数理解析研究所講究録
第731巻 1990年 13-24

13

A new head-normalization algorithm for λ-calculus

野口憲一（Ken-ichi Noguchi）
東京工業大学　理学部　情報科学科

本稿で述べるＬＳアルゴリズムは、主に次の２つを行うアルゴリズムである。（1）λ式のhead-normal-formの概形を高速に求める。（2）λ式のhead-needed-redexをもれなく見つける。ここで言うhead-normal-formの概形とは、$\lambda x_1 x_2 \ldots x_n.\ y\square_1\square_2\ldots\square_m$、つまりhead-variableに続く部分式の中身は明示せず、その個数だけを示した形である。またhead-needed-redexとは、head-normal-formへ到達するために、いつかは必ずreductionしなければならないredexである。ここでは、ＬＳアルゴリズムの考え方、その性質、及び実行効率の計算機実験による評価について述べる。

A new head-normalization algorithm for λ-calculus

Ken-ichi Noguchi
Department of Information Science
Tokyo Institute of Technology
Ookayama, Meguro-Ku, Tokyo 152, Japan

In this paper, we present an algorithm named LS-algorithm, which do the following two things:(1)it finds the outline of the head-normal-form of the given λ-term quickly. (2)it finds all the head-needed-redexes of the given λ-term. By the outline of the head-norm-form we mean $\lambda x_1 \ldots x_n.\ y\square_1\ldots\square_m$, the significant half up to the head-variable and the number of succeeding subterms. The head-needed-redex is a redex which is necessarily contracted in anyreduction to its head-normal-form. We describe the idea of LS-algorithm and itsproperties, and evaluate its computational efficiency by computer experiment.

14

１．はじめに

λ-caluculusは、関数の概念を抽象化した一種の論理体系である。ＬＩＳＰなどのいわゆる関数型言語のプログラムは、そこで使われる関数をλ式の定数と見なせば一種のλ式と考えることができる。従って、λ式をリダクションするということは、関数型言語の世界では、プログラムの評価、実行を行なうことに対応する。

本稿は、このλ-calculusのリダクションに関する研究である。ここでは、λ-calculusのβリダクションを、通常とは異なる手法で実行するアルゴリズムのアイディアを示し、その性質を述べ、さらにこのアルゴリズムの計算機実験による、実行効率の評価に関して述べる。

ここで述べるアルゴリズムは、ＬＳ（Left-Slope）アルゴリズムと言う。このＬＳアルゴリズムは、与えられたλ式Mに対して、主に次の２つを行なうものである。

１．Mのhead-normal-formの概形を高速に求める。（図１．１参照）

２．Mのhead-needed-redexを高速にもれなく見つける。

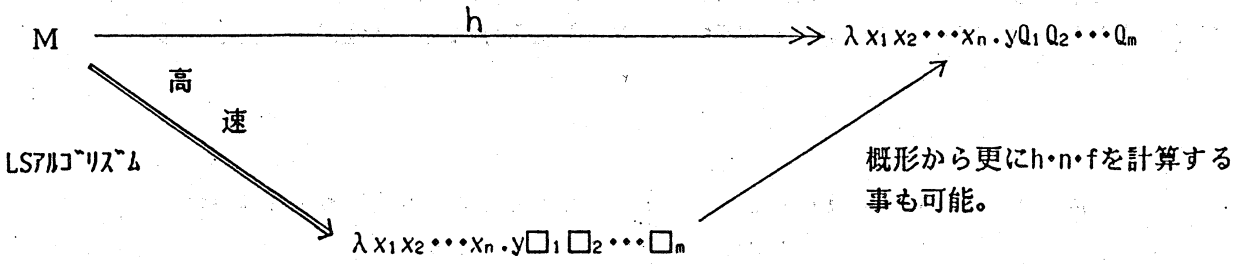


図１．１　ＬＳアルゴリズムのあらまし

head-normal-form の概形とは head-normal-form　$\lambda x_1 x_2 \ldots x_n .\ y Q_1 Q_2 \ldots Q_m$　に対し　$\lambda x_1 x_2 \ldots x_n .\ y \square_1 \square_2 \ldots \square_m$　つまり、head-variableに続くλ部分式 Q_1, Q_2, ... , Q_m に関しては、その中身を明示せず、その個数のみを示すものである。ＬＳアルゴリズムは、より正確に言うと、この概形とこれらの部分式 Q_1, Q_2, ･･･, Q_m を求めるための情報を計算する。従って、ＬＳアルゴリズムは、head-normal-formの概形を高速に求めることをまず先に行なうhead-normalizationアルゴリズムということになる。概形が高速に求まることによって、例えばあるλ式がhead-normal-formを持つか否かを知りたいとき、もし持つ場合には、head-normal-formを持つという情報を早く知ることが出来、また持たない場合でもＬＳアルゴリズムの実行過程から,リダクションに関する有用な情報や予想が読み取れる場合が多い（例えば、リダクションに関する周期性など）。従って、理論的にはhead-normal-formを持つか否かという問題は決定不能であるが、現実的な側面を考えると、あるλ式がhead-normal-formを持つか否かをチェックするための、有効なアルゴリズムであるといえる。

head-needed-redexとは、簡単に言うと、head-normal-formに到達するために、いつかは必ずリダクションしなければならないredexである。関数型プログラミングの世界では、strictness-analysisにおけるstrictな引数に対応するものである（文献［１］）。このhead-needed-redexをいくつか、高速に見つけることにより、最左最外戦略と異なる、効率の良い正規化戦略が実現可能となる。

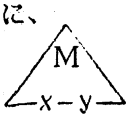
2．λグラフ

λ式のλグラフ表現は、ＬＳアルゴリズムの説明に必要不可欠なものである。λグラフは、λ式のabstractionをunary-operator（abstraction-node）、applicationをbinary-operator（application-node）として、木で表現したものである。（図２．１参照）

λグラフに関して、ここでは次の記法を用いる。
最左変数・・・λグラフの最左斜面の一番下にある変数（λ式の最左にある変数と同じ）。

・・・λ式Mのλグラフを表す。

特に、



・・・変数が明示してある場合、その変数ｘ，ｙは  において自由に現れている変数とする。

M≡λxy.(λz.zz)x((λy.y)y) の場合

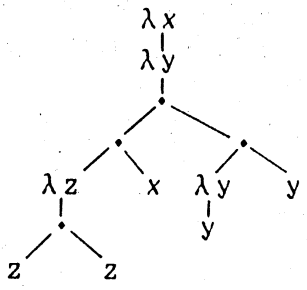


図２．１　λグラフの例

λグラフにおいてhead-redex、head-normal-formは次のように表される。

∘ head-redex

$\lambda x_1 x_2 \ldots x_n .\ \underline{((\lambda y.P)Q)}Q_1 Q_2 \ldots Q_m$

λグラフでは、最左斜面上にあるredexで一番上方にあるredexである。（図２．２）

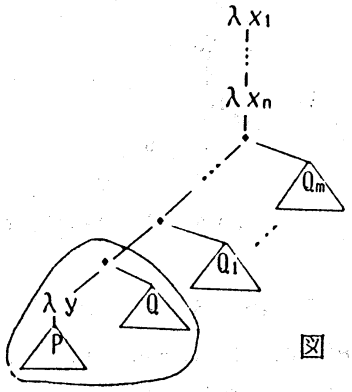


図　２．２

∘ head-normal-form

$\lambda x_1 x_2 \ldots x_n .\ yQ_1 Q_2 \ldots Q_m$

最左斜面上にredexの無いλグラフとなる（図２．３）。
また、
head-normal-formの概形$\lambda x_1 x_2 \ldots x_n .\ y\square_1 \square_2 \ldots \square_m$は、最左斜面のnodeの列で表わされる。（図２．３の枠で囲まれた部分）

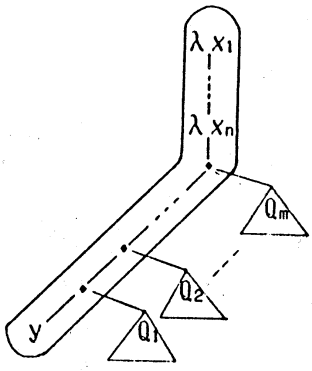


図　２．３

3．Ｌｅｆｔ－Ｓｌｏｐｅ

定義（λ式のleft-slope）

任意のλ式 M に対して、Mのleft-slopeは次のｉ）、ｉｉ）より定義される。（図３．１参照）

ｉ）Mのλグラフの最左斜面（を構成するnodeの列）は、Mのleft-slopeである。

ｉｉ）Mのλグラフの各application-nodeの、右側の部分木の最左斜面（を構成するnodeの列）は、Mのleft-slopeである。　□

図３．１の例では、実線部のnodeの列がleft-slopeであり、計８つのleft-slopeがある。最小のleft-slopeは１個の変数だけからなるものである。

定義（λ式のleft-slope-list）

left-slopeの定義から、最左のleft-slopeを除くすべてのleft-slopeは、その親のapplication-nodeと一対一に対応していることがわかる。（図３．１参照）　そこで各application-nodeに互いに異なる正整数を割当て、各left-slopeに、

- 最左のleft-slope・・・０番
- その他のleft-slope・・・親のapplication-nodeの番号

と名前を付け、抜きだして表にしたものをleft-slope-listと言う。（図３．２）□

例）M≡λx.(λzy.(λxy.x(yv))y)((λw.w)w)((λxv.v)x)の場合、

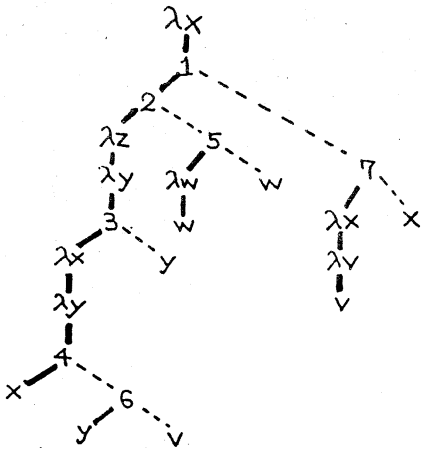


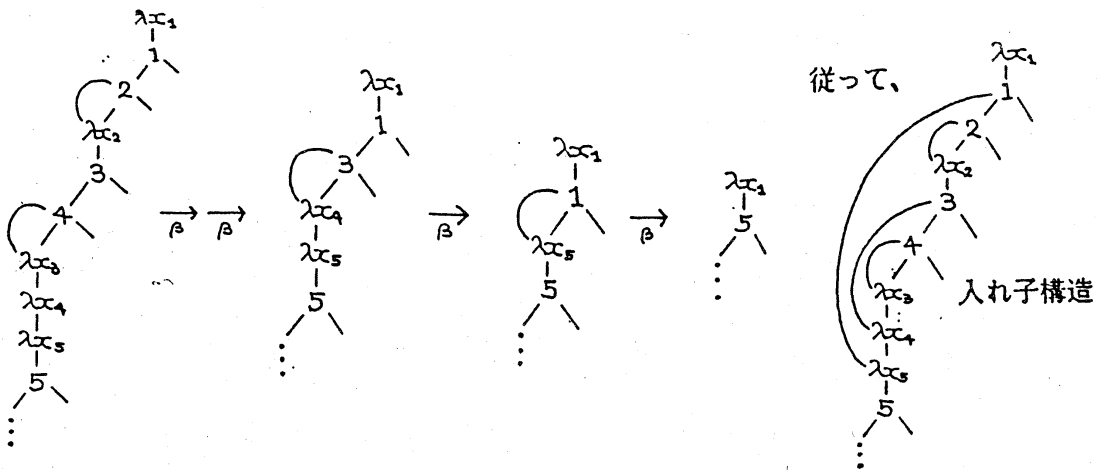
図３．１　Mのleft-slope

0	1	2	3	4	5	6	7
λx	7	5	y	6	w	v	x
1	λx	λw		y			
2	λv	w					
λz	v						
λy							
3							
λx							
λy							
4							
x							

図３．２　Mのleft-slope-list

４．Ｒｅｄｅｘペア

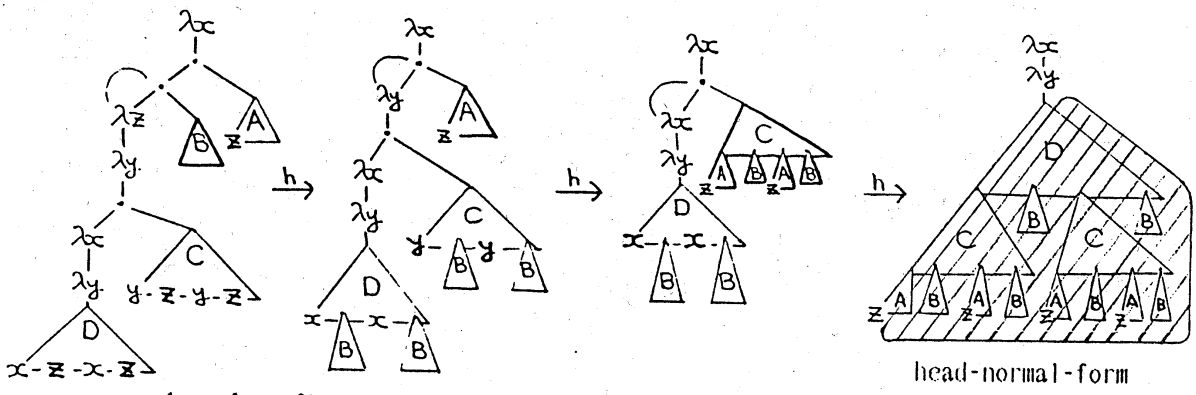
λ式のleft-slope上のnodeに対して、現在redexを構成しているか、または、あるreductionを行うことにより将来redexを構成しうるapplication-nodeとabstraction-nodeの対をredexペアと言う。
例えば、



以上の例でわかる様に、redexペアは元のleft-slopeにおいて、入れ子の関係にある。また、逆に入れ子の関係にある、application-nodeとabstraction-nodeの対はredexペアである。

５．ＬＳアルゴリズムのアイデア

ＬＳアルゴリズムのアイデアを通常のhead-reductionと比較をし、説明を試みる。図５．１は、左にあるλグラフをhead-reductionして、そのhead-normal-formへ到達させたものを表わしている。



但し、Ⓓ Ⓒ Ⓐ の最左斜面には、abstraction-nodeが無いものとする。

図５．１　head-reductionの例

もし、head-normal-formの概形だけを求めたいのなら、head-normal-formの最左斜面のnodeだけがわかれば良いので、図５．１の、head-normal-formの斜線部分への代入は無駄である。Ⓐ Ⓑ Ⓒ Ⓓ が大きなλグラフで、更にα変換が必要であるような場合は、斜線部への代入は非常に大きなタイムロスとなることは容易に理解できよう。そこで、ＬＳアルゴリズムは、head-reductionのようにまずhead-redexを探しそれをreductionするというようなことはしないで、各ステップにおいて 、まずλグラフの最左変数に注目する。そして、head-reductionにより将来この最左変数に代入されるλ部分グラフを見つけ、それを代入するということを行う。

最左変数に代入される部分式を見つけるためには、次の２つの情報が必要となる。

１．最左変数を束縛するabstraction-node

２．１で見つかったabstraction-nodeと将来redexを構成するapplication-node、つまりredexペアの関係にあるapplication-node

また、もし、１のabstraction-node、あるいは、２のapplication-nodeが無いときは、最左変数にもはや何も代入され得ないということになる。

ＬＳアルゴリズムでは、**バイパスを通す**というアイデアを用いることにより、α変換の様な、やっかいな変数変換を行わずに、最左変数を束縛するabstraction-nodeを探すことを可能にしている。

それでは、図５．１の例について、具体的にこのアイデアを説明する。図５．２を見て頂きたい。

図５．２の解説：

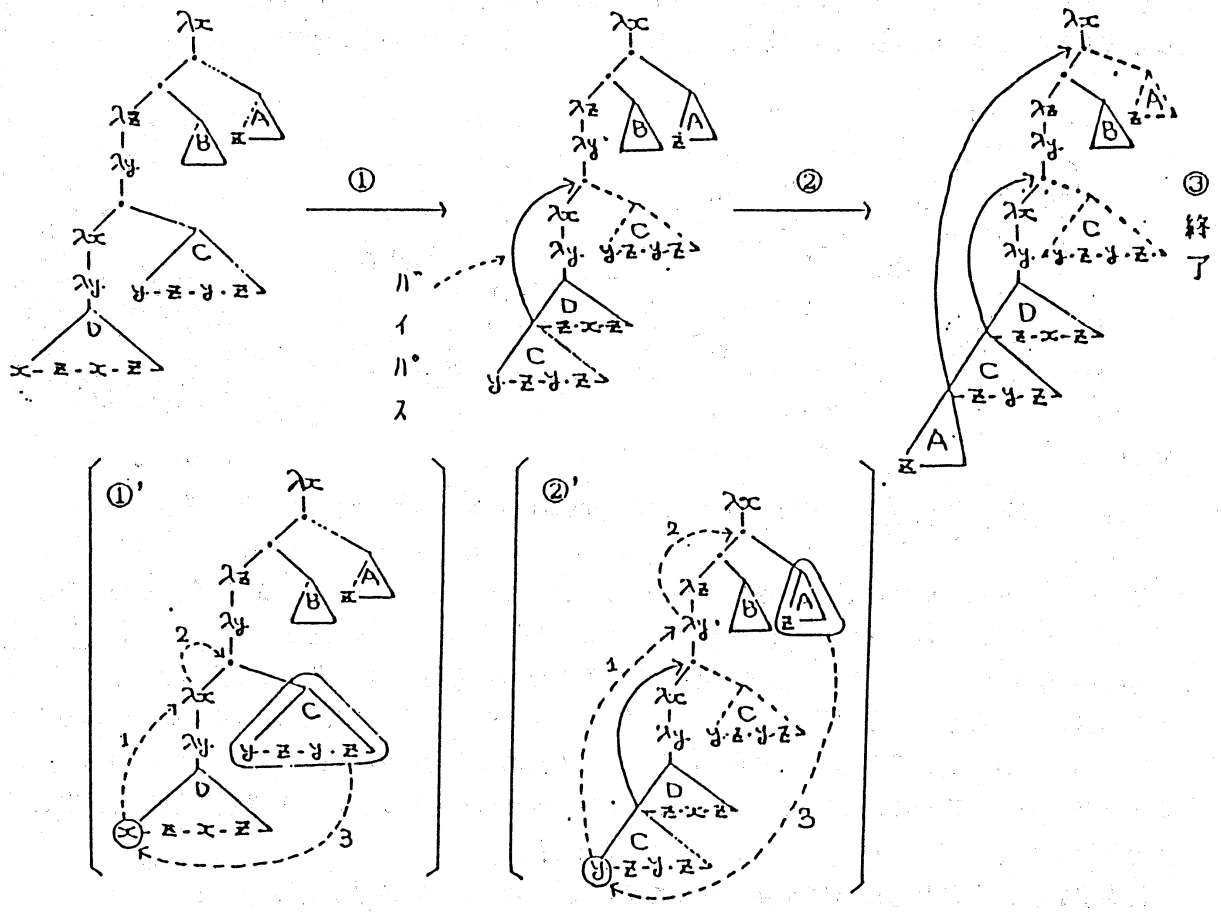
①

１．まず、最左変数ｘを束縛するabstraction-nodeを、最左変数から、λグラフの根に向かって探す。最初に見つかるλｘがｘを束縛するabstraction-nodeである。（①’ $\overset{1}{\dashrightarrow}$ ）

２．束縛するabstraction-nodeが見つかったら、そのabstraction-nodeとredexペアとなるapplication-nodeを探す。（①’ $\overset{2}{\rightarrow}$）

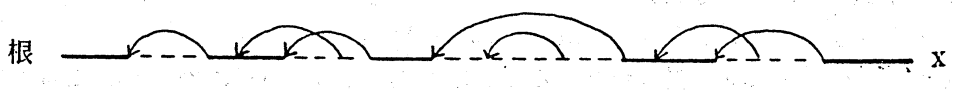
３．最左変数に、２で見つけたapplication-nodeの右側の部分木を代入する。（①’ $\overset{3}{\rightarrow}$）

図５．２　ＬＳアルゴリズムのアイデア



４．３で代入した部分木の根から、その部分木の親であったapplication-nodeへバイパスを通す。
②
１．最左変数ｙを束縛するabstraction-nodeを、λグラフの根へ向かって、バイパスを経由して探す。最初に見つかるλｙが、最左変数ｙを束縛するabstraction-nodeである。（②'- -┗→）　　（注）
２．３．４．・・・①の２、３、４と同様。（②'参照）
③最左変数ｚを束縛するabstraction-nodeをバイパスを経由して、根に向かって探すが見つからない。従って、もはや最左変数ｚには何も代入されないので終了。

（注）バイパスを経由して探す･･･もしあるnodeからバイパスが出ている時は、そのバイパスを経由して（すなわち途中のnodeを飛び越して）バイパスの終点から再び探し始めることを行う。例えば、次の様に、バイパスが通っている場合、実線部のnode上を探す。



以上のアイデアで、ＬＳアルゴリズムは実行される。ここで、このアイディアで出来た、図５．２の最終結果のλグラフの、最左斜面上のnodeで、redexペアとなっているnodeをすべて取り除いてみ

ると、図５．３の様になる。ここで、取り除いた結果の最左斜面のnodeの列（図５．４の枠内）は、図５．１のhead-normal-formを見て頂ければわかる様に、head-normal-formの概形となるのである。

しかしながら、図５．４の斜線部分は、head-normal-formの概形だけを得るのには、まだ不必要な部分である。そこで、ＬＳアルゴリズムは、以上のアイディアをλグラフのleft-slopeを単位に効率良く実行していく。

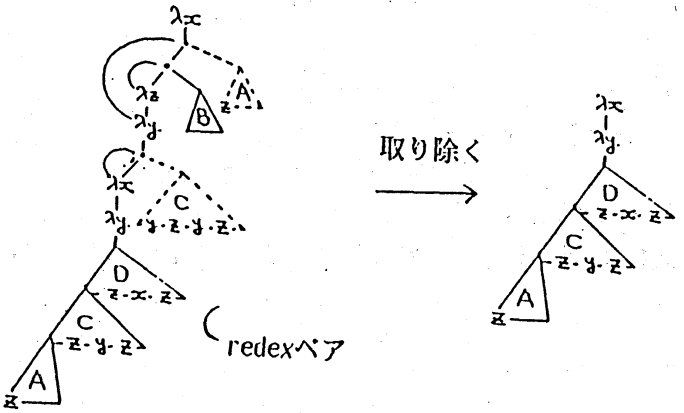


図５．３

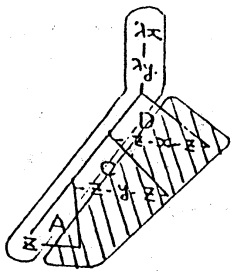


図５．４

６．ＬＳアルゴリズム

ＬＳアルゴリズムは、実際には、“５ ＬＳアルゴリズムのアイデア”で説明した様にλグラフを変形していくのではなく、λ式のleft-slope-listだけを参照して、ＬＳ-tableというものを作成していく。従って、ＬＳアルゴリズムは、次の２つのルーチンからなる。

（１）文字列入力のλ式に対して、left-slope-listを作成する。

（２）left-slope-listを参照しながら、ＬＳ-tableを作成する。

ＬＳ-tableは、図５.２のバイパスつきλグラフの、最左斜面の部分に丁度対応するものである。具体的には、図３．１のλ式の場合、ＬＳ-tableは図６．１の様になる。

M≡λx.(λzy.(λxy.x(yv))y)((λw.w)w)((λxv.v)x)　（λグラフは図３．１参照）

0	1	2	3	4	5	6	7
λx	7	5	y	6	w	v	x
1	λx	λw		y			
2	λv	w					
λz	v						
λy							
3							
λx							
λy							
4							
x							

Mのleft-slope-list

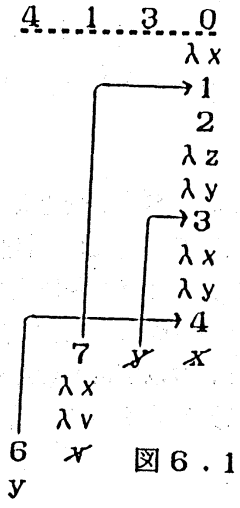


図６．１

但し、

・ＬＳ-tableの最上段の数字の列は、その下にあるleft-slopeの番号の列を表わしている。
・“／”が引かれている変数は、代入によりなくなった変数を表している。

ＬＳアルゴリズムは次のようになる。

ＬＳアルゴリズム：

入力：λ式　Ｍ　　出力：ＭのＬＳ-table

ＳＴＥＰ１）λ式Ｍの left-slope-listを求める。
ＳＴＥＰ２）ＳＴＥＰ１で作成されたleft-slope-listを参照して、次のようにＬＳ-tableを作成する。

ＬＳ-tableの作成：

（０）０番left-slopeをＬＳ-tableに載せる。
（１）ＬＳ-tableの最左変数（注１）を束縛するabstraction-nodeをバイパスを経由して探す。
その時、（注２）
（ａ）束縛するabstraction-nodeが見つかる⟹（２）へ
（ｂ）束縛するabstraction-nodeが見つからない⟹終了（作成されたＬＳ-tableを出力）
（２）（１）で見つけたabstraction-nodeとredexペアとなるapplication-nodeを探す。
その時、（注３）
（ａ）redexペアとなるapplication-nodeが見つかる⟹（３）へ
（ｂ）redexペアとなるapplication-nodeが見つからない⟹終了（作成されたＬＳ-tableを出力）
（３）（２）で見つけたapplication-nodeの番号のleft-slopeを、left-slope-listから抜き出し、ＬＳ-tableの最左変数に代入する（注４）。さらに、代入したleft-slopeの頭から、（２）で見つけたapplication-nodeへバイパスを通す。
（１）へ戻る。

□ {ＬＳアルゴリズム}

注１）ＬＳ-tableの最左変数・・・ＬＳ-tableにおいて最左にあるleft-slopeの（一番下にある）変数（図６.１では、４番left-slopeの変数“ｙ”）
注２）バイパスを経由して探す・・・§１の説明と同様
注３）redexペアとなるapplication-nodeを探す・・・ＬＳ-table上の各left-slopeを、変数を取り除き、つなぎ合わせたものを一つのnodeの列と見なしredexペアを探す。
注４）代入する際、代入するleft-slopeの番号を上段に書く。

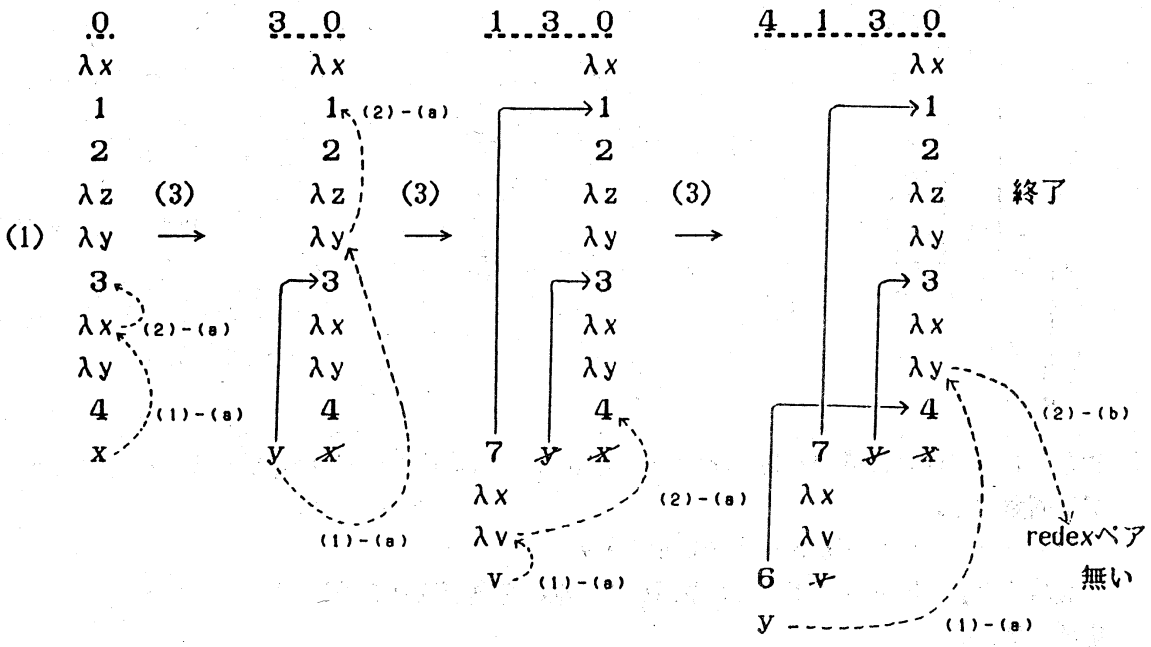
ＬＳアルゴリズムが終了するのは、（１）-（ｂ）、または、（２）-（ｂ）の条件が成り立つ場合、つまり、ＬＳ-tableの最左にあるleft-slopeの変数に、もはや、何も代入され得ない時に終了する。それでは、ＬＳアルゴリズムが、実際にどの様に実行されるか、図６．１の場合の実行例を示す。

実行例：

入力：M≡λx.(λzy.(λxy.x(yv))y)((λw.w)w)((λxv.v)x)

ＳＴＥＰ１）図６．１のleft-slope-listを作成する。

ＳＴＥＰ２）



head-normal-formを持たないλ式　（λｘ．ｘｘ）（λｘ．ｘｘ）　に対して、ＬＳアルゴリズムを実行すると図６．２のようになる。ＬＳアルゴリズムは、head-normal-formを持たないλ式に対しては止まらないアルゴリズムである。

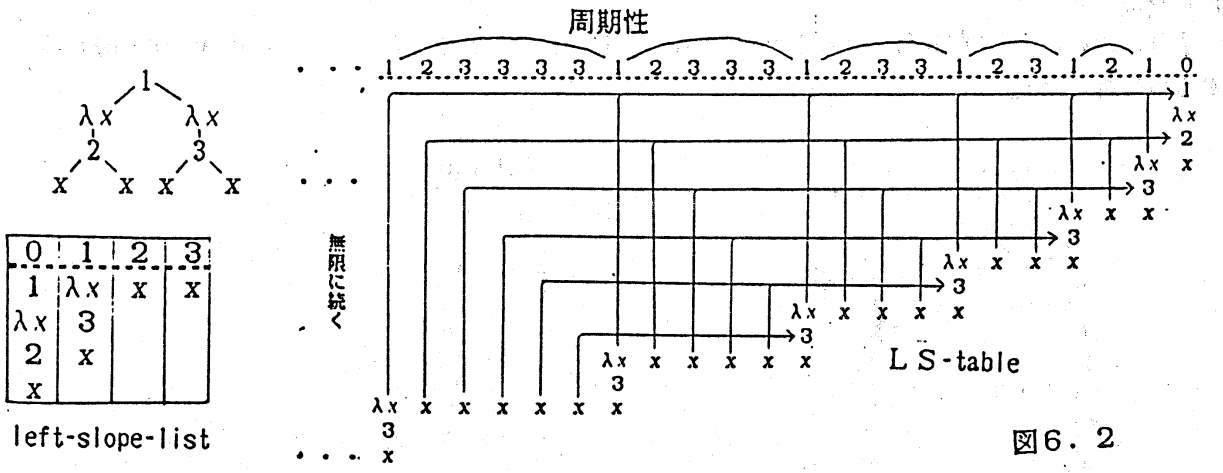


図６．２

７．ＬＳアルゴリズムによって得られる情報

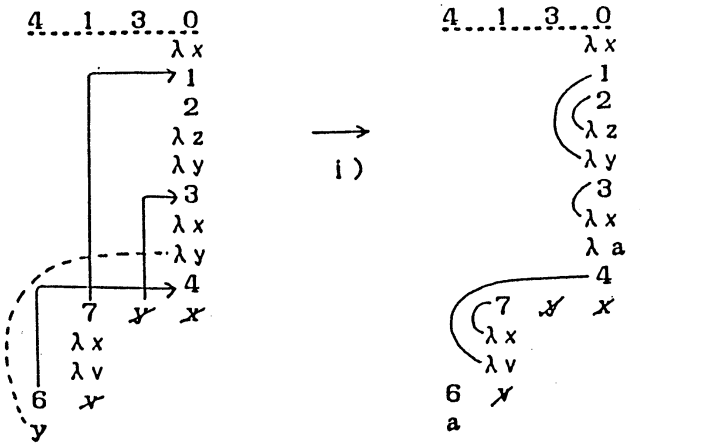
（１）ｈｅａｄ－ｎｏｒｍａｌ－ｆｏｒｍの概形

“５．ＬＳアルゴリズムのアイディア”で説明したように（図５．４）、ＬＳアルゴリズムで作られたＬＳ-tableから、次のｉ），ｉｉ）を順に行う事により求めることが出来る。（図７．１参照）

ｉ）ＬＳ-tableの最左変数と、それを束縛するabstraction-node（バイパス経由で見つけたもの）の変数を、新しい変数に換える（図７．１では、ａ）。

ｉｉ）redexペアとなっているapplication-nodeとabstraction-nodeをすべて取り除く。

出来上がったnodeの列が、head-normal-formの概形を表す。図６．１の例に対しては、図７．１のようになる。



実際、この場合
$M \xrightarrow{h} \lambda x.\lambda y.yv$となるので
一致する事がわかる。　　図７．

また、ＬＳ-tableの情報とleft-slope-listから、head-normal-formも求めることが出来るが、ここでは省略する。

（２）ｈｅａｄ－ｎｅｅｄｅｄ－ｒｅｄｅｘ

文献［１］ Theorem 3.6-(ii) を用いて、次の定理が成り立つ。

定理　７．１

Ｒは、λ式 Ｍ の head-needed-redex である
⟺　Ｒは、ＭのＬＳ-tableに現れるleft-slope上のredexである

証）略□

つまり、図６．１の例の場合、head-needed-redexは、図７．２のλグラフで、abstraction-nodeに＊の付いたredexとなる。

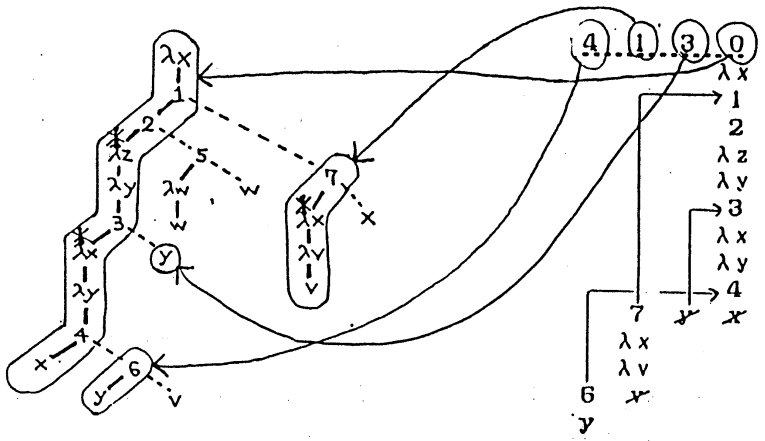


図７．２

８．ＬＳアルゴリズムの実行効率に関して

一般のグラフリダクションと比較して、以下のようなことが言える。

ＬＳアルゴリズムの特徴：

１．グラフリダクションにとって、必要不可欠で厄介なα変換に対応することはやらない。

２．head-normal-formの概形を得るのに不必要な部分はいじらない。

ＬＳアルゴリズムのインプリメントに関して：

ＬＳアルゴリズムは、（１）left-slope-listを作るルーチン、（２）ＬＳ-tableを作るルーチン、の２つからなるが、それぞれに対して

（１）left-slope-listを作るということは、基本的には構文解析を行うことである。従って、λグラフを作るのと同程度の実行効率のインプリメントが出来る。

（２）ＬＳ-tableは、left-slopeの番号の列と、バイパスの情報をうまくコード化したもので表現できる。従って、ＬＳ-tableをコードで表現することにより、いちいちleft-slope-listからleft-slopeを抜き出し、コピーするということが必要ないため、高速にＬＳ-tableを作成することが出来る。

ＬＳアルゴリズムのオーダーに関して：

このＬＳアルゴリズムのオーダーは定まらない、その理由は

１．head-normal-formを持たないλ式に対しては止まらない

２．アルゴリズムが止まる場合、つまりhead-normal-formを持つλ式だけに限ってもオーダーは定まらない。それは、もしオーダーが定まってしまうと、head-normal-formを持つか否かという問題が決定可能となってしまい、有名なＳＣＯＴＴのUndecidability-Theorem（文献［３］、Theorem 5.6）に矛盾してしまうからである。

計算機実験に関して：

ＬＳアルゴリズムは、オーダーが定まらないアルゴリズムであるが、現実的には、graph-head-reductionと比較すると、かなり効率の良いアルゴリズムであることが計算機実験により、確かめられた。いくつかのλ式に対して、次のＡ、Ｂ２つの実験を行った。

Ａ．graph-head-reductionにより、head-normal-formを求める。

Ｂ．ＬＳアルゴリズムにより、ＬＳ-tableを作成する。

λグラフとleft-slope-listは、ほぼ同時間で作成出来るので、Ａ，Ｂからそれぞれ、λグラフ、left-slope-listをつくる時間を、除いた実行時間を調べたところ。ＢはＡより、２倍から２８０倍速く実行されるという結果がでた。ＬＳ-tableが求まれば、ただちちにhead-normal-formの概形、head-needed-redexが求まるので、ＬＳ-アルゴリズムは有効なアルゴリズムであるといえるであろう。実際、ＬＳアルゴリズムのほうが、いくらでも速くなるようなλ式を、人工的に作ることが出来る。

謝辞：
本研究において、非常に貴重な御意見、御助言をいただいた東京工業大学の寳来正子先生に深く感謝致します。

参考文献

[1] H.P.Barendregt, J.R.Kennaway, J.W.Klop, M.R.Sleep, Needed reduction and spine strategies for the lambda calculus (Infomation and computation vol.75 pp.191-231,1987)
[2] H.P.Barendregt,The Lambda Calculus (North-Holland,1981)
[3] J.R.Hindley & J.P.Seldin, Introduction to Combinators and λ-calculus (Cambridge University Press,1986)
[4]S.L.P.Jones,The Implementation of Functional Programing Languages (Prentice Hall,1987)
[5]野口憲一,A new head-normalization algorithm for λ-calculus (東京工業大学 修士論文 1990)
[6] G.Revesz, Lambda-Calculus , Combinators and Functional Programming (Cambridge tracts in theoretical computer science; v.4,1988)