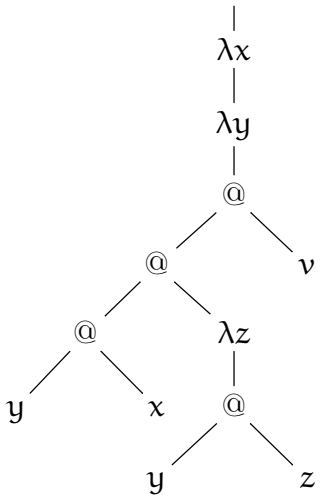
1.8 Стратегии редукции

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.

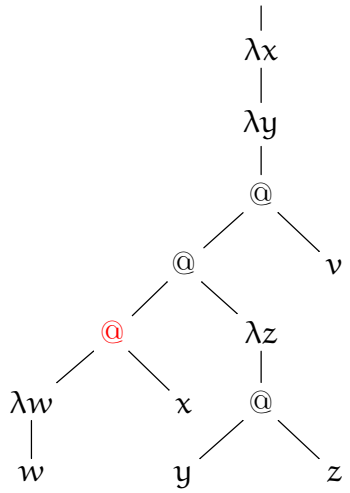
Теорема 7. *Лямбда-терм может иметь одну из двух форм:*

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

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:

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

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

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

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, $\mathbf{K}\,\boldsymbol{\Omega}\,\mathbf{I}$ 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.