

Теория типов

Михайлов Максим

10 сентября 2021 г.

Оглавление

Лекция 1	7 сентября	2
1	Лямбда-исчисление	2
1.1	Определение	2
1.2	Булево исчисление	2
1.3	Числа	3
1.4	Типизированное лямбда-исчисление	4
1.5	Y-комбинатор и противоречивость нетипизированного λ -исчисления .	4

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. λ 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 β -редукции, его мы определим позже, общее понимание у нас есть из вводной лекции функционального программирования.

1.2 Булево исчисление

Определим булево исчисление в λ -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.¹ 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.

¹ 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 τ , 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 λ -исчисления

Мы хотим, чтобы \rightarrow_β 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 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 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.