Краткое введение в лямбда-исчисление

Белкин Дмитрий, студент группы 4362
Бертыш Вадим, студент группы 4374

23 декабря 2015

Введение

В 1928 году Давид Гильберт сформулировал так называемую проблему разрешимости, суть которой примерно выражается в следующем. Было необходимо сформулировать такой алгоритм, который на основании формального языка и утверждения, записанного этим языком, поданных на вход, за конечное число шагов приходил к ответу, является это утверждение ложным или истинным. В 1936 году два математика, Алонзо Чёрч и Алан Тьюринг независимо друг от друга опубликовали работы, в которых заявлялось, что для арифметических утверждений такой алгоритм составить невозможно, а потому и в более общем случае эта проблема неразрешима. Впоследствие это утверждение стало известно как теорема Чёрча-Тьюринга и породило как минимум два средства формализации алгоритмов и понятия вычислимости. В случае с Аланом Тьюрингом это была небезызвестная машина Тьюринга. В случае с Алонзо Чёрчем это было лямбда исчсиление. Итак, что же это такое? Для начала опишем его.

Основными объектами лямбда исчисления являются функции. Но что же мы понимаем под функциями? В современной математике под функцией f чаще всего понимают некоторое соответствие элементам области определения X элементов области значения Y, то есть для каждой $f : X \to Y$ верно, что $f \subseteq X \times Y$. Тем не менее, лямбда исчисление - это теория функций как формул. Это система, позволяющая работать с функциями как с выражениями. Давайте для начала вспомним другой хорошо известный язык выражений, арифметику. Арифметические выражения составляются из переменных $(x, y, z \dots)$, чисел $(1, 2, 3 \dots)$, и операторов (“$+$”, “$-$”, “ $\times$” etc.). Выражение вроде $x+y$ представляет из себя *результат* сложения. Главное преимущество этого языка в том, что выражения можно вкладывать одно в другое, не указывая отдельно промежуточные результаты. Так, например, мы пишем

$$A = (x + y) \times z^2$$

а не

$$\text{Пусть } w = x + y, \text{ тогда пусть } u = z^2, \text{ тогда пусть } A = w \times u$$

Использование второго варианта было бы утомительным и неудобным. Лямбда исчисление расширяет идею языка выражений на функции. Так вместо выражения вида

$$\text{Пусть } f \text{ функция } x \mapsto x^2. \text{ Тогда будем считать, что } A = f(5)$$

в лямбда исчислении мы просто пишем

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

Выражение $\lambda x.x^2$ выражает функцию, которая ставит в соответствие прообразу x образ x^2. Как и в арифметике, мы используем скобки, чтобы группировать термы.

Формализация

Множество лямбда-термов строится из бесконечного множества переменных $\mathcal{V}$ использованием аппликации и абстракции:

$$\mathcal{V} = \{v, v', v'', v''' \dots\}$$

При этом, говоря формально:

$$\begin{aligned} x \in \mathcal{V} &\implies x \in \Lambda && \text{(Переменная является лямбда-термом)} \\ M, N \in \mathcal{V} &\implies MN \in \Lambda && \text{(Аппликация является лямбда-термом)} \\ M \in \mathcal{V}, v \in \mathcal{V} &\implies \lambda v.M \in \Lambda && \text{(Абстракция является лямбда-термом)} \end{aligned}$$

Или же, используя БНФ:

$$\begin{aligned} V &::= v | V' \\ \Lambda &::= V | (\Lambda\ \Lambda) | (\lambda V.\Lambda) \end{aligned}$$

В дальнейшем условимся, что:

- $x, y, z, \cdots$ – произвольные переменные.
- $M, N, L, \cdots$ – произвольные лямбда термы.
- Скобки верхнего уровня опускаются.
- Аппликация правоассоциативна, т.е.

$$F\ M_1\ M_2\ \dots\ M_n \equiv (\dots((F\ M_1)\ M_2)\ \dots\ M_n)$$

- Допустима абстракция сразу по нескольким переменным

$$\lambda x_1\ x_2\ \dots\ x_n.M \equiv \lambda x_1.(\lambda x_2.\ \dots\ (\lambda x_n.M))$$

Свободные и связанные переменные

Множество свободных переменных терма N обозначается $FV(N)$ и индуктивно определяется по следующим правилам:

$$\begin{aligned} FV(x) &\equiv \{x\} \\ FV(M\ N) &\equiv FV(M) \cup FV(N) \\ FV(\lambda x.N) &\equiv FV(N) \setminus \{x\} \end{aligned}$$

Мы будем называть переменную связанной, если она не принадлежит множеству свободных. Множество связанных переменных терма N принято обозначать как $BV(N)$ Заметим, что переменная связана, если она образует абстракцию.

Мы будем называть терм M закрытым (или комбинатором), если $FV(M) \equiv \varnothing$. Множество комбинаторов обозначим как Λ°. Существует раздел математики, тесно связанный с λ-исчислением - комбинаторная логика. Она изучает комбинаторы и вычисления, построенные на них. В комбинаторной логике вводится несколько стандартных комбинаторов:

$$\begin{aligned} \boldsymbol{S}xyz &= xz(yz) \\ \boldsymbol{K}xy &= x \\ \boldsymbol{I}x &= x \end{aligned}$$

α-конверсия

Введем на Λ отношение эквивалентности, задаваемое следующим образом:

$$\forall P.P =_\alpha P$$
$$\lambda x.P =_\alpha \lambda y.P[x := y], \text{ если } y \notin FV(P)$$

Это отношение назвается α-эквивалентностью. Также запишем некоторые аксиомы для него:

$$M =_\alpha M$$
$$M =_\alpha N \Rightarrow N =_\alpha M$$
$$M =_\alpha N, N =_\alpha L \Rightarrow M =_\alpha L$$
$$M =_\alpha M' \Rightarrow M\ Z =_\alpha M'\ Z$$
$$M =_\alpha M' \Rightarrow Z\ M =_\alpha Z\ M'$$
$$M =_\alpha M' \Rightarrow \lambda x.M =_\alpha \lambda x.M'$$

Если $M =_\alpha N$, иногда также пишут $\lambda \models M =_\alpha N$

Обобщая определения выше, термы M и N равны (альфа-эквивалентны), если можно получить один из другого, путем замены имен связанных переменных. Любые два равных терма в одинаковом контексте также будут равны.

Сам процесс замены имени называется α-конверсией и определяется следующим образом:

$$\lambda x.M \to_\alpha \lambda y.(M[x := y]), \text{ если } y \notin FV(M) \qquad (\alpha)$$

Подстановки

Результат подстановки N вместо всех свободных вхождений x в M обозначим $M[x := N]$ и определим как:

$$\begin{aligned} x[x := N] &\equiv N \\ y[x := N] &\equiv y \text{ (если } x \neq y) \\ (M_1\ M_2)[x := N] &\equiv ((M_1[x := N])\ (M_2[x := N])) \\ (\lambda y.M)[x := N] &\equiv (\lambda y.M[x := N]) \end{aligned}$$

Условимся, что подстановка всегда выполняется корректно, то есть заменяемая переменная никогда не является связанной ни в каких внутренних термах. Этой ситуации всегда можно избежать заменой имен во внутреннем терме. Например для следующей подстановки предварительно произведем замену имени во внутреннем терме

$$\lambda a.\lambda b.a\ b[a := b] = \lambda a.(\lambda b.a\ b[b := b'])[a := b] = \lambda b.\lambda b'.b\ b'$$

В силу этого условия допускается некая вольность в обращении с подстановкой, что делает рассуждения компактнее без ущерба сути, пусть и с меньшей долей формализма.

Теперь мы готовы описать λ-исчисление как формальную теорию.

β-редукция

Основная схема λ-исчисления

$$\forall M, N \in \Lambda : (\lambda x.M)\ N = M[x := N] \qquad (\beta)$$

Используя аксиому (β), сформулируем несколько служебных лемм:

Лемма 1. *о множественной подстановке*

$$(\lambda x_1\ x_2\ \ldots\ x_n.M)\ X_1\ X_2\ \ldots\ X_n \equiv M[x_1 := X_1][x_2 := X_2]\ldots[x_n := X_n]$$

Доказательство. Пусть $M' = \lambda x_2 \ldots x_n.M$. По аксиоме (β) мы имеем

$$(\lambda x_1.M')\ X_1\ X_2\ \ldots\ X_n \equiv M'[x_1 := X_1]X_2\ \ldots\ X_n$$

Дальнейшее равенство получаем индукцией по связанным переменным. □

Лемма 2. *о порядке подстановки*

$$M[x := N_1][y := N_2] \equiv M[y := N_2][x := N_1[y := N_2]]$$

Доказательство. Индукция по структуре терма M:

- M $\in \mathcal{V}$ По определению подстановки имеем три случая
  - $M = x \implies N_1[y := N_2] \equiv N_1[y := N_2]$
  - $M = y \implies N_2 \equiv N_2\ (x \notin FV(N_2))$
  - $M = z\ (z \neq x, y) \implies z \equiv z$
- M $= \lambda z.M_1$ В силу соглашений, можно предположить, что $z \neq x, y$ и $z \notin FV(N_1) \cup FV(N_2)$

  $$\begin{aligned}(\lambda z.M_1)[x := N_1][y := N_2] &\equiv \lambda z.M_1[x := N_1][y := N_2] \\ &\equiv \lambda z.M_1[y := N_2][x := N_1[y := N_2]] \\ &\equiv (\lambda z.M_1)[y := N_2][x := N_1[y := N_2]]\end{aligned}$$

  Второе равенстро полученно из индукционного предположения
- $M = M_1M_2$ По индукции лемма верна для M_1 и M_2, далнейшее равенство очевидно

□

Однократное применение аксиомы (β) будем называть β-редукцией и обозанчать как $(\to_\beta)$. Место, где можно применить аксиому (β) будем называть редексом. Применение аксиомы (β) ноль или более раз обозначим как $\twoheadrightarrow_\beta$ (транзитивное замакание $\to_\beta$)

Также введем отношение β-эквивалентности, которое обозначим как $=_\beta$

Любое вычисление представляет собой некоторое количество шагов β-редукции.

Вычисления заканчиваются, когда в терме не остается редексов. Будем говорить, что такие термы находятся в нормальной форме.

Стоит заметить, что β-редукция, хотя и называется редукцией, не всегда сокращает терм, и тем более, не всегда делает терм проще. В качестве примера выражения, которое не сокращается при применении β-редукции, можно привести следующий терм:

$$\omega = (\lambda x.xx)(\lambda x.xx)$$

Такая конструкция за один шаг редуцируется в себя. Очевидно, что у ω никакая цепочка преобразований не приведет к нормальной форме, но тогда возникают вопросы, для каких термов существует нормальная форма, зависит ли она от порядка применения (β) и любой ли порядок редукции ведет к нормальной форме. Мы вернемся к этой теме позже, а пока дополним наше λ-исчисление еще одной аксиомой

$$\lambda x.Mx \equiv M \text{ если } x \notin FV(M) \qquad (\eta)$$

Неподвижная точка

На данном этапе мы можем относительно свободно строить различные термы, однако особый подход требуется для описания рекурсивных функций, о чем сейчас и пойдет речь. Для бестипового лямбда-исчисления справедлива следующая теорема:

Теорема о неподвижной точке.

(i) $\forall F.\exists X.(F\ X = X)$

Более того, существует комбинатор, находящий X

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

(ii) $\forall F.(Y\ F = F\ (Y\ F))$

Доказательство.
Определим $W = \lambda x.F(x\ x)$ и $X = W\ W$ тогда

$$X \equiv W\ W \equiv (\lambda x.F(x\ x))W \to_\beta$$
$$F\ (W\ W) \equiv FX$$

Аналогично

$$Y\ F \to_\beta$$
$$(\lambda x.F\ (x\ x))(\lambda x.F\ (x\ x)) \to_\beta$$
$$F\ ((\lambda x.F\ (x\ x))(\lambda x.F\ (x\ x))) \equiv F\ (Y\ F)$$

□

Нормальный и аппликативный порядки редукции

Как вы уже видели, не все термы имеют нормальную форму. Рассмотрим следующий терм:

$$N = (\lambda xy.y)((\lambda x.xx)(\lambda x.xx))M$$

В этом терме два редекса, можно произвести β-редукцию в двух разных местах:

$$N \to_\beta M$$
$$N \to_\beta (\lambda xy.y)((\lambda x.xx)(\lambda x.xx))M$$

В первом случае мы проредуцировали внешний редекс, во втором внутренний (который оказался уже знакомым термом ω, редуцирующимся в себя). Как вы могли заметить, мы не сможем придти к нормальной форме N, редуцируя внутренний терм, когда как единственная редукция внешнего терма приводит N к его нормальной форме. Это значит, что если нормальная форма существует, к ней не обязательно ведет любой порядок редукции. То, в каком порядке мы производим редукцию, определяет стратегию вычислений. Выделяют различные стратегии, мы же затронем две из них: аппликативную (соответствующую энергичным вычислениям в языках программирования) и нормальную (она соответствует ленивым вычислениям).

При аппликативной стратегии, мы редуцируем термы справа нелево, изнутри наружу. Это соответствует вычислению значения аргументов перед вызовом функции.

Нормальная стратегия предполагают редукцию слева направо, снаружи внутрь. Это соответствует отказу от вычисления значений агрументов перед вызовом функции

Существует утверждение(Карри) о том, что если у терма существует нормальная форма, то к ней можно придти при помощи ленивой стратегии вычислений. Не будем приводить доказательство этого факта ввиду его трудоемкости.

Поговорим о лямбда-исчислении со стороны программирования. Для начала построим булеву алгебру на лямбда исчислении.

Пусть **True** и **False** некие лямбда термы, при этом **True** $\neq_\beta$ **False**. Один из возможных способов сделать это следующий

$$\textbf{True} = \lambda ab.a$$
$$\textbf{False} = \lambda ab.b$$

Определим также термы **and**, **or** и **not**, представляющие соответсвующие операции

$$\textbf{and} = \lambda t_1 t_2 ab.(t_1(t_2 ab)b)$$
$$\textbf{or} = \lambda t_1 t_2 ab.(t_1 a(t_2 ab))$$
$$\textbf{not} = \lambda fab.fba$$

Доходчивый читатель сам удостоверится, что такие определения удовлетворяют аксиомам булевой алгебры. Мы можем проверить различные свойства булевой алгебры, к примеру проверим инволюцию отрицания

$$\textbf{not not}\ x \to_\beta \textbf{not}\ \lambda a'b'.xb'a' \to_\beta \lambda a''b''.(\lambda a'b'.ab'a')b''a'' \twoheadrightarrow_\beta \lambda a''b''.xa''b'' \equiv x$$

Стоит также упомянуть условную конструкцию if

$$\textbf{if} = \lambda t.t$$
$$\textbf{if True}\ a\ b \twoheadrightarrow_\beta a$$
$$\textbf{if False}\ a\ b \twoheadrightarrow_\beta b$$

Следующий пункт - натуральные числа Определим натуральные числа следующим образом:

$$\overline{\mathbf{0}} = \lambda fx.x = \textbf{const}$$
$$\overline{\mathbf{1}} = \lambda fx.f\ x = \textbf{id}$$
$$\dots$$
$$\overline{\boldsymbol{n}} = \lambda f.\lambda x.f^n x$$
$$\text{Где } f^n(x) = \underbrace{f(f(f(\dots f}_{n \text{ раз}}(x))))$$

Такое представление чисел назыается нумералами Чёрча Можно показать, что для них выполняются аксиомы Пеано.

Определим также основные арифметические операции. Самой простой из арифметических операций является инкремент, то есть получение следующего числа. Попробуем сформулировать это на языке лямбда исчисления.

$$\mathbf{succ}\ \overline{\boldsymbol{n}} = \overline{\boldsymbol{n+1}}$$
$$\mathbf{succ} = \lambda nfx.f\ (n\ f\ x)$$

Данную запись можно трактовать так: мы применяем f n раз и еще один раз, что и эквивалентно увеличению на единицу для нумералов Чёрча.

Докажем корректность данного определения:

$$\begin{aligned}
\mathbf{succ}\ \overline{\boldsymbol{n}} &\to_\beta \\
\lambda fx.f\ ((\lambda gy.g^n\ y)\ f\ x) &\to_\beta \\
\lambda fx.f\ (\lambda y.f^n\ y)\ x &\to_\beta \\
\lambda fx.f\ (f^n\ x) \equiv \lambda fx.f^{(n+1)}\ x &\equiv \overline{\boldsymbol{n+1}} \\
&\square
\end{aligned}$$

Сложение, умножение и возведение в степень определяются так:

$$\mathbf{add} = \lambda abfx.(b\ f\ (a\ f)\ x)$$
$$\mathbf{mul} = \lambda abf.a\ (b\ f)$$
$$\mathbf{pow} = \lambda ab.b\ a$$

Определить же вычитание значительно сложнее, ведь мы индуктивно строим n-ое число и остановка за m шагов до n – задача не тривиальная. Мы могли бы определить вычитание относительно просто через декремент, но как же тогда определить сам декремент? Идея решения этой проблемы заключается в следующем, мы можем инкрементировать число и как-то хранить предыдущий результат (число до инкремента) в конечном итоге мы получим пару из $\overline{\boldsymbol{n}}$ и $\overline{\boldsymbol{n-1}}$.

Для реализации этого способа, нам необходимо как-то задавать пары. Будем считать парой такой терм, из которого можно получить первый и второй элемент. Очевидно, что нам также понадобится терм, который будет создавать пару из двух других термов. Попробуем описать это в такой форме:

$$\mathbf{mkPair} = \lambda abf.f\ a\ b$$
$$\mathbf{fst} = \lambda p.p\ \mathbf{True}$$
$$\mathbf{snd} = \lambda p.p\ \mathbf{False}$$

Здесь **mkPair** создает пару, а **fst** и **snd** получают первый и второй элементы пары соответственно. Очевидно также, что получившаяся конструкция задаёт упорядоченные пары, так как мы отличаем первый и второй элемент.

Далее определим “инкремент” пары

$$\mathbf{succP} = \lambda p.\mathbf{mkPair}\ (\mathbf{snd}\ p)\ (\mathbf{succ}\ (\mathbf{fst}\ p))$$

Ну, а имея эту функцию не составит труда определить декремент

$$\mathbf{prev} = \lambda n.\mathbf{fst}\ (n\ \mathbf{succP}\ (\mathbf{mkPair}\ \overline{\mathbf{0}}\ \overline{\mathbf{0}}))$$

и вычитание

$$\mathbf{sub} = \lambda ab.b\ \mathbf{prev}\ a$$