1 Функции

1.1 Процесс переписывания

Когда мы переписываем λ-терм, возникает естественный вопрос: бесконечен ли этот процесс и если нет, то каков конечный результат последовательности таких редукций? Ответ: он может быть как конечным так и бесконечным, в зависимости от терма.

Пример конечного переписывания

Рассмотрим терм $((\lambda x.(yx))z)$. Он может быть переписан в (yz) за один шаг, и никакие последующие редукции невозможны.

Пример бесконечного переписывания

Рассмотрим терм $(\lambda x.(xxx))\lambda x.(xxx)$.

$$(\lambda x.(xxx))\lambda x.(xxx) \Rightarrow ((\lambda x.(xxx))\lambda x.(xxx))\lambda x.(xxx) \Rightarrow$$

$$\Rightarrow (((\lambda x.(xxx))\lambda x.(xxx))\lambda x.(xxx))\lambda x.(xxx) \Rightarrow \cdots$$

таким образом процесс переписывания бесконечен.

1.2 Стратегия редукции

Дан λ-терм t, правила переписывания (редукции) могут применяться к нему в разном порядке для различных редексов. Следовательно, будут получены разные промежуточные результаты в процессе переписывания, в зависимости от порядка, в котором производится редукция над соответствующими редексами.

Стратегии редукции

Две основные стратегии редукции:

- **вызов по значению**: в любом терме вида $((\lambda x.t)s)$ сначала s сводится к s', и только после этого к нему применяется β-редукция и результат сводится к $t[x = s']$.

- **вызов по имени**: к любому терму вида $((\lambda x.t)s)$ сначала применяется β-редукция, а затем результат сводится к $t[x = s]$.

Могут быть и другие, более сложные стратегии.

1.3 Нормальная форма

Определение

λ-терм t находится в **нормальной форме**, если он не содержит подтерма s, такого, что существует некоторый α-эквивалентный к s терм s', образующий β или η редекс в t.

Практический смысл нормальной формы вполне понятен: дальнейшая редукция терма в нормальной форме невозможна, поскольку он не имеет редексов.

Примеры нормальных форм

- $I = \lambda x.x$ находится в нормальной форме
- $(f(tsr))$ находится в нормальной форме
- $(f((\lambda x.(gxh))sr))$ не находится в нормальной форме, потому что он имеет редекс $(\lambda x.(gxh))s$

1.4 Приведение к нормальной форме

Целью переписывания терма является достижение нормальной формы, такой, что последующие редукции становятся невозможны.

Определение

Дан терм t, можно сказать, что терм s является его нормальной формой, тогда и только тогда, когда

- $t \Rightarrow s$
- s находится в нормальной форме.

Замечание

Для данного терма t нормальная форма *может не* существовать. В качестве примера такого терма возьмём $(\lambda x.(xxx))\lambda x.(xxx)$.

Теорема

Для данного терма t вопрос, имеет ли он нормальную форму, является **не разрешимым**, т.е. не существует алгоритма, позволяющего ответить на этот вопрос.

1.5 Достижимость нормальной формы

Даже если для определённого терма существует нормальная форма, её невозможно достичь, используя некоторые стратегии.

Пример использования различных стратегий: вызов по значению и вызов по имени

Рассмотрим терм:

$$(\lambda xy.y)((\lambda x.(xxx))\lambda x.(xxx))$$

Тогда если использовать стратегию вызова по значению, то вычисление первого из аргументов λ-терма $\lambda xy.y$ никогда не закончится. Но если использовать стратегию вызова по имени, можно сразу получить результат в нормальной форме:

$$\lambda y.y$$

Отметим, что стратегия вызова по имени может привести к некоторому увеличению количества редукций, необходимых для приведения терма к нормальной форме. Чтобы показать это, рассмотрим терм:

$$(\lambda x.(xxx))(s) \Rightarrow (sss)$$

1.6 Теорема Черча-Россера

Вопрос: дан λ-терм t, может ли быть так, что существует два *различных* терма в нормальной форме s' и s'' таких, что $t \Rightarrow s'$ и $t \Rightarrow s''$? Ответ дает теорема:

Теорема (Черча-Россера)

Дан некоторый λ-терм t, если для каких-либо термов s' и s'' верно, что $t \Rightarrow s'$ и $t \Rightarrow s''$, то существует такой λ-терм q, что $s' \Rightarrow q$ и $s'' \Rightarrow q$.

В действительности, теорема Чёрча-Россера говорит нам, что порядок, в котором мы применяем редукции к терму, в некотором смысле, не имеет значения: из любого множества промежуточных λ-термов в процессе переписывания можно получить точно такой же терм.

Следствие

Для любого λ-терма t его нормальная форма единственна с точностью до α-эквивалентностей.

1.7 Эквивалентность λ-термов

Определение

Введём **эквивалентность** на λ-термах в виде следующего отношения. Даны два λ-терма t и s, можно сказать, что они эквивалентны, и записать это следующим образом $t \equiv s$, тогда и только тогда, когда существует некоторый λ-терм q такой, что

$$t \Rightarrow q \text{ и } s \Rightarrow q$$

Отметим, что в случае, когда существует нормальная форма n λ-терма t, из эквивалентности $t \equiv s$ следует, что нормальная форма s также существует и совпадает с n с точностью до α-эквивалентности (т.е. переименования связанных переменных).

1.8 Комбинаторное исчисление

Напомним, что **комбинатор** - это λ-терм без констант и свободных переменных.

SKI - комбинаторный базис

Следующие три комбинатора называются **комбинаторным базисом**:

- $I = \lambda x.x$

- $K = \lambda xy.x$
- $S = \lambda xyz.(xz)(yz)$

Комбинаторный терм определяется по индукции:

- комбинатор из комбинаторного базиса I, K, S является комбинаторным термом.
- если a, b - два комбинаторных терма, то (ab) также является комбинаторным термом.

Таким образом, в **комбинаторном исчислении** используется только один оператор: аппликация, без оператора абстракции и каких-либо переменных.

1.9 Редукции в комбинаторном исчислении

Комбинаторное исчисление имеет собственный набор редукций:

Комбинаторные редукции

Следующие правила используются для переписывания комбинаторных термов:

- $(Ia) \Rightarrow_c^I a$
- $(Kab) \Rightarrow_c^K a$
- $(Sabc) \Rightarrow_c^S ((ac)(bc))$

Дальнейшее расширение отношений $\Rightarrow_c^*$ на общее отношение возможности переписывания $\Rightarrow_c$ осуществляется стандартным образом.

1.10 Полнота комбинаторного исчисления

Теорема (полнота комбинаторного исчисления)

Для любого комбинатора c существует такой комбинаторный терм T что

$$c \equiv T$$

Доказательство

Дан комбинатор s, построим соответствующий комбинаторный терм $C(s)$ по индукции:

- $C(x) = x$, если $s = x$ - переменная,
- $C((st)) = (C(s)C(t))$, аппликация
- $C(\lambda x.x) = I$ для любой переменной x
- $C(\lambda x.y) = Ky$, если $x \neq y$
- $C(\lambda x.\lambda ys) = C(\lambda x.C(\lambda y.s))$
- $C(\lambda x.(st)) = SC(\lambda x.s)C(\lambda x.t)$

Этот алгоритм называется *исключением абстракции*. Теперь, индукцией по строению λ-терма докажем, что $C(s) \equiv s$. Случаи $C((st)) = (C(s)C(t))$ и $C(\lambda x.\lambda ys) = C(\lambda x.C(\lambda y.s))$ доказываются непосредственно по предположению индукции. Пусть $s \equiv C(s)$ и $t \equiv C(t)$, т.е. $C(s), s \Rightarrow p$ и $C(t), t \Rightarrow q$ для некоторых p и q. Тогда

$$C(st) = C(s)C(t) \Rightarrow pC(t) \Rightarrow (pq)$$

. Рассмотрим случай $C(\lambda x.y) = Ky$, когда $x \neq y$. Действительно:

$$Ky = (\lambda a.(\lambda b.a))y \Rightarrow_\beta \lambda b.y \Rightarrow_\alpha \lambda x.y$$

. Последний случай, если $C(s), s \Rightarrow p$ и $C(t), t \Rightarrow q$ для некоторых p и q:

$$SC(\lambda x.s)C(\lambda x.t) = (\lambda xyz.xz(yz))\lambda x.p\lambda x.q \Rightarrow$$

$$\Rightarrow \lambda z.((\lambda x.p)z)((\lambda x.q)z) \Rightarrow$$

$$\Rightarrow \lambda z.(p[x = z]q[x = z]) \Rightarrow_\alpha \lambda x.(pq) \Leftarrow \lambda x.(st)$$

.

Эта теорема означает, что комбинаторного базиса I, K, S достаточно для получения всех комбинаторов, выражаемых в λ-исчислении, используя только оператор аппликации.

Замечание

На самом деле достаточно рассматривать только K и S в качестве комбинаторного базиса, потому что можно выразить I как комбинаторный терм от K и S:

$$I \equiv (SKK)$$

1.11 Одноточечный базис комбинаторного исчисления

X комбинатор

Рассмотрим комбинатор $X = \lambda x.xSK$:

- $X(X(XX)) \Rightarrow K$
- $X(X(X(XX))) \Rightarrow S$

Таким образом единственного комбинатора X достаточно для получения всех остальных комбинаторов при помощи аппликации.

X' комбинатор

Рассмотрим комбинатор $X' = \lambda x.xKSK$:

- $(X'X')X' \Rightarrow K$
- $X'(X'X') \Rightarrow S$

Таким образом существует еще один комбинатор X, которого достаточно для получения всех остальных комбинаторов при помощи аппликации.

Эти примеры показывают, что существует 1-точечный (элементный) базис комбинаторных термов.

1.12 Комбинатор неподвижной точки (Y комбинатор)

Y комбинатор

Рассмотрим комбинатор $Y = \lambda h.(\lambda x.h(xx))(\lambda x.h(xx))$: для любого x верно следующее:

$$Yx \equiv x(Yx)$$

Этот комбинатор называется комбинатором **неподвижной точки**.

Проверим, что $a(Ya)$ действительно сводится к Ya:

$$(1)\ a(Ya) = a((\lambda h.(\lambda x.h(xx))(\lambda x.h(xx)))a) \Rightarrow$$

$$a((\lambda x.a(xx))\lambda x.a(xx))$$

$$(2)\ Ya = (\lambda x.a(xx))\lambda x.a(xx) \Rightarrow$$

$$a((\lambda x.a(xx))\lambda x.a(xx))$$

Итак, оба Ya и $a(Ya)$ сводятся к одному и тому же λ-терму, следовательно, они эквивалентны.

1.13 Числа Чёрча

Числа Чёрча

Закодируем натуральные числа комбинаторами (функциями) следующим образом:

- $\underline{0} = \lambda fx.x$
- $\underline{1} = \lambda fx.f(x)$
- $\underline{2} = \lambda fx.f(f(x))$
- $\ldots$
- $\underline{n} = \lambda fx.\underbrace{f(f(\ldots f(}_{n}x)\ldots))$

Инкремент

Если определить комбинатор **инкремента** как: $SUCC = \lambda nfx.f(nfx)$, то

$$SUCC\underline{n} = \underline{n+1}$$

1.14 Арифметика на числах Чёрча

Сложение, умножение, возведение в степень

Если определить сложение, умножение и возведение в степень как

- $PLUS = \lambda mnfx.m\ f\ (n\ f\ x)$
- $MULT = \lambda mnfx.m\ (n\ f)\ x$
- $EXP = \lambda mnfx.(m\ n)\ f\ x$

то

- $PLUS\ \underline{n}\ \underline{m} = \underline{n+m}$
- $MULT\ \underline{n}\ \underline{m} = \underline{n \cdot m}$
- $EXP\ \underline{n}\ \underline{m} = \underline{n^m}$

1.15 Декремент и вычитание

Декремент, вычитание

Если определить декремент и вычитание как

- $PRED = \lambda nf.n\ (\lambda g.\lambda h.h\ (g\ f))\ (\lambda u.x)\ (\lambda u.u)$
- $SUB = \lambda m.(n\ PRED)\ m$

то

- $PRED\ \underline{0} = \underline{0}$
- $PRED\ \underline{n} = \underline{n-1}$ при $n > 0$
- $SUB\ \underline{n}\ \underline{m} = \underline{n-m}$ при $n \geq m$
- $SUB\ \underline{n}\ \underline{m} = \underline{0}$ при $n < m$

1.16 Логические операторы

Логические константы и операторы

Определим логические константы и операторы следующим образом:

- $TRUE = \lambda x.\lambda y.x$
- $FALSE = \lambda x.\lambda y.y$ что равно $\underline{0}$ в числах Чёрча
- $AND = \lambda p.\lambda q.p\ q\ p$
- $OR = \lambda p.\lambda q.p\ p\ q$
- $NOT = \lambda p.p\ FALSE\ TRUE$
- $IF = \lambda p.\lambda a.\lambda b.p\ a\ b$
- $ISZERO = \lambda n.n(\lambda x.FALSE)\ TRUE$
- $LEQ = \lambda m.\lambda n.ISZERO\ (SUB\ m\ n)$

1.17 Пары и списки

Пары и списки

Определим пары и списки следующим образом:

- $PAIR = \lambda x.\lambda y.\lambda f.f\ x\ y$
- $FIRST = \lambda p.p\ TRUE$
- $SECOND = \lambda p.p\ FALSE$
- $NIL = \lambda x.TRUE$
- $NULL = \lambda p.p\ (\lambda x.\lambda y.FALSE)$

Список можно представить в виде пары, содержащей **голову** и **хвост** списка. NIL означает пустой список.

2 Рекурсия

2.1 Y комбинатор и рекурсия

Рекурсия в λ-исчислении

Напомним, что Y-комбинатор: $Y = \lambda h.(\lambda x.h(xx))(\lambda x.h(xx))$ является комбинатором неподвижной точки $Yf \equiv f(Yf)$. Используя Y можно смоделировать рекурсивный вызов функции. В качестве примера возьмем функцию, вычисляющую факториал. Определим

$$F = \lambda fx.(IF\ (ISZERO\ x)\ \underline{1}\ (MULT\ x\ (f\ (PRED\ x))))$$

Тогда функция $FACT = Y\ F$ будет представлять факториал:

$$FACT\underline{n} = \underline{n!}$$

Посмотрим, как работает рекурсия с Y-комбинатором на примере факториала. Для этого вычислим 3!.

$$\begin{gathered}
FACT\ \underline{3} = Y\ F\ \underline{3} \Rightarrow F\ (Y\ F)\ \underline{3} \Rightarrow \\
(\lambda fx.(IF\ (ISZERO\ x)\ \underline{1}\ (MULT\ x\ (f\ (PRED\ x))))\ (Y\ F)\ \underline{3} \Rightarrow \\
IF\ (ISZERO\ \underline{3})\ \underline{1}\ (MULT\ \underline{3}\ ((Y\ F)\ (PRED\ \underline{3}))) \Rightarrow \\
MULT\ \underline{3}\ (Y\ F\ (PRED\ \underline{3})) \Rightarrow MULT\ \underline{3}\ (F\ (Y\ F)\ \underline{2}) \Rightarrow \\
MULT\ \underline{3}\ (\lambda fx.(IF\ (ISZERO\ x)\ \underline{1}\ (MULT\ x\ (f\ (PRED\ x))))\ (Y\ F)\ \underline{2}) \Rightarrow \\
MULT\ \underline{3}\ (IF\ (ISZERO\ \underline{2})\ \underline{1}\ (MULT\ \underline{2}\ ((Y\ F)\ (PRED\ \underline{2})))) \Rightarrow \\
MULT\ \underline{3}\ (MULT\ \underline{2}\ (F\ (Y\ F)\ \underline{1})) \Rightarrow \\
MULT\ \underline{3}\ (MULT\ \underline{2}\ (IF\ (ISZERO\ \underline{1})\ \underline{1}\ (MULT\ \underline{1}\ ((Y\ F)\ (PRED\ \underline{1}))))) \Rightarrow \\
MULT\ \underline{3}\ (MULT\ \underline{2}\ (MULT\ \underline{1}\ (F\ (Y\ F)\ \underline{0}))) \Rightarrow \\
MULT\ \underline{3}\ (MULT\ \underline{2}\ (MULT\ \underline{1}\ \underline{1})) \Rightarrow \underline{6}
\end{gathered}$$

2.2 Z комбинатор и вычисление вызовов по значению

К сожалению, Y комбинатор не будет работать для рекурсии, если использовать вызов по значению в качестве стратегии редукции. Причина: когда мы раскрываем λ-терм $Y\ F\ n$, мы можем применить β-редукцию к редексу $Y\ F$ и получить $F\ (Y\ F)\ n$, затем мы должны раскрыть внутренний аргумент $(Y\ F)$ и получить $F\ (F\ (Y\ F))\ n$ и так далее, следовательно, мы получаем бесконечную последовательность редукций.

Z комбинатор

Чтобы избавиться от бесконечного вычисления первого аргумента в $Y\ F\ n$ можно переписать Y-комбинатор используя обратную η-редукцию:

$$Z = \lambda f.(\lambda x.f(\lambda v.xxv))(\lambda x.f(\lambda v.xxv))$$

Так как $\lambda v.xxv = \lambda v.((xx)v) \Rightarrow_\eta (xx)$, этот λ-терм эквивалентен Y, следовательно, он также является комбинатором неподвижной точки для любого аргумента. Отличие от Y комбинатора состоит в процессе редукции:

$$Z\ f \Rightarrow f\ (\lambda v.(\lambda x.f(\lambda w.xxw))(\lambda x.f(\lambda w.xxw))v)$$

Мы не можем применить β-редукцию к аргументу f (так как он содержит λ).