プログラミング言語　24－7
（1990. 2. 9）

組合せ論理による関数型言語の処理へのグラフ還元の寄与

杉藤　芳雄

電子技術総合研究所　情報アーキテクチャ部　言語システム研究室

あらまし　関数型言語の一処理方式として、その組合せ論理表現形を組合せ論理に関する還元により評価実行するものがある。その際、共有構造の有効利用等の面から注目される“グラフ還元”と称する還元形態がどの程度まで有効に機能するかを、圏論的枠組み（即ち、圏的組合せ論理）の場合を含めて検討する。

Contribution of Graph Reduction in case of Processing Functional Languages via Combinatory Logic

Yoshio SUGITO

Computer Language Section, Computer Science Division,
ELECTROTECHNICAL LABORATORY

1-1-4 Umezono, Tsukuba-shi, Ibaraki-ken, JAPAN

Abstract　As one of the methods of processing functional language, there exists the way which evaluates and executes its corresponding combinatory logic codes by means of combinatory logic's reduction. We investigate how effectively "graph reduction", which has the possibility of enjoying the merits of sharing structures, could perform in case of using the method, and also in case of using the framework of category theory(that is, categorical combinatory logic).

1 はじめに

関数型言語とは何かというメタな議論は控えることにしても、少なくとも構文的には関数名とその引数という組が基本単位になっていることだけは言えよう。そして、その記述形式の簡明さに魅せられて仕様等の表記道具として関数型言語を使用する立場もあるし、更には表記のみに留らずプログラムとして実行するために利用する立場もある。後者の立場では、関数型言語を何らかの方法で実行させる処理系が必要となり、ここに関数型言語の処理系を追求する課題の存在意義があることになる。

この課題を考えるとき、関数型という特徴に注目してその利点を活用しない手はなかろう。幸いにも、関数型言語という呼称が定着する以前から数学の世界では“関数”という概念と直接に対峙する研究分野があり、そこでの成果は関数型言語の処理系に直接あるいは間接に反映されつつある。

本稿で取扱う組合せ論理 (combinatory logic)[1] も、ラムダ計算 (λ-calculus)[2] と同様、このような研究分野の一例である。組合せ論理が、関数そのものを対象としている以上、関数型言語の処理系に利用できることは充分に予想されることである。

実際、Turner[3] は、関数型言語で記述したプログラム（以下では関数型プログラムと称する）という原始コードを、組合せ論理の記号列という目的コードに変換（“コンパイル”に相当）したものを、組合せ論理に基づく還元 (reduction) により評価（“実行”に相当）するという処理方式に関して、目的コード長の最適化を導入することにより関数型プログラムをほぼ実用的に処理できることを約 10 年前に実証した。

ここで還元とは、予め用意された書き換え規則の集まりの中から適用可能なものを入力記号列に対して次々と施しつつ変換していく過程のことであり、適用された書き換え規則の時系列として定義される。

ところで還元の一形態として、書き換え規則を適用する際に入力記号列のデータ構造を積極的に利用する“グラフ還元” (graph reduction) [5] と称するものがある。これは書き換え規則の適用が記号列のデータ構造に強く依存する還元形態である、と言うこともできる。

本稿では、関数型言語の処理方式の一つとして定着しつつある組合せ論理に基づく方法について、特にグラフ還元の利用という観点から検討する。そして、同じ“組合せ論理”という術語を使用し、かつ、圏論 (category theory) における関数の概念を導入した圏的組合せ論理 (categorical combinatory logic)[6] が、やはり関数型言語の一処理方式として近年提案され注目されているので、その場合におけるグラフ還元についても簡単に触れることにする。

以降では、組合せ論理に基づく関数型言語の処理方式を概観したあと、記号列のデータ構造を特定してグラフ還元を導入し、その効果を見る。そして最後に、圏的組合せ論理におけるグラフ還元について言及する。

2 関数型言語と組合せ論理

組合せ論理は、組合せ項 (combinatory term) [後述] を基本構成要素とし、特にコンビネータと称する組合せ項が“演算子”あるいは“関数”の役割を果していて、後続のいくつかの組合せ項の並びを“引数”のように扱いながら、ラムダ計算における λ 変換と同様の作用を、ラムダ計算特有の束縛変数という概念を導入することなく実現するものである。

このように組合せ項を“関数”あるいは“引数”とみなし得る上に、更に括弧表現も援用するという（少なくとも構文的見地からの）事実は、関数型言語を組合せ論理の世界へと誘惑するには十分な背景である。本章では、この形態上の類似性から出発した話が、関数型言語の処理という果実までも引起こすことを大急ぎで見ていくことにする。

先ず、組合せ論理に登場する組合せ項という構成要素は、次のように再帰的に定義されるものである。

変数の無限系列、および定数（“コンビネータ”と称するものをも含む）の有限または無限系列の存在を仮定する。
(1) 単一の変数または定数は、単一の組合せ項。
(2) A および B がそれぞれ組合せ項ならば、(AB) は単一の組合せ項。

組合せ項に登場する括弧対は、それが左詰めの括弧対で囲まれている場合 (および最も外側にある場合) には省略する習慣があるので、例えば ((((AB)C)(DE)) は ABC(DE) と略記する。

次に、組合せ論理の構成要素である組合せ項が関数的な振舞をすることを見るために、代表的なコンビネータ (という組合せ項) の書き換え規則を例示する。ここで記されている英字のうち、各大文字はコンビネータであり、各小文字はコンビネータの“引数”に相当する単一組合せ項である。矢印の右辺は、左辺に登場するコンビネータという“関数”を“引数”に作用させた (書き換え規則の適用) 結果である。尚、各書き換え規則の矢印の左辺の形はリデックス (redex) と称する。

```
S x y z  ==>  x z(y z)        (規則 S)
K x y    ==>  x               (規則 K)
I x      ==>  x               (規則 I)
B x y z  ==>  x(y z)          (規則 B)
```

```
C x y z  ==>  x z y          (規則 C)
Y x      ==>  x(Y x)         (規則 Y)
```

組合せ論理では、与えられた合法的な“形”(即ち、組合せ項の並び)から出発し、上記のような書き換え規則の適用により変換していく操作を可能な限り続けていくこと(つまりリデックスが“形”内に存在する限り続けること)で最終的に得られる“形”(標準形 Normal Form)を最初の“形”に対する値(即ち、評価結果)とみなしている。そして、この変換過程が組合せ論理における還元である。

勿論、変換過程において記号列内に同時期に複数のリデックスが存在する場合、即ち、書き換え規則を適用できる箇所が同時期に複数存在する場合も有り得る。還元は書き換え規則の適用系列として定義されるので、同一入力記号列に対して複数種類の還元が存在する状況の方が、むしろ常態である。このような場合、ラムダ計算の場合と同様に合流性が保証されていれば、複数種類の還元のいずれもが同一の評価値(標準形)に到達することになる。

組合せ論理における還元の例を以下に示すことにする。ここで矢印の上の記号の内、英字は当該コンビネータに関する書き換え規則の適用を意味し、括弧対記号 () は冗長な括弧表現を除去する操作(即ち、上述の略記可能括弧対を除去する操作)を意味する。

[例] BCIK という複合コンビネータは、下記の還元過程により2個の“引数”の並びの後者を選択する標準形を得る。

```
          B            C           I         ()       K
BCIKxy ==> C(IK)xy ==> (IK)yx ==> (K)yx ==> Kyx ==> y
```

ここで注意すべきは、単なるコンビネータの書き換え規則に関する還元だけでは、当然ながら関数型プログラムに含まれる四則計算(例えば plus 文)や比較(例えば eq 文)や制御構文(例えば if 文)等の演算を“実行”できないことである。そこで、還元形式だけでこれらの演算をも実行可能とするには当該演算の関数評価系に関する書き換え規則を追加しておけば良い。

例えば、関数型プログラムに普通登場する演算の関数評価系に関する書き換え規則例は次の様なものである。

```
plus   e1  e2   ==>  (e1 + e2)   の実行値
times  e1  e2   ==>  (e1 * e2)   の実行値
eq     e1  e2   ==>   e1 = e2    のとき  “T”
                      e1 ≠ e2    のとき  “F”
if  e1  e2  e3  ==>   e1 = “T”のとき  e2
                      e1 = “F”のとき  e3
```

関数型プログラムを組合せ論理コードに“コンパイル”するための手順は、[3] に示されているように、以下の4段階である。ここでxは変数または定数の単一組合せ項、Eiは組合せ項である。

(1) < Curry 化>

与えられた関数型プログラムを、(構文に従って)構文要素の2項を単位に括弧対で囲むような表現にする。

例: Def Double x = x + x ==> Def Double x = ((plus x) x)

(2) <移項>

定義式の左辺にある引数を右辺に“移項”して(3)に備える。

例: Def Double x = ((plus x) x) ==> Def Double = [x]((plus x) x)

(3) <アブストラクション>

定義式の右辺に対し、組合せ論理に関する次のようなアブストラクション操作を可能な限り適用して、定数/変数/コンビネータから成る組合せ論理コードに変換する。

```
[x] (E1 E2)  ==> S([x]E1)([x]E2)
[x]x         ==> I
[x]y         ==> Ky                    (yは定数またはx以外の変数)
```

```
例: Def Double = [x]((plus x) x) ==> S([x](plus x))([x] x) ==>
                 S(S([x] plus)([x] x))I ==> S(S(K plus) I)I
```

(4) <最適化>

組合せ論理コード((3)で得られたもの)の記号列の長さを短縮する次のような最適化操作を行う。

```
(a)   S(K E1)(K E2)  ==>  K(E1 E2)
(b)   S(K E1)I       ==>  E1
(c) ((a)(b)が不可能のとき)
      S(K E1)E2      ==>  B E1 E2
(d) ((a)(b)(c)が不可能のとき)
      S E1(K E2)     ==>  C E1 E2

  例:  S(S(K plus) I)I ==> S (plus) I ==> S plus I
```

尚、この目的コードを“実行”するには、入力データを後置したものに組合せ論理／関数評価系の還元を施せばよい。例えば、Double(3) を実行する場合には、Double の目的コードである (S plus I) に入力引数の 3 を後置させた (S plus I)3 を還元すればよい。これは次のような還元により値が得られる。

```
                ()              S               I            plus
(S plus I)3 ==> S plus I 3 ==> plus 3(I 3) ==> plus 3 3 ==> 6
```

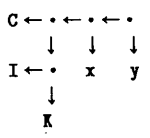
3 グラフ還元の利用

前章で述べた組合せ論理コードに関する還元は、本章で扱うグラフ還元を用いると有効になることが少なくない。ここでグラフ還元とは、操作対象となる記号列のデータ構造を積極的に利用することで効果的な還元をめざす還元の一形態である。とは言うものの、グラフ還元という概念に関する統一見解は見当らないのが実情なので、不本意ながら以降では、次のように仮定する。

即ち、ある特定の形式によるデータ構造で記号列が表現されている状況を想定し、その記号列に適用されるべき書き換え規則が当該データ構造 (の特性) に基づいて設定されている場合の還元形態を“グラフ還元”と称する。

記号列を表現するためのデータ構造形式としては、グラフ還元等の文献 (例えば [3]) で通常利用されている“上昇型” [7] を採用することにする。これは簡単に言えば、記号列を左から右に走査しつつ遭遇する各組合せ項を、2 進木の左下端から根に向けて昇りながら (これが上昇型の命名由来)、配置していく方式である。

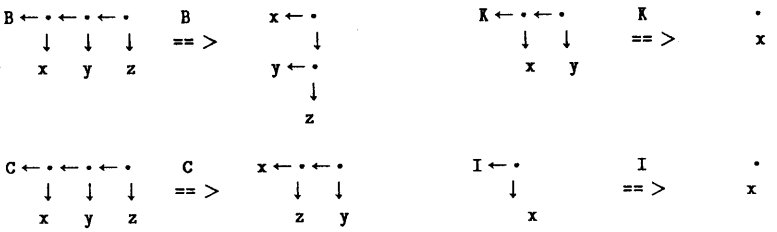
例えば、前章の BCIKxy 還元例の途中の C(IK)xy を上昇型で表現すれば次のようになる。



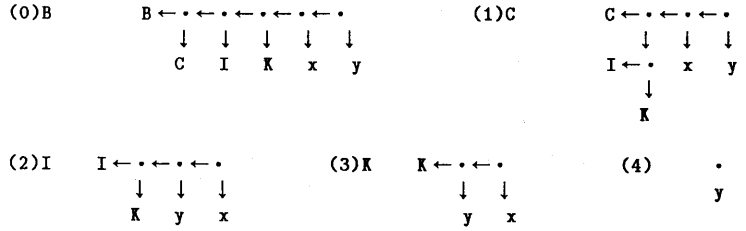
このデータ構造は [7] で述べたように、記号列を 2 進木に密に充填できる利点があるものの、式の読み取りに多少の慣れを必要とする欠点がある。だが最大の利点は、あとで例示されるように、冗長括弧 (即ち、省略可能括弧) の除去操作がその還元過程に一切登場しないことである。

本データ構造上で実際に組合せ論理の書き換え規則を適用するには、やはり当該データ構造で表現された書き換え規則が必要となる。

そこで、とりあえず前章の例 BCIKxy という記号列に登場する各コンビネータの書き換え規則を、上昇型データ構造に基づいて設定すると次のようになる。



以上の準備のもとで、BCIKxy を上昇型データ構造上で上記の書き換え規則により還元 (これがまさにグラフ還元に相当) させると、以下の経過を辿りながら前章と同じ結果を得る。

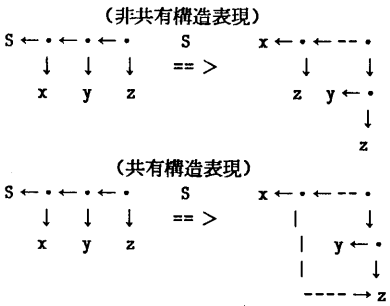


この図解から、前章の還元過程で登場していた冗長括弧の除去操作が、この還元過程では出現していないことが分る。これはグラフ還元による効果(より正確には上昇型データ構造による効果)の一種である。

それでは記号列にコンビネータSが含まれている場合でのグラフ還元はどうであろうか。実はここでようやくグラフ還元の天国と地獄とが共存する共有構造の問題[8]が浮上するのである。

規則Sの矢印の右辺を見ると、コンビネータSの第3"引数"zが2箇所に出現している。そこで、規則Sをデータ構造で表現する場合には、この事実をどのように扱うかで2通りの考え方が存在する。それは即ち、共有構造を採用するか否かである。

そこで規則Sを、上昇型データ構造により2通りで以下に表現してみよう。



グラフ還元がその効果を大いに発揮する局面は、共有構造の積極的な利用においてである。実際、グラフ還元に関連する文献で、規則Sを共有構造で表現していない例を寡聞にして知らないほどである。

しかしながら、共有構造表現に基づくS規則等を含めたまま無防備にグラフ還元を施していくと、快適な筈の航海がやがて暗礁に乗り上げる危険性が多いことになるが、共有構造に関する議論は[8]を参照されたい。

4 圏的組合せ論理とグラフ還元

ラムダ計算が組合せ論理と同様に関数を対象とする以上、それは関数型言語と密接な関係にあることは言うまでもない。ところで、関数型プログラムをラムダ計算におけるλ項表現にしたあと、De Bruijinのname-free記法(簡単に言えばλ項に登場する変数をすべて数字で置き換える方法)に焼き直し、そして最後にそれを圏的組合せ論理(Categorical Combinatory Logic)項表現に変換したものをCAM(categorical abstract machine)で実行する方式がある。

ここで圏的組合せ論理は、圏論における関数概念(具体的にはカルテシアン閉圏 Cartesian closed category)とコンビネータ表現との合体したものとでも言えるものであり、前章まで触れてきた組合せ論理と直接の関係がある訳ではない。しかしながら、上述の方法は基本的には還元により関数型言語を評価/実行する点では、組合せ論理による方法と類似している。そこで、以下では圏的組合せ論理の場合におけるグラフ還元の適用を見ていくことにする。

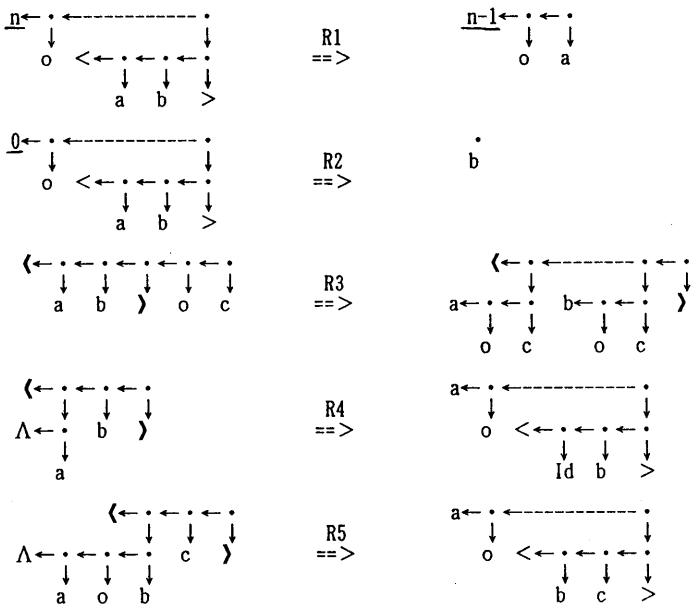
圏的組合せ論理の詳細については関連文献(例えば[6])に譲ることにして、ここでは本稿の主眼であるグラフ還元に直接関係する書き換え規則そのものを先験的に与えることにする。

圏的組合せ論理に登場する書き換え規則の集まりは、創始者Curienの提案をはじめ種々の版があるが、ここではLinsによるもの[4]を考えることにする。まず、その規則の集まりを示そう。但し、原案では特殊な記号が使われているので、ここでは別の記号で代替することを許していただく。即ち、原案での左/右向き白ぬき三角形を本稿ではそれぞれ《、》を用いることにする。また、[R1]、[R2]でnや0等の数字表現に下線が施されているのは、De Bruijn番号という特殊な数値を意味している。

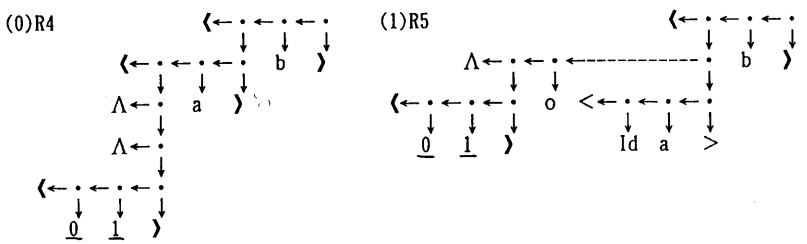
[R 1]　$\underline{n}$ o ＜a, b＞ ==＞ $\underline{(n-1)}$ o a　[但し、n＞0]
[R 2]　$\underline{0}$ o ＜a, b＞ ==＞ b
[R 3]　《a, b》 o c ==＞ 《a o c, b o c》
[R 4]　《Λ (a), b》 ==＞ a o ＜Id, b＞
[R 5]　《Λ (a) o b, c》 ==＞ a o ＜b, c＞

これら以外にも必要に応じて、定数や組込関数のための書き換え規則が用意されるが、ここではそれらの規則は除外しておくことにする。というのは、目的はあくまでもこのような形式の書き換え規則の場合に対するグラフ還元の適応性の検討ということだからである。

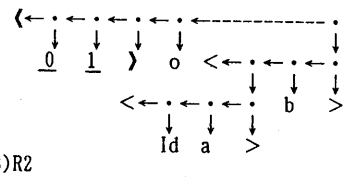
次に、上記の［R 1］から［R 5］までの規則を（上昇型の）データ構造で表現しよう。



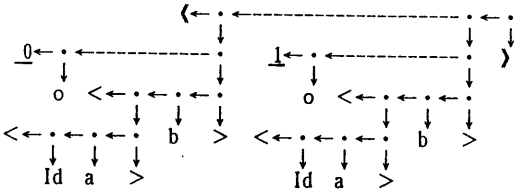
それでは、(λ x λ y.yx)ab　に対応する圏的組合せ論理表現である《《Λ (Λ (《0,1》)),a》,b》 を例題として、今準備した書き換え規則（のデータ構造表現）によるグラフ還元を以下に展開しよう。



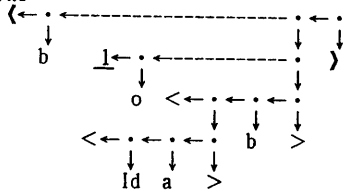
(2)R3



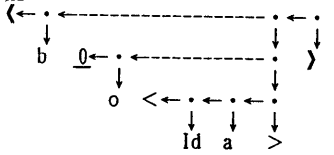
(3)R2



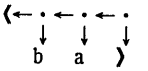
(4)R1



(5)R2



(6)



以上から、《《λ(λ(《0,1》)),a》,b》 $\overset{*}{==>}$ 《b,a》 であることが得られた。

実際、ラムダ計算では (λxλy.yx)ab $\overset{*}{==>}$ ba であるから、上記の還元結果は正しい。

この還元過程だけから、グラフ還元の効果を見ることは早計かもしれないが、少なくとも書き換え規則のデータ構造表現を適当に定めさえすれば、たとえ圏的組合せ論理の場合でもグラフ還元を実行できることだけは確かである。

但し、圏的組合せ論理の一連の書き換え規則を眺めると、いわゆる共有構造の問題が生じることはないように見受けられる。と言うことは、この圏的組合せ論理の場合には、それほどグラフ還元の活躍する場面がないのかもしれない。(但し、不動点コンビネータに相当する書き換え規則を圏的組合せ論理に用意すれば話は別である。)

5 おわりに

組合せ論理に基づく関数型言語の処理方式について、グラフ還元の利用という観点から述べた。そして、"圏的組合せ論理"も組合せ論理の一種と強引に看做して、そこでのグラフ還元についても簡単な例題で試行した。後者については取組んでから日が浅いので今後に残された問題は多い。圏的組合せ論理において、より効果のあるデータ構造の発見とか、グラフ還元の有効事例の摘出とかが待たれるところである。

謝辞 本研究の機会を提供される棟上昭男情報アーキテクチャ部長、およびご助言をいただく当研究室各位をはじめとする関連諸氏に謝意を表わす。

References

[1] 例えば J.R.Hindley,B.Lercher,J.P.Seldin:
"Introduction to Combinatory Logic", London Mathematical Society Student Texts 1, Cambridge University Press,1972.

[2] H.P.Barendregt:"The Lambda Calculus: Its Syntax and Semantics", North-Holland, 1984.

[3] D.A.Turner:"New Implementation Techniques for Applicative Languages", Software-Practice and Experience, Vol.9, pp.31-49, 1979.

[4] A.Diller:"Compiling Functional Languages", John Wiley & Sons LTD, 1988.

[5] J.H.Fasel,R.M.Keller(Eds.):"Graph Reduction", Lecture Notes in Computer Science, No.279, Springer-Verlag, 1987.

[6] P.-L.Curien:"Categorical Combinatory Logic", Lecture Notes in Computer Science, No.194, Springer-Verlag, 1985.

[7] 杉藤：“グラフ還元とデータ構造”、電子情報通信学会ソフトウェアサイエンス研究会、SS88-44(1989年3月10日).

[8] 杉藤：“グラフ還元における共有構造の効用と限界”、電子情報通信学会ソフトウェアサイエンス研究会、SS89-17(1989年9月8日).

盛光印刷所