зависимые типы (MLTT/HoTT).
- ➍ Субструктурные типы: линейные/аффинные, связь с Rust и Haskell.

# Практический план курса



- ① Теоретические задачи на  $\lambda$ -исчисление
- ② Практические задачи на Coq/Agda/TLA+/OCaml
- ③ Задачи вида "опишите условия и следствия теоремы на языке 7-летнего ребёнка"
- ④ Практические задачи CTF-типа на sandbox escape
- ⑤ Задачи на проектирование API на базе теории типов
- ⑥ Каждое домашнее задание будет содержать обязательные задачи и задачи со звёздочкой, которые можно сдать до конца семестра

## Критерий оценивания

На данный момент планируется 6 домашних заданий, за каждое можно получить 40-50 баллов. Также за экзамен можно получить до 150 баллов, критерии оценивания (могут немного поменяться к концу семестра):

- 1 0–79 — 2
- 2 80–149 — 3
- 3 150–199 — 4
- 4 200+ — 5

## TAPL

- Pierce — *Types and Programming Languages*.
- Nederpelt, Geuvers — *Type Theory and Formal Proof*.
- Software Foundations (Vol. 1–2) — <https://softwarefoundations.cis.upenn.edu/>.
- Bertot, Castéran — *Interactive Theorem Proving and Program Development*.
- Mimram — *Program = Proof* (lecture notes).
- Курс Москвитина Д. Н. — *Функциональное программирование* (YouTube).

# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

# Зачем это разработчику и специалисту ИБ

## Инженерная максима

Свести классы ошибок к невозможным по типам и формально проверить и доказать критические свойства систем.

- **Типы как контракты:** сигнатуры фиксируют допустимые состояния и переходы, некорректные программы не компилируются.
- **Раннее обнаружение дефектов:** становятся невозможными целые классы уязвимостей (например, null deref, double free, data races, неправильные состояния протокола) на этапе компиляции.
- **Формальная верификация:** Coq/Agda/... доказывают инварианты безопасности и корректность алгоритмов, инвариант «ошибка не может произойти» выражается как теорема.
- **Композиционная надёжность:** свойства собираются из локальных «контрактов» модулей (ADT, parametricity, зависимые типы, сессионные типы).
- **Знание ограничений подхода:** при всех плюсах данный подход не является панацеей и имеет свои ограничения, которые нужно понимать перед использованием.

## Примеры использования типовых техник

- **Null dereference** ⇒ Option/Maybe, исчерпывающий pattern matching.
- **Use-after-free / double free** ⇒ линейные/аффинные типы, владение/заимствование (ownership/borrowing), времена жизни (lifetimes).
- **Data race** ⇒ неизменяемость по умолчанию, типы эффектов, правила заимствования, сессионные типы.
- **Out-of-bounds / Heartbleed-класс** ⇒ индексированные по длине структуры (напр., Vec  $n \alpha$ ), безопасная индексация.
- **Нарушение протокола / неправильный порядок шагов** ⇒ typestate, сессионные типы ( $!\tau.S$ ,  $?\tau.S$ , end).
- **TOCTTOU** ⇒ capability-токены как линейные ресурсы, атомарные переходы состояний.
- **Инъекции (SQL/XSS)** ⇒ типизированные AST/DSL вместо строк, параметризованные запросы, GADT-кодирование выражений.
- **Неполное покрытие вариантов** ⇒ ADT + исчерпывающий pattern matching.
- **Смешение единиц/домена** ⇒ новые типы/phantomные параметры (Meters, Seconds); запрет неосмысленных операций.

# API как контракт: typestate

## Основная идея

Состояние системы выражается в (фантомных) типах, а нелегальные переходы не типизируются.

```
(* Phantom session state *)
type logged_out
type logged_in

type ('state) session

val login : creds -> logged_out session -> logged_in session
val logout : logged_in session -> logged_out session
val transfer : logged_in session -> amount -> account -> logged_in session
(* Attempt transfer without login: ill-typed *)
```

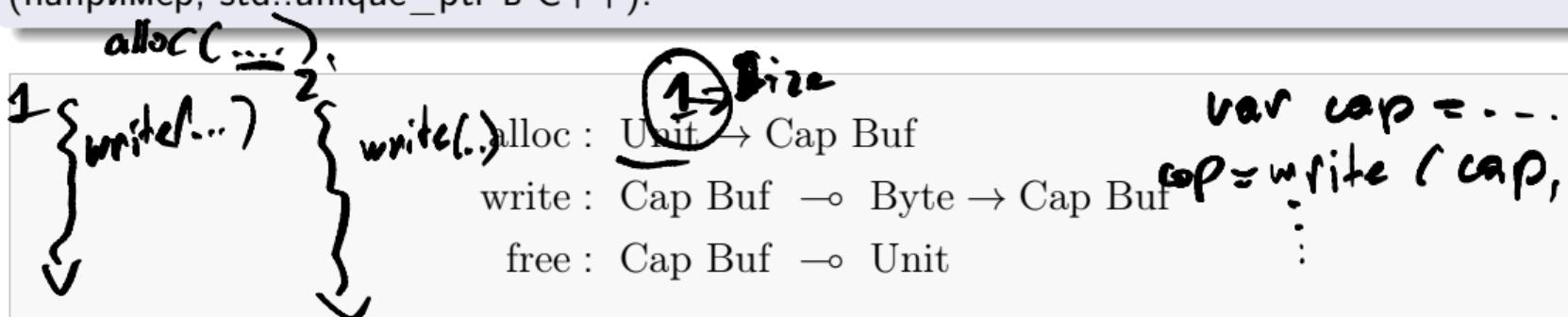
- Контролируем «правильность сценариев» ⇒ отсутствуют целые классы ошибок без ad-hoc проверок в коде.

# Ресурсы и память: линейные типы



## Контракты ресурсов

Субструктурные типы: ресурс потребляется ровно один раз или не более одного раза (например, std::unique\_ptr в C++).



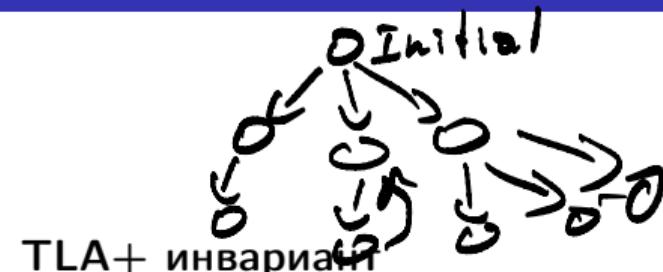
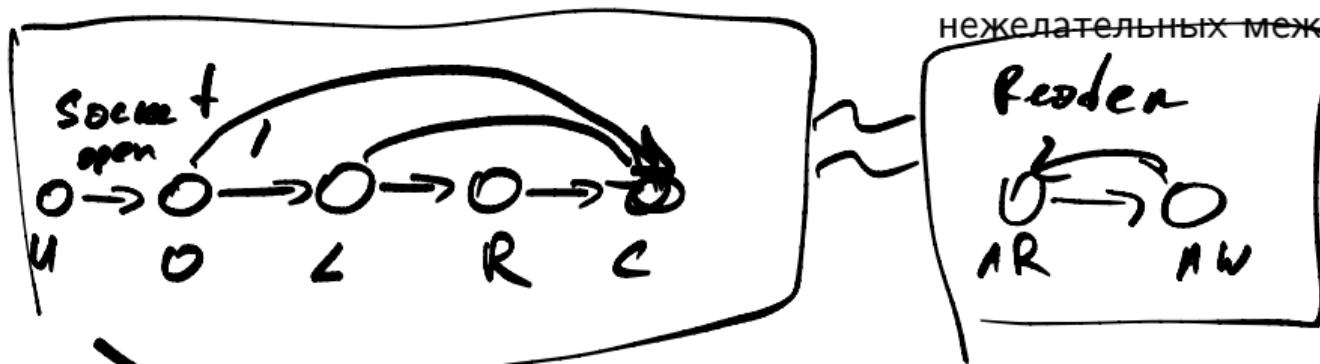
- free ∘ free не типизируется ⇒ исключаем double free на уровне типов.
- Задействование ⇒ одновременная запись из двух мест не проходит проверку типов ⇒ нет data race.

# Протоколы как типы

## Сессионные типы

Client: !Req.?Ack.end Server: ?Req.!Ack.end

Композиция  $\Rightarrow$  корректное взаимодействие, отсутствие перепутанных сообщений и «зависаний». Можно сказать, что в типе кодируется машина состояний протокола.



## TLA+ инвариант

$\text{Inv} \triangleq \forall c \in \text{Clients}:$

$\text{SentReq}[c] \Rightarrow \neg \text{RecvAck}[c] \quad \text{U}$   
 $\text{RecvAck}[c]$

Проверка инварианта Inv model-чекером  
 $\Rightarrow$  (с высокой вероятностью) отсутствие нежелательных межсостояний.

# Формальная верификация: что будем доказывать

- **Инварианты API в Agda/Coq:** сохранение размера очереди, отсутствие чтения удалённого, корректность ролей/прав.
- **Свойства протокола в TLA+:** безопасность (safety) и живость (liveness) для простых handshake/lock-free сценариев.
- **Parametricity & free theorems:** автоматические «бесплатные» законы корректности для полиморфного кода (напр.,  $\text{length}(\text{map } f) = \text{length}$ ).

## Практический эффект

Свойства становятся частью артефакта сборки: *если собрано, значит, проходит доказательства и проверки.*

# Ограничения и зона ответственности

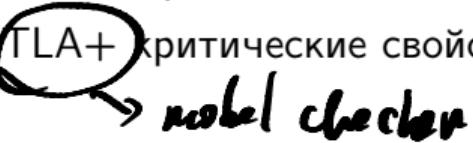
Модель Гиль

Программа

- Типы и доказательства не заменяют криптографию, настройку ОС и операционные и административные регламенты.
- Канальные утечки и физические побочные каналы требуют дополнительных методик (профилирование времени, изоляция, ТЕЕ).
- Политика безопасности должна быть **корректно смоделирована**; неверная модель ⇒ корректная, но неверная система.
- На практике модели и доказательства часто имеют ограниченно смоделированный контекст предметной области.



# Итог мотивации

- ① Типы — это исполнимые спецификации: что запрещено — не скомпилируется.
- ② С помощью Coq/Agda/~~TLA+~~ критические свойства становятся теоремами, а не надеждой на тесты.  
  
надежда на тесты
- ③ Фокус курса — перенести эти принципы в практику: от теории  $\lambda$ -исчисления до применения этого в API и описания протоколов с использованием типов.

# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

- $\lambda$ -исчисление - формальная система, разработанная Алонзо Чёрчем в 1930-х для формализации и анализа понятия вычислимости
- Имеет нетипизированную (untyped) и множество типовых версий
- Позволяет описывать семантику вычислительных процессов
- Является теоретической основой для многих пруверов

Рассмотрим функцию  $f(x) : x^2 + 1$

- эта функция имеет один "вход" (другими словами, зависит от одной переменной) и один "выход":  $f : \mathbb{R} \rightarrow \mathbb{R}$
- в некотором смысле эту функцию можно рассматривать, как отображение  $x \rightarrow x^2 + 1$
- чтобы подчеркнуть "абстрактную" роль  $x$ , используют специальный символ  $\lambda$ :  
 $\lambda x. x^2 + 1$
- данная нотация выражает то, что  $x$  - это не конкретное число, а некоторая абстракция
- для конкретного значения можно "применить" данную функцию:  $(\lambda x. x^2 + 1)(3)$

# Введение в $\lambda$ -исчисление

Из предыдущего слайда следует, что для работы с функциями достаточно двух способов построения выражений:

- ① Абстракция: из выражения  $M$  и переменной  $x$  можно составить новое выражение  $\lambda x.M$  (абстракция  $x$  по  $M$ )
- ② Применение: из двух выражений  $M$  и  $N$  можно составить новое выражение  $MN$

## Введение в $\lambda$ -исчисление: абстракция

$$f(4) = 1 + 2 + 3$$

- ① Пусть  $M = M[x]$  - выражение, возможно содержащее  $x$
- ② Тогда абстракция  $\lambda x.M$  обозначает функцию  $x \rightarrow M[x]$
- ③ Абстракция - способ задать неименованную функцию
- ④ Если  $x$  в  $M[x]$  отсутствует, то  $\lambda x.M$  - константная функция со значением  $M$ .

- ➊ С точки зрения разработки ПО, применение  $F$  к  $X$  - это применение алгоритма ( $F$ ) к данным ( $X$ )
- ➋ Однако явного различия между алгоритмами и данными нет, в частности, возможно самоприменение:  $FF$
- ➌ В общем случае применение - это так называемое  $\beta$ -преобразование:  
$$(\lambda x.M)N \rightarrow_{\beta} M[x := N]$$
- ➍  $M[x := N]$  - это  $M$ , в котором свободные вхождения  $x$  заменены на  $N$

## Введение в $\lambda$ -исчисление: примеры $\beta$ -редукции

$$(\lambda x. x^2 + 1) 3 \rightarrow_{\beta} 3^2 + 1$$

$$\begin{array}{c} (\lambda y. 5) 1 \rightarrow_{\beta} 5 \\ \text{Id} \\ (\lambda x. x) (\lambda y. y) \rightarrow_{\beta} \lambda y. y \end{array}$$

$$\lambda z. ((\lambda x. x) (\lambda y. y)) \rightarrow_{\beta} \lambda z. \lambda y. y$$

# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

# Формальное построение $\lambda$ -исчисления

$$V = \{x_1, x_2, \dots\} \quad |V| = \infty$$

## Definition

Множество  $\lambda$ -термов  $\Lambda$  строится индуктивно:

$$x \in V \Rightarrow x \in \Lambda \quad M, N \in \Lambda \Rightarrow (MN) \in \Lambda \quad M \in \Lambda, x \in V \Rightarrow (\lambda x. M) \in \Lambda$$

- Абстракция  $\lambda x. M$ : анонимная функция.
- Применение  $MN$ : вызов функции  $M$  к аргументу  $N$ .
- Принятые соглашения: применение левоассоциативно, абстракция правоассоциативна, тело абстракции тянется максимально вправо.

## Примеры термов

- $x$
- $(xz)$
- $(\lambda x.(xz))$
- $((\lambda x.(xz))y)$
- $(\lambda y.((\lambda x.(xz))y))$
- $((\lambda y.((\lambda x.(xz))y))w)$
- $(\lambda z.(\lambda w.((\lambda y.((\lambda x.(xz))y))w)))$

# Термы (соглашения)

Общеприняты следующие соглашения:

- Внешние скобки опускаются
- Применение левоассоциативно:

$FXYZ$  обозначает  $((FX)Y)Z$

- Абстракция правоассоциативна:

$\lambda xyz.M$  обозначает  $(\lambda x.(\lambda y.(\lambda z.(M))))$

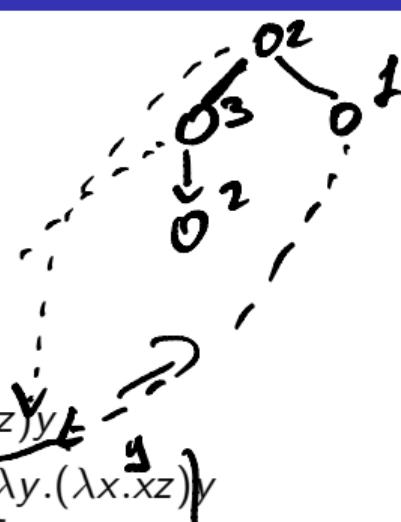
- Тело абстракции простирается вправо насколько это возможно

$\lambda x.MNK$  обозначает  $\lambda x.(MNK)$

$$\lambda y. \lambda y (\lambda x. x) z$$

## Примеры термов

- $x = x$
- $(xz) = xz$
- $(\lambda x.(xz)) = \lambda x.xz$
- $((\lambda x.(xz))y) = (\lambda x.xz)y$
- $(\lambda y.((\lambda x.(xz))y)) = (\lambda y.(\lambda x.xz))y$
- $((\lambda y.((\lambda x.(xz))y))w) = (\lambda y.(\lambda x.xz)y)w$
- $(\lambda z.(\lambda w.((\lambda y.((\lambda x.(xz))y))w))) = \lambda zw.(\lambda y.(\lambda x.xz)y)w$



$\text{② } v \in V \Rightarrow v \in \Lambda$   
 $\text{③ } M, N \in \Lambda \Rightarrow MN \in \Lambda$   
 $\text{④ } M \in \Lambda, x \in V \Rightarrow \lambda x. M \in \Lambda$

# Свободные и связанные переменные

## Definition

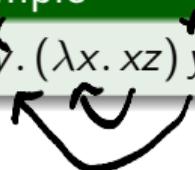
$\text{FV}(\cdot)$  и  $\text{BV}(\cdot)$  задаются рекурсивно:

$$\text{FV}(x) = \{x\}, \quad \text{FV}(MN) = \text{FV}(M) \cup \text{FV}(N), \quad \text{FV}(\lambda x. M) = \text{FV}(M) \setminus \{x\};$$

$$\text{BV}(x) = \emptyset, \quad \text{BV}(MN) = \text{BV}(M) \cup \text{BV}(N), \quad \text{BV}(\lambda x. M) = \text{BV}(M) \cup \{x\}.$$

## Example

$(\lambda y. (\lambda x. xz) y w)$  связаны  $x, y$ , свободны  $z, w$ .



## Упражнение: свободные и связанные переменные

Для каждого терма определите множества  $FV(\cdot)$  и  $BV(\cdot)$ . Обратите внимание на случаи затенения (shadowing) переменных!

1  $\lambda x. x (\lambda y. x y z) u$

$|x| \rightarrow x (y \rightarrow x (y, z), u)$

2  $\lambda f. \lambda x. f (g x) (\lambda y. h y)$

$|p, q| \rightarrow p (|p| \rightarrow q (p, r), s)$

3  $\lambda x. (\lambda x. x y) x z$

4  $(\lambda a. \lambda b. a c (\lambda c. b c)) d$

5  $\lambda p. \lambda q. p (\lambda p. q p r) s$

# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности**
- 6 Комбинаторы и кодирование данных

# Бинарные отношения и отношение эквивалентности

## Definition (Бинарное отношение)

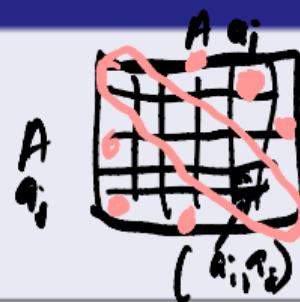
Пусть  $A$  — множество. Бинарное отношение  $R$  на  $A$  — это подмножество  $A \times A$ . Пишем  $a R b$ , если  $(a, b) \in R$ .

$R: A \rightarrow A \rightarrow X$ ,  $X$  — множество пар  $(a, b) \in A \times A$

## Definition (Свойства отношений)

Для отношения  $R \subseteq A \times A$ :

- **Рефлексивность:**  $\forall a \in A a R a$
- **Симметричность:**  $\forall a, b \in A a R b \Rightarrow b R a$
- **Транзитивность:**  $\forall a, b, c \in A a R b \wedge b R c \Rightarrow a R c$



## Definition (Отношение эквивалентности)

Эквивалентность — это отношение, которое одновременно рефлексивно, симметрично и транзитивно. Эквивалентности индуцируют разбиение множества на **классы эквивалентности**.

## $\alpha$ -преобразование (шаг)

Definition ( $\alpha$ -преобразование)

$$\lambda x. M \xrightarrow{\alpha} \lambda y. M[x \mapsto y] \text{ если } y \notin \text{FV}(M).$$

Замечание: Здесь  $M[x \mapsto z]$  — переименование связанного параметра  $x$  в  $M$  на  $z$  (не подстановка терма).

$$M [x := y]$$

## $\alpha$ -преобразование: примеры

$$\lambda y. xy \neq \lambda x. xx$$



- $\lambda x. \lambda y. x(\lambda x. yx) =_{\alpha} \lambda u. \lambda v. u(\lambda x. vx)$
- $\lambda x. (\lambda x. xx) =_{\alpha} \lambda z. (\lambda u. uu)$
- $\lambda y. xy \neq_{\alpha} \lambda x. xx$  — захват свободного  $x$  при  $y \rightarrow x$  недопустим
- $\lambda x. \lambda y. xy \neq_{\alpha} \lambda y. \lambda x. yx$  — перестановка связок не  $\alpha$ -эквивалентность
- $\lambda x. \lambda y. x(\lambda y. y) =_{\alpha} \lambda p. \lambda q. p(\lambda r. r)$

## $\alpha$ -ЭКВИВАЛЕНТНОСТЬ

### Definition

Определим отношение  $=_\alpha$  на  $\Lambda$   
индукцией по структуре термов.

- **Переменная**  $x =_\alpha x$ .
- **Применение** Если  $M_1 =_\alpha N_1$  и  $M_2 =_\alpha N_2$ , то  $M_1 M_2 =_\alpha N_1 N_2$ .
- **Абстракция**  $\lambda x. M =_\alpha \lambda y. N$   
тогда и только тогда, когда существует переменная  $z$  такая, что  
 $z \notin \text{FV}(M) \cup \text{FV}(N)$  и
$$M[x \mapsto z] =_\alpha N[y \mapsto z].$$

$\lambda x y. x y \neq_\alpha \lambda y x. y$

**Замечание 1:** Здесь  $M[x \mapsto z]$  — переименование связанного параметра  $x$  в  $M$  на  $z$  (не подстановка терма).

**Замечание 2:** Далее проверяется (нетрудно), что  $=_\alpha$  — отношение эквивалентности.

# Конвенция Барендрехта (формулировка)

$$\int V = \infty$$

## Соглашение (BVC)

В рассуждениях об  $\lambda$ -термах мы по умолчанию работаем с  $\alpha$ -эквивалентным представителем  $M'$  терма  $M$  таким, что:

- ① все *связанные* переменные используют попарно разные имена (нет затенения);
- ② ни одна связанныя переменная не совпадает ни с одной *свободной* переменной:  
 $BV(M') \cap FV(M') = \emptyset$ .

**Интуиция.** Имена связанных переменных — «временные ярлыки». Их можно переименовать ( $\alpha$ -переименованием) так, чтобы они не мешали подстановкам и доказательствам.

## Почему конвенция применима всегда

- $\alpha$ -эквивалентность разрешает свободное переименование связанных переменных на свежие имена (не попадающие в FV).
- Алфавит переменных бесконечен  $\Rightarrow$  свежие имена всегда существуют.
- Значит, для любого  $M$  существует  $M' =_{\alpha} M$ , удовлетворяющий BVC, и рассуждать «с точностью до  $\alpha$ » корректно.

### Мини-следствие

Если  $y \notin \text{FV}(M)$ , то  $\lambda x. M =_{\alpha} \lambda y. M[x \mapsto y]$ .

## Подстановка без захвата

### Definition (Подстановка $M[x:=N]$ )

Обозначим через  $M[x := N]$  подстановку терма  $N$  вместо свободных вхождений  $x$  в  $M$ , задаваемую правилами

$$x[x := N] = N, \quad y[x := N] = y \quad (y \neq x),$$

$$(P Q)[x := N] = (P[x := N])(Q[x := N]),$$

$$(\lambda y. P)[x := N] = \begin{cases} \lambda y. P, & y = x; \\ \lambda y. (P[x := N]), & y \neq x, y \notin \text{FV}(N); \\ \lambda z. (P[y \xrightarrow{y \neq x} z])[x := N], & y \neq x, y \in \text{FV}(N), z \notin \text{FV}(P) \cup \text{FV}(N). \end{cases}$$

## $\beta$ -редукция

Definition ( $\beta$ -редукция)

$$(\lambda x. M) N \rightarrow_{\beta} M[x := N].$$

Definition (Редекс)

Редекс (reducible expression) - это выражение вида  $(\lambda x. M) N$ .

- $\beta$ -редукция — основной механизм вычисления в  $\lambda$ -исчислении
- Моделирует применение функции к аргументу
- Подстановка должна избегать захвата переменных

# Примеры $\beta$ -редукции

$\rightarrow_{\beta}$   $\rightarrow_{\beta}$

Example

$$\begin{array}{c} \text{Id} \quad \text{Id} \quad \text{Id} \\ (\lambda x. x)(\lambda y. y) \rightarrow_{\beta} \lambda y. y \\ | \\ (\lambda f. \lambda x. f x)(\lambda y. y) \rightarrow_{\beta} \lambda x. ((\lambda y. y) x) \rightarrow_{\beta} \lambda x. x \\ (\lambda f. \lambda g. \lambda x. f (g x))(\lambda y. y y)(\lambda z. z) \\ F \quad x \quad y \quad \begin{aligned} &\rightarrow_{\beta} (\lambda g. \lambda x. (\lambda y. y y) (g x))(\lambda z. z) \\ &\rightarrow_{\beta} \lambda x. (\lambda y. y y)((\lambda z. z) x) \\ &\rightarrow_{\beta} \lambda x. (\lambda y. y y) x \\ &\rightarrow_{\beta} \lambda x. x x \end{aligned} \end{array}$$

# Замкнутость относительно контекстов

Понятие: «замкнутое относительно контекстов»

Под контекстом понимаем терм с одной «дыркой»  $C[\cdot]$ , строящийся по грамматике:

$$C ::= [\cdot] \mid CM \mid MC \mid \lambda x. C.$$

Замкнутость относительно контекстов означает: если  $M \sim N$ , то для любого  $C$  имеем  $C[M] \sim C[N]$ . Не путать с замкнутым термом ( $\text{FV}(M) = \emptyset$ ).

Example

Пусть  $C = \lambda z. [\cdot] z$ . Тогда

$$C[\lambda x. x] =_{\alpha} C[\lambda y. y] \quad \text{т.е.} \quad \lambda z. (\lambda x. x) z =_{\alpha} \lambda z. (\lambda y. y) z.$$

## Definition ( $\beta$ -эквивалентность)

$\beta$ -эквивалентность  $=_{\beta}^*$  — наименьшее отношение эквивалентности, замкнутое относительно контекстов и содержащее  $\beta$ -преобразование:

- ① (Основа)  $(\lambda x. M) N \rightarrow_{\beta} M[x := N] \Rightarrow (\lambda x. M) N =_{\beta}^* M[x := N]$
- ② (Рефлексивность)  $M =_{\beta}^* M$
- ③ (Симметричность)  $M =_{\beta}^* N \Rightarrow N =_{\beta}^* M$
- ④ (Транзитивность)  $M =_{\beta}^* N, N =_{\beta}^* L \Rightarrow M =_{\beta}^* L$
- ⑤ (Совместимость)  $M =_{\beta}^* N \Rightarrow \lambda x. M =_{\beta}^* \lambda x. N, ML =_{\beta}^* NL, LM =_{\beta}^* LN$

## $\beta$ -эквивалентность: цепочки редукций



### Definition («Конструктивное» определение)

Обозначим  $M \leftrightarrow_{\beta} N$ , если  $M \rightarrow_{\beta} N$  или  $N \rightarrow_{\beta} M$  (один шаг в любую сторону). Тогда  $M =_{\beta} N$  тогда и только тогда, когда существует конечная цепочка

$$M_0 \leftrightarrow_{\beta} M_1 \leftrightarrow_{\beta} \cdots \leftrightarrow_{\beta} M_k, \quad M_0 = M, \quad M_k = N.$$

### Замечание

Эквивалентно:  $=_{\beta}$  — рефлексивно–симметрично–транзитивное замыкание  $\rightarrow_{\beta}$  и наименьшая конгруэнция, содержащая  $\rightarrow_{\beta}$ .

## $\beta$ -эквивалентность: пример

Пусть  $M \equiv (\lambda f. \lambda x. f(f x)) g x$ ,  $N \equiv g(g x)$ . Тогда

$$M \rightarrow_{\beta} (\lambda x. g(g x)) x \rightarrow_{\beta} g(g x) = N,$$

откуда  $M =_{\beta}^{*} N$ .

Ещё цепочка (шаги в обе стороны  $\leftrightarrow_{\beta}$ ):

$$(\lambda x. x)((\lambda y. y) t) \leftrightarrow_{\beta} (\lambda y. y) t \leftrightarrow_{\beta} t.$$

*$\beta$ -расширение  
+  $\rightarrow_{\beta} (\lambda y. y) y$*

Эти примеры иллюстрируют конструктивное определение через конечные цепочки  $\leftrightarrow_{\beta}$  и существование общего редукта.

# $\eta$ -преобразование

## Definition ( $\eta$ -преобразование)

$$(\lambda x. M)x =_{\eta} M, \text{ если } x \notin FV(M).$$

## Definition ( $\eta$ -эквивалентность)

$\eta$ -эквивалентность  $=_{\eta}$  — это наименьшее отношение эквивалентности, замкнутое относительно контекстов и содержащее  $\eta$ -преобразование:

- ① (Основа)  $\lambda x. Mx =_{\eta} M$ , если  $x \notin FV(M)$
- ② (Рефлексивность)  $M =_{\eta} M$
- ③ (Симметричность)  $M =_{\eta} N \Rightarrow N =_{\eta} M$
- ④ (Транзитивность)  $M =_{\eta} N, N =_{\eta} L \Rightarrow M =_{\eta} L$
- ⑤ (Совместимость)  $M =_{\eta} N \Rightarrow \lambda x. M =_{\eta} \lambda x. N, ML =_{\eta} NL, LM =_{\eta} LN$

# Примеры $\eta$ -эквивалентности

## Экстенсиональность функций

$\eta$ -эквивалентность выражает принцип **экстенсиональности**: две функции равны, если они дают одинаковые результаты для всех возможных аргументов.

## Example

$$\lambda x. f x =_{\eta} f \quad (\text{если } x \notin \text{FV}(f))$$

$$\lambda y. (\lambda z. z) y =_{\eta} \lambda z. z$$

$$\lambda a. (\lambda b. \lambda c. b) a =_{\eta} \lambda b. \lambda c. b$$

$$\lambda x. x x \neq_{\eta} x \quad (x \in \text{FV}(x x))$$

# Отношения эквивалентности

## Definition (Отношение эквивалентности)

**Эквивалентность** — это отношение, которое одновременно рефлексивно, симметрично и транзитивно. Эквивалентности индуцируют разбиение множества на *классы эквивалентности*.

## Контекст лекции

$\equiv_\alpha$ ,  $\equiv_\beta$  и  $\equiv_\eta$  — отношения эквивалентности на термах. Мы работаем с термами с точностью до этих эквивалентностей:

- по  $\alpha$ : переименование связанных переменных несущественно;
- по  $\beta$ : вычисления/подстановки не меняют «смысл» терма;
- по  $\eta$ : функции, совпадающие по действию на всех аргументах, считаются одинаковыми (экстенсиональность).

Отсюда удобно рассуждать о классах эквивалентности  $[M] = \{N \mid N \equiv M\}$  и формулировать свойства вроде «НФ единственна (с точностью до  $\alpha$ )».

## Отношения эквивалентности: пример

### Example

$$\lambda x. x =_{\alpha} \lambda y. y, \quad (\lambda x. x) t =_{\beta}^{*} t, \quad \lambda x. f x =_{\eta} f \quad (x \notin \text{FV}(f)).$$

# План лекции

- 1 Оргчасть и устройство курса
- 2 Мотивация
- 3 Введение в  $\lambda$ -исчисление
- 4 Формальный синтаксис и нотация
- 5 Редукции и отношения эквивалентности
- 6 Комбинаторы и кодирование данных

## Definition

**Комбинатор (замкнутый  $\lambda$ -терм)**  $M$  - это такой  $\lambda$ -терм, что  $FV(M) = \emptyset$ . Множество всех замкнутых термов обозначается  $\Lambda^0$ .

- $I = \lambda x.x$
- $\omega = \lambda x.xx$
- $\Omega = \omega\omega = (\lambda x.xx)(\lambda x.xx)$
- $K = \lambda xy.x$
- $KI = \lambda xy.y$
- $S = \lambda fgx.fx(gx)$
- $B = \lambda fgx.f(gx)$

# Кодирование Чёрча: булеаны, пары, числа

Булеаны

$K$

$$\text{true} \equiv \lambda t. \lambda f. t, \quad \text{false} \equiv \lambda t. \lambda f. f$$

$$\text{if} \equiv \lambda b. \lambda x. \lambda y. b x y$$

Пары

$$\text{pair} \equiv \lambda x. \lambda y. \lambda p. p x y$$

$$\pi_1 \equiv \lambda p. p(\lambda x. \lambda y. x), \quad \pi_2 \equiv \lambda p. p(\lambda x. \lambda y. y)$$

$KI$

Натуральные (Чёрч)

$$0 \equiv \lambda f. \lambda x. x, \quad 1 \equiv \lambda f. \lambda x. f x, \dots$$

$$\text{succ} \equiv \lambda n. \lambda f. \lambda x. f(n f x)$$

$$\text{plus} \equiv \lambda m. \lambda n. \lambda f. \lambda x. m f (n f x)$$

## Что почитать

- Pierce, гл. 5–7 (untyped  $\lambda$ );
- Software Foundations:  $LF$ , главы о  $\beta$ -редукции.
- Более продвинутое: Barendregt — *The Lambda Calculus*.