

# Нетипизированное $\lambda$ -исчисление

Юрий Литвинов

15.03.2019г

# Лямбда-исчисление

## Математическая основа функционального программирования

- ▶ Формальная система, основанная на  $\lambda$ -нотации, ещё одна формализация понятия «вычисление», помимо машин Тьюринга (и нормальных алгоритмов Маркова, если кто-то про них помнит)
- ▶ Введено Алонзо Чёрчем в 1930-х для исследований в теории вычислимости
- ▶ Имеет много разных модификаций, включая «чистое»  $\lambda$ -исчисление и разные типизированные  $\lambda$ -исчисления
- ▶ Реализовано в языке LISP, с тех пор прочно вошло в программистский обиход (даже анонимные делегаты в C# называют лямбда-функциями, как вы помните)

# Лямбда-нотация

Способ вводить функции, не придумывая для них название каждый раз

$$x \rightarrow t[x] \implies \lambda x. t[x]$$

Например,

$$\lambda x. x$$

$$\lambda x. x^2$$

# Применение функции (или аппликация)

Математически привычно

$$f(x)$$

Но непонятно, о чём идёт речь — о функции  $f$ , принимающей аргумент  $x$ , или о результате применения  $f$  к  $x$ .

В лямбда-исчислении  $f(x)$  обозначается как

$$f\ x$$

При этом принято, что

$$\lambda x.x + y = \lambda x.(x + y), \quad \lambda x.x + y \neq (\lambda x.x) + y$$

Примеры записи:

$$(\lambda x.x^2)\ 5 = 25$$

$$(\lambda x.\lambda y.x + y)\ 2\ 5 = 7$$

# Каррирование (Currying)

В λ-исчислении не нужны функции нескольких переменных:

$$\lambda x. \lambda y. x + y \stackrel{def}{=} \lambda x y. x + y$$

Можно понимать как функцию, которая возвращает функцию:

$$\lambda x. \lambda y. x + y \equiv \lambda x. (\lambda y. x + y)$$

$$\mathbb{R} \rightarrow (\mathbb{R} \rightarrow \mathbb{R})$$

Частичное применение:

$$(\lambda x. \lambda y. x + y) 5 \equiv \lambda x. (x + 5)$$

# $\lambda$ -исчисление как формальная система

Внезапно, математика на парах по проге

Всё, что было выше, хорошо, но неформально. Формализуем, чтобы иметь возможность применять математические методы.

## **Нетипизированное лямбда-исчисление:**

- ▶ Всё —  $\lambda$ -термы (числа и операции вводятся через них)
  - ▶ Не делается различий между данными и функциями, можно применять функцию к функции (вообще говоря, есть только функции, они же являются данными)
- ▶ Процесс вычисления вводится как набор формальных преобразований над  $\lambda$ -термами
  - ▶ **Операционная семантика**

# $\lambda$ -термы

$\lambda$ -терм — это:

- ▶ Переменная:  $v \in V$ , где  $V$  — некоторое множество, называемое множеством переменных
- ▶ Аппликация: если  $A$  и  $B$  —  $\lambda$ -термы, то  $AB$  —  $\lambda$ -терм.
- ▶  $\lambda$ -абстракция: если  $A$  —  $\lambda$ -терм, а  $v$  — переменная, то  $\lambda v.A$  —  $\lambda$ -терм
- ▶ Других способов получить  $\lambda$ -терм нет

# Соглашения об ассоциативности

Чтобы не надо было писать кучу скобок

- ▶ Аппликация левоассоциативна:  $F X Y = (F X) Y$
- ▶ λ-абстракция правоассоциативна:  $\lambda x y.M = \lambda x.(\lambda y.M)$
- ▶ λ-абстракция распространяется вправо настолько, насколько возможно:  $\lambda x.M N = (\lambda x.M N)$



## Свободные и связанные переменные

- ▶ λ-абстракция  $\lambda x. T[x]$  **связывает** переменную  $x$  в терме  $T[x]$
- ▶ Если значение выражения зависит от значения переменной, то говорят, что переменная **свободно** входит в выражение

Пример:

$$\sum_{m=1}^n m = \frac{n(n+1)}{2}$$

Здесь  $n$  входит свободно, а  $m$  связана. Имя связанной переменной можно менять:

$$\int_0^x 2y + a \, dy = x^2 + ax \longrightarrow \int_0^x 2z + a \, dz = x^2 + ax$$

НО

$$\int_0^x 2a + a \, da \neq x^2 + ax$$

## Свободные и связанные переменные, формально

Как обычно, определение рекурсивно по структуре терма:

- ▶  $FV(x) = x$
- ▶  $FV(S\ T) = FV(S) \cup FV(T)$
- ▶  $FV(\lambda x.S) = FV(S) \setminus \{x\}$
- ▶  $BV(x) = \emptyset$
- ▶  $BV(S\ T) = BV(S) \cup BV(T)$
- ▶  $BV(\lambda x.S) = BV(S) \cup \{x\}$

Примеры:

$$S = (\lambda x\ y.x)(\lambda x.z\ x) \Rightarrow FV(S) = z, BV(S) = \{x, y\}$$

# Подстановка

$T[x := S]$  - подстановка в терме  $T$  терма  $S$  вместо всех свободных вхождений переменной  $x$  (например,  $x[x := T] = T$ ).

Проблема:

$$(\lambda y. x + y)[x := y] = \lambda y. y + y$$

Решения:

- ▶ Запретить свободным переменным иметь одинаковые имена и называться так же, как связанные (соглашение Барендрегта)
- ▶ Переименовывать связанные переменные «на лету» перед выполнением подстановки

## Подстановка, формально

- ▶  $x[x := T] = T$
- ▶  $y[x := T] = y$
- ▶  $(S_1 S_2)[x := T] = S_1[x := T] S_2[x := T]$
- ▶  $(\lambda x.S)[x := z] = \lambda x.S$
- ▶  $(\lambda y.S)[x := T] = \lambda y.(S[x := T])$ , если  $y \notin FV(T)$  или  $x \notin FV(S)$
- ▶  $(\lambda y.S)[x := T] = \lambda z.(S[y := z][x := T])$ , иначе ( $z$  при этом выбирается так, что  $z \notin FV(S) \cup FV(T)$ )

## Зачем мы это делали

Можно ввести отношение **равенства** над термами, имеющее физический смысл «термы означают одно и то же» и отношение **редукции**, означающее «термы имеют одинаковое **значение**», что нужно для определения **вычисления** (хотя заметьте, что пока в формальной системе даже понятия «значение» нет).  
Делать это мы будем, определив аксиомы и правила вывода над термами, через **преобразования** термов.

# Преобразования

**α-преобразование** :  $\lambda x. S \rightarrow_\alpha \lambda y. S[x := y]$  при условии, что  $y \notin FV(S)$ .

Даёт возможность переименовывать связанные переменные.

**β-преобразование** :  $(\lambda x. S) T \rightarrow_\beta S[x := T]$ . Определяет процесс вычисления.

**η-преобразование** :  $\lambda x. T x \rightarrow_\eta T$ , если  $x \notin FV(T)$ . Обеспечивает **экстенциональность** — две функции экстенционально эквивалентны, если на всех одинаковых входных данных дают одинаковый результат:

$$\forall x F x = G x$$

## Аксиомы равенства λ-термов

$$\frac{S \rightarrow_{\alpha} T \text{ или } S \rightarrow_{\beta} T \text{ или } S \rightarrow_{\eta} T}{S = T}$$

$$\overline{T = T}$$

$$\overline{S = T}$$

$$\overline{T = S}$$

$$\frac{S = T \wedge T = U}{S = U}$$

$$\overline{S = T}$$

$$\overline{SU = TU}$$

$$\overline{S = T}$$

$$\overline{US = UT}$$

$$\overline{S = T}$$

$$\overline{\lambda x. S = \lambda x. T}$$

## Вычисление, что мы хотим

Очевидно, что равенство — это отношение эквивалентности. Оно «не даёт терять информацию», потому что всегда можно вернуться к исходному терму. А мы хотим вычислять значение терма, то есть всё-таки терять информацию о синтаксисе терма, сохраняя его «смысл». Так что уберём симметричность, получив отношение  **$\beta$ -редукции**, которое уже не эквивалентность и позволяет делать с термом что-то осмысленное.



Аксиомы  $\beta$ -редукции

$$\frac{S \rightarrow_{\alpha} T \text{ или } S \rightarrow_{\beta} T \text{ или } S \rightarrow_{\eta} T}{S \rightarrow_{\beta} T}$$

$$\overline{T \rightarrow_{\beta} T}$$

$$\frac{S \rightarrow_{\beta} T \wedge T \rightarrow_{\beta} U}{S \rightarrow_{\beta} U}$$

$$\frac{S \rightarrow_{\beta} T}{SU \rightarrow_{\beta} TU}$$

$$\frac{S \rightarrow_{\beta} T}{US \rightarrow_{\beta} UT}$$

$$\frac{S \rightarrow_{\beta} T}{\lambda x. S \rightarrow_{\beta} \lambda x. T}$$

# Пример

Редукция не всегда уменьшает размер терма

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

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

$$(\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) (\lambda x. x x x) \rightarrow_{\beta} \dots$$

так что

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

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

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

НО

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

# Редексы

## Reducible expressions

**Редэксом** называется пара термов, в которой можно выполнить подстановку, или выражение вида

$$(\lambda x. S) T$$

По правилу  $\beta$ -редукции

$$(\lambda x. S) T \rightarrow_{\beta} S[x := T]$$

Например,

$$(\lambda f. \lambda x. f \ x \ x) + \rightarrow_{\beta} \lambda x. + \ x \ x$$

Терм без редэков называется термом в **нормальной форме** (он вычислен, его нельзя дальше упростить)

## Стратегии редукции

При выполнении редукции можно выбрать, какой редэкс заменять, это и есть стратегия редукции.

**аппликативная** стратегия — заменяем самый левый редэкс, не содержащий в себе других редэксов (самое маленькое подвыражение)

**нормальная** стратегия — заменяем самый левый самый внешний редэкс

Аппликативная стратегия соответствует передаче параметра по значению (сначала вычисляем параметр, потом передаём его в функцию), нормальная стратегия соответствует передаче параметра по имени (или ленивому вычислению), когда мы откладываем вычисление параметра до последнего, в надежде, что он нам не понадобится.

# Какая стратегия лучше

## Теорема (Карри о нормализации)

*Если у терма есть нормальная форма, то последовательное сокращение самого левого внешнего редекса приводит к ней.*

## Теорема (Чёрча-Россера)

*Если терм  $M$   $\beta$ -редукцией редуцируется к термам  $N$  и  $K$ , то существует терм  $L$  такой, что к нему редуцируются и  $N$ , и  $K$ .*

То есть нормальная форма не всегда есть (см. пример про  $(\lambda x.x x x) (\lambda x.x x x)$ ), но если она есть, её можно получить нормальной стратегией, причём нормальная форма единственная.

# Комбинаторы

**Комбинатор** формально — это  $\lambda$ -терм без свободных переменных.  
 Неформально — функция, которая позволяет комбинировать функции, без упоминания данных.

Известные комбинаторы:

$I \equiv \lambda x. x$  — тождественная функция

$\omega \equiv \lambda s. s s$  — комбинатор самоприменимости

$\Omega \equiv \omega \omega \equiv (\lambda s. s s)(\lambda s. s s)$  — расходящийся комбинатор

$K \equiv \lambda x y. x$  — канцеллятор (первый элемент пары)

$K_* \equiv \lambda x y. y$  — второй элемент пары

$S \equiv \lambda x y z. x z (y z)$  — коннектор

$B \equiv \lambda f g x. f (g x)$  — композиция

# Комбинаторы, примеры

$$I I \equiv (\lambda x.x)(\lambda x.x) \rightarrow_{\beta} \lambda x.x \equiv I$$

$$\begin{aligned} K I &\equiv (\lambda x.\lambda y.x)(\lambda x.x) \rightarrow_{\beta} \\ &\rightarrow_{\beta} \lambda y.(\lambda x.x) \rightarrow_{\alpha} \lambda x.\lambda y.y \equiv K_* \end{aligned}$$

# Комбинатор неподвижной точки

## Теорема (О неподвижной точке)

Для любого  $\lambda$ -терма  $F$  существует неподвижная точка:

$$\forall F \exists X : F X = X$$

## Теорема (О комбинаторе неподвижной точки)

Существует комбинатор неподвижной точки

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

такой, что

$$\forall F \quad F (Y F) = Y F$$

## Доказательство.

$$Y F \equiv (\lambda x. F (x x)) (\lambda x. F (x x)) = F (\lambda x. F (x x)) (\lambda x. F (x x)) = F (Y F)$$



## Зачем это надо

Рекурсия. Проблема  $\lambda$ -исчисления в том, что у функций нет имён, поэтому они не могут вызывать сами себя, вообще.

Например,

$$factorial = \lambda n. if (isZero\ n) 1 (mult\ n (factorial\ (pred\ n)))$$

Но так писать нельзя, *factorial* в правой части. Перепишем, применив  $\eta$ -преобразование:

$$factorial = (\lambda f. \lambda n. if (isZero\ n) 1 (mult\ n (f\ (pred\ n)))) factorial$$

Внезапно,

$$factorial = Y(\lambda f. \lambda n. if (isZero\ n) 1 (mult\ n (f\ (pred\ n))))$$

(ну,  $F(YF) = YF$ , тут *factorial* выступает в роли неподвижной точки, а *F* — штуки в скобках).

# Пример

$$\begin{aligned}
 factorial\ 3 &= (Y\ F)\ 3 \\
 &= F\ (Y\ F)\ 3 \\
 &= if\ (isZero\ 3)\ 1\ (mult\ 3\ ((Y\ F)\ (pred\ 3))) \\
 &= mult\ 3\ ((Y\ F)\ 2) \\
 &= mult\ 3\ (F\ (Y\ F)\ 2) \\
 &= mult\ 3\ (mult\ 2\ ((Y\ F)\ 1)) \\
 &= mult\ 3\ (mult\ 2\ (mult\ 1\ ((Y\ F)\ 0))) \\
 &= mult\ 3\ (mult\ 2\ (mult\ 1\ 1)) \\
 &= 6
 \end{aligned}$$

(очень рекомендую курс “Системы типизации лямбда-исчисления” Дениса Москвина на <https://www.lektorium.tv>, примеры и часть дальнейшего изложения — оттуда)

## Булевы выражения

Пока что на λ-исчислении факториал не написать, нет чисел и *if*-ов. Начнём с булевых выражений:

$$TRUE \equiv \lambda x. \lambda y. x$$

$$FALSE \equiv \lambda x. \lambda y. y$$

Ну и оператор **IF**:

$$IF \equiv \lambda b. \lambda t. \lambda f. b \ t \ f$$

— обратите внимание, булевы константы вводились так, чтобы *IF* получился таким простым

## Булевы операторы

Ввести булевы операторы очень просто через *IF*:

$$AND \equiv \lambda a b. IF\ a\ b\ FALSE$$

$$OR \equiv \lambda a b. IF\ a\ TRUE\ b$$

$$NOT \equiv \lambda b. IF\ b\ FALSE\ TRUE$$

Какова нормальная форма терма *NOT*?

# Ответ

$$\begin{aligned}
 NOT &= \lambda b. IF\ b\ FALSE\ TRUE \\
 &= \lambda b. ((\lambda b'. \lambda t. \lambda f. b' t f)\ b\ (\lambda x. \lambda y. y)\ (\lambda x. \lambda y. x)) \\
 &\rightarrow_{\beta} \lambda b. ((\lambda t. \lambda f. b\ t f)\ (\lambda x. \lambda y. y)\ (\lambda x. \lambda y. x)) \\
 &\rightarrow_{\beta} \lambda b. ((\lambda f. b\ (\lambda x. \lambda y. y)\ f)\ (\lambda x. \lambda y. x)) \\
 &\rightarrow_{\beta} \lambda b. (b\ (\lambda x. \lambda y. y)\ (\lambda x. \lambda y. x))
 \end{aligned}$$

А если  $b$  может быть только  $TRUE$  и  $FALSE$ , всё проще:

$$NOT = \lambda b\ t\ f. b\ f\ t$$

Легко убедиться подстановкой  $TRUE$  и  $FALSE$  в обе формулы

# Нумералы Чёрча

Теория чисел может быть введена через λ-исчисление. Числа вводятся так (нумералы Чёрча):

$$0 \equiv \lambda s z. z$$

$$1 \equiv \lambda s z. s z$$

$$2 \equiv \lambda s z. s (s z)$$

$$3 \equiv \lambda s z. s (s (s z))$$

$$4 \equiv \lambda s z. s (s (s (s z)))$$

# Арифметические операции

$$S \equiv \lambda n. \lambda f. \lambda x. f((n f) x)$$

то есть

$$\begin{aligned} S n &= (\lambda n f x. f(n f x)) n \\ &\rightarrow_{\beta} \lambda f x. f(n f x) \\ &= n + 1 \end{aligned}$$

Сложение:

$$ADD \equiv \lambda m n. \lambda f x. (m f) ((n f) x)$$

или

$$ADD \equiv \lambda m n. (m S) n$$

## Умножение и степень, проверка на 0

$$MUL \equiv \lambda m n.m (ADD n) 0$$

$$EXP \equiv \lambda m n.m (MUL n) 1$$

$$ISZRO \equiv \lambda n.n (\lambda x.FALSE) TRUE$$



# Пары

Конструктор пары:

$$PAIR \equiv \lambda x y f.f x y$$

идея такая же, как у булевых констант и *IF*-а — обернуть значения в аппликацию функции. Конкретная пара:

$$PAIR a b = \lambda f.f a b$$

Проекции:

$$FST \equiv \lambda p.p \ TRUE$$

$$SND \equiv \lambda p.p \ FALSE$$

## Почему

Потому что мы так определили *TRUE* и *FALSE*:

$$\begin{aligned}
 FST (PAIR\ a\ b) &= PAIR\ a\ b\ TRUE \\
 &\equiv (\lambda x\ y\ f.f\ x\ y)\ a\ b\ TRUE \\
 &= TRUE\ a\ b \\
 &= (\lambda x.\lambda y.x)\ a\ b \\
 &= a
 \end{aligned}$$

$$\begin{aligned}
 SND (PAIR\ a\ b) &= PAIR\ a\ b\ FALSE \\
 &\equiv (\lambda x\ y\ f.f\ x\ y)\ a\ b\ FALSE \\
 &= FALSE\ a\ b \\
 &= (\lambda x.\lambda y.y)\ a\ b \\
 &= b
 \end{aligned}$$

## Функция предшествования

Вспомогательные функции:

$$ZP \equiv PAIR\ 0\ 0$$

$$SP \equiv \lambda p. PAIR\ (SND\ p)\ (SUCC\ (SND\ p))$$

- ▶  $ZP$  — ZeroPair
- ▶  $SP\ (PAIR\ i\ j) = PAIR\ j\ (j + 1)$

Функция предшествования:

$$PRED \equiv \lambda m. FST\ (m\ SP\ ZP)$$

# Примитивная рекурсия

$$XZ \equiv \lambda x. PAIR\ x\ 0$$

$$FS \equiv \lambda f\ p. PAIR\ (f\ (FST\ p)\ (SND\ p))\ (SUCC\ (SND\ p))$$

$$REC \equiv \lambda m\ f\ x. FST\ (m\ (FS\ f)\ (XZ\ x))$$

Получаем:

$$(x, 0) \rightarrow$$

$$(f\ x\ 0, 1) \rightarrow$$

$$(f\ (f\ x\ 0)\ 1, 2) \rightarrow$$

$$(f\ (f\ (f\ x\ 0)\ 1)\ 2, 3) \rightarrow \dots$$

В частности,

$$PRED = \lambda m. REC\ m\ (\lambda x\ y. y)\ 0$$

# Списки

$$NIL \equiv \lambda c n. n$$

$$CONS \equiv \lambda e l c n. c e (l c n)$$

Например,

$$[] \equiv NIL \equiv \lambda c n. n$$

$$[5; 3; 2] \equiv CONS\ 5\ (CONS\ 3\ (CONS\ 2\ NIL)) \equiv \lambda c n. c\ 5\ (c\ 3\ (c\ 2\ n))$$

Стандартные функции:

$$EMPTY \equiv \lambda l. l\ (\lambda h t. FALSE)\ TRUE$$

$$HEAD \equiv \lambda l. l\ (\lambda h t. h)\ 0$$

$$TAIL \equiv \lambda l. FST\ (l\ (\lambda a b. PAIR\ (SND\ b)\ (CONS\ a\ (SND\ b)))\ (PAIR\ NIL\ NIL))$$