Лекция 1. Основы лямбда-исчисления

Пережогин А.С.

11 октября 2012 г.

Лямбда-исчисление основано на формальной нотации лямбда-терма, составляемого из переменных и некоторого фиксированного набора констант с использованием операции применения функции и лямбда-абстрагирования. Сказанное означает, что все лямбда-выражения можно разделить на четыре категории:

1. **Переменные:** обозначаются произвольными строками, составленными из букв и цифр.
2. **Константы:** также обозначаются строками; отличие от переменных будем определять из контекста.
3. **Комбинации:**, т.е. применения функции S к аргументу T; и S и T могут быть произвольными лямбда-термами. Комбинация записывается как $S\ T$.
4. **Абстракции** произвольного лямбда-терма S по переменной x, обозначаемые как $\lambda x.S$.

Таким образом, лямбда-терм определяется рекурсивно и его грамматику можно определить в виде следующей формы Бэкуса-Наура:

$$Exp = Var \mid Const \mid Exp\ Exp \mid \lambda\ Var\ .\ Exp$$

В соответствие с этой грамматикой лямбда-термы представляются в виде синтаксических деревьев, а не в виде последовательности символов. Отсюда следует, что соглашения об ассоциативности операции применения функции, эквивалентность выражений вида $\lambda x\, y.S$ и $\lambda x.\lambda y.S$, неоднозначность в именах констант и переменных проистекают только из необходимости представления лямбда-термов в удобном человеку виде, и не являются частью формальной системы.

В данном разделе мы формализуем данное ранее интуитивное представление о свободных и связанных переменных. Множество свободных переменных $FV(S)$ лямбда-терма S можно определить рекурсивно следующим образом:

$$\begin{aligned} FV(x) &= \{x\} \\ FV(c) &= \emptyset \\ FV(S\ T) &= FV(S) \cup FV(T) \\ FV(\lambda x.S) &= FV(S) \backslash \{x\} \end{aligned}$$

Аналогично множество связанных переменных $BV(S)$ определяется следующими формулами:

$$\begin{aligned} BV(x) &= \emptyset \\ BV(c) &= \emptyset \\ BV(S\ T) &= BV(S) \cup BV(T) \\ BV(\lambda x.S) &= BV(S) \cup \{x\} \end{aligned}$$

Здесь предполагается, что c — некоторая константа.
Для терма $S = (\lambda x\, y.x)\ (\lambda x.z\ x)$ можно показать, что $FV(S) = \{z\}$ и $BV(S) = \{x, y\}$.

Интуитивно ясно, что применение терма $\lambda x.S$ как функции к аргументу T дает в результате терм S, в котором все свободные вхождения переменной x заменены на T. Как ни странно, формализовать это интуитивное представление оказывается нелегко. Будем обозначать операцию подстановки терма S вместо переменной x в другом терме T как $T[x := S]$. Также, как и в определение свободных и связанных переменных, правила подстановки также можно определить рекурсивно. Трудность состоит в том, что

необходимо наложить дополнительные ограничения, позволяющие избегать конфликта в именах переменных.

$$
\begin{aligned}
x[x := T] &= T \\
y[x := T] &= y, \text{ если } x \neq y \\
c[x := T] &= c \\
(S_1\ S_2)[x := T] &= S_1[x := T]\ S_2[x := T] \\
(\lambda x.S)[x := T] &= \lambda x.S \\
(\lambda y.S)[x := T] &= \lambda y.(S[x := T]), \text{ если } x \neq y \text{ и } x \notin FV(S), \text{ либо } y \notin FV(\\
(\lambda y.S)[x := T] &= \lambda z.(S[y := z][x := T]) \text{ иначе, где } z \notin FV(S) \cup FV(T)
\end{aligned}
$$

Лямбда-исчисление основано на трех операциях конверсии, которые позволяют переходить от одного терма к другому, эквивалентному ему. По сложившейся традиции эти конверсии обозначают греческими буквами α, β и η. Они определяются следующим образом:

- α-конверсия: $\lambda x.S \xrightarrow{\alpha} \lambda y.S[x := y]$ при условии, что $y \notin FV(S)$. Например, $\lambda u.u\ v \xrightarrow{\alpha} \lambda w.w\ u$.
- β-конверсия: $(\lambda x.S)\ T \xrightarrow{\beta} S[x := T]$.
- η-конверсия: $\lambda x.T\ x \xrightarrow{\eta} T$, если $x \notin FV(T)$. Например, $\lambda u.v\ u \xrightarrow{\eta} v$.

Для нас наиболее важна β-конверсия, поскольку она соответствует вычислению значения функции от аргумента. α-конверсия является вспомогательным механизмом для того, чтобы изменять имена связанных переменных, а η-конверсия интересна в основном при рассмотрении лямбда-исчисления с точки зрения логики, а не программирования.

Используя введенные правила конверсии, можно формально определить понятие равенства лямбда-термов. Два терма равны, если от одного из них можно перейти к другому с помощью конечной последовательности конверсий. Определим понятие равенства следующими выражениями, в которых горизонтальные линии следует понимать как "если утверждение над чертой выполняется, то выполняется и утверждение под ней":

$$\frac{S \xrightarrow{\alpha} T \text{ или } S \xrightarrow{\beta} T \text{ или } S \xrightarrow{\eta} T}{S = T} \quad (1)$$

$$\frac{}{T = T} \quad (2)$$

$$\frac{S = T}{T = S} \quad (3)$$

$$\frac{S = T \text{ и } T = U}{S = U} \tag{4}$$

$$\frac{S = T}{S\ U = T\ U} \tag{5}$$

$$\frac{S = T}{U\ S = U\ T} \tag{6}$$

$$\frac{S = T}{\lambda x.S = \lambda x.T} \tag{7}$$

Следует отличать понятие равенства, определяемое этими формулами, от понятия синтаксической эквивалентности, которую мы будем обозначать специальным символом $\equiv$. Например,

$\lambda x.x \not\equiv \lambda y.y$, но $\lambda x.x = \lambda y.y$. Часто можно рассматривать синтаксическую эквивалентность термов с точностью до α-конверсий. Такую эквивалентность будем обозначать символом $\equiv_\alpha$. Это отношение определяется так же, как равенство лямбда-термов, за тем исключением, что из всех конверсий допустимы только α-конверсии. Таким образом, $\lambda x.x \equiv_\alpha \lambda y.y$.

η-конверсия в лямбда-исчислении выражает собой принцип *экстенсиональности*. В общефилософском смысле два свойства называются экстенсионально эквивалентными, если они принадлежат в точности одним и тем же объектам. В математике, например, принят экстенсиональный взгляд на множества, т.е. два множества считаются одинаковыми, если они содержат одни и те же элементы. Аналогично мы говорим, что две функции равны, если они имеют одну и ту же область определения, и для любых значений аргумента из этой области определения вычисляют один и тот же результат. Вследствие наличия η-конверсии определенное нами выше отношение равенства лямбда-термов экстенсионально. Действительно, если $f\ x$ и $g\ x$ равны для любого x, то в частности $f\ y = g\ y$, где y не является свободной переменной ни в f, ни в g. Следовательно, по последнему правилу в определении равенства лямбда-термов, имеем $\lambda y.f\ y = \lambda y.g\ y$. Теперь, если несколько раз применить η-конверсию, можно получить, что $f = g$. И обратно,

экстенсиональность дает то, что каждое применение η-конверсии действительно приводит к равенству, поскольку по правилу β-конверсии $(\lambda x.T\ x)\ y = T\ y$ для любого y, если x не свободна в T.

Отношение равенства лямбда-термов, разумеется, симметрично. Хотя оно хорошо отражает понятие эквивалентности лямбда-термов, с вычислительной точки зрения более интересно будет рассмотреть асимметричный вариант. Определим отношение редукции

(обозначаемое символом $\longrightarrow$) следующим образом:

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

$$\frac{}{T \longrightarrow T}$$

$$\frac{S \longrightarrow T \text{ и } T \longrightarrow U}{S \longrightarrow U}$$

$$\frac{S \longrightarrow T}{S\ U \longrightarrow T\ U}$$

$$\frac{S \longrightarrow T}{U\ S \longrightarrow U\ T}$$

$$\frac{S \longrightarrow T}{\lambda x.S \longrightarrow \lambda x.T}$$

Давайте отвлечемся от теоретических рассуждений и вспомним важность рассматриваемых вопросов для функционального программирования. Функциональная программа представляет собой *выражение*, и выполнение ее означает вычисление этого выражения. Употребляя введенные термины, можно сказать, что мы начинаем с соответствующего терма и последовательно применяем редукции до тех пор, пока это возможно. Но какую же именно редукцию применять на каждом конкретном шаге? Отношение редукции не детерминистично, т.е. для некоторого терма t существует несколько различных t_i, таких что $t \longrightarrow t_i$. Иногда выбор между ними означает выбор между конечной и бесконечной последовательности редукций, т.е. между завершением и зацикливанием программы. Например, если мы начнем редукцию с самого внутреннего *редекса*[1] в

следующем примере, мы получим бесконечную последовательность редукций:

$$\begin{aligned} & (\lambda x.y)\ ((\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)) \\ \to\ & (\lambda x.y)\ ((\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)) \\ \to\ & (\lambda x.y)\ ((\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)) \\ \to\ & \ldots \end{aligned}$$

Однако если мы начнем с самого внешнего редекса, то мы сразу получим:

$$(\lambda x.y)\ ((\lambda x.x\ x\ x)\ (\lambda x.x\ x\ x)) \to y,$$

и больше нельзя применить никакую редукцию.
Следующая теорема утверждает, что наблюдение, сделанное в предыдущем примере, верно и в более общем смысле.

Theorem

Если $S \to T$, где T находится в нормальной форме, то последовательность редукций, начинающаяся с S, и заключающаяся в том, что для редукции всегда выбирается самый левый и самый внешний редекс, гарантированно завершится и приведет терм в нормальную форму.

Понятие «самого левого и самого внешнего» редекса можно определить индуктивно: для терма $(\lambda x.S)\ T$ это будет сам терм; для любого другого терма $S\ T$ это будет самый левый и самый внешний редекс в S, и для абстракции $\lambda x.S$ это будет самый левый самый внешний редекс в S. В терминах нашего конкретного синтаксиса мы всегда редуцируем редекс, чей символ λ находится левее всего. Следующая теорема, известная как теорема Черча–Россера, утверждает, что если мы начнем с терма T и проведем две произвольные конечные последовательности редукций, всегда будут

существовать еще две последовательности редукций, которые приведут нас к одному и тому же терму (хотя он может и не находиться в нормальной форме).

Theorem

Если $t \longrightarrow s_1$ *и* $t \longrightarrow s_2$, *то существует терм* u, *такой, что* $s_1 \longrightarrow u$ *и* $s_2 \longrightarrow u$.

Эта теорема имеет несколько важных следствий.

Corollary

Если $t_1 = t_2$, *то существует терм* u *такой, что* $t_1 \longrightarrow u$ *и* $t_2 \longrightarrow u$.

Corollary

Если $t = t_1$ и $t = t_2$, где t_1 и t_2 находятся в нормальной форме, то $t_1 \equiv_\alpha t_2$, т.е. t_1 и t_2 равны с точностью до переименования переменных.

Следовательно, нормальная форма, если она существует, единственна с точностью до α-конверсии.
С вычислительной точки зрения это означает следующее. В некотором смысле, стратегия редуцирования самого левого самого внешнего редекса (будем называть ее *нормализованной стратегией*) является наилучшей, поскольку она приведет к результату, если он достижим с помощью какой-либо стратегии. С другой стороны, *любая* завершающаяся последовательность редукций всегда приводит к одному и тому же результату. Более того, никогда не поздно прекратить выполнять редукции по заданной стратегии и вернуться к нормализованной стратегии.

[1] От англ. redex (REDucible EXpression)