

Math-Net.Ru

Общероссийский математический портал

Д. А. Березун, Полная головная линейная редукция, *Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление*, 2017, том 10, выпуск 3, 59–82

DOI: <https://doi.org/10.18721/JCSTCS.10306>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением


<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 195.144.231.194

18 октября 2021 г., 18:23:35





Программное обеспечение вычислительных, телекоммуникационных и управляющих систем

DOI: 10.18721/JCSTCS.10306

УДК 519.682.1

ПОЛНАЯ ГОЛОВНАЯ ЛИНЕЙНАЯ РЕДУКЦИЯ

Д.А. Березун

Санкт-Петербургский государственный университет,
Санкт-Петербург, Российская Федерация

Головная линейная редукция (head linear reduction) представляет собой стратегию редукции лямбда-термов, производящую минимальное количество подстановок для достижения псевдоголовной нормальной формы (quasi-head normal form). Статья посвящена обобщению понятия головной линейной редукции до полной головной линейной редукции (complete head linear reduction), позволяющей полностью нормализовать лямбда-терм и определить новый подход к вычислениям — трассирующую нормализацию (traversal-based normalization). Оба подхода формализованы в виде систем переходов (transition system). В статье также показана корректность обеих стратегий редукций: головной линейной редукции относительно головной редукции — головная линейная редукция завершается в псевдоголовной нормальной форме терма тогда и только тогда, когда завершается головная, и полной головной линейной редукции относительно эффективной редуцирующей стратегии — головная линейная редукция завершается в нормальной форме терма тогда и только тогда, когда последняя существует.

Ключевые слова: лямбда-исчисление; редукция; линейная редукция; головная линейная редукция; полная головная линейная редукция; трассирующая нормализация.

Ссылка при цитировании: Березун Д.А. Полная головная линейная редукция // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2017. Т. 10. № 3. С. 59–82. DOI: 10.18721/JCSTCS.10306

COMPLETE HEAD LINEAR REDUCTION

D.A. Berezun

St. Petersburg State University, St. Petersburg, Russian Federation

In lambda calculus, head linear reduction is a reduction strategy which reaches a quasi-head normal form of terms in the minimum number of linear substitution steps. The paper is dedicated to the generalization of head linear reduction to a complete head linear reduction which yields normal forms when they exist. The formal presentation of both head linear reduction and complete head linear reduction via transition systems is provided. We also proved that both reduction strategies are correct: head linear reduction with respect to head reduction, i.e., that head linear reduction terminates in quasi-head normal form if and only if head reduction terminates, and we proved that

complete head linear reduction is an effective reduction strategy, i.e., it terminates if and only if the normal form exists.

Keywords: lambda-calculus; reduction strategy; linear reduction; head linear reduction; complete head linear reduction; traversal-based normalization.

Citation: Berezun D.A. Complete Head Linear Reduction. St. Petersburg State Polytechnical University Journal. Computer Science. Telecommunications and Control Systems. 2017, Vol. 10, No. 3, Pp. 59–82. DOI: 10.18721/JCSTCS.10306

1 Введение

Головная линейная редукция играет особую роль в различных подходах к вычислениям, таких как игровая семантика (game semantics) [22–25], оптимальные редукции (optimal reductions) [16], геометрия взаимодействия (geometry of interaction) [21], сети доказательств (proof nets) [20] и др. Основанная на расширенном понятии редекса, именуемого *простым редексом* (prime redex), позволяющим явно линейризовать последовательность подстановок, головная линейная редукция не является редукцией лямбда-термов в её классическом понимании. Неформально говоря, головная линейная редукция вместо устранения редекса как такового и произведения подстановки в терм, запоминает редекс и производит линейные подстановки «по надобности», когда это необходимо, для достижения *псевдоголовной нормальной формы* (quasi-head normal form), т. е. такой формы терма, в которой головной переменной не соответствует ни один редекс. Головная нормальная форма получается из псевдоголовной нормальной формы путём подстановки всех простых редексов (см. раздел 3) во всех аргументах терма.

В статье приводится обобщение понятия головной линейной редукции до *полной головной линейной редукции* (complete head linear reduction), *приводящей терм в нормальную форму*. Полная головная линейная редукция позволяет определить новый подход к вычислениям — *трассирующую нормализацию* (traversal-based normalization) [6] — не использующую классические приемы, такие как контексты, замыкания и др.

В статье также показана корректность обеих стратегий редукций: приведено формальное доказательство того, что головная и полная головная линейные редукции завершаются в псевдоголовной нормальной и нормальной формах входного терма, со-

ответственно, тогда и только тогда, когда последние существуют. Иными словами, в общепринятой терминологии [5] полная головная линейная редукция является *эффективной редуцирующей стратегией* (effective reduction strategy).

2 Лямбда-исчисление

Лямбда-исчисление (lambda-calculus) впервые было предложено в 30-х гг. американским математиком Алонсо Чёрчем (Alonso Church) с целью формализации и анализа понятия вычислимости. Формальная система, предложенная Чёрчем, основана всего на трёх примитивах: переменной, абстракции, т. е. анонимной функции, и применении функции к аргументу. *Лямбда-выражением* (лямбда-термом, λ -термом или просто термом, Λ -term) называется выражение, удовлетворяющее грамматике $\Lambda^1 ::= V \mid \Lambda @^2 \Lambda \mid \lambda V$. Λ , где V — множество конечных строк, называемых *переменными*, над некоторым фиксированным алфавитом $\Sigma \setminus \{ , . , @ , \lambda \}$, выражение вида $\lambda x.e$ называется *абстракцией* по переменной x (λ -abstraction), а выражение вида $e_1 @ e_2$ — *применением* (application) терма e_1 к терму e_2 . Представленное в таком виде исчисление также называется *чистым* или *нетипизированным*, или *бестиповым* лямбда-исчислением.

Вхождение переменной в терм бывает двух видов: свободное и связанное.

¹ Мы также будем использовать скобки (“(”, “)”) как незначащие символы грамматики в тех ситуациях, когда из конкретного синтаксиса невозможно однозначное восстановление абстрактного.

² Символ @ иногда опускается: $\Lambda @ \Lambda \equiv \Lambda \Lambda \equiv \Lambda \Lambda$.

³ Мы будем считать, что приоритет оператора “@” выше чем у “.”, то есть $\lambda x.x @ y \equiv \lambda x.(x @ y)$, а ассоциативность у оператора “@” — левая, то есть $x @ y @ z = (x @ y) @ z$.

Связанными называются все вхождения переменных, по которым выше в дереве разбора были произведены абстракции, остальные вхождения переменных называются *свободными*. Так, например, в терме $\lambda y.x @ y @ (\lambda x.x)$ первое вхождение переменной x является свободным, второе — связанным, как и единственное вхождение переменной y .

Основной формой эквивалентности лямбда-термов является так называемая α -эквивалентность, утверждающая равенство термов, получающихся друг из друга переименованием связанных переменных. Так, например, термы $\lambda x.x$ и $\lambda y.y$ α -эквивалентны, поскольку первый получается из второго переименованием связанной переменной y в x .

Нередко термы в лямбда-исчислении рассматриваются с точностью до α -эквивалентности. Поэтому вместо терминов связанное и свободное вхождение переменной часто используют термины *связанная* и *свободная* переменная, соответ-

ственно, подразумевая, что одна и та же переменная не входит в терм свободно и связано одновременно. Более того, мы будем придерживаться *соглашения Барендрегта (Barendregt's convention), подразумевающего*, что все переменные имеют различные имена. Формально определения множеств свободных (FV) и связанных (BV) переменных приведены на рис. 1.

Основной аксиомой лямбда-исчисления является β -редукция. Интуитивно, β -редукция производит применение функции, т. е. лямбда-абстракции, к её аргументу путём замены всех вхождений переменной, связанной этой абстракцией, на тело аргумента:

$$(\lambda x.e_1) e_2 =_{\beta} e_1[x / e_2].$$

Такая замена называется *подстановкой* аргумента e_1 в терм e_2 , а само выражение вида $(\lambda x.e_1) e_2$ — *редексом*. Снабжённое аксиомой β -редукции λ -исчисление обладает свойством *полноты по Тьюрингу*, определяя тем самым одну из простейших моделей

Синтаксис
$\Lambda ::= V \mid \Lambda @ \Lambda \mid \lambda V.\Lambda$
$V ::= \{x, y, \dots\}$
Подстановка
$x[x/r] = r$
$y[x/r] = y, \text{ если } x \neq y$
$(e_1 e_2)[x/r] = (e_1[x/r])(e_2[x/r])$
$(\lambda x.e)[x/r] = \lambda x.e$
$(\lambda y.e)[x/r] = \lambda y.e[x/r], \text{ если } x \neq y \text{ и } y \notin FV(e)$
β -редукция
$(\lambda x.e_1) e_2 =_{\beta} e_1[x/e_2]$
Множество свободных переменных
$FV(x) = \{x\}$
$FV(\lambda x.e) = FV(e) \setminus \{x\}$
$FV(e_1 e_2) = FV(e_1) \cup FV(e_2)$
η -конверсия
$f \sim_{\eta} \lambda x.f x, \text{ если } x \notin FV(f)$

Синтаксис в нотации де Брауна

$$\Lambda^d ::= \mathbb{N} \mid \Lambda^d \Lambda^d \mid \lambda \Lambda^d$$

Сдвиг в Λ^d

$$\uparrow_c^d(k) = \begin{cases} k, & k < c \\ k + d, & k \geq c \end{cases} \quad \uparrow_c^d(\lambda e) = \lambda \uparrow_{c+1}^d(e)$$

Подстановка в Λ^d

$$k[j/r] = \begin{cases} r, & k = j \\ k, & \text{иначе} \end{cases} \quad (\lambda e)[j/r] = \lambda e[j + 1 / \uparrow_0^1 r]$$

β -редукция в Λ^d

$$(\lambda e_1) e_2 \rightarrow \uparrow_0^{-1}(e_1[0 / \uparrow_0^1(e_2)])$$

Множество связанных переменных

$$BV(x) = \emptyset$$

$$BV(\lambda x.e) = BV(e) \cup \{x\}$$

$$BV(e_1 e_2) = BV(e_1) \cup BV(e_2)$$

α -эквивалентность

$$\lambda x.e \sim_{\alpha} \lambda y.e[x/y], \text{ если } y \notin FV(e)$$

Рис. 1. Чистое лямбда-исчисление

вычислений. Термы, получаемые друг из друга по правилу β -редукции, называются β -эквивалентными. Запись $s \rightarrow_{\beta} t$ означает, что терм t получается из терма s за один шаг β -редукции, а запись $s \rightarrow_{\beta}^* t$ означает, что терм t получается из терма s за ноль или более шагов β -редукции.

Заметим, что подстановка не является тривиальной операцией. Так, например, рассмотрим терм $(\lambda x.x y) x$. Согласно правилу α -эквивалентности, он эквивалентен терму $(\lambda z.z y) x$, который, в свою очередь, β -эквивалентен терму $\lambda z.z x$. Если произвести подстановку $[x / y]$ непосредственно в терме $\lambda x.x y$, то результатом будет терм $\lambda x.x x$, который уже не эквивалентен терму $\lambda z.z x$. Таким образом, для подстановки важно, чтобы переменные, встречающиеся свободно в подставляемом терме, не связывались при подстановке. Такая подстановка называется *свободной от связывания (capture-avoiding substitution)*, далее — просто подстановка. Формальные правила подстановки приведены на рис. 1.

В начале 70-х гг. голландским математиком Николасом де Брауном⁴ (Nicolaas Govert de Bruijn) была предложена нотация, получившая название *представление де Брауна* [2], позволяющая полностью избавиться от имён в лямбда-термах. В представлении де Брауна терм полностью лишается имён переменных: вхождения переменных заменяются на натуральные числа, называемые *индексами де Брауна (de Bruijn indexes)*, а абстракции становятся анонимными. По сути, термы в представлении де Брауна представляют собой классы эквивалентности термов, факторизованных по правилу α -эквивалентности. *Индекс де Брауна* суть натуральное число, представляющее собой вхождение переменной, равное количеству абстракций между этим вхождением и абстракцией, связывающей саму переменную, включительно. Так, например, терм $\lambda x.\lambda y.\lambda z.x z (y z)$, также известный как S

комбинатор, в нотации де Брауна имеет следующий вид: $\lambda \lambda \lambda 3 1 (2 1)$. Заметим, что в представлении де Брауна при осуществлении подстановки может потребоваться пересчёт индексов (см. рис. 1).

Ещё одно преобразование иногда считается стандартным для лямбда-исчисления — η -конверсия (эта-конверсия, η -coersion), интуитивно, утверждающее, что функция и абстракция этой функции, применённой к переменной, по которой осуществлена абстракция, суть одно и то же (см. рис. 1). Преобразование функции f к виду $\lambda x.fx$ получило название η -расширения (η -expansion), а преобразование, ему обратное, — η -редукции (η -reduction).

2.1 Стратегии вычислений

Говорят, что терм находится в *нормальной форме*, если он не содержит редексов, т.е. к нему неприменима аксиома β -редукции, а два терма называются *равными*, с точностью до α -конверсии, если они имеют одну и ту же нормальную форму. Вычисление в лямбда-исчислении есть вычисление нормальной формы терма — нормализация. Конечно, ввиду Тьюринг-полноты λ -исчисления не все термы имеют нормальную форму, например, вычисление терма, также известного как терм омега (ω , omega), $\omega = \sigma\sigma = (\lambda x.x x) (\lambda x.x x)$ расходится. Важным свойством в контексте нормализации является свойство Чёрча–Россера (Church–Rosser), также известное как свойство ромба, утверждающее, что если два терма s и t равны, то существует такой терм m , что $s \rightarrow_{\beta}^* m$ и $t \rightarrow_{\beta}^* m$. Прямым следствием свойства ромба является единственность нормальной формы терма, если таковая существует. Тем не менее, в аксиоме β -редукции ничего не сказано про порядок, в котором должна происходить β -редукция терма. Далее в этой части мы рассмотрим существующие *порядки* (стратегии, reduction strategies) *редукций* и их свойства.

Разнообразие стратегий вычислений в лямбда-исчислении обусловлено, в том числе, тем, что разные языки программирования изобретались для различных целей и, соответственно, имеют разные свойства и реализации. Несмотря на то, что не все

⁴ В русскоязычной литературе также встречаются следующие варианты перевода фамилии «de Bruijn»: «де Брюйн», «де Брейн», «де Бройн».

Редукция аргументов	Редукция под абстракцией	
	Да	Нет
Да	(Сильная) нормальная форма $S ::= \lambda x.N \mid x N_1 \dots N_n$	Слабая нормальная форма $W ::= \lambda x.e \mid x W_1 \dots W_n$
Нет	Головная нормальная форма $H ::= \lambda x.H \mid x e_1 \dots e_n$	Слабая головная нормальная форма $E ::= \lambda x.e \mid x e_1 \dots e_n$

где $\forall i \in \mathbb{N}$, e_i — произвольный лямбда-терм

Рис. 2. Нормальные формы⁶

порядки редукций приводят терм к нормальной форме, они нередко завершаются в некоторой «нормальной»⁵ для этого порядка форме, которая с точки зрения конкретного порядка редукций считается *значением*, т. е. термом, который не редуцируется дальше, даже если содержит редексы. Основными свойствами стратегий редукций являются следующие два: произведение редукций под абстракцией и редукция аргументов. В таблице на рис. 2 приведены «нормальные» формы, получаемые при всех комбинациях этих свойств, описанные в терминах контекстно-свободных грамматик⁶.

2.1.1 Слабые порядки редукций

Слабые порядки редукций характеризуются тем, что они считают абстракцию значением и, соответственно, не производят редукцию под абстракцией.

Вызов по имени (call-by-name). Интуитивно, стратегия вычисления по имени сначала вычисляет функцию до значения, лямбда-абстракции, после чего редуцирует её применение к аргументу путём постановки тела аргумента вместо каждого вхождения пере-

менной, по которой эта абстракция произведена, что может привести к повторным вычислениям аргументов, если вхождений переменной, по которой производилась абстракция функции, несколько. Формально, на каждом шаге раскрывается самый левый, самый внешний редекс, который не находится под лямбда-абстракцией, и, таким образом, вызов по имени завершается в слабой головной нормальной форме.

Вызов по значению (call-by-value) в отличие от вызова по имени сначала производит редукцию аргументов и лишь потом применения функции к ним. На каждом шаге раскрывается самый левый, самый внутренний редекс, который не находится под лямбда-абстракцией, и, таким образом, вызов по значению завершается в слабой нормальной форме.

Вызов по необходимости (call-by-need) является вызовом по имени, в который добавлена *мемоизация*, т. е. при вычислении аргумента его значение сохраняется и не перевычисляется при последующем повторном использовании аргумента. В *чистых функциональных языках*, где нет побочных эффектов, результат вызова по имени и вызова по необходимости будет одинаковым, а значит, вызов по необходимости завершается в слабой головной нормальной форме. Одним из наиболее популярных языков, использующих вызов по необходимости, является язык функционального программирования Haskell.

2.1.2 Сильные порядки редукций

Нормальный порядок (applicative order) отличается от стратегии вызова по имени тем, что он производит *сильную* редукцию, т. е. редукцию под абстракцией, а также

⁵ Данная форма не является нормальной формой в том смысле, что терм, находящийся в ней, может содержать редексы. Для того, чтобы такая терминология не вводила в заблуждение, нормальную форму часто называют *сильной* нормальной формой, а остальные «нормальные» формы имеют некоторый дополнительный префикс, описывающий «какая именно это нормальная форма», например, головная нормальная форма (см. главу 2.2.3 и рис. 2).

⁶ Таблица заимствована из статьи Питера Сестофта (Peter Sestoft) [7].

редукцию аргументов. Первым на каждом шаге раскрывается самый левый, самый внешний редекс. В отличие от других порядков редукций, нормальный порядок редукций является *нормализующим* (normalizing), т. е. завершается в сильной нормальной форме терма тогда и только тогда, когда последняя существует. С другой стороны, как и вызов по имени, нормальный порядок редукций может проделывать одну и ту же работу несколько раз. Например, $(\lambda x.xx)((\lambda y.y)a) \rightarrow ((\lambda y.y)a)((\lambda y.y)a) \rightarrow a((\lambda y.y)a) \rightarrow aa$ дважды редуцирует редекс $(\lambda y.y)a$.

Апplikативный порядок (applicative order) В то время как нормальный порядок является сильной версией вызова по имени, аппликативный порядок является сильной версией вызова по значению. Иными словами, на каждом шаге первым раскрывается самый левый, самый глубокий редекс. Например, $(\lambda x.xx)((\lambda y.y)a) \rightarrow (\lambda x.xx)a \rightarrow aa$. Если аппликативный порядок завершается, то приводит терм в сильную нормальную форму. Тем не менее, в отличие от нормального порядка редукций, аппликативный порядок может не завершаться, даже если терм имеет нормальную форму. Нередко это происходит тогда, когда какой-нибудь аргумент функции не имеет нормальной формы и при этом не используется в её теле. Например, терм $(\lambda x.\lambda y.x)a((\lambda x.xx)(\lambda x.xx))$ имеет нормальную форму a , в то время как его аргумент $((\lambda x.xx)(\lambda x.xx))$, уже известный нам терм ω нормальной формы не имеет, и его вычисление расходится. Более того, аппликативный порядок редукций не способен редуцировать терм, являющийся применением рекурсивной функции к аргументу, даже если комбинатор рекурсии, используемый для определения рекурсивной функции, является специальным комбинатором для вызова по значению.

Головная редукция (head reduction, leftmost head reduction) производит редукцию только головных редексов. Редекс называется *головным*, если его предками, в смысле абстрактного синтаксического дерева терма, являются лишь абстракции. Например, редекс $(\lambda y.e_1)e_2$ является головным в терме $\lambda x_1 \dots \lambda x_n.(\lambda y.e_1)e_2 \dots e_m$. Завершается голов-

ная редукция в головной нормальной форме. Заметим, что повторное применение головной редукции к аргументам полученной головной нормальной формы нормализует терм. Более того, такая стратегия редукций является нормализующей.

Существуют и другие порядки редукций, нередко являющиеся лишь вариациями и разнообразными комбинациями описанных выше порядков редукций, такие как *гибридный* (hybrid) аппликативный порядок, гибридный нормальный порядок или *спиновая головная редукция* (spine head reduction) и др. Мы опустим их в данной статье, обратив читателя к [5, 7].

2.2 Головная линейная редукция и трассирующая нормализация

Как уже отмечалось ранее, лямбда-исчисление в первую очередь представляет собой модель вычислений, и, соответственно, одним из основных вопросов, стоящих перед ней, является вопрос о *сложности* (complexity) вычислений. Иными словами, положим, дан некоторый терм, и зафиксирована стратегия вычислений. Какова сложность вычислений – нормализации – данного терма. Простейшая мера – количество шагов β -редукции, необходимых для нормализации терма. К сожалению, такой подход не соответствует сложности вычислений на реальных вычислителях, поскольку сложность операции подстановки находится в непосредственной нелинейной зависимости от терма.

Работы [13–18] представляют различные подходы к описанию сложности вычислений в лямбда-исчислении и определению оптимальных стратегий редукций, основанных на той или иной мере сложности вычислений. Одним из таких подходов является *головная линейная редукция* (head linear reduction) [3, 4], на каждом шаге вместо обычных подстановок производящая так называемые *линейные подстановки* (linear substitution) – подстановки только одного вхождения переменной в терм. В этой главе мы рассмотрим классические определения головной линейной редукции, трассирующей нормализации и связь между ними.

Терм T	Условие	Список головных абстракций $\lambda_h(T)$	Список простых редексов $pr(T)$
x		$[\]$	$[\]$
UV	$\lambda_h(U) = [\]$	$[\]$	$pr(U)$
UV	$\lambda_h(U) = \lambda x : l$	l	$(\lambda x, V) : pr(U)$
$\lambda x.U$		$\lambda x : \lambda_h(U)$	$pr(U)$

Рис. 3. Определение списков головных абстракций и простых редексов

2.2.1 Головная линейная редукция: классическое определение

Для формального определения понятия головной линейной редукции нам понадобится ввести некоторое количество дополнительных определений. *Хребетными* или *спинальными подтермами* (spine sub-terms) терма T называются сам терм T , а также все спинальные подтермы терма U , если $T = UV$ или $T = \lambda x.U$ соответственно. Заметим, что любой лямбда-терм имеет ровно один спинальный подтерм-переменную. Такое вхождение переменной, так же как и сам подтерм, называется *головным* (head occurrence, hoc). Ещё два понятия: *список головных абстракций* (head λ -list, $\lambda_h(T)$) и *простые редексы* (prime redexes) — определяются по индукции по структуре терма T . Простой редекс представляет собой пару $(\lambda x, N)$, первый и второй элементы которой называются *абстракцией* и *аргументом простого редекса* соответственно. Индуктивное определение списков простых редексов и головных абстракций приведены

на рис. 3, где $pr(T)$ обозначает список простых редексов терма T .

Редексом головного вхождения переменной (hoc redex) терма T называется простой редекс $(\lambda x, V)$, где x является головной переменной, если таковой редекс существует.

Интуитивно, головная линейная редукция линеаризует последовательность подстановок путём замены на каждом шаге лишь одного вхождения переменной (головного), оставляя сам головной редекс нетронутым. Если же он не определён, головная линейная редукция завершается в так называемой *псевдоголовной нормальной форме* (quasi-head-normal form, qhn).

Пусть $r = (\lambda x, \dots)$ и $s = (\lambda y, \dots)$ — два простых редекса терма T . Говорят, что простой редекс r *содержит* простой редекс s , если λy является вершиной поддерева, начинающегося в вершине λx . Более того, простые редексы r и s называются *последовательными*, если r содержит s , и не существует такого простого редекса t , который содержится в r , но не содержится в s .

Пример. Пусть $T = \lambda s.(\lambda x.(\lambda y.(\lambda w.w @ s) @ y) @ x) @ (\lambda z.z)$, тогда

$$\begin{cases} \lambda_h(T) = [\lambda s] \\ pr(T) = [(\lambda x, (\lambda z.z)), (\lambda y, x), (\lambda z, y)] \iff [r, s, t] \end{cases}$$

Пары (r, s) и (s, t) образуют последовательные простые редексы.

Теорема 1. Пусть N — произвольный терм, тогда если терм M является результатом применения головной линейной редукции к терму N , то:

- $M \equiv_{\beta} N$;
- Головная линейная редукция завершается тогда и только тогда, когда завершается головная редукция.

Первое утверждение может быть показано индукцией по числу шагов головной линейной редукции. Что же касается вто-

рого утверждения, то его необходимость устанавливается в силу того, что головная редукция терма N завершится за число шагов, равное количеству простых редексов терма M . Действительно, никакая β -редукция терма M не способна произвести подстановку головного вхождения переменной, а значит, и не способна создать новый простой редекс. Таким образом, терм имеет головную нормальную норму, что в свою очередь гарантирует завершаемость [5] го-

Синтаксис по Карри	Типизация по Карри	Типизация по Чёрчу
$\Lambda^{Cu} ::= \Lambda$		
Синтаксис по Чёрчу	$\frac{x : \tau \in \Gamma}{\Gamma \vdash_{Cu} x : \tau}$	$\frac{x : \tau \in \Gamma}{\Gamma \vdash_{Ch} x : \tau}$
$\Lambda^{Ch} ::= V \mid \Lambda^{Ch} \Lambda^{Ch}$		
$\quad \mid \lambda V^{\tau} . \Lambda^{Ch}$	$\frac{\Gamma, x : \tau \vdash_{Cu} e : \sigma}{\Gamma \vdash_{Cu} \lambda x . e : \tau \rightarrow \sigma}$	$\frac{\Gamma, x : \tau \vdash_{Ch} e : \sigma}{\Gamma \vdash_{Ch} \lambda x^{\tau} . e : \tau \rightarrow \sigma}$
Типы		
$\tau ::= \iota \mid \tau \rightarrow \tau$	$\frac{\Gamma \vdash_{Cu} e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash_{Cu} e_2 : \tau_2}{\Gamma \vdash_{Cu} e_1 e_2 : \tau_2}$	$\frac{\Gamma \vdash_{Ch} e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash_{Ch} e_2 : \tau_2}{\Gamma \vdash_{Ch} e_1 e_2 : \tau_2}$
Контексты		
$\Gamma ::= \emptyset \mid \Gamma, V : \tau$		

Рис. 4. Простое типизированное лямбда-исчисление

ловной редукции. Тем не менее доказать достаточность утверждения 2 не так просто. Для этого мы введём формальное определение головной линейной редукции в виде системы переходов, с помощью которой формально и докажем утверждение теоремы (см. главу 3).

2.2.2 Простое типизированное лямбда-исчисление

Простое типизированное лямбда-исчисление (simply-typed lambda-calculus, STLC) имеет схожий с бестиповым синтаксис, тем не менее, не все термы бестипового исчисления являются *допустимыми* (valid) термами простого типизированного. Типы в простом типизированном лямбда-исчислении описываются следующей грамматикой: $\tau ::= \iota \mid \tau \rightarrow \tau$, где ι представляет собой некоторый *базовый тип* (ground type), а $\tau_1 \rightarrow \tau_2$ — *функциональный тип* (стрелочный тип, arrow type), ассоциативность у оператора *стрелка* \rightarrow правая. Принято выделять два похода, два стиля типизации: стиль *Карри* (типизация по Карри, \a la Curry) и стиль *Чёрча* (типизация по Чёрчу, \a la Church). Эти два подхода являются принципиально разными: при типизации по Карри сначала задаётся грамматика термов — синтаксис, затем определяется их поведение — семантика, и наконец вводится система типов — типизация, отвергающая термы, обладающие нежелательным поведением, в то время как в стиле Чёрча типизация предшествует семантике⁷. Иными словами, при типизации по Чёрчу семантикой — смыслом — наделены исключительно *пра-*

ильно типизированные (well-formed, well-typed) термы, а типизация по Карри даёт возможность рассуждать о поведении лямбда-термов вне зависимости от того, являются ли они правильно типизированными или нет. Синтаксис и типизации по Чёрчу и по Карри простого типизированного лямбда-исчисления в виде правил вывода (inference rules) приведены на рис. 4. Суждение вида $\Gamma \vdash \Lambda : \tau$ называется *терм в контексте* (term-in-context), где *контекст типизации* Γ (typing context) является множеством текущих предположений о типах термов.

Одним из основных отличий двух систем типизации является свойство единственности типа: при типизации по Чёрчу терм имеет не более одного типа, в то время как при типизации по Карри терм может иметь один или несколько типов, или быть не типизируемым вовсе. Например, терм $\lambda x : \tau . x$ (при типизации по Чёрчу) имеет един-

⁷ Иногда под типизацией по Чёрчу понимают явно типизированные системы (типы переменных явно указываются в синтаксисе языка), а под типизацией по Карри — неявно типизированные (типы присваиваются в процессе компиляции или интерпретации программы). Путаница возникла ввиду того, что сам Чёрч описывал своё исчисление в явном стиле, в то время как Карри использовал неявный стиль. Несмотря на то, что такой взгляд на стили типизации является исторически сложившимся: явно типизированные системы часто описывались и описываются в стиле Чёрча, а неявно типизированные — в стиле Карри, — он всё же является ошибочным.

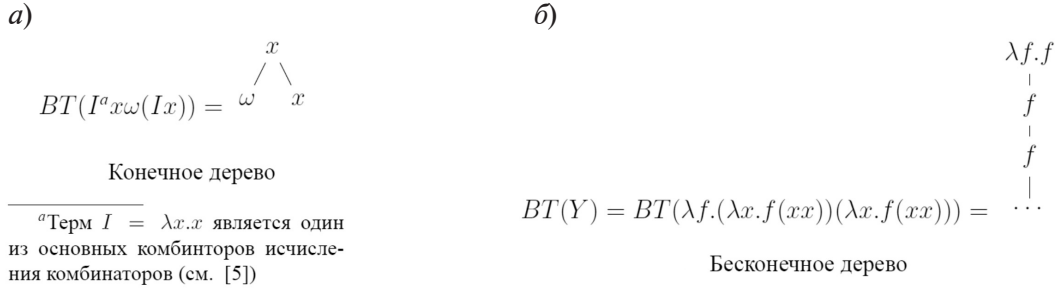


Рис. 5. Пример: деревья Бёма для термов, не имеющих нормальной формы

ственный тип $\tau \rightarrow \tau$, где τ – некоторый конкретный тип, указанный в синтаксисе, например, ι или $(\iota \rightarrow \iota \rightarrow \iota) \rightarrow \iota \rightarrow \iota \rightarrow \iota$ и т. д., а аналогичному ему терму $\lambda x.x$ в представлении Карри соответствует всё множество типов вида $\alpha \rightarrow \alpha$.

Как уже упоминалось, не все термы бестипового исчисления являются термами простого типизированного. Более того, в отличие от бестипового лямбда-исчисления, простое типизированное является *строго нормализуемым* (strongly normalized), т. е. каждый допустимый терм имеет нормальную форму, а процедура его нормализации всегда завершается. Так, например, терм бестипового лямбда-исчисления ω не имеет нормальной формы и не является допустимым термом простого типизированного. Исчерпывающее описание простого типизированного лямбда-исчисления и его свойств, а также более богатых систем типов читатель может найти в [5, 10, 11].

2.2.3 Головная нормальная форма и дерево Бёма

Произвольный лямбда-терм M можно записать в следующей форме⁸:

$$BT(M) := \begin{cases} \omega & \text{, если } M \text{ не имеет hnf} \\ \lambda \vec{x}.y \begin{matrix} / \\ BT(V_1) \\ \dots \\ \backslash \\ BT(V_n) \end{matrix} & \text{, если } \text{phnf}(M) = \lambda \vec{x}.yV_1 \dots V_n \end{cases}$$

Как уже отмечалось ранее, повторное применение головной редукции к аргумен-

⁸ Утверждение может быть легко доказано индукцией по структуре терма M .

$$M = \lambda \vec{v}.UV_1 \dots V_n, \text{ где } U = \begin{bmatrix} y \\ (\lambda y.P)Q \end{bmatrix} \quad (1)$$

В первом случае говорят, что терм M находится в *головной нормальной форме* (head normal form), во втором случае $(\lambda y.P)Q$ образует головной редекс. Заметим, что по определению терм находится в головной нормальной форме тогда и только тогда, когда он не содержит головных редексов.

Важным следствием этого определения является то, что головная нормальная форма не обладает свойством единственности: один и тот же терм может иметь более одной головной нормальной формы. Например, головной нормальной формой терма $y((\lambda x.x)z)$ является как терм сам по себе, так и его нормальная форма yz . Среди множества головных нормальных форм терма выделяют *основную* головную нормальную форму (principal head normal form, phnf), получаемую посредством головной редукции терма, записанного в виде (1), если редукция завершается.

Деревом Бёма (Böhm tree) $BT(M)$ терма M называется дерево, построенное по следующим правилам:

там головной нормальной формы нормализует терм. Таким образом, если терм имеет нормальную форму, то дерево Бёма, представляющее этот терм, вычислимо и конечно, и наоборот [5, 27, 28]. Также отметим,

что если терм нормальной формы не имеет, то дерево Бёма, ему соответствующее, либо конечно, а процедура построения его следующего уровня расходится (см. рис. 5 а), либо является бесконечным (см. рис. 5 б).

2.2.4 η-длинная форма терма

η-длинной форма (η-long form) терма получается путём его полного η-расширения и заменой бинарного оператора применения на оператор *длинного применения* @^l (long application). Заметим, что каждая переменная, непосредственным предком которой не является абстракция, также подлежит η-расширению, путём введения анонимной (dummy) абстракции ($x \mapsto \lambda.x$). Например, η-длинной формой терма $\lambda x^{1 \rightarrow 1}.x$ является терм $\lambda x^{1 \rightarrow 1}.a^1 b^1.x(\lambda.b)(\lambda.a)$. Заметим, что понятие η-длинной форм имеет смысл только в типизированном исчислении,

$$\Lambda_{odd}^{lf} ::= \lambda x_1^{\tau_1} \dots x_p^{\tau_p}. \Lambda_{even}^{lf}$$

$$\Lambda_{even}^{lf} ::= x \Lambda_{1odd}^{lf} \dots \Lambda_{nodd}^{lf} \mid \Lambda_{0odd}^{lf} @^l \Lambda_{1odd}^{lf} \dots \Lambda_{modd}^{lf}, \text{ где } m \in \mathbb{N}, n, p \in \mathbb{N}_0$$

Иными словами, нечётные уровни в абстрактном синтаксическом дереве представляют собой абстракции по произвольному числу аргументов, а чётные — применением переменной к нулю или более аргументам, либо же применение другого η-длинного терма к одному или более аргументам, получившее название оператора длинного применения @^l.

2.2.5 Трассирующая нормализация

В 70-х гг. XX в. Гордон Плоткин [8] сформулировал проблему построения *полностью абстрактной модели вычислений* (fully abstract model of computations) для языка программирования вычислимых функций PCF (Programming Computable Functions), заключающуюся в построении абстрактной модели языка, являющейся одновременно *полной* (complete) — каждый терм языка представим в его модели — и *согласованной* (sound) — каждый элемент абстрактной модели имеет прообраз в языке. Впервые предложенная Плоткиным проблема была решена в начале 90-х гг. независимо двумя группами исследователей (Мартинотом Хуландом и Люком Онгом [23], и Самсоном Абрамски и Гаем МакКаскером [22])

т. к. в нём η-расширение ограничено хотя бы размером типа терма, в то время как в бестиповом случае η-расширять терм можно до бесконечности.

β-нормальной η-длинной является форма терма, не содержащая β-редексов, но являющаяся η-длинной. Например, β-нормальной η-длинной формой терма $(\lambda x^{1 \rightarrow 1}.x) y$ является терм $\lambda a^1. y(\lambda.a)$, которой произведением полной η-редукции, которая, разумеется, всегда завершается, η-эквивалентен терму $y^{1 \rightarrow 1}$, являющемуся, в свою очередь, нормальной формой исходного терма.

Заметим, что абстрактный синтаксис, используемый в определении η-длинной формы терма отличается от стандартного абстрактного синтаксиса λ-исчисления, введённого в начале главы 2, а именно, он описывается следующей грамматикой:

посредством *игровой семантики* (game semantics) программ, являющейся одним из способов задания формальной семантики языков программирования. Игровая семантика рассматривает вычисления, как *игру* (game) между *пропонентом(-ами)* (proponent, player) и *оппонентом(-ами)* (opponent) — программой и её окружением, а семантика — смысл — программы — *стратегия* (strategy), которой игрок должен придерживаться. Подробнее с игровой семантикой читатель может ознакомиться в работах [22–26].

В 2015 г. Люком Онгом было замечено [1], что из игровой семантики программ для простого типизированного лямбда-исчисления естественным образом вытекает нестандартная процедура его нормализации. Отличительной особенностью данной процедуры является то, что вместо изменения терма посредством β-редукции, используя стандартные приёмы, такие как окружения или замыкания⁹, *трассирующая нормализация* (traversal-based normalization, normalization by traversals) оставляет входной терм нетронутым, производя его нормализацию путём обхода абстрактного синтаксического дерева терма, запоминая историю этого



обхода. Оригинальная процедура нормализации, предложенная Онгом, требует преобразования термов в η -длинную форму, завершаясь в β -нормальной η -длинной форме.

Интуитивно, процедура трассирующей нормализации, предложенная Онгом, производит обход абстрактного синтаксического дерева терма в глубину по самому левому пути, запоминая историю обхода, называемую *трассой* (traversal). Достигая вхождения связанной переменной, процедура нормализации «перепрыгивает» на поддереву, соответствующее динамическому аргументу, с которым связана абстракция этой переменной. Заметим, что такой аргумент всегда найдётся, в силу того, что терм находится в η -длинной форме, а значит, все абстракции применены к некоторым аргументам. При достижении вхождения свободной переменной, процедура нормализации «разделяется», т. е. продолжается независимыми обходами каждого из аргументов этой переменной, порождая тем самым множество трасс, каждая из которых впоследствии определит уникальный путь от вершины до некоторого листа в абстрактном дереве η -длинной β -нормальной

формы входного терма. Процедура обхода и построения трасс является синтаксически управляемой и детерминированной. Стоит также отметить, что предложенная процедура нормализации в некотором смысле¹⁰ соответствует головной линейной редукции термов.

2.3 Системы переходов

Одним из подходов к изучению поведения дискретных систем являются *системы переходов* (transition system, [9]) — суть четвёрка (S, I, F, \rightarrow) , где S — множество состояний системы, $I \subseteq S$ — множество исходных состояний системы, $F \subseteq S$ — множество конечных состояний системы, $\rightarrow \subseteq S \times S$ — отношение, означающее дискретный переход системы из одного состояния в другое, обозначаемое $(s_1, s_2) \in \rightarrow$ или, для удобства чтения, $s_1 \rightarrow s_2$, где $s_1, s_2 \in S$.

Система переходов называется *детерминированной*, если текущее состояние системы однозначно определяет последующее, в противном случае система называется *недетерминированной*.

Например, алгоритм Евклида может быть записан следующей системой переходов:

$$\begin{aligned} S &= \mathbb{N} \times \mathbb{N} \\ I &= S \\ F &= \{(n, n) \mid n \in \mathbb{N}\} \\ \rightarrow &= \{((m, n), (m - n, n)) \mid m > n\} \cup \{((m, n), (m, n - m)) \mid m < n\} \end{aligned}$$

3 Головная линейная редукция (HLR)

В этой главе мы введём формальное определение головной линейной редукции в виде системы переходов и докажем её согласованность с головной редукцией.

3.1 HLR как система переходов

Определение. Система переходов для головной линейной редукции:

⁹ *Замыканием* (closure) называется функция первого порядка, определяющая значения переменных определённых вне тела функции, в случае лямбда-исчисления — свободных переменных. Использование замыканий является одной из стандартных способов реализации функциональных языков программирования.

1. Состоянием является тройка $\langle A[\underline{B}]; \Gamma; \Delta \rangle$, где

- $A[\underline{B}]$ — λ -терм, в котором вершина, являющаяся корнем поддерева B , выделена¹¹.

¹⁰ Внимательный читатель мог заметить, что головная линейная редукция завершается в псевдоголовной нормальной форме, в то время как трассирующая нормализация в стиле Онга — в β -нормальной η -длинной форме. Тем не менее, с точностью до вхождения свободной переменной, имеющей не пустой список аргументов, и η -эквивалентности, если входной терм находится в η -длинной форме, эти формы равны.

¹¹ Выделение является синтаксической пометкой вершины абстрактного синтаксического дерева терма и обозначается *подчёркиванием* в примерах и правилах.

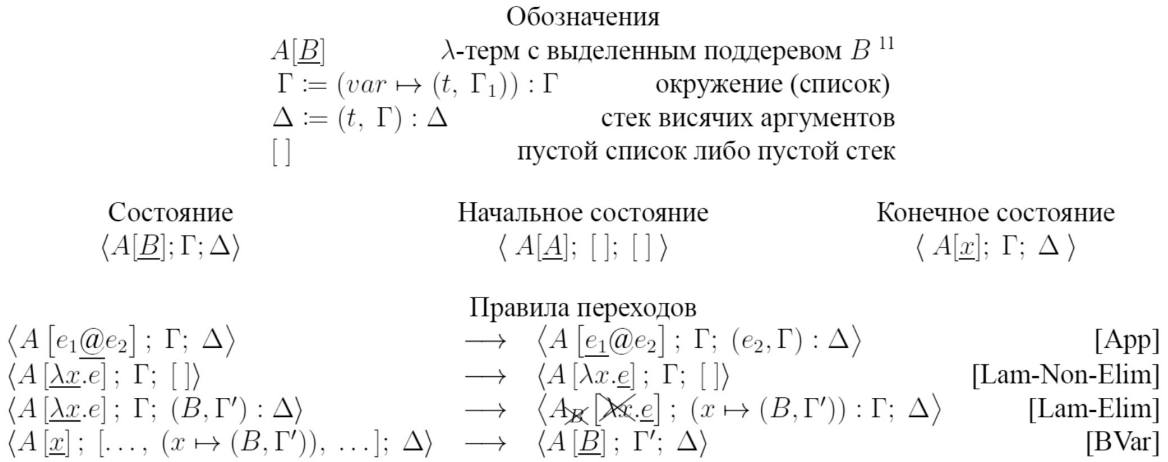


Рис. 6. Система переходов для головной линейной редукции

• $\Gamma := (var \mapsto (t, \Gamma_1)) : \Gamma$ — окружение — список связываний переменных, где var — переменная, t — λ-терм, а Γ_1 — сохранённое окружение, соответствующее терму t .

• $\Delta := (t, \Gamma) : \Delta$ — стек висячих аргументов — пар вида (t, Γ) , где t является λ-термом, а Γ — соответствующим ему окружением.

2. Начальным является состояние $\langle A[A]; []; [] \rangle$, где $[\]$ — пустые окружение и стек висячих аргументов, соответственно, а $A[A]$ — входной терм с выделенным корнем.

3. Конечным является состояние вида $\langle A[x]; \Gamma; \Delta \rangle$, где $A[x]$ — исходный λ-терм с выделенной переменной x , $x \notin \Gamma$.

4. Правила переходов приведены на рис. 6.

• В случае применения (правило [App]), выделенным становится его левый аргумент, а в стек висячих аргументов помещается аргумент с текущим окружением. В дальнейшем этот аргумент может использоваться для формирования простого редекса.

• Если выделенной вершиной является абстракция, то либо, если стек висячих аргументов пуст, и переменная абстракции не связывается ни с каким аргументом (правило [Lam-Non-Elim]), либо, если стек висячих аргументов не пуст, формирует простой редекс с его вершиной (правило [Lam-Elim]). В этом случае абстракция и соответствующие применение и его аргу-

мент вычёркиваются¹² из текущего дерева, а простой редекс сохраняется в текущем окружении. Очевидно, что правила напрямую согласованы с определением простого редекса.

• Если же выделенной является переменная, то, если она является свободной, система переходов достигла своего конечного состояния, ежели переменная является связанной, то либо существует простой редекс, аргумент которого может быть подставлен вместо вхождения этой головной переменной (правило [BVar]), либо такого редекса нет, и опять же система переходов достигла конечного состояния.

Заметим, что система переходов является детерминированной и синтаксически управляемой, а выбор правила зависит лишь от первого элемента состояния (формально, выбор между правилами [Lam-*] зависит от текущего состояния стека висячих аргументов, но он также напрямую зависит от входного терма). Единственным случаем, когда система переходов может достичь своего конечного состояния, является случай выделенной переменной, которая либо свободна, либо не связана никаким про-

¹² Вычёркивание является синтаксической пометкой вершин в дереве, а не полным удалением подтерма. С этого момента мы будем использовать обозначение $A_x[\lambda x. e]$ для терма A , у которого подтерм B , вершина “λx” и соответствующая вершина оператора применения помечены — вычеркнуты.

стым редексом. Более того, правила гарантируют, что выделенной может быть лишь вершина, находящаяся на самом левом пути терма. Иными словами, если система

переходов достигает конечного состояния, вычисления завершаются, нос-переменная выделена и не связана никаким простым редексом.

Пример. Рассмотрим терм $(\lambda x . x) @ (\lambda y . y)$.

$$\begin{aligned} \langle (\lambda x . x) @ (\lambda y . y); \quad []; \quad [] \rangle &\xrightarrow{[App]} \quad (2) \\ \langle (\lambda x . x) @ (\lambda y . y); \quad []; \quad [((\lambda y . y), [])] \rangle &\xrightarrow{[Lam-Elim]} \quad (3) \\ \langle \cancel{(\lambda x . x)} @ \cancel{(\lambda y . y)}; \quad [x \mapsto ((\lambda y . y), [])]; \quad [] \rangle &\xrightarrow{[BVar]} \quad (4) \\ \langle \cancel{(\lambda x . \lambda y . y)} @ \cancel{(\lambda y . y)}; \quad []; \quad [] \rangle &\xrightarrow{[Lam-Non-Elim]} \quad (5) \\ \langle \cancel{(\lambda x . \lambda y . y)} @ \cancel{(\lambda y . y)}; \quad []; \quad [] \rangle &\not\rightarrow \quad (6) \end{aligned}$$

Как и ожидалось, первым элементом конечного состояния является псевдоголовная нормальная форма входного терма: $(\lambda x . \lambda y . y) @ (\lambda y . y)$. Головной формой терма является терм $\lambda y . y$, в данном случае получаемый простым «выбрасыванием» вычеркнутых вершин из первого элемента конечного состояния системы переходов. В общем случае нам потребуется определить специальную функцию *exp*, которая строит соответствующий терм из конечного состояния, и показать её согласованность с головной редукцией (см. гл. 3.2). ■

Более показательный пример приведён на рис. 9.

$$exp \langle M[A]; \Gamma \bullet [(x \mapsto (B, \Gamma'))]; \Delta \rangle = exp \langle M[A[x/B[\Gamma']]]; \Gamma; \Delta \rangle \quad (7)$$

$$exp \langle M_B[A]; []; (B, \Gamma') : \Delta \rangle = exp \langle M_{B[\Gamma']}[A]; []; \Delta \rangle \quad (8)$$

$$exp \langle M; []; [] \rangle = M' \quad (9)$$

где $B[\Gamma'] = exp \langle B; \Gamma'; [] \rangle$, а терм M' получается из терма M удалением всех вычеркнутых вершин. (7) производит подстановку терма $B[\Gamma']$ вместо всех вхождений переменной x в такое поддерево терма M , которое имеет корнем выделенную вершину (т. е. поддерево A). Каждый рекурсивный вызов (8) производит подстановку всех переменных в соответствии с контекстом Γ' исключительно висячем аргументе B . Заметим, мы придерживаемся соглашения Барендрегта (Barendregt's convention): предполагается, что все переменные имеют уникальные имена. Следование этой концепции в данном случае позволяет избежать захвата имён переменных при подстановке. Далее мы будем называть *расширением* состояния системы переходов результат применения функции расширения *exp* к этому состоянию.

3.2 Согласованность с головной редукцией

В данной главе приводится доказательство согласованности головной линейной и головной редукций. Сначала определяется дополнительная функция *расширения* – *exp*.

Функция расширения. По данному состоянию системы переходов функция *exp* возвращает λ -терм. Интуитивно, функция *exp* производит последовательную редукцию всех простых редексов, накопившихся на данный момент времени. Поскольку последовательность простых редексов совпадает с последовательностью головных редексов, головная и линейная головная редукции согласованы. Формально¹³

Отметим некоторые важные для нас свойства системы переходов головной линейной редукции:

- Исходя из определения функции *exp* (конкретно уравнений (7), (8)), единственное правило системы переходов, при применении которого расширение получаемого состояния не равно расширению состояния, из которого оно получено, – [Lam-Elim], все остальные правила оставляют расширение неизменным.

- Количество переходов системы без применения правила [Lam-Elim] конечно. Данный факт является прямым следствием определения контекста, конечности размера входного терма и того, что в силу опре-

¹³ $\alpha \cdot \beta$ означает конкатенацию контейнеров α и β .

деления только правило [Lam-Elim] может изменить контекст. С этого момента мы будем обозначать последовательное применение правил без применения правила [Lam-Elim] через \rightarrow .

$$\begin{aligned} \text{exp}(2) &= \text{exp}(\langle (\lambda x . x) @ (\lambda y . y); []; [] \rangle) = (\lambda x . x) @ (\lambda y . y) \\ \text{exp}(3) &= \text{exp}(\langle (\lambda x . x) @ (\lambda y . y); []; [(\lambda y . y), []] \rangle) = \text{exp}(\langle (\lambda x . x) @ (\lambda y . y); []; [] \rangle) \\ &= (\lambda x . x) @ (\lambda y . y) \\ \text{exp}(4) &= \text{exp}(\langle \cancel{(\lambda x . x)} @ \cancel{(\lambda y . y)}; [(x \mapsto ((\lambda y . y), []))]; [] \rangle) \\ &= \text{exp}(\langle \cancel{(\lambda x . \lambda y . y)} @ \cancel{(\lambda y . y)}; []; [] \rangle) = \lambda y . y \\ \text{exp}(5) &= \text{exp}(\langle \cancel{(\lambda x . \lambda y . y)} @ \cancel{(\lambda y . y)}; []; [] \rangle) = \lambda y . y \\ \text{exp}(6) &= \text{exp}(\langle \cancel{(\lambda x . \lambda y . y)} @ \cancel{(\lambda y . y)}; []; [] \rangle) = \lambda y . y \end{aligned}$$

Идея доказательства согласованности головной и линейной головной редукций состоит в следующем: расширение не может быть изменено никаким правилом, кроме [Lam-Elim]. Следовательно, после применения функции расширения к состояниям системы переходов каждое применение правила [Lam-Elim] соответствует одному шагу головной редукции. Так, например, $(\lambda x . x) @ (\lambda y . y)$ соответствует первому шагу, $\lambda y . y$ соответствует трём оставшимся шагам.

Теорема 2. Пусть $\langle \dots \rangle$ – некоторое состояние системы переходов, расширение которого обозначено ... (см. иллюстрацию к теореме, рис. 7), такое, что на следующем шаге должно быть применено правило [Lam-Elim], и результатом его применения является состояние $\langle M_i; \Gamma_i; \Delta_i \rangle$, расширением которого является некоторый терм M'_i . Далее система переходов делает некоторое конечное число шагов до следующего применения правила [Lam-Elim], состояние $\langle M_i; \Gamma_i; \Delta_i \rangle$, расширением которого также является терм M'_i , тогда если может быть применено правило [Lam-Elim], расширяющее окружение Γ новым связыванием $(x \mapsto (B, \Gamma'))$, и расширением результата

Пример. В данном примере показан результат применения функции exp к каждому состоянию системы переходов для примера, приведённого в разделе 3.1.

применения которого является некоторый терм M'_{i+1} , то терм M'_{i+1} получается из термина M_i за одним шаг головной редукции.

Доказательство. Индукция по количеству применений правила [Lam-Elim].

База индукции. Очевидно, т. к. первый элемент, добавляемый в Γ , является головным редексом по определению.

Индукционный переход. Согласно индукционному предположению, после i -го применения правила [Lam-Elim] текущим является состояние вида $\langle M_i; \Gamma_i; \Delta_i \rangle$, расширением которого является некоторый терм M_i . Как уже отмечалось ранее, \rightarrow не изменяет расширения и число шагов \rightarrow конечно. Таким образом, существует два возможных случая:

1. Система переходов достигла конечного состояния, и нечего доказывать.

2. Система не достигла конечного состояния, и может быть применено правило [Lam-Elim]. В данном случае, согласно определению правила [Lam-Elim], мы знаем вид исходного состояния: $\langle A[\lambda x . e]; \Gamma; (B, \Gamma') : \Delta \rangle$, обозначим его за (i) , и вид результата применения правила [Lam-Elim] к нему: $\langle A_x[\cancel{\lambda x . e}]; (x \mapsto (B, \Gamma')) : \Gamma; \Delta \rangle$, обозначим

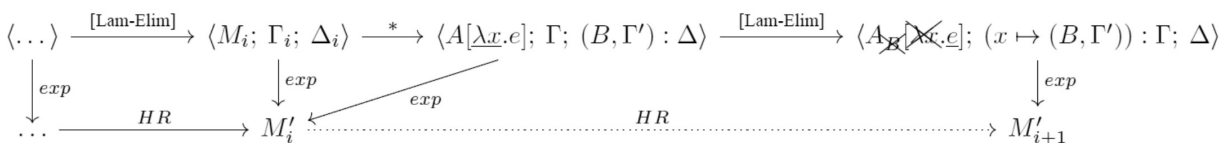


Рис. 7. Иллюстрация к теореме 2



его за (ii). Заметим, что нам надо показать, что результат применения функции exp к состоянию (ii) совпадает с результатом

$$exp \langle A[\underline{\lambda x}. e]; \Gamma; (B, \Gamma') : \Delta \rangle \quad (10)$$

$$\xrightarrow{* exp} exp \langle A[\underline{\lambda x}. e[\Gamma]]; []; (B, \Gamma') : \Delta \rangle \quad \text{за конечное число шагов, по (7)} \quad (11)$$

$$\xrightarrow{exp} exp \langle A_{B[\Gamma']}[\underline{\lambda x}. e[\Gamma]]; []; \Delta \rangle \quad \text{за один шаг (8)} \quad (12)$$

$$\xrightarrow{exp} \dots \quad (13)$$

Теперь применим функцию exp к состоянию (ii) :

$$exp \langle A_{\cancel{x}}[\cancel{\lambda x}. \underline{e}]; (x \mapsto (B, \Gamma')) : \Gamma; \Delta \rangle \quad (14)$$

$$\xrightarrow{* exp} exp \langle A_{\cancel{x}}[\cancel{\lambda x}. \underline{e}[\Gamma]]; [(x \mapsto (B, \Gamma'))]; \Delta \rangle \quad \text{устраняя } \Gamma, \text{ согласно (7)} \quad (15)$$

$$\xrightarrow{exp} exp \langle A_{\cancel{x}}[\cancel{\lambda x}. \underline{e}[\Gamma][x/B[\Gamma']]]; []; \Delta \rangle \quad \text{за шаг (7)} \quad (16)$$

$$\xrightarrow{exp} \dots \quad (17)$$

Легко заметить, что терм, являющийся первым компонентом состояния (12), $A_{B[\Gamma']}[\underline{\lambda x}. e[\Gamma]]$, имеет головным редексом $(\lambda x, B)$, поскольку контекст уже является пустым. Таким образом, если применить шаг головной редукции к данному терму, результатом будет терм $A_{\cancel{x}}[\cancel{\lambda x}. \underline{e}[\Gamma][x/B[\Gamma']]]$, по определению головной редукции, соответствующий первой компоненте состояния (16). Согласно (7) и (8), если продолжить применение функции exp к состояниям (12) и (16), соответственно, то в обоих случаях один и тот же стек висячих аргументов Δ не произведёт никаких изменений ни в аргументе B , ни в выделенном поддереве e . Следовательно, редекс $(\lambda x, B)$ является головным и для терма M_i , а значит, и расширением состояния (ii) является терм M_{i+1} , что и требовалось доказать ■

Теорема 2 сопоставляет каждому шагу головной редукции шаг системы переходов, соответствующий применению правила [Lam-Elim]. Таким образом, прямым следствием теоремы является тот факт, что *головная линейная редукция завершается тогда и только тогда, когда завершается головная редукция*. Ввиду того, что функция расширения не способна изменить путь от корня дерева терма до корня его выделенного поддеревя, первая компонента конечного состояния системы переходов для головной линейной редукции содержит

одного шага головной редукции терма M_i , т. е. термом M_{i+1} . Применим функцию exp к состоянию (i) :

терм такой, что самый левый путь в дереве, его представляющем, является частью головной нормальной формы, а значит, и расширение конечного состояния является вершиной дерева Бёма. Таким образом, повторное применение головной линейной редукции ко всем оставшимся аргументам (полная головная линейная редукция, см. гл. 4) нормализует терм.

4 Полная головная линейная редукция

Полная головная линейная редукция (complete head linear reduction – CHLR) является расширением головной линейной редукции, рекурсивно применяющим последнюю к аргументам при достижении конечного состояния. В данной главе приводится формальное описание CHLR в виде системы переходов, являющейся истинным расширением системы переходов для HLR.

4.1 Система переходов для полной головной линейной редукции

Система переходов для CHLR отличается от системы переходов для HLR тремя новыми правилами [FVar-*]. Неформально говоря, эти правила обрабатывают ситуацию, когда система переходов HLR достигла своего конечного состояния, т. е. первым компонентом состояния системы переходов является некоторый терм с



Рис. 8. Система переходов для полной головной линейной редукции

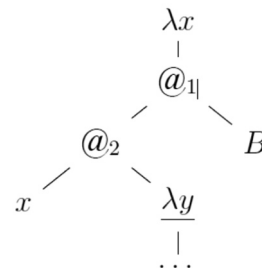
выделенной свободной или динамически несвязанной с аргументом переменной. Под *динамически несвязанной с аргументом переменной* понимается связанная переменная, чья абстракция не применена ни к какому аргументу. Итак, система переходов для CHLR обладает следующими изменениями по сравнению с системой переходов для HLR.

- Стек висячих аргументов Δ расширяется специальным символом $\$$. Символ $\$$ играет роль разделителя, запрещающего связывать абстракцию с аргументом, если путь от этого аргумента до соответствующей абстракции не является левым путём. Иными словами, разделитель $\$$ гарантирует, что все редексы в контексте являются простыми для данного подтерма. Например, пусть дан некоторый входной терм, и в ходе полной головной линейной редукции получено состояние $\langle M[\lambda y]; \Gamma; [\$, (B, \Gamma_0)] \rangle$, где терм

$M[\lambda y]$ изображён справа. В данном примере разделитель $\$$ запрещает связывание вершины λy с висячим аргументом B , поскольку они не образуют простого редекса.

- *Начальным* является состояние $\langle \lambda_1; []; [] \rangle$, где λ_1 является входным термом с выделенным корнем.

- *Конечным* состоянием является $\langle M[x]; \Gamma; [] \rangle$, где $x \notin dom(\Gamma)$. Заметим, что в отличие от головной линейной редукции, в конечном состоянии стек висячих аргументов должен быть пустым.



- Результатом CHLR является терм в нормальной форме; в терминах системы переходов это означает, что он может быть получен из первой компоненты конечного состояния системы переходов путём удаления из неё всех вычеркнутых вершин.

- На рис. 8 приведены правила для переходов для TS для {CHLR}. Для удобства чтения изменения по сравнению с соответ-

ствующими правилами для HLR выделены с помощью прямоугольников: выделение.

4.2 Функция расширения

Как и в случае HLR, мы определим функцию exp расширения состояния системы переходов до терма. В данном случае, по сравнению с системой переходов для HLR, добавляется лишь одно правило – (20).

$$exp \langle M[\underline{A}]; \Gamma \bullet (x \mapsto (B, \Gamma')); \Delta \rangle = exp \langle M[\underline{A}[x/B[\Gamma']]]; \Gamma; \Delta \rangle \quad (18)$$

$$exp \langle M_B[\underline{A}]; []; (B, \Gamma') : \Delta \rangle = exp \langle M_{B[\Gamma]}[\underline{A}]; []; \Delta \rangle \quad (19)$$

$$\boxed{exp \langle M_B[\underline{A}]; []; \$: \Delta \rangle} = \boxed{exp \langle M_B[\underline{A}]; []; \Delta \rangle} \quad (20)$$

$$exp \langle M; []; [] \rangle = M' \quad (21)$$

$$\text{где } B[\Gamma'] = exp \langle \underline{B}; \Gamma'; [] \rangle \quad (22)$$

$$M' \text{ получается из } M \text{ удалением вычеркнутых вершин} \quad (23)$$

Теорема 3. Полная головная линейная редукция завершается тогда и только тогда, когда терм имеет нормальную форму. Более того, если M – некоторый терм, $s_{final} ::= \langle M'; \Gamma'; \Delta' \rangle$ – конечное состояние системы переходов полной головной линейной редукции для терма M , обозначим за M_{exp} расширение состояния s_{final} , тогда:

1. M_{exp} есть M' за исключением вычеркнутых поддеревьев;

2. M_{exp} не содержит редексов;

3. M_{exp} является нормальной формой терма M .

Доказательство. Заметим, что правила [FVar-*] не могут изменить расширение состояния системы переходов. Следовательно, доказательство корректности CHLR является прямым следствием корректности HLR (см. следствия к теореме 2), а значит, полная головная линейная редукция завершается тогда и только тогда, когда завершается полная головная редукция терма,

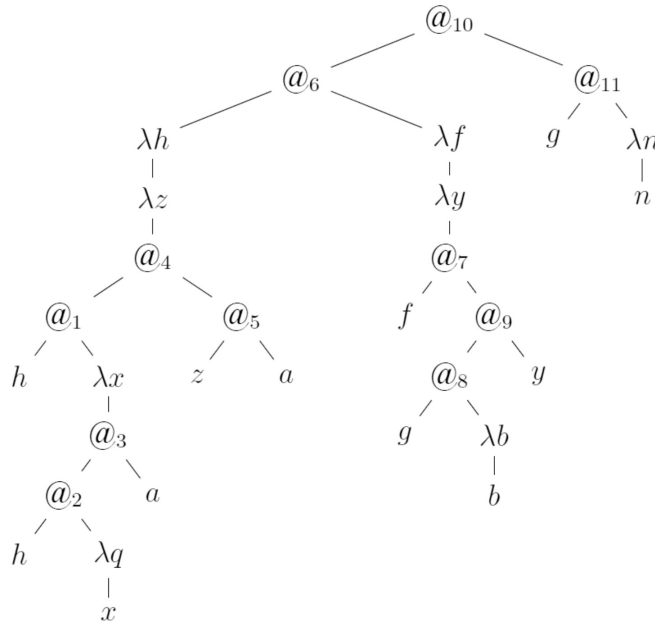


Рис. 9. Синтаксическое дерево терма $\langle NPR \rangle$

что в свою очередь эквивалентно конечности дерева Бёма, представляющего входной терм, а значит, и эквивалентно существованию нормальной формы входного терма. Иными словами, полная головная линейная редукция завершается тогда и только

тогда, когда у входного терма существует нормальная форма. ■

Далее мы рассмотрим пример применения системы переходов для головной линейной и полной головной линейной редукций соответственно на примере терма $\langle N P R \rangle$, где

$$N = \lambda h . \lambda z . h @ (\lambda x . (h @ (\lambda q . x) @ a)) @ (z @ a)$$

$$P = \lambda f . \lambda y . f @ ((g @ (\lambda b . b)) @ y)$$

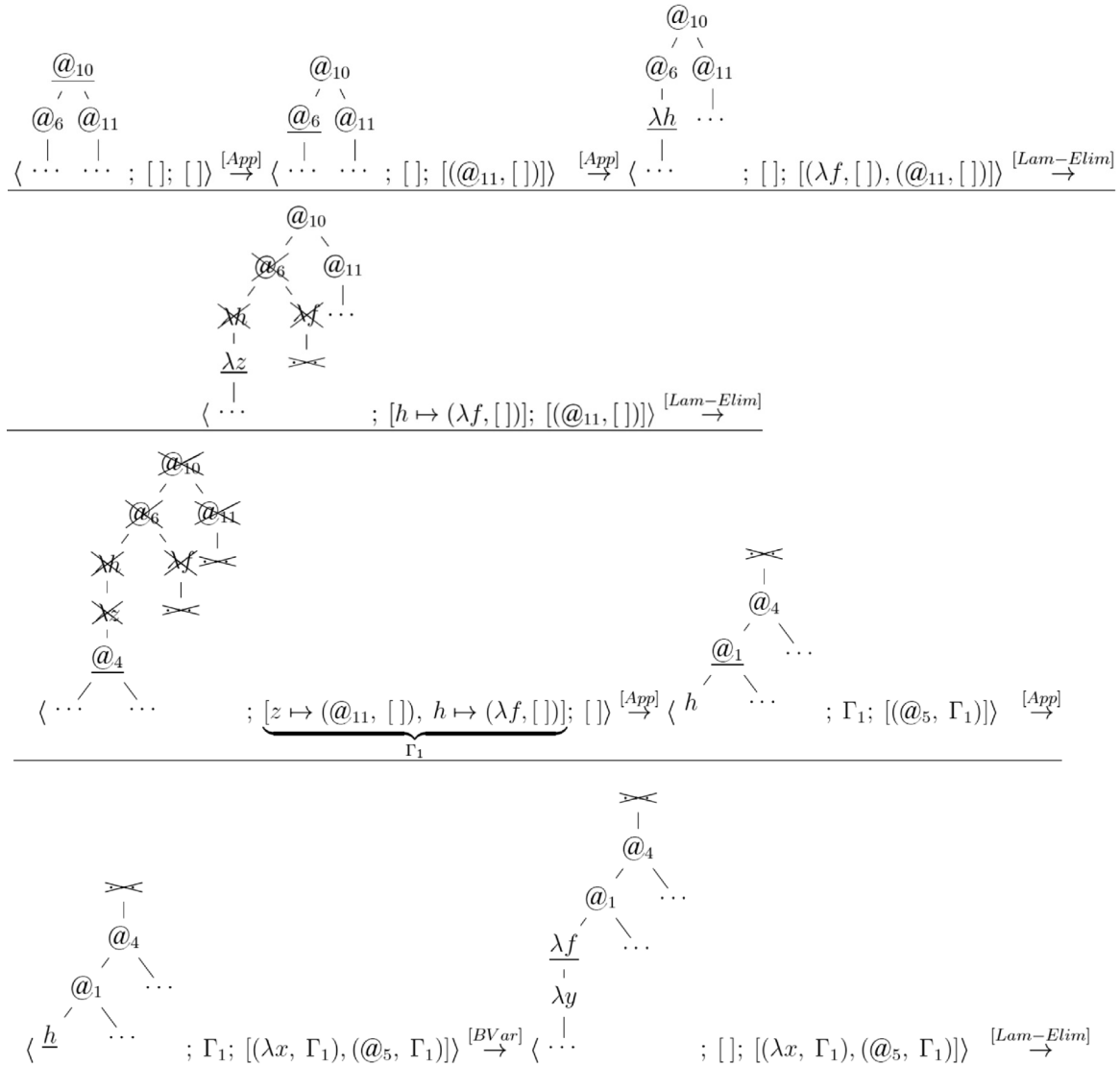
$$R = g @ (\lambda n . n)$$

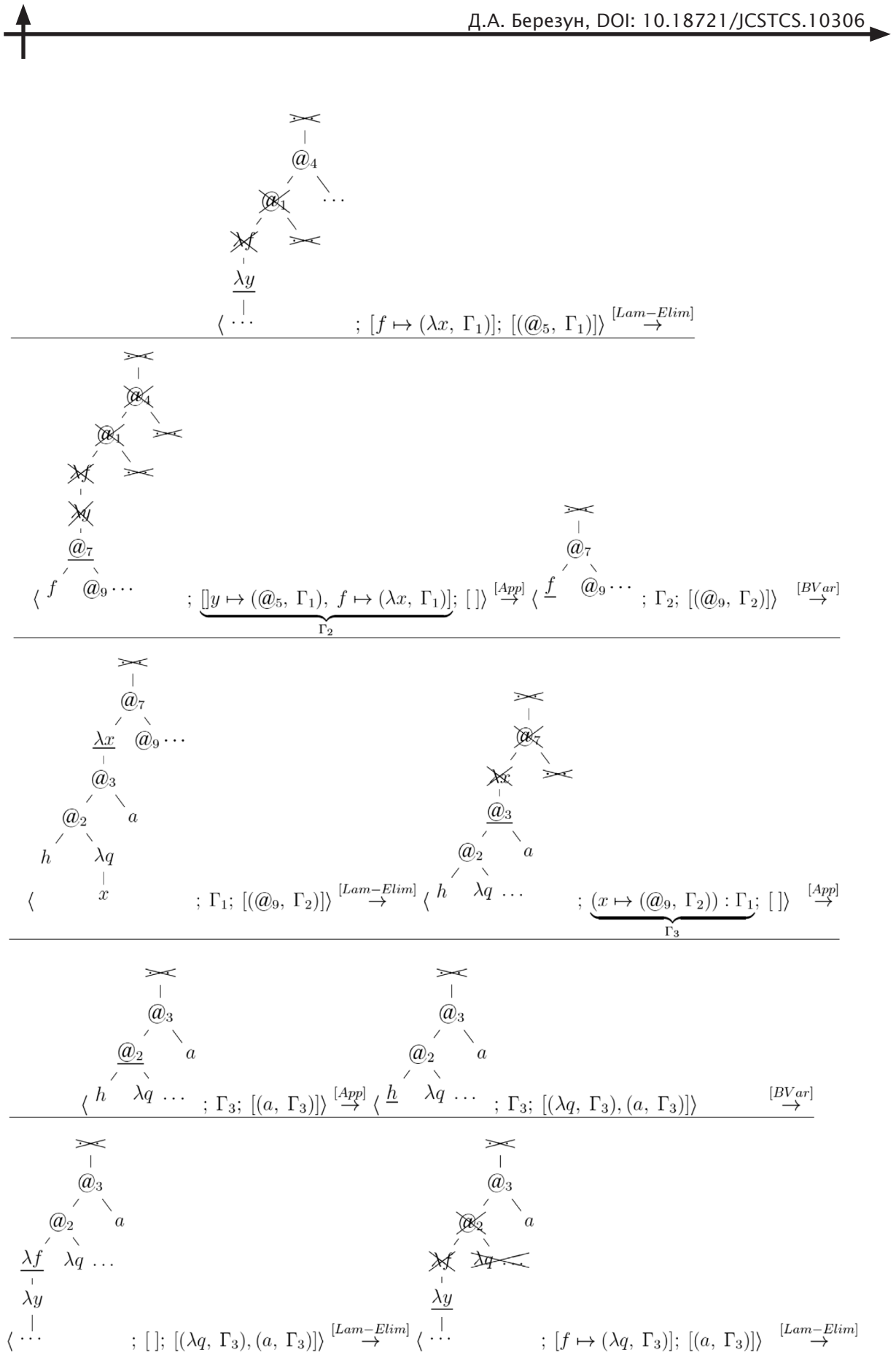
$$N P R = ((\lambda h \lambda z . (h @_1 (\lambda x . ((h @_2 (\lambda q . x)) @_3 a))) @_4 (z @_5 a)) @_6 (\lambda f \lambda y . f @_7 ((g @_8 (\lambda h . h)) @_9 y))) @_{10} (g @_{11} (\lambda n . n))$$

Для наглядности в нашем примере термы будут представляться частью их синтаксического

дерева. Синтаксическое дерево терма $\langle N P R \rangle$ может быть найдено на рис. 9.

Система переходов HLR для терма $\langle N P R \rangle$





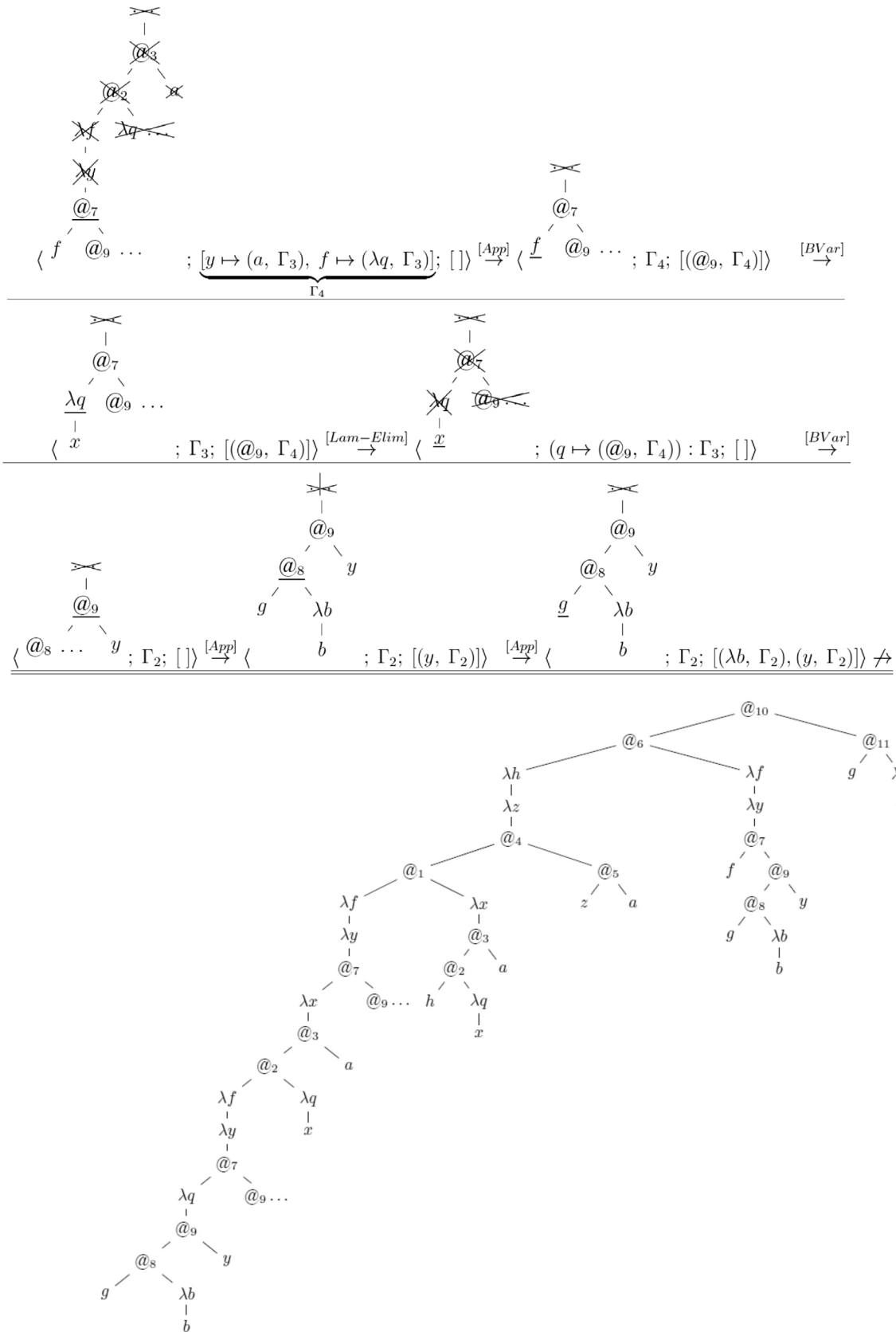
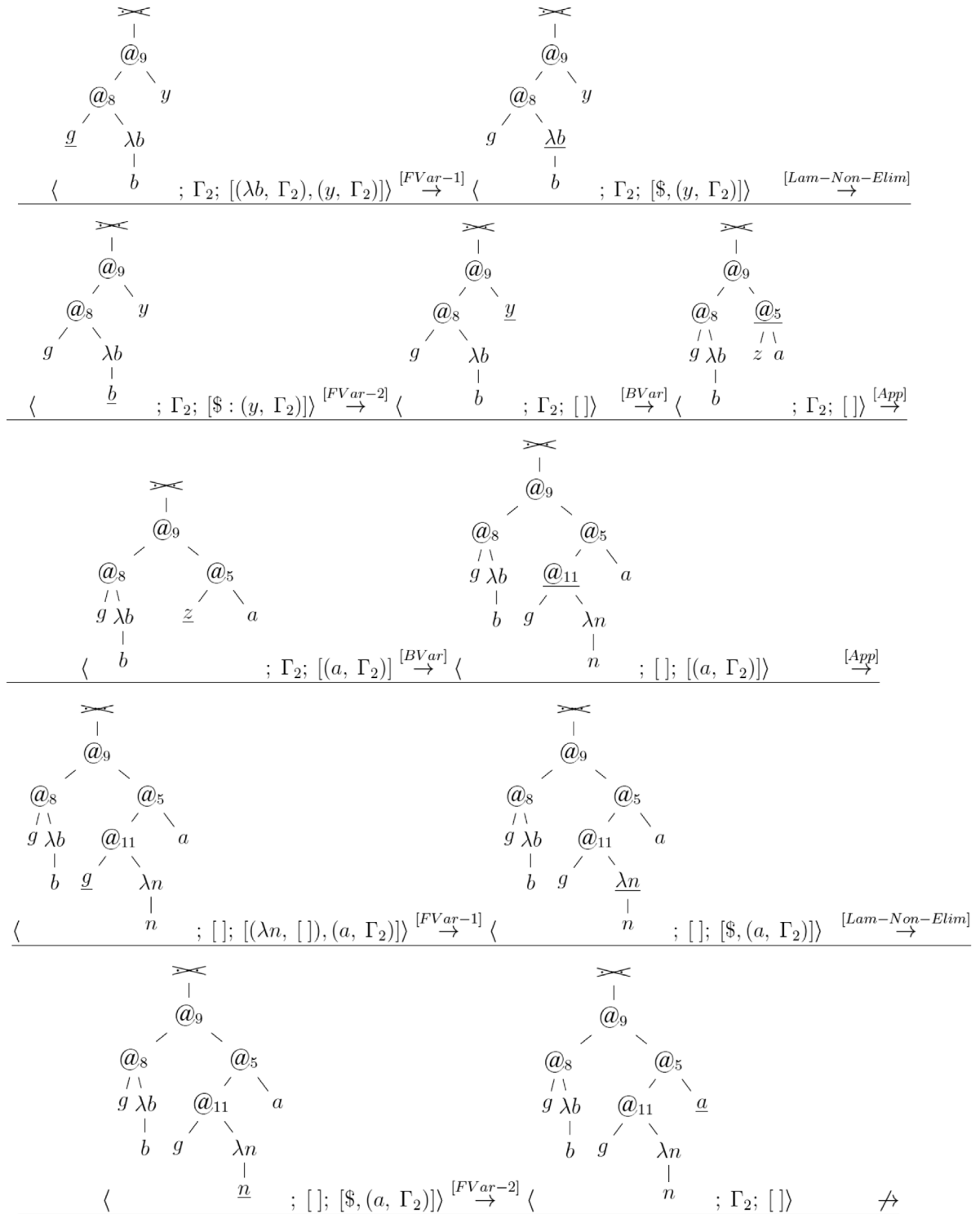


Рис. 10. Псевдоголовная нормальная форма (qhn) термина NPR

Замечание. Результатом головной линейной редукции является первый компонент конечного состояния системы переходов (NB: вычеркнутые элементы

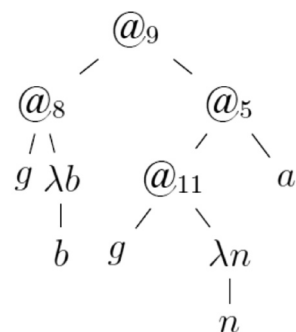
являются частью qhn). Таким образом, результатом головной линейной редукции является терм, представленный на рис. 10.

Система переходов для CHLR для терма $\langle NPR \rangle$



Заметим, что вплоть до первого применения одного из правил [FVar-*] система переходов для полной головной линейной редукции в точности повторяет шаги системы переходов для головной линейной редукции, поэтому мы просто продолжим построение нашего примера, начиная с конечного состояния системы переходов для головной линейной редукции.

Напомним, что результатом полной головной линейной редукции является первый компонент конечного состояния системы переходов, в котором все вычеркнутые вершины удалены:



СПИСОК ЛИТЕРАТУРЫ

1. Ong C.-H. Luke Normalisation by Traversals. CoRR, abs/1511.02629 // URL: <http://arxiv.org/abs/1511.02629>, 2015.
2. de Bruijn N.G. Lambda Calculus Notation with Nameless Dummies: A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem // *Indagationes Mathematicae*. Elsevier, 1972. No. 34. Pp. 381–392.
3. Danos V., Regnier L. Head linear reduction. Unpublished, 2004.
4. Danos V., Herbelin H., Regnier L. Game semantics and abstract machines // Proc. of the 11th Annual IEEE Symp. on Logic in Computer Science. IEEE Computer Society, Washington, USA, 1996. 394 p.
5. Barendregt H.P. The lambda calculus: its syntax and semantics. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam, New-York, Oxford, 1981.
6. Berezun D., Jones N.D. Compiling untyped lambda calculus to lower-level code by game semantics and partial evaluation (invited paper) // Proc. of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation. ACM, NY, USA, 2017. Pp. 1–11.
7. Sestoft P. Demonstrating lambda calculus re-

Заключение

Головная линейная редукция является основой для различных подходов к вычислениям [16, 20–25], в том числе и игровой семантики, из которой естественным образом вытекает процедура нормализации термов простого типизированного лямбда-исчисления [1]. Обобщение этого подхода до бестипового лямбда-исчисления позволяет определить новый подход к вычислениям – трассирующую нормализацию [1, 6].

В статье приведена формализация понятия головной линейной редукции для нетипизированного лямбда-исчисления в виде системы переходов, формально показана её корректность относительно головной редукции, впервые введено понятие полной головной линейной редукции, являющейся истинным обобщением головной линейной редукции, формализованное в виде системы переходов, а также показано, что последняя является эффективной редуцирующей стратегией, что является первым шагом в формальном доказательстве корректности трассирующей нормализации.

duction // *The Essence of Computation*. Springer-Verlag New York, Inc., 2002. Pp. 420–435.

8. Plotkin G.D. LCF considered as a programming language // *Theory of Computer Science*. 1977. No. 5(3). Pp. 223–255.
9. Keller R.M. Formal Verification of Parallel Programs // *Communications of the ACM*. 1976. Vol. 19. No. 7. Pp. 371–384.
10. Pierce B.C. Types and Programming Languages. The MIT Press, 2002.
11. Pierce B.C. Advanced Topics in Types and Programming Languages. The MIT Press, 2004.
12. Blum W. The safe lambda calculus. University of Oxford, UK, 2009.
13. Wadsworth Ch.P. Semantics and Pragmatics of the Lambda-Calculus. Oxford University, 1971.
14. François-Régis Sinot. Complete laziness: a natural semantics // *Electr. Notes Theor. Comput. Sci.* 2008. No. 204. Pp. 129–145.
15. Launchbury J. A natural semantics for lazy evaluation // In Conf. Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 1993. Pp. 144–154.
16. Jean-Jacques Levy. Optimal reductions in the lambda-calculus // *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formal-*



ism. Academic Press, 1980.

17. **Lamping J.** An algorithm for optimal lambda calculus reduction // Proc. of the 17th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. ACM, 1990. Pp. 16–30.

18. **Shivers O., Wand M.** Bottom-up beta-reduction: Uplinks and lambda-dags // Programming Languages and Systems. 14th European Symp. on Programming. 2005. Pp. 217–232.

19. **Kees-Lan can de Looij Vincent van Oostrom, Mariln Zwitterlood.** Lambdascope another optimal implementation of the lambda-calculus // Workshop on Algebra and Logic on Programming Systems. 2004.

20. **Mascari G., Pedicini M.** Head Linear Reduction and Pure Proof Net Extraction // Theor. Comput. Sci. 1994. Vol. 135. No. 1. Pp. 111–137.

21. **Baillot P., Pedicini M.** Elementary Complexity and Geometry of Interaction // Fundam. Inf. 2001. Vol. 45. No. 1-2. Pp. 1–31.

22. **Abramsky S., McCusker G.** Game semantics. In Computational Logic // Proc. of the 1997

Marktoberdorf Summer School. Springer Verlag, 1999. Pp. 1–56.

23. **Hyland J.M.E., Luke Ong C.-H.** On full abstraction for PCF: I, II, and III // Inf. Comput. 2000. Vol. 163(2). Pp. 285–408.

24. **Ker A.D., Nickau H., Ong C.-H. Luke** Innocent game models of untyped lambda-calculus // Theor. Comput. Sci. 2002. Vol. 272(1-2). Pp. 247–292.

25. **Blum W., Ong C.-H. Luke.** A concrete presentation of game semantics // Galop 2008: Games for Logic and Programming Languages. 2008.

26. **Abramsky S., Ong C.-H. Luke.** Full abstraction in the lazy lambda calculus // Inf. Comput. 1993. Vol. 105(2). Pp. 159–267.

27. **Ker A.D., Nickau H., Ong C.-H. Luke** A Universal Innocent Game Model for the Böhm Tree Lambda Theory // Computer Science Logic, 13th Internat. Workshop, 8th Annual Conf. of the EACSL. Madrid, Spain, 1999. Pp. 405–419.

28. **Gérard Huet.** Regular Böhm Trees // Math. Struct. in Comp. Science. 1998. No. 8. Pp. 671–680.

Статья поступила в редакцию 22.06.2017

REFERENCES

1. **Ong C.-H. Luke** Normalisation by Traversals. *CoRR*, abs/1511.02629. Available: <http://arxiv.org/abs/1511.02629>, 2015.

2. **de Bruijn N.G.** Lambda Calculus Notation with Nameless Dummies: A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indagationes Mathematicae*. Elsevier, 1972, No. 34, Pp. 381–392.

3. **Danos V., Regnier L.** Head linear reduction. Unpublished, 2004.

4. **Danos V., Herbelin H., Regnier L.** Game semantics and abstract machines. *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society, Washington, DC, USA, 1996, 394 p.

5. **Barendregt H.P.** *The lambda calculus: its syntax and semantics*. Studies in logic and the foundations of mathematics. North-Holland, Amsterdam, New-York, Oxford, 1981.

6. **Berezun D., Jones N.D.** Compiling untyped lambda calculus to lower-level code by game semantics and partial evaluation (invited paper). *Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*. ACM, NY, USA, 2017, Pp. 1-11.

7. **Sestoft P.** *Demonstrating lambda calculus reduction. The Essence of Computation*. Springer-Verlag New York, Inc., 2002, Pp. 420–435.

8. **Plotkin G.D.** LCF considered as a programming language. *Theory of Computer Science*, 1977,

No. 5(3), Pp. 223–255.

9. **Keller R.M.** Formal Verification of Parallel Programs. *Communications of the ACM*, 1976, Vol. 19, No. 7, Pp. 371–384.

10. **Pierce B.C.** *Types and Programming Languages*. The MIT Press, 2002.

11. **Pierce B.C.** *Advanced Topics in Types and Programming Languages*. The MIT Press, 2004.

12. **Blum W.** *The safe lambda calculus*. University of Oxford, UK, 2009.

13. **Wadsworth Ch.P.** *Semantics and Pragmatics of the Lambda-Calculus*. Oxford University, 1971.

14. **François-Régis Sinot.** Complete laziness: a natural semantics. *Electr. Notes Theor. Comput. Sci.*, 2008, No. 204, Pp. 129–145.

15. **Launchbury J.** A natural semantics for lazy evaluation. *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1993, Pp. 144–154.

16. **Jean-Jacques Levy.** Optimal reductions in the lambda-calculus. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

17. **Lamping J.** An algorithm for optimal lambda calculus reduction. *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 1990, Pp. 16–30.

18. **Shivers O., Wand M.** Bottom-up beta-reduction: Uplinks and lambda-dags. *Programming Languages and Systems, 14th European Symposium*

on Programming, 2005, Pp. 217–232.

19. **Kees-Lan van de Looij Vincent van Oostrom, Mariln Zwitterlood.** Lambdascope another optimal implementation of the lambda-calculus. *Workshop on Algebra and Logic on Programming Systems*, 2004.

20. **Mascari G., Pedicini M.** Head Linear Reduction and Pure Proof Net Extraction. *Theor. Comput. Sci.*, 1994, Vol. 135, No. 1, Pp. 111–137.

21. **Baillet P., Pedicini M.** Elementary Complexity and Geometry of Interaction. *Fundam. Inf.*, 2001, Vol. 45, No. 1-2, Pp. 1–31.

22. **Abramsky S., McCusker G.** Game semantics. In Computational Logic. *Proceedings of the 1997 Marktoberdorf Summer School*. Springer Verlag, 1999. Pp. 1–56.

23. **Hyland J.M.E., Luke Ong C.-H.** On full abstraction for PCF: I, II, and III. *Inf. Comput.*, 2000,

Vol. 163(2), Pp. 285–408.

24. **Ker A.D., Nickau H., Ong C.-H. Luke** Innocent game models of untyped lambda-calculus. *Theor. Comput. Sci.*, 2002, Vol. 272(1-2), Pp. 247–292.

25. **Blum W., Ong C.-H. Luke.** A concrete presentation of game semantics. *Galop 2008: Games for Logic and Programming Languages*, 2008.

26. **Abramsky S., Ong C.-H. Luke.** Full abstraction in the lazy lambda calculus. *Inf. Comput.*, 1993, Vol. 105(2), Pp. 159–267.

27. **Ker A.D., Nickau H., Ong C.-H. Luke** A Universal Innocent Game Model for the Böhm Tree Lambda Theory. *Computer Science Logic, 13th International Workshop, 8th Annual Conference of the EACSL*, Madrid, Spain, 1999, Pp. 405–419.

28. **Gérard Huet.** Regular Böhm Trees. *Math. Struct. in Comp. Science*, 1998, No. 8, Pp. 671–680.

Received 22.06.2017

СВЕДЕНИЯ ОБ АВТОРАХ / THE AUTHORS

Березун Даниил Андреевич

Berezun Daniil A.

E-mail: d.berezun@2009.spbu.ru