Лямбда-исчисление

Денис Москвин

31 августа 2020 г.

Содержание

1 Бестиповая система

Лямбда-исчисление представляет собой формализм, проистекающий из идеи представить вычисление выражения как подстановку в определение функции фактических значений аргумента вместо формальных параметров [9, 2, 5, 3]. Например, имея функцию, вычисляющую значение выражения $3^n + n$ по переданному в нее аргументу n, нотация

$$\lambda n.\, 3^n + n,$$

мы хотим уметь передавать ей фактическое значение аргумента, например, 2, нотация

$$(\lambda n.\, 3^n + n)\, 2,$$

и производить шаг вычисления, выполняя подстановку:

$$(\lambda n.\, 3^n + n)\, 2 \longrightarrow 3^2 + 2.$$

Три эти конструкции иллюстрируют три ключевых понятия лямбда исчисления: *лямбда-абстракцию*, *аппликацию* и *редукцию*. Лямбда-абстракция описывает синтаксис определения функции на основе параметризованного выражения, представляющего ее тело. Аппликация задает синтаксис применения функции к ее фактическим аргументам. Редукция определяет отношение вычисления, основывающееся на подстановке фактических параметров вместо формальных.

Аппликация, то есть применение функции F к аргументу X, в лямбда-исчислении записывается как $F\,X$. Скобки вокруг аргумента не ставятся, они используются исключительно для группировки[1]. Отметим, что для выполнения редукции необходима аппликация, в которой слева стоит лямбда-абстракция. Такая аппликация носит название *редекса*.

Функции нескольких переменных в лямбда-исчислении не требуют специального синтаксиса. Имея выражение $2m+3n$ с двумя параметрами мы можем абстрагироваться по каждому из них получив выражения

$$\lambda m.\, 2m + 3n,$$
$$\lambda n.\, 2m + 3n.$$

Первое выражение остается параметризованным переменной n. Говорят, что оно содержит свободную переменную n и связанную m. Второе, наоборот, содержит свободную m и связанную n. Мы можем продолжить процесс лямбда-абстрагирования по свободным переменным, абстрагируясь, например, по m в $\lambda n.\, 2m+3n$. Получится выражение, в котором все переменные связаны:

$$\lambda m\,.(\lambda n.\, 2m + 3n).$$

[1] То есть не будет ошибкой написать $F\,(X)$ или $(F)\,X$ или $(F)\,(X)$ или даже $((F)\,(X))$, но это будет одно и то же выражение. Однако если на месте F или X будут стоять выражения, имеющие внутреннюю структуру, скобки могут потребоваться.

Это выражение и представляет собой функцию двух переменных. Для его вычисления нам следует передать туда два аргумента, причем последовательно. Если, например, мы хотим вычислить значение этого выражения при $m = 15$ и $n = 4$, мы должны записать две аппликации

$$((\lambda m.(\lambda n.\, 2m + 3n))\, 15)\, 4,$$

а затем выполнить две редукции[2]

$$((\lambda m.(\lambda n.\, 2m + 3n))\, 15)\, 4 \longrightarrow (\lambda n.\, 2 \cdot 15 + 3n)\, 4 \longrightarrow 2 \cdot 15 + 3 \cdot 4.$$

Таким образом в лямбда-исчислении функция k переменных[3] — это функция одной переменной, возвращающая (при аппликации к фактическому значению аргумента) функцию $k - 1$ переменной. В нашем примере подвыражение $\lambda n.\, 2 \cdot 15 + 3n$ (функция одной переменной) есть результат частичного применения исходной функции двух переменных к аргументу 15.

Ниже мы будем рассматривать так называемое *чистое* лямбда-исчисление. В отличие от приведенных выше примеров базовые строительные блоки будут представлять собой не числа и стандартные операторы над ними, а неинтерпретируемые в рамках формализма переменные.

1.1 Термы

Термы бестипового лямбда-исчисления, множество которых мы будем обозначать Λ, определяются индуктивно:

$$\begin{array}{lcr} x \in V & \Rightarrow & x \in \Lambda, \\ M, N \in \Lambda & \Rightarrow & (M\,N) \in \Lambda, \\ M \in \Lambda, x \in V & \Rightarrow & (\lambda x.\, M) \in \Lambda. \end{array} \tag{1}$$

Здесь V — бесконечное множество переменных; его элементами будем считать латинские буквы в нижнем регистре, возможно снабженные индексами[4]. При этом произвольные термы будем обозначать буквами латинского алфавита в верхнем регистре, как в приведенном выше определении. Терм может быть переменной, например,

$$\begin{array}{l} x \in \Lambda, \\ y \in \Lambda, \end{array}$$

аппликацией (*применением*), например,

$$\begin{array}{l} (x\,y) \in \Lambda, \\ ((x\,y)\,z) \in \Lambda, \\ ((x\,(y\,x)) \in \Lambda, \end{array}$$

[2]Обратим внимание, что в первом выражении имеются две аппликации, но только одна из них образует редекс.

[3]Часто функцию k переменных называют функцией *арности* k.

[4]Отметим, что здесь и ниже мы используем идентификатор x двояким образом: как конкретную переменную исчисления и как метапеременную-образец, обозначающую произвольный элемент V в индуктивных определениях и рекурсивных алгоритмах. Иногда для этих целей будут использоваться и другие идентефикаторы.

или *лямбда-абстракцией*, например,

$$(\lambda x.\, x) \in \Lambda,$$
$$(\lambda z.\, ((x\,(y\,x))) \in \Lambda,$$
$$(\lambda x.\, (\lambda y.\, ((x\,y)\,(z\,q)))) \in \Lambda.$$

Из примеров видно, что любой терм строятся из более простых термов в соответствии с индуктивным определением (1). Эти более простые термы называют *собственными подтермами* данного терма. Отношение «быть подтермом» удобно сделать рефлексивным, включив в него еще и сам анализируемый терм. Про него говорят, что он является (для самого себя) *несобственным* подтермом. Более формально, множество $\mathrm{subterms}(Q)$ ***подтермов*** терма Q определяется индуктивно

$$\begin{array}{lcl} \mathrm{subterms}(x) & = & \{x\}, \\ \mathrm{subterms}(M\,N) & = & \{M\,N\} \cup \mathrm{subterms}(M) \cup \mathrm{subterms}(N), \\ \mathrm{subterms}(\lambda x.\, M) & = & \{\lambda x.M\} \cup \mathrm{subterms}(M). \end{array} \tag{2}$$

Будем придерживаться стандартных соглашений: внешние скобки и скобки вокруг тела лямбда-абстракции опускаются[5]; операция применения термов ассоциативна влево; тело лямбда-абстракции простирается вправо насколько это возможно; несколько последовательных абстракций объединяют в один блок. То есть, например, $\lambda x\, y.\, x\, y\, (z\, q)$ — компактное обозначение для $(\lambda x.\, (\lambda y.\, ((x\,y)\,(z\,q))))$.

Пусть в терме Q есть подтерм-переменная x. Говорят, что абстракция $\lambda x.\, Q$ *связывает* дотоле *свободную* переменную x в терме Q. Например, в терме

$$(\lambda y.\, (\lambda x.\, x\, z)\, y)\, w$$

переменные x и y — связанные, а z и w — свободные. Более формально, множество $\mathrm{FV}(Q)$ ***свободных (free) переменных*** в терме Q определяется индуктивно

$$\begin{array}{lcl} \mathrm{FV}(x) & = & \{x\}, \\ \mathrm{FV}(M\,N) & = & \mathrm{FV}(M) \cup \mathrm{FV}(N), \\ \mathrm{FV}(\lambda x.\, M) & = & \mathrm{FV}(M) \setminus \{x\}. \end{array} \tag{3}$$

Множество $\mathrm{BV}(Q)$ ***связанных (bound) переменных*** в терме Q:

$$\begin{array}{lcl} \mathrm{BV}(x) & = & \varnothing, \\ \mathrm{BV}(M\,N) & = & \mathrm{BV}(M) \cup \mathrm{BV}(N), \\ \mathrm{BV}(\lambda x.\, M) & = & \mathrm{BV}(M) \cup \{x\}. \end{array} \tag{4}$$

Связывание переменной при построении терма может происходить несколько раз, например, здесь

$$(\lambda x.\, (\lambda x.\, x\, z)\, x)\, x$$

переменная x — связанная (дважды) и свободная, а z — свободная.

[5]Первым соглашением мы уже воспользовались в предыдущем определении.

Если в терме Q нет свободных переменных, то есть $FV(Q) = \varnothing$, его называют ***замкнутым*** термом или ***комбинатором***. Вот список классических комбинаторов и их общеупотребительных имен:

$$\begin{array}{lcl}
\mathbf{I} & = & \lambda x.\, x \\
\boldsymbol{\omega} & = & \lambda x.\, x\, x \\
\boldsymbol{\Omega} & = & \boldsymbol{\omega}\, \boldsymbol{\omega} \;=\; (\lambda x.\, x\, x)(\lambda x.\, x\, x) \\
\mathbf{K} & = & \lambda x\, y.\, x \\
\mathbf{K}_* & = & \lambda x\, y.\, y \\
\mathbf{C} & = & \lambda f\, x\, y.\, f\, y\, x \\
\mathbf{B} & = & \lambda f\, g\, x.\, f\, (g\, x) \\
\mathbf{S} & = & \lambda f\, g\, x.\, f\, x\, (g\, x)
\end{array}$$

Мы будем использовать эти имена как простые сокращения при записи лямбда-термов.

1.2 Вычисления

Определим отношение вычисления (или редукции). Очевидно, что вычисления можно выполнить не в любом терме. Для осуществления в аппликации подстановки, нужно, чтобы левый аппликанд был лямбда-абстракцией. Имеется специальный термин для термов такой структуры: терм вида

$$(\lambda x.\, M)\, N$$

называется ***редексом*** или β-***редексом***, от английского reducible expression. Тогда ***одношаговая*** β-***редукция*** может быть определена для редекса следующим образом

$$(\lambda x.\, M)\, N \;\longrightarrow_\beta\; [x \mapsto N]\, M. \tag{5}$$

Правая часть содержит еще не определенную формально подстановку, результат этой подстановки называется ***сокращением*** β-редекса. Примеры

$$\begin{array}{l}
\mathbf{I}\, z = (\lambda x.\, x)\, z \;\longrightarrow_\beta\; [x \mapsto z]\, x = z, \\
\mathbf{K}\, u = (\lambda x\, y.\, x)\, u \;\longrightarrow_\beta\; [x \mapsto u]\, (\lambda y.\, x) = \lambda y.\, u.
\end{array}$$

Отметим, что сама по себе нотация подстановки является частью «вычислителя», а не формализма, в том смысле, что, например, $[x \mapsto z]\, x$ не является корректным термом лямбда-исчисления[6].

К сожалению, наше определение редукции не совсем удовлетворительно. В следующий цепочки вычислений первый переход не соответствует правилу (5):

$$\mathbf{K}\, u\, v = ((\lambda x\, y.\, x)\, u)\, v \longrightarrow_\beta (\lambda y.\, u)\, v \;\longrightarrow_\beta\; u.$$

[6]Иногда рассматривают системы лямбда исчисления с явной подстановкой, в которых понятие терма расширяют подобными конструкциями. Например, во многих функциональных языках имеется конструкция `let` $x = N$ `in` M во многих отношениях эквивалентная $[x \mapsto N]\, M$.

Проблема в том, что в выражении $((\lambda x\, y.\, x)\, u)\, v$ левым аппликандом является не абстракция, а аппликация $(\lambda x\, y.\, x)\, u$. Редекс присутствует в выражении, но как собственный подтерм, а не на верхнем уровне. Решением этой проблемы служит дополнения определения (5) для одношаговой β-редукции следующими правилами:

$$\begin{array}{lcl} M \longrightarrow_\beta N & \Rightarrow & Z\,M \longrightarrow_\beta Z\,N, \\ M \longrightarrow_\beta N & \Rightarrow & M\,Z \longrightarrow_\beta N\,Z, \\ M \longrightarrow_\beta N & \Rightarrow & \lambda x.\, M \longrightarrow_\beta \lambda x.\, N. \end{array} \tag{6}$$

Эти правила позволяют осуществлять поиск редексов среди всех подтермов данного терма и сокращать их. В частности, в приведенном выше примере для первого сокращения работает второе из этих новых правил.

Дадим теперь формальное определение подстановки. ***Подстановка*** терма N вместо свободных вхождений переменной x в терм M (нотация $[x \mapsto N]\, M$) задается следующими правилами:

$$\begin{array}{lcll} [x \mapsto N]\ x & = & N, & \\ [x \mapsto N]\ y & = & y, & \\ & & & \\ [x \mapsto N]\ (P\,Q) & = & ([x \mapsto N]\, P)\, ([x \mapsto N]\, Q), & \\ & & & \\ [x \mapsto N]\ (\lambda x.\, P) & = & \lambda x.\, P, & \\ [x \mapsto N]\ (\lambda y.\, P) & = & \lambda y.\, [x \mapsto N]\ P, & \text{если } y \notin \mathrm{FV}(N), \\ [x \mapsto N]\ (\lambda y.\, P) & = & \lambda z.\, [x \mapsto N]\ ([y \mapsto z]\, P), & \text{если } y \in \mathrm{FV}(N). \end{array} \tag{7}$$

Здесь подразумевается, что переменные x и y различны, а переменная z — «свежая», то есть мы выбираем для нее имя не задействованное как свободное в P и N: $z \notin \mathrm{FV}(P) \cup \mathrm{FV}(N)$. Пример подстановки

$$[x \mapsto u\, v]((\lambda x.\, (\lambda x\,.\, x\, z)\, x)\, x) \;=\; (\lambda x.\, (\lambda x\,.\, x\, z)\, x)\, (u\, v)$$

иллюстрирует, что предметом замены при подстановке служат только свободные вхождения переменной.

Оговорка про свойства y в последних пунктах определения (7) позволяет избежать так называемого «захвата переменной». Рассмотрим следующую подстановку, выполненную по последнему, 6 правилу этого определения

$$[x \mapsto y]\, (\lambda y.\, x\, y) = \lambda z.\, y\, z.$$

Если бы мы действовали по предпоследнему, 5 правилу, мы получили бы совершенно неудовлетворительный результат $\lambda y.\, y\, y$, в котором свободный внешний y неотличим от связанного внутреннего y.

Почему приведенная выше замена имени переменной y на z имеет смысл? Дело в том, что переименование связанной переменной никак не влияет на «вычислительное»

поведение терма. Например, для произвольного терма N оба вычисления дают один и тот же результат

$$(\lambda x.\, x)\, N \longrightarrow_\beta [x \mapsto N]\, x = N,$$
$$(\lambda u.\, u)\, N \longrightarrow_\beta [u \mapsto N]\, u = N.$$

Иными словами, нет никакой вычислительной разницы между следующими способами записи комбинатора $\mathbf{I}$: $\lambda x.\, x$, $\lambda u.\, u$, $\lambda f.\, f$ и т.д. Это же относится и к другим термам со связанными переменными: конкретные имена этих переменных не важны для описания вычислительного поведения терма.

Формализовать это наблюдение можно, задав на термах так называемое отношение α-эквивалентности:

$$\lambda y.\, M =_\alpha \lambda z.\, [y \mapsto z]\, M, \text{ если } z \notin \mathrm{FV}(M). \tag{8}$$

Можно считать, что термы нас интересуют с точностью до α-эквивалентности. В дальнейшем эта оговорка будет подразумеваться.

С операционной точки зрения мы теперь можем интерпретировать последнее правило в определении (7) следующим образом. Если нам нужно выполнить подстановку $[x \mapsto N]\,(\lambda y.\, P)$ и мы обнаружили, что y свободно в N, то следует выполнить в $\lambda y.\, P$ α-преобразование, заменив связанный здесь y на свежую переменную. После этого можно пользоваться предпоследним правилом в (7).

Упражнение. Выполните подстановку $[x \mapsto y\, y](\lambda y\, z.\, z\, x\, y)$. Можно ли в качестве «свежей» переменной в этой подстановке выбрать z?

Лемма 1 (подстановки)**.** *Пусть* $M, N, L \in \Lambda$. *Предположим* $x \not\equiv y$ *и* $x \notin \mathrm{FV}(L)$. *Тогда*

$$[y \mapsto L]\, [x \mapsto N]\, M = [x \mapsto [y \mapsto L]\, N]\, [y \mapsto L]\, M. \tag{9}$$

Доказательство. Нудная индукция по всем 6 случаям. ■

Определим теперь отношение ***многошаговой* β*-редукции*** (нотация $\twoheadrightarrow_\beta$) как рефлексивное транзитивное замыкание одношаговой. Формально

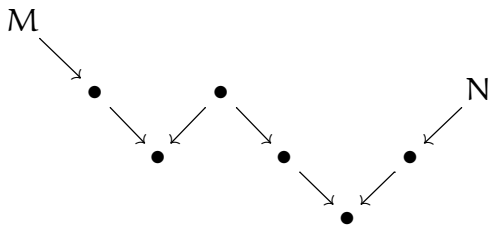
$$\begin{array}{lcl} M \longrightarrow_\beta N & \Rightarrow & M \twoheadrightarrow_\beta N, \\ & & M \twoheadrightarrow_\beta M, \\ M \twoheadrightarrow_\beta N,\ N \twoheadrightarrow_\beta L & \Rightarrow & M \twoheadrightarrow_\beta L. \end{array} \tag{10}$$

Первое из этих трех правил говорит, что термы связаны отношением многошаговой редукции, если они связаны отношением редукции в один шаг, второе — что терм связан сам с собой отношением многошаговой редукции в 0 шагов. Третье правило, транзитивность, позволяет выполнять редукцию за число шагов, большее единицы.

Делая симметричное транзитивное замыкание многошаговой редукции, получаем отношение ***β-эквивалентности*** или ***β-конвертируемости*** (нотация $=_\beta$)

$$\begin{array}{lcl} M \twoheadrightarrow_\beta N & \Rightarrow & M =_\beta N, \\ M =_\beta N & \Rightarrow & N =_\beta M, \\ M =_\beta N,\ N =_\beta L & \Rightarrow & M =_\beta L. \end{array} \tag{11}$$

Интуитивно: два терма M и N связаны отношением $=_\beta$, если есть связывающая их цепочка $\longrightarrow_\beta$ -стрелок:



Подобного рода визуализации можно формализовать, введя понятие ***редукционного графа***. Редукционный граф терма $M \in \Lambda$ (нотация $G_\beta(M)$) — это ориентированный мультиграф с вершинами в $\{N \mid M \twoheadrightarrow_\beta N\}$ и дугами $\longrightarrow_\beta$. Например,

$$G_\beta\,(\mathbf{I}\,(\mathbf{I}\,x)) \;=\; \bullet \rightleftarrows \bullet \longrightarrow \bullet$$

Отношение β-эквивалентности в общем случае не является разрешимым. Это значит, что для пары термов M и N не существует универсального алгоритма, проверяющего факт $M =_\beta N$. Однако для широкого класса термов с «хорошими» вычислительными свойствами[7] отношение β-эквивалентности разрешимо.

Помимо β-редукции часто рассматривают еще одно «вычислительное» отношение над термами — η-редукцию, которая задается следующим правилом

$$\lambda x.\, M\,x \;\longrightarrow_\eta\; M, \text{ если } x \notin \mathrm{FV}(M). \tag{12}$$

При этом полное определение содержит также правила, аналогичные (6), мы не будем выписывать их явно. Аналогично (10) и (11) вводятся отношения ***многошаговой η-редукции*** и ***η-эквивалентности***. Выражение $\lambda x.\, M\,x$ (конечно при условии $x \notin \mathrm{FV}(M)$) называется ***η-редексом***, а преобразование (12) — его сокращением.

Смысл η-эквивалентности $\lambda x.\, M\,x =_\eta M$ в том, что аппликативное поведение термов слева и справа от знака η-равенства одинаково; для произвольного N верно

$$(\lambda x.\, M\,x)\,N =_\beta M\,N.$$

Иными словами $\lambda x.\, M\,x$ и M — это одна и та же «функция».

η-преобразование обеспечивает принцип ***экстенсиональности***: две функции F и G считаются *экстенсионально эквивалентными*, если они дают одинаковый результат при любом одинаковом вводе:

$$\forall N \quad F\,N =_\beta G\,N.$$

Одного β-правила недостаточно, чтобы получить отсюда равенство F и G, но при наличии η-эквивалентности это можно сделать. Действительно, выбирая $y \notin \mathrm{FV}(F) \cup \mathrm{FV}(G)$, получаем

$$\begin{array}{ll} & F\,y =_\beta G\,y \\ \Rightarrow & \lambda y.\, F\,y =_\beta \lambda y.\, G\,y \\ \Rightarrow & F =_{\beta\eta} G. \end{array}$$

[7] Это будут термы, для которых цепочка редукций завершается получением *значения*, то есть терма, в котором нет редексов.

1.3 Значения

Для любой модели вычислений важной задачей является определение того, что является результатом этих вычислений. Как определить, что вычислительная система достигла финального состояния? В лямбда-исчислениии вычисление основывается на сокращении редексов, поэтому естественно считать, что терм, в котором редексы отсутствуют, является таким финальным состоянием.

Говорят, что λ-терм M находится в ***β-нормальной форме*** (β-NF), если в нем нет подтермов, являющихся β-редексами. λ-терм M имеет β-нормальную форму, если для некоторого N выполняется $M =_\beta N$ и N находится в β-NF. Аналогично можно определить ***η-нормальную форму***.

Примеры. Терм $\lambda x\, y.\, x\, (\lambda z.\, z\, x)\, y$ находится в β-нормальной форме. Терм $(\lambda x.\, x\, x)\, y$ не находится в β-нормальной форме, но имеет в качестве β-NF терм $y\, y$, достигаемый за один шаг β-редукции. Терм $\lambda x\, y\, z.\, x\, x\, y\, z$ находится в β-NF, но не в η-NF. За два шага η-редукции он приводится к βη-NF $\lambda x.\, x\, x$.

Сделаем ряд более общих наблюдений относительно многошаговой β-редукции и достижения β-нормальной формы в результате этого процесса.

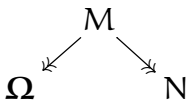
(1) Не все термы имеют β-нормальную форму. Например,

$$\begin{array}{rcl} \boldsymbol{\Omega} & = & \omega\, \omega \\ & = & (\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \\ & \longrightarrow_\beta & (\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \\ & \longrightarrow_\beta & \ldots \end{array}$$

Или в виде редукционного графа

$$G_\beta\, (\boldsymbol{\Omega}) \;=\; \circlearrowright\bullet$$

Этот пример показывает, что процесс β-редукций может быть бесконечным. С вычислительной точки зрения можно сказать, что у нас имеются *расходящиеся* вычисления. Отметим, что этот пример пока не является строгим доказательством отсутствия у $\boldsymbol{\Omega}$ β-нормальной формы. Возможно существует терм N в β-NF, такой что $\boldsymbol{\Omega} =_\beta N$. Это может выглядеть, например, так



Позже станет понятно, почему такая конструкция невозможна.

(2) Бывают термы, «удлинняющиеся» при редукции.

$$\begin{array}{rcl} \boldsymbol{\Omega_3} & = & \omega_3\, \omega_3 \\ & = & (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x) \\ & \longrightarrow_\beta & (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x) \\ & \longrightarrow_\beta & (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x)\, (\lambda x.\, x\, x\, x) \\ & \longrightarrow_\beta & \ldots \end{array}$$

Отметим, что здесь термы, получающиеся на каждом из бесконечной цепочки последовательных редукционных шагов, всегда содержит ровно один редекс. В виде редукционного графа

$$\mathsf{G}_\beta\,(\mathbf{\Omega_3}) \;=\; \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \dots$$

Мы видим, что не всегда редукционные графы конечны.

(3) Не все последовательности редукций приводят к β-NF. В терме $\mathbf{K\,I\,\Omega}$ имеются два редекса: внутренний, в $\mathbf{\Omega}$, и внешний, на верхнем уровне. Попытка сокращения внутреннего приводит к расходимости, а сокращение внешнего позволяет достигнуть β-NF за 2 шага:

$$\begin{array}{lcl}
\mathbf{K\,I\,\Omega} & = & \mathbf{K\,I}\,((\lambda x.\,x\,x)\,(\lambda x.\,x\,x)) \\
& \longrightarrow_\beta & \mathbf{K\,I}\,((\lambda x.\,x\,x)\,(\lambda x.\,x\,x)) \\
& \longrightarrow_\beta & \dots
\end{array}$$

$$\begin{array}{lcl}
\mathbf{K\,I\,\Omega} & = & (\lambda x\,y.\,x)\,\mathbf{I\,\Omega} \\
& \longrightarrow_\beta & (\lambda y.\,\mathbf{I})\,\mathbf{\Omega} \\
& \longrightarrow_\beta & \mathbf{I}
\end{array}$$

Здесь синим отмечен сокращаемый редекс. Этот пример иллюстрирует важность понятия *стратегии редукции*. Стратегия должна отвечает на вопрос, какой из имеющихся в терме редексов необходимо сокращать. Редукционный граф для этого терма выглядит так:

$$\mathsf{G}_\beta\,(\mathbf{K\,I\,\Omega}) \;=\; \bullet \longrightarrow \bullet \longrightarrow \bullet$$

(4) Не все бесконечные редукционные графы не имеют β-NF.

$$\mathsf{G}_\beta\,((\lambda x.\,\mathbf{I})\,\mathbf{\Omega_3}) \;=\; \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \bullet \longrightarrow \dots$$

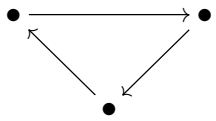
Этот пример иллюстрирует еще одно крайне важное свойство нашей вычислительной системы. Как бы далеко мы не забрались по «неправильному», то есть ведущему к расходимости пути, всегда есть возможность «вернуться обратно». Отметим, что это свойство, конечно же, требует доказательства, которое будет приведено ниже.

Упражнение. Приведите пример терма, не находящегося ни в β-NF, ни в η-NF, но превращающийся в βη-NF за один шаг любой из редукций.

Упражнение. Может ли при β-редукции возникнуть η-редекс?

Упражнение. Приведите пример терма, редукционный граф которого имеет вид

Упражнение. Приведите пример терма, редукционный граф которого имеет вид



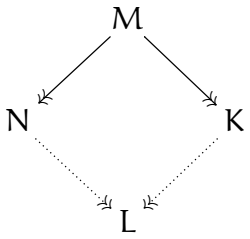
1.4 Свойства вычислений

Основной теоремой, из которой следуют многие важные свойства нашей вычислительной системы служит теорема Чёрча-Россера.

Теорема 1 (Чёрча-Россера)**.** *Если* $M \longrightarrow_\beta N$, $M \longrightarrow_\beta K$, *то существует* L, *такой что* $N \longrightarrow_\beta L$ *и* $K \longrightarrow_\beta L$.

Доказательство будет дано ниже.

Можно изобразить это свойство в виде диаграммы:



На таких диаграммах обычно подразумевают, что то, что дано в условии, изображено сплошными линиями, а то, что требуется доказать, прерывистыми. Иногда подобное свойство называют *свойством ромба*. Также используют термин *конфлюентность*.

Следующие факты являются следствиями теоремы Чёрча-Россера.

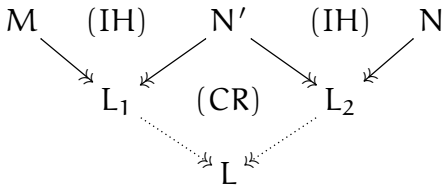
Теорема 2 (о существовании общего редукта)**.** *Если* $M =_\beta N$, *то существует* L, *такой что,* $M \twoheadrightarrow_\beta L$ *и* $N \twoheadrightarrow_\beta L$.

Доказательство (индукция по генерации $=_\beta$):

Случай $M =_\beta N$, поскольку $M \twoheadrightarrow_\beta N$. Возьмем $L = N$.

Случай $M =_\beta N$, поскольку $N =_\beta M$. По гипотезе индукции имеется общий β-редукт L_1 для N, M. Возьмем $L = L_1$.

Случай $M =_\beta N$, поскольку $M =_\beta N'$, $N' =_\beta N$. Тогда



Здесь (IH) указывает на применение гипотезы индукции, а (CR) — теоремы Черча-Россера. ■

Теорема 3 (о редуцируемости к NF)**.** *Если* M *имеет* N *в качестве* β*-NF, то* $M \twoheadrightarrow_\beta N$.

Доказательство (упражнение).

Теперь мы можем доказать отсутствие нормальной формы у $\mathbf{\Omega}$. Действительно, если бы у $\mathbf{\Omega}$ была β-NF, скажем N, то выполнялось бы $\mathbf{\Omega} \twoheadrightarrow_\beta \mathsf{N}$. Но комбинатор $\mathbf{\Omega}$ редуцируется лишь сам к себе и не является β-NF.

Теорема 4 (о единственности β-NF)**.** *λ-терм имеет не более одной β-NF.*

Доказательство (упражнение).

Единственность β-NF дает нам простой алгоритм проверки β-эквивалентности двух термов, имеющих β-нормальные формы. Достаточно провести редукцию до нормальной формы для каждого из этих термов и сравнить результаты[8]. Напомним, что в общем случае проверка β-эквивалентности термов неразрешима.

1.5 Доказательство теоремы Чёрча-Россера

Мы следуем идеям доказательства из [3]. Доказательство во многом базируется на том соображении, что при редукциях редекс не может «развалиться». Иными словами, если работать в синтаксисе с обязательными скобками и покрасить некоторый редекс следующим образом

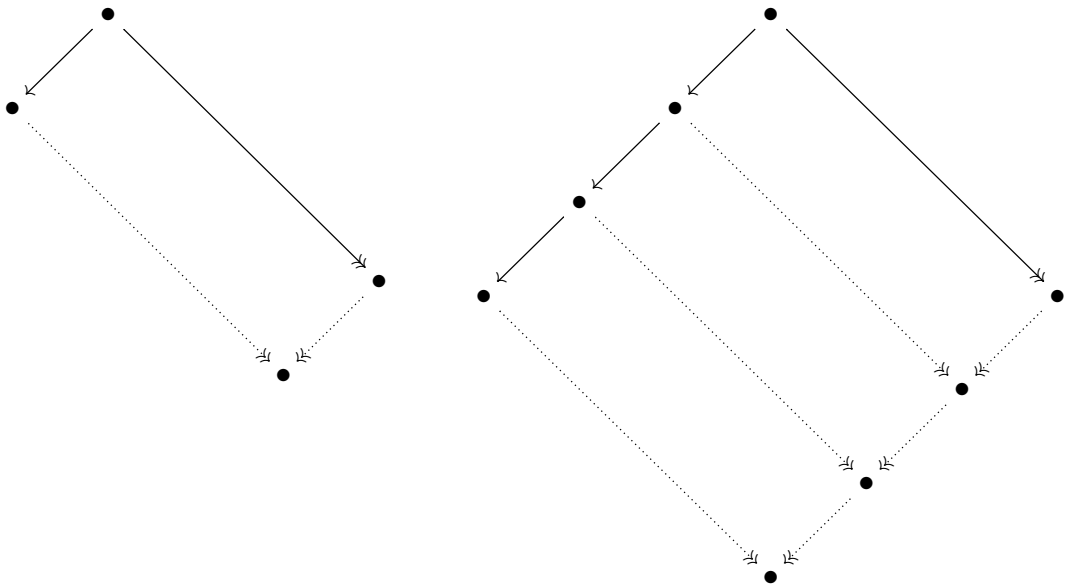
$$\ldots((\lambda x.\, \mathsf{M})(\mathsf{N}))\ldots$$

то при вычислениях этот покрашенный редекс может полностью пропасть (либо его сокращением, либо под действием внешних факторов), размножится (пар цветных скобок станет несколько), M и N могут видоизменится произвольным (в общем случае) образом. Но пока пара цветных скобок цела, они всегда неотрывно будет следовать друг за другом. В доказательстве мы будем использовать более экономную раскраску редексов.

Технически доказательство устроено так: сначала доказываем Лемму полоски, а

[8] С точностью до α-эквивалентности.

затем из полосок составляем «ромб».



Термы *подкрашенного* лямбда-исчисления, множество которых мы будем обозначать Λ, определяются индуктивно:

$$\begin{array}{lcr} x \in V & \Rightarrow & x \in \Lambda, \\ M, N \in \Lambda & \Rightarrow & (M\,N) \in \Lambda, \\ M \in \Lambda, x \in V & \Rightarrow & (\lambda x.\, M) \in \Lambda, \\ M, N \in \Lambda, x \in V & \Rightarrow & (\lambda x.\, M)\,N \in \Lambda. \end{array} \tag{13}$$

То есть в редексах (и только в них) некоторые лямбды могут быть покрашены.

Подкрашенные редукции (одношаговые и многошаговые) определяются стандартным образом на базе правил сокращения:

$$\begin{array}{l} (\lambda x.\, M)\,N \longrightarrow_\beta [x \mapsto N]\,M, \\ (\lambda x.\, M)\,N \longrightarrow_\beta [x \mapsto N]\,M. \end{array} \tag{14}$$

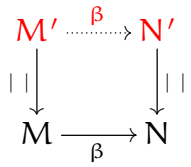
То есть вычисления игнорируют раскраску.

Вводится операция *стирания подкраски*. Если $M \in \Lambda$, то $|M| \in \Lambda$ получается заменой в M всего красного на черное.

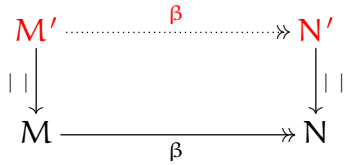
Задается операция $\varphi : \Lambda \to \Lambda$, заключающаяся в сокращении всех подкрашенных редексов изнутри наружу:

$$\begin{array}{lcl} \varphi(x) & = & x, \\ \varphi(M\,N) & = & \varphi(M)\,\varphi(N), \\ \varphi(\lambda x.\, M) & = & \lambda x.\, \varphi(M), \\ \varphi((\lambda x.\, M)\,N) & = & [x \mapsto \varphi(N)]\,\varphi(M). \end{array} \tag{15}$$

Лемма 2.



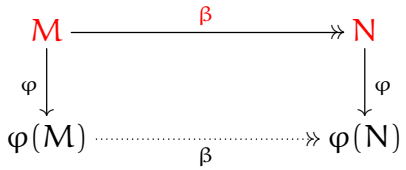
Лемма 3.



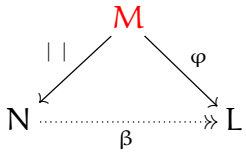
Лемма 4.

$$\varphi([x \mapsto N]\, M) = [x \mapsto \varphi(N)]\, \varphi(M).$$

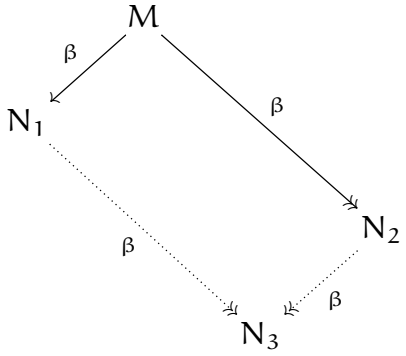
Лемма 5.



Лемма 6.

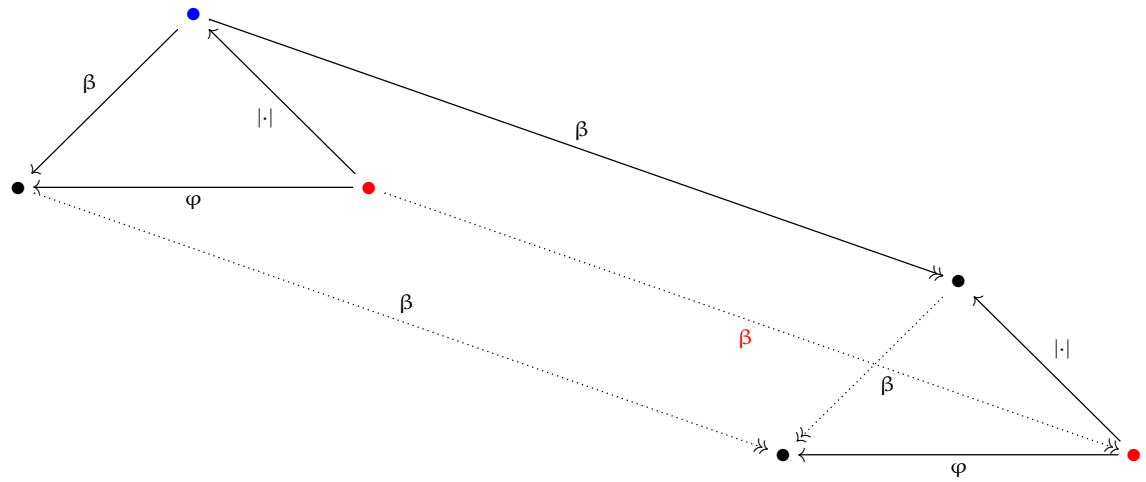


Лемма 7 (полоски)**.**



Доказательство. Берем в M редекс, сокращаемый для получения N_1, и подкра-

шиваем его. Смысл верхнего левого треугольника при этом очевиден.



Грани призмы, содержащие красные вершины, представляют собой предыдущие леммы, а оставшаяся грань представляет собой лемму полоски. ■

1.6 Кодирование в лямбда-исчислении

Хотя чистое лямбда-исчисление устроено довольно минималистично, тем не менее, используя его, можно сконструировать полноценный язык программирования.

Булевы значения можно определить следующим образом:

$$\begin{aligned} \texttt{tru} &= \lambda\, \texttt{t}\, \texttt{f}.\, \texttt{t} \\ \texttt{fls} &= \lambda\, \texttt{t}\, \texttt{f}.\, \texttt{f} \end{aligned}$$

Тогда стандартные булевы операции кодируются так:

$$\begin{aligned} \texttt{if} &= \lambda\, \texttt{b}\, \texttt{x}\, \texttt{y}.\, \texttt{b}\, \texttt{x}\, \texttt{y} \\ \texttt{not} &= \lambda\, \texttt{b}.\, \texttt{b}\ \texttt{fls}\ \texttt{tru} \\ \texttt{and} &= \lambda\, \texttt{x}\, \texttt{y}.\, \texttt{x}\, \texttt{y}\ \texttt{fls} \\ \texttt{or} &= \lambda\, \texttt{x}\, \texttt{y}.\, \texttt{x}\ \texttt{tru}\ \texttt{y} \end{aligned}$$

Несложно проверить, что ожидаемые характеристические свойства логических операторов выполняются:

$$\begin{aligned} &\texttt{if tru A B} = \texttt{tru A B} = \texttt{A}; \\ &\texttt{if fls A B} = \texttt{fls A B} = \texttt{B}. \end{aligned}$$

$$\begin{aligned} &\texttt{and tru A} = \texttt{tru A fls} = \texttt{A}; \\ &\texttt{and fls A} = \texttt{fls A fls} = \texttt{fls}. \end{aligned}$$

И т.д. В качестве упражнения попробуйте найти более «короткую» версию оператора «НЕ».

Пару (двухэлементный кортеж) можно определить так:

$$\texttt{pair} \equiv \lambda\, x\, y\, f.\, f\, x\, y$$

Стандартные операции для пары (проекции):

$$\begin{aligned} \texttt{fst} &\equiv \lambda\, p.\, p\ \texttt{tru} \\ \texttt{snd} &\equiv \lambda\, p.\, p\ \texttt{fls} \end{aligned}$$

Проверьте, что ожидаемые свойства проекций выполняются:

$$\begin{aligned} &\texttt{fst}\ (\texttt{pair}\ A\ B) = A; \\ &\texttt{snd}\ (\texttt{pair}\ A\ B) = B. \end{aligned}$$

Если функция двух аргументов задана в традиционном стиле $f\,(\texttt{pair}\ x\ y)$ (на паре, т.е. декартовом произведении), то перейти к стандартной записи можно *каррированием*:

$$\texttt{curry} \ = \lambda\, f\, x\, y.\, f\,(\texttt{pair}\ x\ y)$$

Реализуйте обратную процедуру, `uncurry`.

Числа (нумералы Чёрча)

$$\begin{aligned} 0 &\equiv \lambda\, s\, z.\, z \\ 1 &\equiv \lambda\, s\, z.\, s\, z \\ 2 &\equiv \lambda\, s\, z.\, s\,(s\, z) \\ 3 &\equiv \lambda\, s\, z.\, s\,(s\,(s\, z)) \\ 4 &\equiv \lambda\, s\, z.\, s\,(s\,(s\,(s\, z))) \\ &\ldots \end{aligned}$$

Выражение $F^n(X)$, где $n \in \mathbb{N}$, а $F, X \in \Lambda$, определим индуктивно:

$$\begin{aligned} F^0(X) &\equiv X; \\ F^{n+1}(X) &\equiv F(F^n(X)). \end{aligned}$$

Тогда n-ое число Чёрча :

$$n \equiv \lambda\, s\, z.\, s^n(z).$$

Проверка числа на ноль ($0 \equiv \lambda\, s\, z.\, z$):

$$\texttt{iszro} \equiv \lambda\, n.\, n\ (\lambda\, x.\, \texttt{fls})\ \texttt{tru}$$

Проверьте, что ожидаемые свойства `iszro` выполняются. Попробуйте найти более «короткую» версию `iszro`.

Функция следования для чисел Чёрча

$$\texttt{succ} \equiv \lambda n\, s\, z.\, s\,(n\, s\, z)$$

Проверьте, что ожидаемые свойства `succ` выполняются. Попробуйте найти другое определение `succ`.

Функция сложения чисел Чёрча

$$\texttt{plus} \equiv \lambda m\, n\, s\, z.\, m\, s\, (n\, s\, z)$$

Проверьте, что ожидаемые свойства `plus` выполняются. Попробуйте найти определение `plus` с использованием `succ`.

Функция умножения чисел Чёрча

$$\begin{aligned} \texttt{mult1} &\equiv \lambda m\, n.\, m\, (\texttt{plus}\ n)\ \texttt{0} \\ \texttt{mult2} &\equiv \lambda m\, n\, s\, z.\, m\, (n\, s)\, z \end{aligned}$$

Проверьте, что ожидаемые свойства умножения выполняются. Можно ли `mult2` записать короче?

Реализуйте функцию возведения чисел Черча в степень, например

$$\texttt{pow}\ 2\ 3 = 8$$

Функция предшествования для чисел Чёрча. Вспомогательные функции

$$\begin{aligned} \texttt{zp} &\equiv \texttt{pair 0 0} \\ \texttt{sp} &\equiv \lambda \texttt{p}.\, \texttt{pair}\ (\texttt{snd p})\ (\texttt{succ}\ (\texttt{snd p})) \end{aligned}$$

Вторая работает так

$$\texttt{sp}\ (\texttt{pair i j}) = \texttt{pair j}\ (\texttt{j} + \texttt{1})$$

$$\begin{aligned} \texttt{sp}^{\texttt{0}}\ (\texttt{zp}) &= \texttt{pair 0 0} \\ \texttt{sp}^{\texttt{m}}\ (\texttt{zp}) &= \texttt{pair}\ (\texttt{m} - \texttt{1})\ \texttt{m} \end{aligned}$$

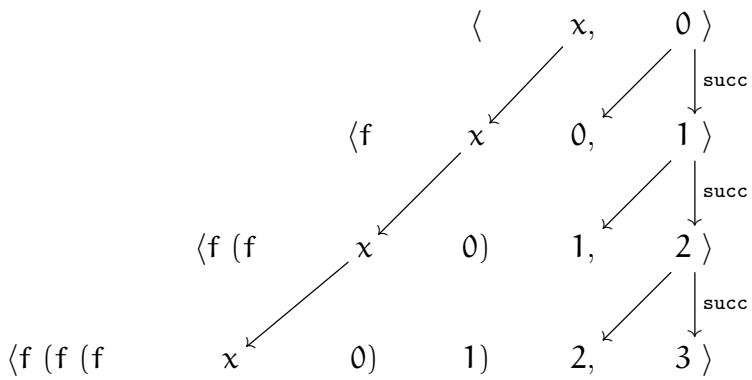
(здесь $m > 0$). Тогда функция предшествования:

$$\texttt{pred} = \lambda m.\, \texttt{fst}\ (m\ \texttt{sp zp})$$

Какая у неё временная сложность? Что нужно поменять, чтобы вышел факториал?

Числа Чёрча: **примитивная рекурсия**.
Обобщим предыдущую схему

$$\begin{aligned} \texttt{xz} &\equiv \lambda x.\, \texttt{pair}\ x\ \texttt{0} \\ \texttt{fs} &\equiv \lambda f\, p.\, \texttt{pair}\ (f\ (\texttt{fst p})\ (\texttt{snd p}))\ (\texttt{succ}\ (\texttt{snd p})) \\ \texttt{rec} &\equiv \lambda m\, f\, x.\, \texttt{fst}\ (m\ (\texttt{fs f})\ (\texttt{xz x})) \end{aligned}$$



В частности,

$$\texttt{pred} \;=\; \lambda m.\, \texttt{rec}\; m\; (\lambda x\, y.\, y)\; \texttt{0}$$

Реализуйте факториал через комбинатор примитивной рекурсии `rec`.
Реализуйте функцию суммирования чисел от 1 до n.
Реализуйте функцию нахождения n-ой частичной суммы ряда $\sum_{k=1}^{n} f(k)$.
Напишите функции: `minus`, вычитающую числа Чёрча, `equals`, сравнивающую два числа Чёрча на предмет равенства, а также всевозможные неравенства, строгие и нестрогие, `lt`, `gt`, `le`, `ge`.

Конструкторы **списков** можно определить так:

$$\begin{array}{rcl} \texttt{nil} & \equiv & \lambda c\, n.\, n \\ \texttt{cons} & \equiv & \lambda e\, l\, c\, n.\, c\, e\, (l\, c\, n) \end{array}$$

Например,

$$\begin{array}{l} [\,] = \texttt{nil} \;=\; \lambda c\, n.\, n \\ [5,3,2] \;=\; \texttt{cons}\; 5\; (\texttt{cons}\; 3\; (\texttt{cons}\; 2\; \texttt{nil})) \;=\; \lambda c\, n.\, c\; 5\; (c\; 3\; (c\; 2\; n)) \end{array}$$

Функция, определяющая пуст ли список

$$\texttt{empty} \;\equiv\; \lambda l.\, l\; (\lambda h\, t.\, \texttt{fls})\; \texttt{tru}$$

Проверьте правильность работы `empty`.
Попробуйте найти более «короткую» версию `empty`.
Постройте функцию `head`, возвращающую голову списка, например

$$\texttt{head}\; [5,3,2] \;=\; 5$$

Постройте функцию `sum` суммирующую элементы списка, например

$$\texttt{sum}\; [5,3,2] \;=\; 10$$

Постройте функцию `length` вычисляющую длину списка, например

$$\texttt{length}\ [5,3,2]\ =\ 3$$

Постройте функцию `tail`, возвращающую хвост списка, например

$$\texttt{tail}\ [5,3,2]\ =\ [3,2]$$

1.7 Рекурсия

Отношение β-эквивалентности даёт возможность решать простейшие уравнения на термы. Пусть, например, стоит задача найти терм F, такой что $\forall M, N, L$ выполнялось бы $F\,M\,N\,L =_\beta M\,L\,(N\,L)$. Решение:

$$\begin{array}{lll} F\,M\,N\,L & =_\beta & M\,L\,(N\,L) \\ F\,M\,N & =_\beta & \lambda l.\, M\,l\,(N\,l) \\ F\,M & =_\beta & \lambda n.\, \lambda l.\, M\,l\,(n\,l) \\ F & =_\beta & \lambda m\,n\,l.\, m\,l\,(n\,l) \end{array}$$

Каждый переход легко обосновать, если двигаться по приведенному списку равенств снизу вверх.

А если уравнение рекурсивное, например, $F\,M =_\beta M\,F$? Оказывается, имеется универсальный способ решения! Он базируется на следующих теоремах.

Теорема 5 (о неподвижной точке)**.** *Для любого* λ*-терма* F *существует неподвижная точка:*

$$\forall F \in \Lambda\ \ \exists X \in \Lambda\ \ F\,X =_\beta X.$$

Доказательство. Введем $W = \lambda x.\, F\,(x\,x)$ и $X = W\,W$. Тогда

$$X = W\,W = (\lambda x.\, F(x\,x))\,W =_\beta F\,(W\,W) = F\,X. \quad \blacksquare$$

Теорема 6 (о комбинаторе неподвижной точке)**.** *Существует комбинатор неподвижной точки* $\mathbf{Y}$*, такой что* $\forall F$ *выполнено* $F(\mathbf{Y}\,F) =_\beta \mathbf{Y}\,F$.

Доказательство. Введём

$$\mathbf{Y} = \lambda f.\, (\lambda x.\, f\,(x\,x))(\lambda x.\, f\,(x\,x)).$$

Для произвольного F имеем

$$\mathbf{Y}\,F = (\lambda x.\, F\,(x\,x))(\lambda x.\, F\,(x\,x)) =_\beta F\,(\underbrace{(\lambda x.\, F\,(x\,x))(\lambda x.\, F\,(x\,x))}_{\mathbf{Y}\,F}) =_\beta F(\mathbf{Y}\,F). \quad \blacksquare$$

Теперь мы можем решать рекурсивные уравнения. Пусть, например, стоит задача найти терм F, такой что $\forall M$ выполнялось бы $F\,M =_\beta M\,F$. Решение:

$$\begin{array}{lll} F\,M & =_\beta & M\,F \\ F & =_\beta & \lambda m.\, m\,F \\ F & =_\beta & (\lambda f\, m.\, m\,f)\,F \\ F & =_\beta & \mathbf{Y}(\lambda f\, m.\, m\,f) \end{array}$$

Еще один пример, с функциями над числами Черча. Факториал рекурсивно:

$$\mathtt{fac} = \lambda n.\, \mathtt{if}\,(\mathtt{iszro}\ n)\ \mathtt{1}\ (\mathtt{mult}\ n\,(\mathtt{fac}\,(\mathtt{pred}\ n)))$$

Переписываем в виде

$$\mathtt{fac} = \underbrace{(\lambda f\, n.\, \mathtt{if}\,(\mathtt{iszro}\ n)\ \mathtt{1}\ (\mathtt{mult}\ n\,(f\,(\mathtt{pred}\ n))))}_{\mathtt{fac}'}\ \mathtt{fac}$$

Отсюда видно, что $\mathtt{fac}$ — неподвижная точка для вспомогательной функции $\mathtt{fac}'$:

$$\mathtt{fac} = \mathbf{Y}\ \mathtt{fac}'$$

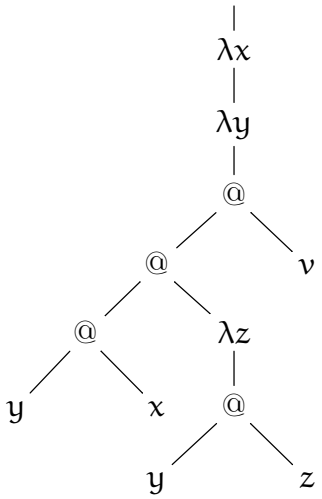
Как работает $\mathtt{fac}$?

$$\begin{array}{rcl} \mathtt{fac}\ 3 & = & (\mathbf{Y}\,\mathtt{fac}')\,3 \\ & = & \mathtt{fac}'\,(\mathbf{Y}\,\mathtt{fac}')\,3 \\ & = & \mathtt{if}\,(\mathtt{iszro}\ 3)\ \mathtt{1}\ (\mathtt{mult}\ 3\,((\mathbf{Y}\,\mathtt{fac}')\,(\mathtt{pred}\ 3))) \\ & = & \mathtt{mult}\ 3\,((\mathbf{Y}\,\mathtt{fac}')\,2) \\ & = & \mathtt{mult}\ 3\,(\mathtt{fac}'\,(\mathbf{Y}\,\mathtt{fac}')\,2) \\ & = & \mathtt{mult}\ 3\,(\mathtt{mult}\ 2\,((\mathbf{Y}\,\mathtt{fac}')\,1)) \\ & = & \mathtt{mult}\ 3\,(\mathtt{mult}\ 2\,(\mathtt{mult}\ 1\,((\mathbf{Y}\,\mathtt{fac}')\,0))) \\ & = & \mathtt{mult}\ 3\,(\mathtt{mult}\ 2\,(\mathtt{mult}\ 1\ 1)) \\ & = & 6 \end{array}$$

Отметим, что приведенный выше комбинатор Карри $\mathbf{Y}$ — не единственный лямбда-терм являющийся комбинатором неподвижной точки. Проверьте, что комбинатор Тьюринга $\mathbf{\Theta} = A\,A$, где $A = \lambda x\, y.\, y\,(x\,x\,y)$, тоже является комбинатором неподвижной точки.

1.8 Стратегии редукции

В соответствии с определением (1) мы можем рассматривать лямбда-терм как дерево с тремя типами узлов: аппликация задает бинарные узлы (их часто маркируют символом @), абстракция унарные, а переменные — листья, то есть узлы арности 0. Например, терм $\lambda x\, y.\, y\,x\,(\lambda z.\, y\,z)\,v$ имеет вид



Части лямбда-терма имеют специальные названия: в $\lambda x\, y.\, y\, x\, (\lambda z.\, y\, z)\, v$ часть $\lambda x\, y.$ называют *абстрактором*, а часть $y\, x\, (\lambda z.\, y\, z)\, v$ — *телом* терма. Абстрактор, фактически, представляет собой список переменных, и, в отличие от тела, может быть пустым. Тело терма, в свою очередь, не являясь по определению абстракцией, может быть либо переменной либо аппликацией. Так называемая аппликативная структура терма задается следующей теоремой.

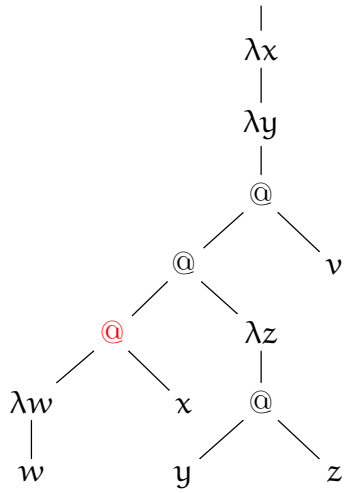
Теорема 7. *Лямбда-терм может иметь одну из двух форм:*

$$\begin{array}{lcl} \lambda\overrightarrow{x}.\, y\ \overrightarrow{N} & \equiv & \lambda x_1 \ldots x_n.\, y\, N_1 \ldots N_k \\ \lambda\overrightarrow{x}.\, (\lambda z.\, P)\ Q\ \overrightarrow{N} & \equiv & \lambda x_1 \ldots x_n.\, (\lambda z.\, P)\, Q\, N_1 \ldots N_k \end{array} \tag{16}$$

Здесь $n \geqslant 0$, $k \geqslant 0$, а переменная y может совпадать с одной из x_i, и *обязана* совпадать, если терм замкнут. Первая форма называется ***головной нормальной формой*** (HNF), переменная y называется ***головной переменной***. Во второй форме редекс $(\lambda z.\, P)\, Q$ называют ***головным редексом***.

Терм находится в нормальной форме, если он находится в головной нормальной форме, а все $N_1, \ldots, N_k$ тоже находятся в нормальной форме. Естественно, что у нормальной формы для любого подтерма в головной позиции стоит переменная.

Предыдущее синтаксическое дерево задавало терм в нормальной форме. У терма $\lambda x\, y.\, (\lambda w.\, w)\, x\, (\lambda z.\, y\, z)\, v$ имеется головной редекс, его синтакстческое дерево имеет вид



Две основные стратегии редукции — нормальная и аппликативная. *Нормальная стратегия* сокращает самый левый внешний редекс (leftmost outermost). Пример:

$$\mathbf{K}\,(\mathbf{I}\,\mathbf{I})\,\mathbf{\Omega} = (\lambda x\,y.\,x)\,(\mathbf{I}\,\mathbf{I})\,\mathbf{\Omega} \longrightarrow_\beta (\lambda y.\,\mathbf{I}\,\mathbf{I})\,\mathbf{\Omega} \longrightarrow_\beta \mathbf{I}\,\mathbf{I} \longrightarrow_\beta \mathbf{I}$$

Аппликативная стратегия сокращает самый левый внутренний редекс (leftmost innermost). Пример:

$$\mathbf{K}\,(\mathbf{I}\,\mathbf{I})\,\mathbf{\Omega} = \mathbf{K}\,((\lambda x.\,x)\,\mathbf{I})\,\mathbf{\Omega} \longrightarrow_\beta \mathbf{K}\,\mathbf{I}\,\mathbf{\Omega} = (\lambda x\,y.\,x)\,\mathbf{I}\,\mathbf{\Omega} \longrightarrow_\beta (\lambda y.\,\mathbf{I})\,(\boldsymbol{\omega}\,\boldsymbol{\omega}) \longrightarrow_\beta \ldots$$

Синим в этих примерах выделен сокращаемый редекс.

Теорема 8 (о нормализации)**.** *Если терм* M *имеет* β*-NF, то последовательное сокращение самого левого внешнего редекса приводит к этой нормальной форме.*

То есть нормальная стратегия гарантированно нормализует нормализуемое. Можем доказывать отсутствие β-NF. Например, $\mathbf{K}\,\mathbf{\Omega}\,\mathbf{I}$ не имеет β-нормальной формы.

Недостаток нормальной стратегии — ее возможная неэффективность, достоинство в том, что она не считает ничего лишнего. Пусть N — вычислительно сложный терм, то есть содержащий много редексов. Если использовать нормальную стратегию, то в выражении

$$(\lambda x.\,F\,x\,(G\,x)\,x)\,N \longrightarrow_\beta F\,N\,(G\,N)\,N$$

в процессе дальнейших редукций редексы в N придётся сокращать три раза. Зато в

$$(\lambda x\,y\,z.\,y)\,N \longrightarrow_\beta \lambda y\,z.\,y$$

нормальная стратегия не вычисляет N ни разу. Аппликативная стратегия в обоих примерах вычислит N один раз.

Аппликативная стратегия похожа на стратегию вычислений («энергичную», eager) большинства языков программирования. Сначала вычисляется значение аргумента, затем происходит применение функции. Нормальная стратегия похожа на способ вычисления в «ленивых» (lazy) языках (Haskell, Clean).

Для решения проблем с эффективностью в «ленивых» языках используют *механизм разделения*. Вместо непосредственной подстановки терма вместо формального параметра подставляют указатель на этот терм (например, N в примере выше), который теперь хранится в памяти как *отложенное вычисление*. Когда (и если) в процессе дальнейших редукций нам потребуется значение терма, то вычисление форсируется, но, поскольку на этот терм может быть выставлено несколько указателей, его значение окажется вычисленным для всех дальнейших обращений по любому из указателей. Механизм разделения обычно реализуют через вычисления в контекстах или через редукцию на графах. [1]

Нет необходимости всегда доводить редукцию до NF. На практике часто ограничиваются так называемой слабой головной нормальной формой. *Слабая головная нормальная форма* (WHNF) — это HNF или лямбда-абстракция, то есть не редекс на верхнем уровне.

Вычисление только до WHNF позволяет избежать захвата переменной при редуцировании *замкнутого* терма. (Попробуйте доказать этот факт.)

При наличии констант[9] понятие WHNF и HNF дополняют частично применёнными константными функциями, например

$$\mathbf{and\ true}$$

поскольку его можно записать в η-эквивалентном WHNF-виде

$$\lambda x.\, \mathbf{and\ true}\ x$$

В Haskell к WHNF относят и конструктор данных, примененный полностью или частично.

Механизм вызова — термин, применяемый при исследовании высокоуровневых языков программирования. В функциональных языках:

- *вызов по значению* — аппликативный порядок редукций до WHNF;
- *вызов по имени* — нормальный порядок редукций до WHNF;
- *вызов по необходимости* — «вызов по имени» плюс разделение.

1.9 Индексы де Брёйна

Имеется другой способ описания лямбда-термов, так называемые ***индексы де Брёйна*** [10, 6]. Вместо того, чтобы именовать переменные в терме и использовать при связывании в лямбда-абстракции ссылку по имени, используется натуральное число, указывающее, сколько открытых областей видимости (лямбд) надо перепрыгнуть, чтобы добраться до той, где переменная связана. В этой нотации, например, комбинатор

[9] В расширенных системах лямбда-исчисления.

$K = \lambda x.\,\lambda y.\,x$ имеет вид $\lambda\,\lambda\,1$, а $K_* = \lambda x.\,\lambda y.\,y$ выглядит так: $\lambda\,\lambda\,0$. Сами лямбды при этом уже не содержат имен переменных, представляя собой структурные маркеры, задающие места связывания индексов, иногда называемые *биндерами* (binder).

Кодирование незамкнутых термов тоже не представляет сложности: индекс свободной переменной должен быть больше или равен количеству объемлющих позицию лямбд. То есть, например, $\lambda f.\,f\,x\,y$ кодируется как $\lambda\,0\,1\,2$. Порядок нумерации свободных переменных подразумевается фиксированным через набор биндеров в некотором внешнем окружении[10].

Важным преимуществом индексов де Брейна служит тот факт, что все α-эквивалентные термы с именованными переменными в этом представлении выражаются совершенно одинаково. Еще одним преимуществом служит отсутствие при вычислениях коллизий, подобных захвату свободной переменной. Однако процедура подстановки требует теперь пересчета всех свободных индексов подставлямого терма в каждом месте подстановки.

Дадим формальное описание индексов де Брейна. Термы задаются индуктивно:

$$\begin{array}{lcl} n \in \mathbb{N} & \Rightarrow & n \in \Lambda, \\ M, N \in \Lambda & \Rightarrow & (M\,N) \in \Lambda, \\ M \in \Lambda & \Rightarrow & (\lambda\,M) \in \Lambda. \end{array} \tag{17}$$

Стандартные соглашения остаются прежними: внешние скобки опускаются, операция применения термов ассоциативна влево, то есть $0\,1\,2$ обозначает $((0\,1)\,2)$. Тело терма простирается вправо насколько это возможно, то есть $\lambda\,0\,0$ обозначает $\lambda\,(0\,0)$, а не $(\lambda\,0)\,0$.

Базовой операцией для реализации преобразований над термами в представлении де Брёйна служит операция *сдвига* или *подъема* (shift или lifting), нотация $\uparrow_m^k$. Величина m называется *отсечением*, если значение индекса де Брейна больше или равно значению отсечения, то к нему применяется увеличения его значения на k.

$$\begin{array}{lcl} \uparrow_m^k n & = & \begin{cases} n, & \text{если } n < m, \\ n + k, & \text{если } n \geqslant m, \end{cases} \\ \uparrow_m^k (P\,Q) & = & (\uparrow_m^k P)\,(\uparrow_m^k Q), \\ \uparrow_m^k (\lambda\,P) & = & \lambda\,(\uparrow_{m+1}^k P). \end{array} \tag{18}$$

Последнее равенство говорит, что при переходе через каждую лямбду величина отсечения увеличивается, то есть операция подъема затрагивает только свободные переменные. На практике обычно используется версия с нулевым отсечением, для нее вводят нотацию $\uparrow^k = \uparrow_0^k$.

Операция псевдоподстановки $[i \rightsquigarrow N]\,M$ терма N вместо свободной переменной i в

[10] В системах с типами это линейно упорядоченный контекст типизации.

терм M выражается в терминах подъема следующим образом:

$$\begin{aligned}
[i \rightsquigarrow N]\, n &= \begin{cases} n, & \text{если } n < i, \\ \uparrow^n N, & \text{если } n = i, \\ n-1, & \text{если } n > i, \end{cases} \\
[i \rightsquigarrow N]\, (P\, Q) &= ([i \rightsquigarrow N]\, P)\, ([i \rightsquigarrow N]\, Q), \\
[i \rightsquigarrow N]\, (\lambda\, P) &= \lambda\, ([i+1 \rightsquigarrow N]\, P).
\end{aligned} \tag{19}$$

При этом одношаговая β-редукция определяется через псевдоподстановку так

$$(\lambda\, M)\, N \longrightarrow_\beta [0 \rightsquigarrow N]\, M. \tag{20}$$

Вместо псевдоподстановки (19), заточенной под то, чтобы правило (20) выглядело стандартным образом, можно использовать настоящую подстановку:

$$\begin{aligned}
[i \mapsto N]\, n &= \begin{cases} N, & \text{если } n = i, \\ n, & \text{если } n \neq i, \end{cases} \\
[i \mapsto N]\, (P\, Q) &= ([i \mapsto N]\, P)\, ([i \mapsto N]\, Q), \\
[i \mapsto N]\, (\lambda\, P) &= \lambda\, (\left[i+1 \mapsto \uparrow^1 N\right] P).
\end{aligned} \tag{21}$$

В терминах подстановки одношаговая β-редукция определяется так

$$(\lambda\, M)\, N \longrightarrow_\beta \uparrow^{-1} [0 \mapsto \uparrow^1 N]\, M. \tag{22}$$

Реализация с псевдоподстановкой, в которой сосредоточены все операции сдвига, обычно оказывается слегка эффективнее, поскольку позволяет выполнить все сдвиги в N единомоментно.

Аналогично вводится η-редукция: если в терме M нет ссылок на внешний биндер, то

$$\lambda\, M\, 0 \longrightarrow_\eta \uparrow^{-1} M. \tag{23}$$

2 Простые типы

Помимо описанного выше бестипового лямбда-исчисления имеется большое количество его типизированных версий [8, 3, 4, 10, 11]. Идея использования типов для классификации лямбда-термов имеет давнюю историю и связана с именами Хаскелла Карри и Алонсо Черча. Типы рассматриваются *синтаксические* конструкции, приписываемые термам по определенным правилам:

$$M : \sigma$$

Здесь M — терм, σ — тип, а двоеточие — оператор, задающий отношение типизации[11]. Для того, чтобы построить систему типов нужно задать, во-первых, правила конструирования типов и, во-вторых, правила приписывания типов термам.

В λ-исчислении с типами выделяют два семейства систем типов:

- Системы в стиле Карри. Термы те же, что и в бестиповой теории. Каждый терм может обладать множеством различных типов (пустое, одно- или многоэлементное, бесконечное).

- Системы в стиле Чёрча. Термы — аннотированные версии бестиповых термов. Каждый терм имеет тип (обычно уникальный), выводимый из способа, которым терм аннотирован.

Можно рассматривать системы типов с двух различных профессиональных точек зрения. Подход программиста таков: термы интерпретируются как программы, а типы — как их частичные спецификации. При этом системы в стиле Карри характерны для языков с неявной типизацией (например, Haskell, Ocaml). Системы в стиле Чёрча больше похожи на языки с явной типизацией, к ним относится большинство других типизированных языков.

Подход логиков к системам типов иной: типы интерпретируются как высказывания, а термы — как их доказательства. Связь между системами типов и логическими системами называют *соответствием Карри-Говарда* или *изоморфизмом Карри-Говарда*, если это соответствие описано на формальном математическом языке.

В большинстве систем типов имеется базовый способ конструирования типа. Это стрелка, задающая функциональный тип. Например, функции succ, возвращающей натуральное число, следующее за переданным ей натуральным аргументом, может быть приписан тип $\mathbb{N} \to \mathbb{N}$

$$\mathrm{succ} : \mathbb{N} \to \mathbb{N}$$

Стрелка — общепринятый в математике способ связывать область определения и множество значений функций. В системах типов для лямбда исчисления такая практика формализуется.

Приведенный пример содержит конкретный тип данных натуральных чисел $\mathbb{N}$. Если рассматривать неспецифицированные типы, то для их обозначения принято использовать греческие буквы в нижнем регистре. В общем случае типом функции из типа α в тип β является тип $\alpha \to \beta$.

В чистом лямбда-исчислении, например, тождественному комбинатору $\mathbf{I} \equiv \lambda x.\, x$ может быть приписан тип $\alpha \to \alpha$

$$\mathbf{I} : \alpha \to \alpha$$

Если имеется некоторый y типа α, то применение функции $\mathbf{I}$ к этому y, то есть выражение $\mathbf{I}\, y$ тоже имеет тип α. Это довольно естественное наблюдение позже будет

[11] Считается, что оператор типизации «двоеточие» имеет очень низкий приоритет; если M и σ — выражения сконструированные каким-то образом, то их не нужно заключать в скобки.

оформлено как правило типизации. Гипотезу о том, что терм-переменная имеет некоторый тип, удобно оформлять в виде *контекста*[12]:

$$y : \alpha \vdash \mathbf{I}\, y : \alpha$$

Читается это так: в предположении, что переменная y имеет тип α, терм $\mathbf{I}\, y$ имеет тип α.

2.1 Просто типизированное λ-исчисление

Самая простая система типов это *просто типизированное* λ*-исчисление*. Так же ее называют системой $\lambda_{\to}$ или Simple Type Theory (STT). В этой системе единственным способом конструирования типов является использование функциональной стрелки.

Множество типов $\mathbb{T}$ системы $\lambda_{\to}$ определяется индуктивно:

$$\alpha, \beta, \ldots \in \mathbb{T} \qquad \text{(переменные типа)}$$

$$\sigma, \tau \in \mathbb{T} \Rightarrow (\sigma \to \tau) \in \mathbb{T} \qquad \text{(типы пространства функций)}$$

или, в абстрактном синтаксисе:

$$\mathbb{T} \ ::= \ \mathbb{V} \mid (\mathbb{T} \to \mathbb{T})$$

Здесь $\mathbb{V} = \{\alpha, \beta, \ldots\}$ — множество типовых переменных. Мы будем следовать соглашению: буквы из начала греческого алфавита $\alpha, \beta, \gamma, \ldots$ используются для типовых переменных, а $\sigma, \tau, \rho, \ldots$ — как метапеременные, для произвольных типов. Часто типовые переменные называют константами, иногда их число ограничивают заданным конечным набором.

При записи типов внешние скобки, как обычно, опускают, а функциональную стрелку считают *правоассоциативным* оператором: если $\sigma_1, \ldots, \sigma_n \in \mathbb{T}$, то

$$\sigma_1 \to \sigma_2 \to \ldots \to \sigma_n \ \equiv \ (\sigma_1 \to (\sigma_2 \to \ldots \to (\sigma_{n-1} \to \sigma_n) \ldots))$$

Примеры типов:

$$(\alpha \to \beta) \equiv \alpha \to \beta$$

$$(\alpha \to (\beta \to \gamma)) \equiv \alpha \to \beta \to \gamma$$

$$((\alpha \to \beta) \to \gamma) \equiv (\alpha \to \beta) \to \gamma$$

$$((\alpha \to \beta) \to ((\beta \to \gamma) \to (\alpha \to \gamma))) \equiv (\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$$

$$((\alpha \to \beta) \to (((\alpha \to \beta) \to \beta) \to \beta)) \equiv (\alpha \to \beta) \to ((\alpha \to \beta) \to \beta) \to \beta$$

[12]Приоритет оператора $\vdash$ (*турникет* или *штопор*) полагают еще более низким, чем оператора типизации «двоеточие».

Первый и третий примеры являются типом функции одного аргумента, второй и пятый — двух.

Всякий тип в $\lambda_{\to}$ может быть записан в виде

$$\sigma_1 \to \sigma_2 \to \ldots \to \sigma_n \to \alpha$$

где, α — некоторая переменная. Действительно, если «в конце» типа стоит не переменная, то в простой системе это может быть только стрелка, и, исходя из правоассоциативности, мы можем добавить в список аргументов типа то, что окажется слева от этой стрелки.

Каковы правила приписывания типов термам? Если терм *переменная*, это допустимо делать произвольным образом. Переменной может быть приписан любой допустимый тип:

$$\begin{aligned} &x : \alpha \\ &y : \alpha \to \beta \\ &z : (\alpha \to \beta) \to ((\alpha \to \beta) \to \beta) \to \beta \end{aligned}$$

Если же терм представляет собой *аппликацию* $M\,N$, то

- M должно быть функцией, то есть иметь стрелочный тип, $M : \sigma \to \tau$;
- N должно быть «подходящим» аргументом, то есть иметь тип σ, совпадающий с типом аргумента стрелки;
- вся аппликация при этом будет иметь тип результата функции: $M\,N : \tau$.

Примеры приписывания типов для аппликации

$$\begin{aligned} x{:}\alpha, y{:}\alpha \to \beta &\quad\vdash\quad y\,x : \beta \\ x{:}\alpha, y{:}\alpha \to \beta, z{:}\beta \to \gamma &\quad\vdash\quad z\,(y\,x) : \gamma \end{aligned}$$

Отметим использование контекстов для приписывания типов свободным переменным целевого терма.

Упражнение. Какие должны иметь типы x и y, чтобы $x\,(y\,x){:}\gamma$?

Если терм является *абстракцией* $\lambda x.\,M$, то

- его тип должен быть стрелочным $\lambda x.\,M : \sigma \to \tau$;
- тип переменной x, по которой происходит абстракция, должен быть σ, то есть совпадать с типом аргумента стрелки;
- тип тела M должен быть τ, то есть совпадать с типом результата стрелки.

Например, в предположении, что $x{:}\alpha$, имеем

$$\lambda x.\,x : \alpha \to \alpha$$

А надо ли здесь в контексте указывать, что $x : \alpha$? Имеет ли смысл запись $x : \alpha \;\vdash \lambda x.\,x : \alpha \to \alpha$? Ответ — нет, это не очень удачное решение. Переменная x — связанная,

иначе говоря, локальная, она доступна только внутри связывающей ее лямбды. В более широкой области видимости может присутствовать другая переменная с тем же именем. Контекст же глобален, поэтому упоминание в нем имен локальных переменных может приводить к неоднозначности.

Однако если совсем не иметь возможности указать, что $x{:}\alpha$, то допустимы и типизация $\lambda x.\, x : \beta \to \beta$ и даже $\lambda x.\, x : (\alpha \to \beta) \to (\alpha \to \beta)$. Это так называемая типизация в *стиле Карри*. В системах Карри если терму удалось приписать какой-то тип, то это оказывается можно сделать бесконечным количеством способов.

Альтернатива — указать тип переменой при абстракции, внутри абстрактора: $\lambda x^{\alpha}.\, x : \alpha \to \alpha$. В этом случае тип терма определяется однозначно. Это типизация в *стиле Чёрча*.

Упражнение. Типизируйте по Чёрчу: $\lambda x^{?}.\, \lambda y^{?}.\, x\,(y\,x) : ?$

Правила ассоциативности для типов (вправо) и аппликации (влево) хорошо согласованы друг с другом: они позволяют работать с функциями многих переменных без использования скобок. В предположении о типах термов

$$\begin{array}{lcl} F & : & \alpha \to (\beta \to (\gamma \to \delta)) \\ M & : & \alpha \\ N & : & \beta \\ P & : & \gamma \end{array}$$

имеем

$$\begin{array}{lcl} F\,M & : & \beta \to (\gamma \to \delta) \\ (F\,M)\,N & : & \gamma \to \delta \\ ((F\,M)\,N)\,P & : & \delta \end{array}$$

Во всех приведенных примерах зелёные скобки необязательны и обычно опускаются в соответствии с соглашениями об ассоциативности.

Правила ассоциативности для типов (вправо) и абстракций (тоже вправо) тоже хорошо согласованы. В предположении $Q : \gamma$

$$\begin{array}{lcl} \lambda y^{\beta}.\, Q & : & \alpha \to \gamma \\ \lambda x^{\alpha}.\, (\lambda y^{\beta}.\, Q) & : & \alpha \to (\beta \to \gamma) \end{array}$$

2.2 Формализм $\lambda_{\to}$

2.2.1 Предтермы

Начнем с системы $\lambda_{\to}$ в стиле Карри. Множество Λ ее *предтермов* (или *псевдотермов*) строится из переменных из множества $V = \{x, y, z, \ldots\}$ с помощью аппликации и абстракции:

$$\begin{array}{lcl} x \in V & \Rightarrow & x \in \Lambda \\ M, N \in \Lambda & \Rightarrow & (M\,N) \in \Lambda \\ M \in \Lambda, x \in V & \Rightarrow & (\lambda x.\, M) \in \Lambda \end{array}$$

В абстрактном синтаксисе

$$\Lambda \;::=\; V \mid (\Lambda\,\Lambda) \mid (\lambda V.\, \Lambda)$$

То есть предтермы системы в стиле Карри — это в точности термы бестипового λ-исчисления. Все соглашения о правилах опускания скобок такие же как и для термов бестипового исчисления.

Само понятие предтерма возникает из того соображения, что не всем термам в изучаемой системе можно будет приписать тип. Те предтермы, которые смогут быть типизированными, получат название *допустимых* термов (или просто термов). Остальные так и останутся в статусе предтермов.

Примеры предтермов в стиле Карри:

$$\lambda x\, y.\, x$$
$$\lambda f\, g\, x.\, f\,(g\, x)$$
$$\lambda x.\, x\, x$$

Первым двум из них мы позже сможем приписать тип в $\lambda_{\to}$, а третьему нет.

Для систем в стиле Черча предтермы определяются похожим образом. Единственным, но ключевым отличием является указание типа переменной при абстракции. Множество $\Lambda_{\mathbb{T}}$ предтермов системы в стиле Черча строится из переменных из $V = \{x, y, z, \ldots\}$ с помощью аппликации и аннотированной типами абстракции:

$$\begin{array}{rcl} x \in V & \Rightarrow & x \in \Lambda_{\mathbb{T}} \\ M, N \in \Lambda_{\mathbb{T}} & \Rightarrow & (M\, N) \in \Lambda_{\mathbb{T}} \\ M \in \Lambda_{\mathbb{T}}, x \in V, \sigma \in \mathbb{T} & \Rightarrow & (\lambda x^{\sigma}.\, M) \in \Lambda_{\mathbb{T}} \end{array}$$

В абстрактном синтаксисе

$$\Lambda_{\mathbb{T}} \quad ::= \quad V \mid (\Lambda_{\mathbb{T}}\, \Lambda_{\mathbb{T}}) \mid (\lambda V^{\mathbb{T}}.\, \Lambda_{\mathbb{T}})$$

Все соглашения о скобках и ассоциативности те же, что и в системе Λ.

Примеры предтермов в стиле Черча мы записываем ниже в двух стилях. Первый, с типизацией связываемых переменных в виде степени, мы ввели выше. Второй, в котором тип связываемой переменной указывается с помощью стандартного оператора типизации «двоеточие», тоже допустим, но менее удобен.

$$\begin{array}{lcl} \lambda x^{\alpha}\, y^{\beta}.\, x & \equiv & \lambda x{:}\alpha.\, \lambda y{:}\beta.\, x \\ \lambda x^{\alpha}\, y^{\alpha}.\, x & \equiv & \lambda x{:}\alpha.\, \lambda y{:}\alpha.\, x \\ \lambda f^{\alpha}\, g^{\beta}\, x^{\gamma}.\, f\,(g\, x) & \equiv & \lambda f{:}\alpha.\, \lambda g{:}\beta.\, \lambda x{:}\gamma.\, f\,(g\, x) \\ \lambda f^{\beta\to\gamma}\, g^{\alpha\to\beta}\, x^{\alpha}.\, f\,(g\, x) & \equiv & \lambda f{:}(\beta \to \gamma).\, \lambda g{:}(\alpha \to \beta).\, \lambda x{:}\alpha.\, f\,(g\, x) \\ \lambda x^{\alpha}.\, x\, x & \equiv & \lambda x{:}\alpha.\, x\, x \end{array}$$

Первым двум и четвертому из них мы позже сможем приписать тип в $\lambda_{\to}$, а остальным нет. Это связано с тем, что мы приписали тип всем связываемым термовым переменным в виде переменных типа. Это неверно, поскольку в теле терма некоторые из этих термовых переменных используются в аппликациях справа, то есть должны иметь стрелочный тип. Четвертый пример исправляет эту проблему, он является допустимым термом. В последнем же примере даже замена α на какой бы то ни было стрелочный тип не поможет: получившейся предтерм все равно не удастся типизировать.

2.2.2 Контексты

Утверждением типизации называется приписывание терму типа

$$M : \tau$$

где $M \in \Lambda$ и $\tau \in \mathbb{T}$. Тип τ иногда называют *предикатом*, а терм M — *субъектом* утверждения. Это определение верно для систем Карри, для $\lambda_{\to}$ «а ля Чёрч» надо лишь заменить Λ на $\Lambda_{\mathbb{T}}$.

Примеры (верных) утверждений типизации:

Система в стиле Карри	Система в стиле Чёрча
$\lambda x.\, x : \alpha \to \alpha$	$\lambda x^{\alpha}.\, x : \alpha \to \alpha$
$\lambda x.\, x : (\alpha \to \beta) \to \alpha \to \beta$	$\lambda x^{\alpha\to\beta}.\, x : (\alpha \to \beta) \to \alpha \to \beta$
$\lambda x\, y.\, x : \alpha \to \beta \to \alpha$	$\lambda x^{\alpha}\, y^{\beta}.\, x : \alpha \to \beta \to \alpha$

То, что эти утверждения типизации верны, пока только декларация. Вскоре, введя правила типизации, мы сможем доказывать подобные факты.

Объявление — это утверждение типизации с термовой переменной в качестве субъекта. Примеры объявлений

$$\begin{gathered} x : \alpha \\ y : \beta \\ f : \alpha \to \beta \\ g : (\alpha \to \beta) \to \gamma \end{gathered}$$

Контекст — это множество объявлений с *различными* термовыми переменными в качестве субъекта:

$$\{x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n\}$$

Контекст иногда называют *базисом* или *окружением*. Пустой контекст также допустим, его обозначают как пустое множество $\emptyset$.

Для именования контекстов принято использовать греческие буквы в верхнем регистре. Фигурные скобки множества часто опускают:

$$\Gamma \;=\; x{:}\alpha, y{:}\beta, f{:}\alpha \to \beta, g{:}(\alpha \to \beta) \to \gamma$$

В выкладках часто удобнее записывать объявления в контекстах не с помощью оператора типизации, а в виде степени:

$$x{:}\alpha, y{:}\beta, f{:}\alpha \to \beta, g{:}(\alpha \to \beta) \to \gamma \;\equiv\; x^{\alpha}, y^{\beta}, f^{\alpha\to\beta}, g^{(\alpha\to\beta)\to\gamma}$$

Контексты можно *расширять*, добавляя объявление типизации свежей для данного контекста переменной:

$$\Delta \;=\; \Gamma, z^{\alpha\to\gamma} \;=\; x^{\alpha}, y^{\beta}, f^{\alpha\to\beta}, g^{(\alpha\to\beta)\to\gamma}, z^{\alpha\to\gamma}$$

Контекст можно рассматривать как (частичную) функцию из множества переменных V в множество типов $\mathbb{T}$.

2.2.3 Правила типизации

Сформулируем теперь правила типизации $\lambda_{\to}$ «а ля Карри».

Утверждение $M : \tau$ называется *выводимым* в контексте Γ, обозначение

$$\Gamma \vdash M : \tau$$

если его вывод может быть произведен по правилам:

$$\begin{array}{rcl} x^\sigma \in \Gamma & \Rightarrow & \Gamma \vdash x : \sigma \\ \Gamma \vdash M : \sigma \to \tau,\ \ \Gamma \vdash N : \sigma & \Rightarrow & \Gamma \vdash M\,N : \tau \\ \Gamma, x^\sigma \vdash M : \tau & \Rightarrow & \Gamma \vdash \lambda x.\, M : \sigma \to \tau \end{array}$$

Если существуют Γ и τ, такие что $\Gamma \vdash M : \tau$, то предтерм M называют *(допустимым) термом.*

Удобно записывать эти правила в несколько другом виде.

(аксиома) $\Gamma \vdash x{:}\sigma$, если $x^\sigma \in \Gamma$

$$(\to \text{Elim}) \quad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau}$$

$$(\to \text{Intro}) \quad \frac{\Gamma, x^\sigma \vdash M : \tau}{\Gamma \vdash \lambda x.\, M : \sigma \to \tau}$$

Сверху от черты расположены необходимые посылки, а снизу — заключения правила вывода. Это позволяет оформлять вывод утверждения типизации в виде дерева. Аксиома задает листья этого дерева, правило введения стрелки ($\to$ Intro) формирует ствол и ветви, а правило удаления стрелки ($\to$ Elim) отвечает за ветвления.

Например для для комбинатора $\mathbf{K}$ вывод утверждения типизации $\lambda x\, y.\, x : \alpha \to \beta \to \alpha$ в виде дерева таков

$$\dfrac{\dfrac{x^\alpha, y^\beta \vdash x : \alpha}{x^\alpha \vdash \lambda y.\, x : \beta \to \alpha}\ (\to \text{I})}{\emptyset \vdash \lambda x\, y.\, x : \alpha \to \beta \to \alpha}\ (\to \text{I})$$

Пустой контекст часто обозначают не в виде значка пустого множества, а просто оставляя место слева от турникета пустым: $\vdash \lambda x\, y.\, x : \alpha \to \beta \to \alpha$.

Отметим, что приведенный выше вывод годится не только для конкретных переменных типа α и β. Заменяя их на произвольные $\sigma, \tau \in \mathbb{T}$, получим правильное дерево

вывода для $\vdash \lambda x\, y.\, x : \sigma \to \tau \to \sigma$. Таким образом мы видим, что написав для некоторого терма одно дерево вывода типов в системе Карри, мы можем породить бесконечное количество таких деревьев, и, следовательно, бесконечное количество типов для исходного терма.

Приведем еще один пример вывода типов, для комбинатора **B**:

$$\dfrac{\dfrac{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash f : \beta \to \gamma \qquad \dfrac{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash g : \alpha \to \beta \qquad f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash x : \alpha}{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash g\, x : \beta}\,(\to \mathrm{E})}{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash f\,(g\, x) : \gamma}\,(\to \mathrm{E})}{f^{\beta\to\gamma}, g^{\alpha\to\beta} \vdash \lambda x.\, f\,(g\, x) : \alpha \to \gamma}\,(\to \mathrm{I})}{f^{\beta\to\gamma} \vdash \lambda g\, x.\, f\,(g\, x) : (\alpha \to \beta) \to \alpha \to \gamma}\,(\to \mathrm{I})}{\vdash \lambda f\, g\, x.\, f\,(g\, x) : (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma}\,(\to \mathrm{I})$$

Отметим, что если для комбинатора может быть выведен тип, то он всегда может быть выведен в пустом контексте.

Сформулируем правила типизации $\lambda_{\to}$ для черчевской версии $\lambda_{\to}$.

(аксиома)	$\Gamma \vdash x : \sigma$, если $x^{\sigma} \in \Gamma$
$(\to \mathrm{E})$	$\dfrac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\, N : \tau}$
$(\to \mathrm{I})$	$\dfrac{\Gamma, x^{\sigma} \vdash M : \tau}{\Gamma \vdash \lambda x^{\sigma}.\, M : \sigma \to \tau}$

Как видим визуально отличия минимальны. Однако вывод типа окажется проще, поскольку не нужно делать предположений о типах связанных переменных.

Пример дерева вывода типа для $\lambda x^{\alpha}\, y^{\beta}.\, x$:

$$\dfrac{\dfrac{x^{\alpha}, y^{\beta} \vdash x : \alpha}{x^{\alpha} \vdash \lambda y^{\beta}.\, x : \beta \to \alpha}\,(\to \mathrm{I})}{\vdash \lambda x^{\alpha}\, y^{\beta}.\, x : \alpha \to \beta \to \alpha}\,(\to \mathrm{I})$$

Заменяя переменные типа α и β в этом дереве на произвольные $\sigma, \tau \in \mathbb{T}$, снова получим правильное дерево вывода. То есть для каждой пары $\sigma, \tau \in \mathbb{T}$ верно $\vdash \lambda x^{\sigma}\, y^{\tau}.\, x : \sigma \to \tau \to \sigma$. Однако термы в черчевской системе содержат типовую аннотацию, поэтому для каждой такой пары мы типизируем *разные* термы. Можно формально доказать, что у каждого замкнутого терма в системе «а ля Чёрч» имеется единственный тип[13].

Приведем в заключение пример вывода типа для комбинатора **B** в черчевской системе:

[13] А у каждого замкнутого предтерма — не более одного.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash f : \beta\to\gamma \qquad \dfrac{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash g : \alpha\to\beta \qquad f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash x : \alpha}{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash g\,x : \beta}\,(\to \mathrm{E})}{f^{\beta\to\gamma}, g^{\alpha\to\beta}, x^{\alpha} \vdash f\,(g\,x) : \gamma}\,(\to \mathrm{E})}{f^{\beta\to\gamma}, g^{\alpha\to\beta} \vdash \lambda x^{\alpha}.\, f\,(g\,x) : \alpha\to\gamma}\,(\to \mathrm{I})}{f^{\beta\to\gamma} \vdash \lambda g^{\alpha\to\beta}\, x^{\alpha}.\, f\,(g\,x) : (\alpha\to\beta)\to\alpha\to\gamma}\,(\to \mathrm{I})}{\vdash \lambda f^{\beta\to\gamma}\, g^{\alpha\to\beta}\, x^{\alpha}.\, f\,(g\,x) : (\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma}\,(\to \mathrm{I})$$

2.3 Свойства простой системы

Мы в этом разделе не доказываем формальным образом почти никаких утверждений, за формальными индуктивными доказательствами отсылаем к [3]. Если конкретная система (Карри или Черча) не указана, то утверждение относится к обеим системам. В случае расхождения свойств утверждение о системе Карри синее, а о системе Черча — красное.

2.3.1 Вспомогательные леммы, нетипизируемые предтермы

Начнем знакомство со свойствами систем $\lambda_{\to}$ с ряда вспомогательных технических лемм.

Лемма 8 (инверсии (генерации)).

- $\Gamma \vdash x : \sigma \;\Rightarrow\; x^{\sigma} \in \Gamma$.
- $\Gamma \vdash M\,N : \tau \;\Rightarrow\; \exists\sigma\ [\Gamma \vdash M : \sigma\to\tau \;\wedge\; \Gamma \vdash N : \sigma]$.
- $\Gamma \vdash \lambda x.\, M : \rho \;\Rightarrow\; \exists\sigma,\tau\ [\Gamma, x^{\sigma} \vdash M : \tau \;\wedge\; \rho \equiv \sigma\to\tau]$.
  ($\lambda_{\to}$ а ля Карри)
- $\Gamma \vdash \lambda x^{\sigma}.\, M : \rho \;\Rightarrow\; \exists\tau\ [\Gamma, x^{\sigma} \vdash M : \tau \;\wedge\; \rho \equiv \sigma\to\tau]$.
  ($\lambda_{\to}$ а ля Чёрч)

Эта лемма представляет собой прочитанные наоборот, снизу вверх, правила приписывания типов термам. Отметим, что это не механическая операция, верная для любых систем типов. Лемма верна потому, что для каждого из трех способов формирования термов (переменная, аппликация, абстракция) в системе $\lambda_{\to}$ есть в точности одно правило типизации.

Лемма 9 (о типизируемости подтерма). *Пусть M' — подтерм M. Тогда $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma' \vdash M' : \sigma'$ для некоторых Γ' и σ'. То есть, если терм имеет тип, то каждый его подтерм тоже имеет тип.*

Следующая группа лемм описывает свойства контекстов.

Лемма 10 («разбавления» (Thinning))**.** *Пусть* Γ *и* Δ *— контексты, причём* $\Delta \supseteq \Gamma$*. Тогда* $\Gamma \vdash M : \sigma \;\Rightarrow\; \Delta \vdash M : \sigma$*. Расширение контекста не влияет на выводимость утверждения типизации.*

Лемма 11 (о свободных переменных)**.** $\Gamma \vdash M : \sigma \;\Rightarrow\; \mathsf{FV}(M) \subseteq \mathrm{dom}(\Gamma)$*. Свободные переменные типизированного терма должны присутствовать в контексте.*

Лемма 12 (сужения)**.** $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma \restriction \mathsf{FV}(M) \vdash M : \sigma$*. Сужение контекста до множества свободных переменных терма не влияет на выводимость утверждения типизации.*

Вместе эти леммы отвечают на вопрос: какой контекст требуется, чтобы произвести присваивание типов? Ответ следующий: в контексте обязательно должны присутствовать свободные переменные типизируемого терма; все остальные переменные опциональны, не влияют на типизацию и могут быть безболезненно отброшены[14].

Приведенных лемм достаточно, чтобы продемонстрировать, что в системах $\lambda_{\to}$ есть нетипизируемые предтермы. Рассмотрим предтерм содержащий самоприменение, например $x\,x$. Предположим, что это терм. Тогда имеются контекст Γ и тип τ, такие что

$$\Gamma \vdash x\,x : \tau$$

По лемме об инверсии (пункт 2, аппликация) существует такой σ, что правый подтерм $x : \sigma$, а левый подтерм (тоже x) имеет тип $\sigma \to \tau$. По лемме о свободных переменных $x \in \mathrm{dom}(\Gamma)$ и должен иметь там *единственное* связывание по определению контекста. Возникает требование $\sigma = \sigma \to \tau$. Но тип не может быть несобственным подвыражением самого себя, поскольку (*и пока*) типы конечны. Поэтому предтерму $x\,x$ в в системах $\lambda_{\to}$ не может быть приписан никакой тип.

По лемме о типизируемости подтерма предтермы $\boldsymbol{\omega} = \lambda x.\, x\,x$, $\boldsymbol{\Omega} = \boldsymbol{\omega}\,\boldsymbol{\omega}$ и $\boldsymbol{Y} = \lambda f.\, (\lambda x.\, f(x\,x))(\lambda x.\, f(x\,x))$ не имеют типа в $\lambda_{\to}$. Иногда этот факт записывают следующим образом

$$x^{\sigma} \nvdash x\,x : \tau, \quad \nvdash \boldsymbol{\omega} : \sigma, \quad \nvdash \boldsymbol{\Omega} : \sigma, \quad \nvdash \boldsymbol{Y} : \sigma.$$

2.3.2 Подстановка типа, подстановка терма, редукция субъекта

Обсудим теперь какие преобразования сохраняют утверждение типизации. Начнем с преобразований над типами.

Для $\sigma, \tau \in \mathbb{T}$ *подстановку* типа τ вместо всех вхождений переменной α в тип σ обозначим $[\alpha \mapsto \tau]\sigma$. Например, для типа $\alpha \to \beta \to \alpha$ подстановка в него типа $\gamma \to \gamma$ вместо переменной α записывается

$$[\alpha \mapsto (\gamma \to \gamma)](\alpha \to \beta \to \alpha)$$

[14] Это не означает, что они совсем не нужны. При построении дерева вывода типа нам на промежуточных этапах могут требоваться более широкие контексты, чем это необходимо для типизации текущего «уровня».

и дает в результате тип

$$(\gamma \to \gamma) \to \beta \to \gamma \to \gamma$$

Подстановку можно так же осуществлять в контекст, при этом она выполняется во все типы-предикаты объявлений этого контекста. В системе Черча термы содержат типы, в этом случае можно определить подстановку типа в терм.

Мы уже отмечали, что замена переменной типа на произвольный тип не портит дерева вывода типов: все правила вывода остаются верными при этой операции. Поэтому верно следующее утверждение.

Лемма 13 (подстановки типа)**.**
$\Gamma \vdash M : \sigma \;\Rightarrow\; [\alpha \mapsto \tau]\Gamma \vdash M : [\alpha \mapsto \tau]\sigma$. *($\lambda_{\to}$ Карри)*
$\Gamma \vdash M : \sigma \;\Rightarrow\; [\alpha \mapsto \tau]\Gamma \vdash [\alpha \mapsto \tau]M : [\alpha \mapsto \tau]\sigma$. *($\lambda_{\to}$ Чёрч)*
Подстановка в выводимое утверждение типизации некоторого типа вместо переменной типа порождает выводимое утверждение типизации.

Например, подстановка $[\alpha \mapsto (\gamma \to \gamma)]$ в утверждение типизации для системы Черча

$$x^{\alpha} \;\vdash\; \lambda y^{\alpha}\, z^{\beta}.\, x : \alpha \to \beta \to \alpha$$

осуществляется и в тип, и в терм, и в контекст и дает новое утверждение типизации

$$x^{\gamma\to\gamma} \;\vdash\; (\lambda y^{\gamma\to\gamma}\, z^{\beta}.\, x) : (\gamma \to \gamma) \to \beta \to \gamma \to \gamma$$

Поскольку первое утверждение типизации выводимо (проверьте это), то и второе тоже выводимо.

Перейдем теперь к обсуждению преобразований над термами, сохраняющими утверждение типизации. По аналогии с типами рассмотрим подстановку терма вместо термовой переменной. Очевидно, что для сохранения типизации они должны быть одного типа.

Лемма 14 (подстановки терма)**.** *Пусть* $\Gamma, x^{\sigma} \vdash M : \tau$ *и* $\Gamma \vdash N : \sigma$*, тогда* $\Gamma \vdash [x \mapsto N]M : \tau$*. Подходящая по типу подстановка терма сохраняет тип.*

Пример. Берём выводимое утверждение типизации

$$x^{\gamma\to\gamma} \;\vdash\; \lambda y^{\beta}.\, x : \beta \to \gamma \to \gamma$$

и подставляем в него вместо свободной переменной x типа $\gamma \to \gamma$ терм $\lambda z^{\gamma}.\, z$ подходящего типа $\gamma \to \gamma$. Получаем

$$\vdash\; \lambda y^{\beta}\, z^{\gamma}.\, z : \beta \to \gamma \to \gamma$$

Упражнение. Что произойдет с деревом вывода типа при такой подстановке?

Лемма подстановки терма позволяет доказать теорему о сохранении типа в процессе вычислений.

Теорема 9 (о редукции субъекта)**.** *Пусть* $M \twoheadrightarrow_\beta N$. *Тогда* $\Gamma \vdash M : \sigma \;\Rightarrow\; \Gamma \vdash N : \sigma$. *Тип терма сохраняется при* β*-редукциях.*

С практической точки зрения это одно из ключевых свойств любой системы типов, системы которые нарушают это свойство сложно было бы использовать.

Следствие 1. *Множество типизируемых в* $\lambda_\to$ *термов замкнуто относительно редукции.*

В обратную сторону эта теорема (и следствие из нее) не верны для $\lambda_\to$. (TODO примерчики этого.)

Изучаемые до сих пор свойства были общими для систем Карри и Черча. Следующее свойство, единственности типа верно только для черчевской версии $\lambda_\to$.

Теорема 10 (о единственности типа для $\lambda_\to$ а ля Чёрч)**.** *Пусть* $\Gamma \vdash M : \sigma$ *и* $\Gamma \vdash M : \tau$. *Тогда* $\sigma \equiv \tau$. *Терм в* $\lambda_\to$ *а ля Чёрч имеет единственный тип.*

Следствие 2. *Пусть* $\Gamma \vdash M : \sigma$, $\Gamma \vdash N : \tau$ *и* $M =_\beta N$. *Тогда* $\sigma \equiv \tau$. *Типизируемые* β*-конвертируемые термы имеют одинаковый тип в* $\lambda_\to$ *а ля Чёрч.*

Для системы а ля Карри единственности типа нет. Несложно привести пример, иллюстрирующий этот факт. Оба приведенных ниже типа подходят для $\mathbf{K} = \lambda x\, y.\, x$ в $\lambda_\to$ а ля Карри:

$$\vdash \lambda x\, y.\, x : \alpha \to (\delta \to \gamma \to \delta) \to \alpha$$
$$\vdash \lambda x\, y.\, x : (\gamma \to \gamma) \to \beta \to \gamma \to \gamma$$

В целом между системами Карри и Чёрча имеется тесная связь. Можно задать стирающее отображение $|\cdot| : \Lambda_{\mathbb{T}} \to \Lambda$:

$$\begin{aligned} |x| &\equiv x \\ |M\,N| &\equiv |M|\,|N| \\ |\lambda x^\sigma.\, M| &\equiv \lambda x.\, |M| \end{aligned}$$

С помощью этого отображения все атрибутированные типами термы из версии Чёрча $\lambda{\to}$ «проектируются» в термы в версии Карри:

$$M \in \Lambda_{\mathbb{T}} \;\wedge\; \Gamma \vdash_{\text{Ч}} M : \sigma \;\Rightarrow\; \Gamma \vdash_{\text{К}} |M| : \sigma$$

Обратно, типизированные термы из версии Карри $\lambda{\to}$ могут быть «подняты» в термы из версии Чёрча, приписыванием переменным в лямбдах типовых атрибутов, однозначно восстанавливаемых из типа терма:

$$M \in \Lambda \;\wedge\; \Gamma \vdash_{\text{К}} M : \sigma \;\Rightarrow\; \exists N \in \Lambda_{\mathbb{T}}\, [\Gamma \vdash_{\text{Ч}} N : \sigma \;\wedge\; |N| \equiv M]$$

Для произвольного типа $\sigma \in \mathbb{T}$ выполняется

$$\sigma \text{ обитаем в } \lambda{\to}\text{-Карри} \;\Leftrightarrow\; \sigma \text{ обитаем в } \lambda{\to}\text{-Чёрч}$$

2.3.3 Прочая метатеория

Для любой системы типов важную роль имеют проблемы разрешимости основных практических задач: можем ли мы алгоритмически осуществить проверку выводимости утверждения типизации, генерации типа по терму или терма по типу.

Следующая таблица описывает эти проблемы и их названия

$\vdash M : \sigma?$	Задача проверки типа	ЗПТ
	Type Checking Problem	TCP
$\vdash M : ?$	Задача синтеза/приписывания/вывода типа	ЗСТ, ЗВТ
	Type Synthesis/Assgnment/Inference Problem	TSP, TAP
$\vdash ? : \sigma$	Задача обитаемости типа	ЗОТ
	Type Inhabitation Problem	TIP

Для обеих версий системы $\lambda_{\to}$, и в стиле Чёрча, и в стиле Карри, все эти задачи разрешимы. Задача проверки выглядит проще задачи синтеза, но в системах Карри они эквивалентны: проверка $M\,N : \sigma?$ требует синтеза $N : ?$.

Следующий метатеоретический вопрос о системе типов касается завершимости вычислений над типизируемыми термами. Введем несколько связанных с этим понятий. Терм называют *слабо (weak) нормализуемым* (WN), если *существует* последовательность редукций, приводящих его к нормальной форме. Терм называют *сильно (strong) нормализуемым* (SN), если *любая* последовательность редукций, приводит его к нормальной форме.

Например, терм $\mathbf{K\,I\,K}$ сильно нормализуем, терм $\mathbf{K\,I\,\Omega}$ слабо нормализуем, терм $\mathbf{\Omega}$ — не нормализуем.

Понятия слабой и сильной нормализуемости можно распространить с отдельных термов на систему типов. Систему типов называют *слабо нормализуемой* если все её допустимые термы слабо нормализуемы. Систему типов называют *сильно нормализуемой* если все её допустимые термы сильно нормализуемы.

Теорема 11 (о нормализации $\lambda_{\to}$)**.** *Обе системы $\lambda_{\to}$ (и Карри, и Чёрча) сильно нормализуемы.*

То есть любой допустимый терм в $\lambda_{\to}$ всегда редуцируется к нормальной форме независимо от выбранной стратегии редукции.

3 Типы пересечения

Системы с типами-пересечениями расширяют простую систему дополнительным способом конструирования типов (см., напр., [7, 3, 4, 13], изложение ниже идет по [7]). К стрелочному типу добавляется тип $\sigma \cap \tau$. Этот тип интуитивно рассматривается как тип для терма, которому можно приписать и тип σ, и тип τ. Например, комбинатору $\mathbf{I}$

можно приписать типы $\alpha \to \alpha$ и $(\beta \to \gamma) \to \beta \to \gamma$, поэтому в системе с пересечениями можно написать

$$\lambda x.\, x : (\alpha \to \alpha) \cap ((\beta \to \gamma) \to \beta \to \gamma).$$

Следует отличать тип-пересечения от типа пары: для конструирования пары типа $\sigma \times \tau$ нам нужно иметь *два* выражения, одно типа σ, другое типа τ. Для пересечения нам необходимо иметь *одно* выражение, которому можно приписать оба типа.

Вводится также универсальный тип U[15], который интуитивно рассматривается как пустое пересечение. Универсальный тип можно (по определению) приписать любому терму, отсюда его название. Ясно, что подобное приписывание никак не специфицирует терм; термам, обладающих «пристойным» вычислительным поведением, можно будет приписывать более информативные типы. Под словом «пристойный» понимается вычислительное поведение, формирующее некоторую персистентную структуру, не изменяющуюся при дальнейших вычислениях — головную нормальную форму. Сам терм при этом может не иметь нормальной формы. Таков, например, $\mathbf{Y}$-комбинатор $\mathbf{Y} \equiv \lambda f.\, \mathbf{A}\,\mathbf{A}$, где $\mathbf{A} \equiv \lambda x.\, f\,(x\,x)$. Его редукционное поведение приводит к формированию устойчивого аппликативного префикса[16]

$$\mathbf{Y} \equiv \lambda f.\, \mathbf{A}\,\mathbf{A} \longrightarrow_\beta \lambda f.\, f\,(\mathbf{A}\,\mathbf{A}) \longrightarrow_\beta \lambda f.\, f\,(f\,(\mathbf{A}\,\mathbf{A})) \longrightarrow_\beta \lambda f.\, f\,(f\,(f\,(\mathbf{A}\,\mathbf{A}))) \longrightarrow_\beta \ldots$$

Ниже мы покажем, что $\mathbf{Y}$-комбинатору может быть приписан тип $(\mathsf{U} \to \tau) \to \tau$. В отличие от него, ненормализуемому «непристойному» комбинатору $\mathbf{\Omega}$ может быть приписан только универсальный тип U.

3.1 Формализм

Множество типов $\mathbb{T}_\cap$ системы $\lambda_\cap$ определяется индуктивно:

$$\alpha, \beta, \ldots \in \mathbb{T}_\cap \qquad \text{(переменные типа)}$$

$$\mathsf{U} \in \mathbb{T}_\cap \qquad \text{(универсальный тип)}$$

$$\sigma, \tau \in \mathbb{T}_\cap \Rightarrow (\sigma \to \tau) \in \mathbb{T}_\cap \qquad \text{(стрелочные типы)}$$

$$\sigma, \tau \in \mathbb{T}_\cap \Rightarrow (\sigma \cap \tau) \in \mathbb{T}_\cap \qquad \text{(типы-пересечения)}$$

Соглашения: α, β, γ используем для типовых переменных, а σ, τ, ρ — для произвольных типов. Бинарные связки стрелки и пересечения считаются *правоассоциативными*: если $\sigma_1, \ldots, \sigma_n \in \mathbb{T}$, то

$$\sigma_1 \to \sigma_2 \to \ldots \to \sigma_n \;\equiv\; (\sigma_1 \to (\sigma_2 \to \ldots \to (\sigma_{n-1} \to \sigma_n)\ldots)),$$

$$\sigma_1 \cap \sigma_2 \cap \ldots \cap \sigma_n \;\equiv\; (\sigma_1 \cap (\sigma_2 \cap \ldots \cap (\sigma_{n-1} \cap \sigma_n)\ldots))$$

[15] В XX веке универсальный тип обозначали ω.

[16] Более строгая формулировка этой интуиции приводит к так называемой аппроксимационной семантике (см. напр. [13]).

Существует огромное количество версий формализмов присваивания термам типов-пересечений. Мы изложим систему Coppo-Dezani (на нее обычно ссылаются как на $\lambda \cap \mathsf{CD}$). (Mario Coppo and Mariangiola Dezani, Марио Коппо и Марианджола Дезани, первое изложение 1978 год.) Отметим, что эта система в стиле Карри.

$$\Gamma \vdash M : \mathsf{U} \qquad (\mathrm{Ax}\ \mathsf{U})$$

$$\Gamma \vdash x : \sigma, \text{ если } (x : \sigma) \in \Gamma \qquad (\mathrm{Ax})$$

$$\frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M\,N : \tau} \qquad (\to \mathsf{E})$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.\, M : \sigma \to \tau} \qquad (\to \mathrm{I})$$

$$\frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \sigma} \quad \frac{\Gamma \vdash M : \sigma \cap \tau}{\Gamma \vdash M : \tau} \qquad (\cap \mathsf{E})$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma \vdash M : \tau}{\Gamma \vdash M : \sigma \cap \tau} \qquad (\cap \mathrm{I})$$

$$\frac{\Gamma \vdash \lambda x.\, M\,x : \sigma}{\Gamma \vdash M : \sigma}, \text{ если } x \notin \mathrm{FV}(M) \qquad (\eta)$$

Примеры.
Самоприменение типизируемо в $\lambda_\cap$. Вот дерево вывода типа $\alpha \cap (\alpha \to \beta) \to \beta$ для комбинатора $\boldsymbol{\omega} \equiv \lambda x.\, x\,x$:

$$\dfrac{\dfrac{\dfrac{x^{\alpha \cap (\alpha \to \beta)} \vdash x : \alpha \cap (\alpha \to \beta)}{x^{\alpha \cap (\alpha \to \beta)} \vdash x : \alpha \to \beta}\,(\cap \mathsf{E}) \quad \dfrac{x^{\alpha \cap (\alpha \to \beta)} \vdash x : \alpha \cap (\alpha \to \beta)}{x^{\alpha \cap (\alpha \to \beta)} \vdash x : \alpha}\,(\cap \mathsf{E})}{x^{\alpha \cap (\alpha \to \beta)} \vdash x\,x : \beta}\,(\to \mathsf{E})}{\vdash \lambda x.\, x\,x : \alpha \cap (\alpha \to \beta) \to \beta}\,(\to \mathrm{I})$$

Универсальный тип позволяет организовывать странные конструкции:

$$\dfrac{\dfrac{}{x^{\mathsf{U} \to \mathsf{U}} \vdash x : \mathsf{U}}\,(\mathrm{Ax}\ \mathsf{U})}{\vdash \lambda x.\, x : (\mathsf{U} \to \mathsf{U}) \to \mathsf{U}}\,(\to \mathrm{I})$$

Однако прагматика его использования — маркировать потенциально расходящиеся вычисления:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f^{U\to\tau}, x^{U\to\tau} \vdash x\,x : U}\,(\mathrm{Ax}\ U)}{f^{U\to\tau}, x^{U\to\tau} \vdash f\,(x\,x) : \tau}\,(\to \mathrm{E})}{f^{U\to\tau} \vdash \lambda x.\, f\,(x\,x) : (U \to \tau) \to \tau}\,(\to \mathrm{I}) \qquad \dfrac{\dfrac{\dfrac{}{f^{U\to\tau}, x^{U} \vdash x\,x : U}\,(\mathrm{Ax}\ U)}{f^{U\to\tau}, x^{U} \vdash f\,(x\,x) : \tau}\,(\to \mathrm{E})}{f^{U\to\tau} \vdash \lambda x.\, f\,(x\,x) : U \to \tau}\,(\to \mathrm{I})}{f^{U\to\tau} \vdash (\lambda x.\, f\,(x\,x))\,(\lambda x.\, f\,(x\,x)) : \tau}\,(\to \mathrm{E})}{\vdash \mathbf{Y} : (U \to \tau) \to \tau}\,(\to \mathrm{I})$$

3.2 Метатеория

3.2.1 Конверсия субъекта

Для системы $\lambda\cap$ выполняется не только теорема о редукции субъекта, но и теорема об экспансии субъекта, которые могут быть объединены в следующую теорему.

Теорема 12 (о β-конверсии субъекта)**.** *Если* $M =_\beta N$*, то для любых* Γ *и* τ

$$\Gamma \vdash M : \tau \;\Leftrightarrow\; \Gamma \vdash N : \tau.$$

Три примера ниже иллюстрируют, как разрешается проблемы с нарушением сохранения типа при экспансии субъекта, имевшие место в простой системе. Это (1) специализация наиболее общего типа (или полная утрата типа) из-за возникновения вхождения фиктивной переменной; (2) возникновение нетипизируемого подтерма из-за унификации двух ранее несвязанных вхождений подтерма; (3) специализация наиболее общего типа из-за унификации двух ранее несвязанных вхождений подтерма.

(1) В простой системе терм $\lambda f\,x.\,(\lambda y.\,x)(f\,x)$ имеет наиболее общий тип $(\beta \to \gamma) \to \beta \to \beta$. После шага β-редукции терм превращается в $\mathbf{K}_* = \lambda f\,x.\,x$ с наиболее общим типом $\alpha \to \beta \to \beta$. Очевидно, что этот тип нельзя (в $\lambda_\to$) приписать исходному терму: при экспансии возникает новое вхождение переменной f, диктующее для нее стрелочный тип. Покажем, что в $\lambda_\cap$ такое приписывание возможно:

$$\dfrac{\dfrac{\dfrac{\dfrac{f^{\alpha}, x^{\beta}, y^{U} \vdash x : \beta}{f^{\alpha}, x^{\beta} \vdash \lambda y.\,x : U \to \beta}\,(\to \mathrm{I}) \qquad \dfrac{}{f^{\alpha}, x^{\beta} \vdash f\,x : U}\,(\mathrm{Ax}\ U)}{f^{\alpha}, x^{\beta} \vdash (\lambda y.\,x)(f\,x) : \beta}\,(\to \mathrm{E})}{f^{\alpha} \vdash \lambda x.\,(\lambda y.\,x)(f\,x) : \beta \to \beta}\,(\to \mathrm{I})}{\vdash \lambda f\,x.\,(\lambda y.\,x)(f\,x) : \alpha \to \beta \to \beta}\,(\to \mathrm{I})$$

Техника приписывания универсального типа терму, собирающемуся исчезнуть при редукции, очевидно работает, даже если этот терм вообще не типизируем.

(2) Терм $\boldsymbol{\omega}\,\mathbf{I}$ не имеет типа в простой системе, как содержащая нетипизируемый подтерм $\boldsymbol{\omega}$. Однако один шаг β-редукции превращает ее в типизируемый терм $\mathbf{I}\,\mathbf{I}$. То есть при экспансии возникает нетипизируемый подтерм. В $\lambda_\cap$, как показано выше, $\boldsymbol{\omega}$ типизируема стрелочным типом с аргументом-пересечением. Поэтому для типизации аппликации $\boldsymbol{\omega}\,\mathbf{I}$ нам нужно приписать $\mathbf{I}$ тип-пересечение. Введем обозначения $\sigma \equiv \alpha \to \alpha$, $\Gamma \equiv x^{\sigma\cap(\sigma\to\sigma)}$. Тогда

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash x : \sigma \cap (\sigma \to \sigma)}{\Gamma \vdash x : \sigma \to \sigma}\,(\cap E) \quad \dfrac{\Gamma \vdash x : \sigma \cap (\sigma \to \sigma)}{\Gamma \vdash x : \sigma}\,(\cap E)}{\Gamma \vdash x\,x : \sigma}\,(\to E)}{\vdash \lambda x.\, x\,x : \sigma \cap (\sigma \to \sigma) \to \sigma}\,(\to I) \qquad \dfrac{\dfrac{\dfrac{z : \alpha \vdash z : \alpha}{\vdash \lambda z.\, z : \sigma}\,(\to I) \quad \dfrac{z : \sigma \vdash z : \sigma}{\vdash \lambda z.\, z : \sigma \to \sigma}\,(\to I)}{\vdash \lambda z.\, z : \sigma \cap (\sigma \to \sigma)}\,(\cap I)}{\vdash (\lambda x.\, x\,x)(\lambda z.\, z) : \sigma}\,(\to E)$$

Итак мы видим, что в $\lambda_\cap$ верно $\vdash \boldsymbol{\omega}\,\mathbf{I} : \alpha \to \alpha$.

(3) **Упражнение.**
Покажите, что в $\lambda_\to$ терм $\mathbf{K}\,\mathbf{I}\,(\mathbf{S}\,\mathbf{I})$ имеет наиболее общий тип $\alpha \to \alpha$.
Покажите, что в $\lambda_\to$ терм $(\lambda x.\, \mathbf{K}\,x\,(\mathbf{S}\,x))\,\mathbf{I}$ имеет наиболее общий тип $(\beta \to \gamma) \to \beta \to \gamma$.
Покажите, что в $\lambda_\cap$ терму $(\lambda x.\, \mathbf{K}\,x\,(\mathbf{S}\,x))\,\mathbf{I}$ можно приписать тип $\alpha \to \alpha$. Это означает сохранение этого типа при такой экспансии $\mathbf{K}\,\mathbf{I}\,(\mathbf{S}\,\mathbf{I})$.

3.2.2 Нормализация

Если вычисления в терме потенциально завершаются, то есть терм слабо нормализуем, то ему можно приписать тип-пересечение, не содержащий U. Например, при нормальной стратегии $\mathbf{Y}\,\mathbf{K}_* \longrightarrow_\beta \mathbf{I}$, при этом $\mathbf{K}_* : \mathsf{U} \to \sigma \to \sigma$, что дает $\mathbf{Y}\,\mathbf{K}_* : \sigma \to \sigma$ и согласуется с типом $\mathbf{I}$.

Верно и обратное: если терму в $\lambda_\cap$ можно приписать тип-пересечение, не содержащий U, то он имеет нормальную форму.

Теорема 13 (о типах слабо нормализуемых термов)**.** *Терм* M *имеет* β*-нормальную форму тогда и только тогда, когда имеются контекст* Γ *и тип* τ*, не содержащие* U*, такие что* $\Gamma \vdash M : \tau$.

Алгоритмически вывод типа-пересечения без U для терма в системе $\lambda_\cap$ можно выполнить за два шага: сначала нормализуем, потом типизируем. Второй шаг оказывается довольно простым, а первый, естественно, неразрешим в общем случае, поскольку давал бы решение проблемы останова. Тот факт, что типизация нормальной формы порождает правильную типизацию исходного терма следует из теоремы об экспансии субъекта.

Имеется более сильный аналог предыдущей теоремы, доказанный van Bakel [12]. Терм сильно нормализуем тогда и только тогда, когда можно построить дерево вывода его типа не содержащее U[17]. Отметим, что если мы ограничиваемся системой без универсального типа, экспансия субъекта уже не имеет места.

[17]Отметим, что любой терм, содержащий $\mathbf{Y}$ в качестве подтерма, если и нормализуем, то только слабо.

3.2.3 Термы имеющие решение

Замкнутый терм M называют ***имеющим решение*** (solvable[18]), если существуют термы $Q_1, \ldots, Q_m$, такие что $M\,Q_1\,\ldots\,Q_m$ слабо нормализуем. Например, хотя $\mathbf{Y}$-комбинатор ненормализуем, он имеет решение: $\mathbf{Y}\,\mathbf{K}_* \longrightarrow_\beta \mathbf{I}$. С другой стороны очевидно, что $\mathbf{\Omega}$ не имеет решения. Терм со свободными переменными называют ***имеющим решение***, если разрешимо его замыкание.

Теорема 14 (о характеризации имеющих решение термов)**.** *Терм M имеет решение тогда и только тогда, когда имеются контекст Γ и тип τ, отличный от U, такие что $\Gamma \vdash M : \tau$.*

Из этой теоремы следует, что, в отличие от $\mathbf{Y}$, комбинатор $\mathbf{\Omega}$ типизируем единственным образом $\mathbf{\Omega} : \mathsf{U}$.

3.3 Отношение вложения на типах

В 1983 году Хенк Барендрегт, Марио Коппо и Марианджола Дезани переформулировали систему с пересечениями на языке отношения вложения типов как множеств. На эту систему обычно ссылаются как на $\lambda_\cap$BCD.

Отношение $\preccurlyeq$ на типах вводится индуктивно с помощью следующих схем аксиом и правил:

$$
\begin{array}{ll}
(A1) & \sigma \preccurlyeq \sigma \\
(A2) & \sigma \preccurlyeq \mathsf{U} \\
(A3) & \mathsf{U} \preccurlyeq \mathsf{U} \to \mathsf{U} \\
(A4) & \sigma \preccurlyeq \sigma \cap \sigma \\
(A5) & \sigma \cap \tau \preccurlyeq \sigma \\
(A6) & \sigma \cap \tau \preccurlyeq \tau \\
(A7) & (\sigma \to \tau_1) \cap (\sigma \to \tau_2) \preccurlyeq \sigma \to \tau_1 \cap \tau_2
\end{array}
$$

$$
\begin{array}{ll}
(R1) & \sigma \preccurlyeq \sigma', \tau \preccurlyeq \tau' \implies \sigma \cap \tau \preccurlyeq \sigma' \cap \tau' \\
(R2) & \sigma \preccurlyeq \sigma', \tau \preccurlyeq \tau' \implies \sigma' \to \tau \preccurlyeq \sigma \to \tau' \\
(R3) & \tau_1 \preccurlyeq \tau_2, \tau_2 \preccurlyeq \tau_3 \implies \tau_1 \preccurlyeq \tau_3
\end{array}
$$

Отношение $\sim$ на типах определяется так

$$\sigma \sim \tau \iff \sigma \preccurlyeq \tau \text{ и } \tau \preccurlyeq \sigma$$

Задача проверки выполнения $\sigma \preccurlyeq \tau$ алгоритмически разрешима.

Теорема 15 (об эквивалентности)**.** *Правило типизации*

$$\frac{\Gamma \vdash \lambda x.\, M\,x : \sigma}{\Gamma \vdash M : \sigma}, \text{ если } x \notin \mathrm{FV}(M) \qquad (\eta)$$

[18]Мы отказались от перевода «разрешимый», чтобы избежать путаницы с разрешимостью (decidability) задачи вывода типа, о которой шла речь чуть выше.

может быть заменено на

$$\frac{\Gamma \vdash M : \sigma \quad \sigma \preccurlyeq \tau}{\Gamma \vdash M : \tau} \qquad (\preccurlyeq)$$

и наоборот; при этом выводимость утверждения $\Gamma \vdash M : \tau$ *сохранится.*

Рассмотрим несколько интересных подслучаев доказательства.
Покажем, что имея (η)-правило, можно получить (A3). Для этого нужно показать, что любому терму M можно приписать тип $U \to U$.

$$\dfrac{\dfrac{\dfrac{}{x^U \vdash M\,x : U}\,(\mathrm{Ax}\ U)}{\vdash \lambda x.\, M\,x : U \to U}\,(\to \mathrm{I})}{\vdash M : U \to U}\,(\eta)$$

Покажем, что имея (η)-правило, можно получить (A7). Для этого нужно показать, что терму $M : (\sigma \to \tau_1) \cap (\sigma \to \tau_2)$ можно приписать тип $\sigma \to \tau_1 \cap \tau_2$. Вводя $\rho \equiv (\sigma \to \tau_1) \cap (\sigma \to \tau_2)$ и используя свежую для вывода $\Gamma \vdash M : \rho$ переменную y, имеем

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash M : \rho}{\Gamma, y^\sigma \vdash M : \rho}\,(\mathrm{Thin})}{\Gamma, y^\sigma \vdash M : \sigma \to \tau_1}\,(\cap \mathrm{E}) \qquad \Gamma, y^\sigma \vdash y : \sigma}{\Gamma, y^\sigma \vdash M\,y : \tau_1}\,(\to \mathrm{E}) \qquad \dfrac{\dfrac{\dfrac{\Gamma \vdash M : \rho}{\Gamma, y^\sigma \vdash M : \rho}\,(\mathrm{Thin})}{\Gamma, y^\sigma \vdash M : \sigma \to \tau_2}\,(\cap \mathrm{E}) \qquad \Gamma, y^\sigma \vdash y : \sigma}{\Gamma, y^\sigma \vdash M\,y : \tau_1}\,(\to \mathrm{E})}{\Gamma, y^\sigma \vdash M\,y : \tau_1 \cap \tau_2}\,(\cap \mathrm{I})}{\Gamma \vdash \lambda y.\, M\,y : \sigma \to \tau_1 \cap \tau_2}\,(\to \mathrm{I})}{\Gamma \vdash M : \sigma \to \tau_1 \cap \tau_2}\,(\eta)$$

Покажем, что имея (η)-правило, можно обосновать правило (R2). Для этого нужно показать, что терму $M : \sigma' \to \tau$ можно приписать тип $\sigma \to \tau'$. При этом предполагается, что $\sigma \preccurlyeq \sigma'$ и $\tau \preccurlyeq \tau'$. Доказательство проводится индукцией по структуре типа. Используя свежую для вывода $\Gamma \vdash M : \sigma' \to \tau$ переменную y, имеем

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash M : \sigma' \to \tau}{\Gamma, y^\sigma \vdash M : \sigma' \to \tau}\,(\mathrm{Thin}) \qquad \dfrac{\Gamma, y^\sigma \vdash y : \sigma}{\Gamma, y^\sigma \vdash y : \sigma'}\,(\mathrm{IH})}{\Gamma, y^\sigma \vdash M\,y : \tau}\,(\to \mathrm{E})}{\Gamma, y^\sigma \vdash M\,y : \tau'}\,(\mathrm{IH})}{\Gamma \vdash \lambda y.\, M\,y : \sigma \to \tau'}\,(\to \mathrm{I})}{\Gamma \vdash M : \sigma \to \tau'}\,(\eta)$$

Список литературы

[1] Харрисон П. Филд А. *Функциональное программирование.* М.: Мир, 1993.

[2] Барендрегт Х. *Лямбда-исчисление, его синтаксис и семантика.* М.: Мир, 1985.

[3] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, Dov M. Gabbay, and S. E. Maibaum, editors, *Handbook of Logic in Computer Science (Vol. 2)*, pages 117–309. Oxford University Press, Inc., New York, NY, USA, 1992.

[4] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types.* Cambridge University Press, New York, NY, USA, 2013.

[5] H.P. Barendregt. *The Lambda Calculus: ist syntax and semantics.* Studies in logic and the Foundations of Mathematics. North-Holland, 1985.

[6] N.G. Bruijn, de. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[7] J. Roger Hindley. Types with intersection: An introduction. *Formal Aspects of Computing*, 4(5):470–486, Sep 1992.

[8] J. Roger Hindley. *Basic Simple Type Theory.* Cambridge University Press, New York, NY, USA, 1997.

[9] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction.* Cambridge University Press, New York, NY, USA, 2 edition, 2008.

[10] Benjamin C. Pierce. *Types and Programming Languages.* MIT Press, 2002.

[11] Benjamin C. Pierce, editor. *Advanced Topics in Types and Programming Languages.* MIT Press, 2005.

[12] Steffen van Bakel. Complete restrictions of the intersection type discipline. *Theoretical Computer Science*, 102(1):135 – 163, 1992.

[13] Steffen van Bakel. Strict intersection types for the lambda calculus. *ACM Comput. Surv.*, 43(3), April 2011.