ソフトウェア基礎論　30−6
（1989. 9. 8）

グラフ還元における共有構造の効用と限界

ON PROS AND CONS OF SHARING STRUCTURES IN GRAPH REDUCTION

杉藤　芳雄

Yoshio SUGITO

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

Computer Language Section, Computer Science Division, ELECTROTECHNICAL LABORATORY

あらまし　グラフ還元は、記号列への書き換え規則の適用を、記号列のデータ構造表現上での書き換えの実行とみなし、当該データ構造の特徴を積極的に利用する還元形態である。本稿では、組合せ論理の世界でのグラフ還元という状況において、記号列のデータ構造に殆ど必然的に登場し、かつ、還元過程の重要な鍵を握る共有構造に関する効用と限界を、関数型言語プログラムの組合せ論理表現という事例研究を交えて検討する。

Abstract　Graph reduction, where applications of rewrite rules into strings of symbols are regarded as an execution of rewrite rules in the data structure of the strings, makes good use of the characteristic of its data structure. In this report, in the circumstances of graph reduction applied in the frame of Combinatory Logic, including the case study for analysis of Combinatory Logic representation of a functional language program, we discuss the effects and limitations of sharing structures, which appear almost necessarily in the data strucures of strings and which play an important role in the reduction process.

1. はじめに

入力記号列の一部分を別の記号列で置き換えるという操作、即ち、与えられた入力記号列(それをSとする)内に、或る記号列(それをLとする)に一致するものが見出される場合、その箇所を(別の)或る記号列(それをRとする)で置き換えるという操作によりSを変形して出力記号列とすることを、書き換え規則(rewrite rule または reduction rule)[今の場合 L==>R と記す]のSへの適用と称することにする。

いくつかの書き換え規則を用意しておき、与えられた入力記号列に対して適用可能な書き換え規則を次々と施すことにより最終的に得られる出力記号列を、初期の記号列に対する評価値と規定する演算の枠組は、例えば構文規則という書き換え規則による入力記号列の構文解析の場合のように、計算機科学でしばしば登場するものである。このとき、一連の書き換え規則の適用系列のことを還元(reduction)と称する。

この還元は、初期の入力記号列から出発して或る時点での書き換え規則の適用結果としての出力記号列を次の時点での書き換え規則の適用の際の新たな入力記号列として継承する操作過程であるため、過去の書き換え規則適用の履歴は現時点の記号列に凝縮されていることになる。

この場合、過去に順次形成されてきた書き換え規則の適用結果である中間出力記号列は一般には保存されることなく、存在するのは常に最新の適用結果のみである。これが還元という演算方式の大きな特徴であり、いわば後戻りを期待できない(あるいは期待しない)単方向性の処理である。

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)[2][3][4]と称するものがある。見方を変えれば、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ような還元形態である。

本稿では、関数型言語の処理方式の一つとして有力視されている組合せ論理(Combinatory Logic)[1]の還元という世界で上述のグラフ還元について検討するが、特にグラフ還元に必然的に関連する記号列の共有構造という、いわば毒にも薬にもなり得る両刃の剣であるデータ構造問題を中心に議論する。

以降では、組合せ論理とその還元について簡単に触れたあと、記号列のデータ構造を特定してグラフ還元を導入する。そして、組合せ論理に関するグラフ還元を試行し、その有効性を確認する。その際、共有構造が鍵を握ることを調べるために、事例研究として関数型言語プログラムのコンパイル・コードとしての組合せ論理表現におけるグラフ還元を、還元過程の図解入りで比較検討する。

2. 組合せ論理と還元

組合せ論理は、後述のように定義される組合せ項(combinatory term)を基本構成要素とし、特にコンビネータ[5]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を“引数”のように扱いながら、ラムダ計算(λ-Calculus)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。

従って、組合せ論理は、束縛変数という概念に伴う厄介な規則に悩まされる必要がないので演算の簡明化をもたらす利点がある反面、その記号列表現の意味が直観的に甚だ把握しにくいという(或る意味では重大な)欠点がある。

先ず、組合せ論理に登場する書き換え規則、即ち、コンビネータに関する書き換え規則を簡単に紹介する。

組合せ項(combinatory term)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含む)の有限または無限系列の存在を仮定する。これらの変数や定数はアトムと称する。

(1) 各アトムは単一の組合せ項。

(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、((((AB)(CD))E)(FG)) は AB(CD)E(FG) と略記される。

略記形から省略されている括弧対を復元するには、記号列を左端から右方向に走査しつつ、先ず最左端の単一組合せ項を見出し、次にその右隣の単一組合せ項をさがし、それが存在すればそれら2つの単一組合せ項を括弧で囲んで新たに1つの単一組合せ項とし、再びその右隣の単一組合せ項をさがすという操作を反復していき、最後は全体が1つの単一組合せ項となればよい。今の例では、新たに復元された括弧対を[ ]で表現することにすれば、次のようになる。

AB(CD)E(FG)→[AB](CD)E(FG)→[(AB)(CD)]E(FG)→[((AB)(CD))E](FG)→[(((AB)(CD))E)(FG)]

以上の準備のもとに、代表的なコンビネータの書き換え規則を次に例示してみよう。ここで記されている英字のうち、大文字はコンビネータ(という組合せ項)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。矢印の右辺はコンビネータを“引数”に作用させた結果である。尚、各書き換え規則の矢印の左辺の形はリデックス(redex)と称される。

```
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)
W x y    ==>  x y y           (規則 W)
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、組合せ項の系列)から出発して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できる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)とみなしているが、この変換過程が組合せ論理での還元である。

組合せ論理における還元の例を以下に3件示すことにする。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1] BC(KI)I 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”の順序を交換する標準形を得る。

```
          B              ()             K          ()         C        I
BC(KI)Ixy ==> C((KI)I)xy ==> C(KII)xy ==> C(I)xy ==> CIxy ==> Iyx ==> yx
```

本例について若干の補則をすると、上記の還元過程の最初に Ixy という部分記号列が登場し、一見すると規則B以外に規則Iも適用可能のように思える。しかしながら、省略された括弧対を復元したものを常に念頭に置いて考える必要があることに注意すべきである。今の例では、(((((BC)(KI))I)x)y) という記号列がその復元形であり、この形では規則Iを適用できないことが次のようにして分る。

実はコンビネータの書き換え規則例をいくつか示したときに、明言しないでおいたが、いずれも括弧対に関しては略記形である。例えば、規則Iについて括弧対を復元して表示すれば (Ix) ==> x となり、規則Bについては (((Bx)y)z) ==> (x(yz)) となる。これらの事実を承知していれば、今の場合に規則Iのリデックス (Ix) が (((((BC)(KI))I)x)y) に含まれていないことは明らかである。

このように組合せ論理の還元では、記号列内の括弧対で明示されていないものに意外な落とし穴があるので、取扱いには注意が必要である。

勿論、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、同じ入力記号列から出発する複数種類の還元が存在する状況の方が、むしろ普通である。この場合、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。

尚、冗長括弧対の除去を書き換え規則として表現することを試みると次のようになろう。ここでxは単一組合せ項を、αおよびβは組合せ項の1つ以上の並びを、それぞれ表わすものとする。

(x) ==> x

((α)β) ==> (α β)

この2つの書き換え規則を一括して以降では“規則()”とよぶことにする。

[例2] S(KS)K という複合コンビネータは、下記の還元過程により3個の“引数”の内の第2と第3の“引数”を単一組合せ項にする(即ち、括弧対で囲む)標準形を得るので、コンビネータBと同等の作用をすることになる。

```
           S              ()             K           S             ()
S(KS)Kxyz ==> (KS)x(Kx)yz ==> KSx(Kx)yz ==> S(Kx)yz ==> (Kx)z(yz) ==>
         K
Kxz(yz) ==> x(yz)
```

[例3] SS(KI) という複合コンビネータは、下記の還元過程により第2の“引数”を反復する標準形を得るので、コンビネータWと同等の作用をすることになる。

```
          S              S                ()           K          I
SS(KI)xy ==> Sx((KI)x)y ==> xy(((KI)x)y) ==> xy(KIxy) ==> xy(Iy) ==>
      ()
xy(y) ==> xyy
```

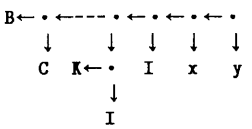
3. グラフ還元

第1章でも触れたように、グラフ還元とは記号列のデータ構造を積極的に利用する還元形態である。これだけでは漠然としているが、グラフ還元に関する統一見解は見当らないのが実情である。

そこで以降では、記号列がある特定のデータ構造で表現されていると想定し、その上で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”と称することにする。

先ず、記号列の表現のためのデータ構造としては、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”[6]を採用することにする。これは、簡単に言えば2進木の左下端から根に向けて(これが上昇型の命名由来である)、記号列を左→右に走査しつつ各組合せ項を配置していく方式である。

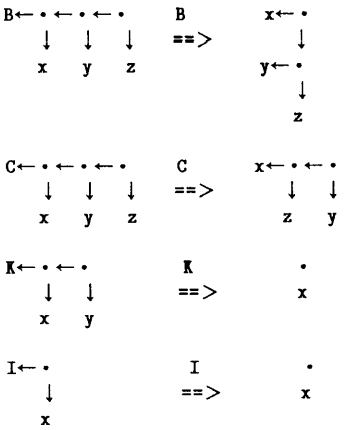
例えば、前章の例1の BC(KI)Ixy を上昇型で表現すれば次のようになる。



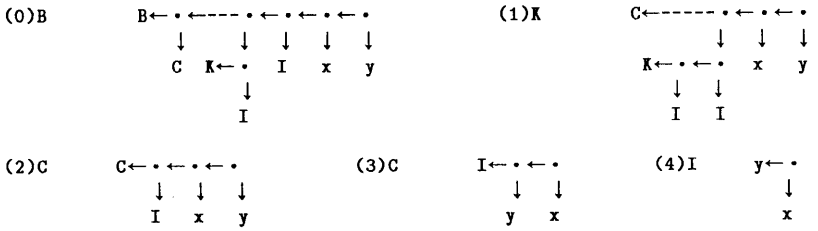
このデータ構造は[6]で述べたように、密に記号列を充填できる利点があるものの、式の読み取りに多少の慣れを必要とする欠点がある。そして最大の特徴あるいは特長は、あとで示すように、冗長括弧の除去操作である規則()がその還元過程に一切登場しないことである。このことは還元過程の本質部分の議論のためにも、還元過程の図解の際の紙面の節約のためにも好都合なので、採用の最大の根拠になっている。

本データ構造上で実際にコンビネータの書き換え規則を適用するには、やはりそのデータ構造向きの(より正確には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1である BC(KI)Ixy 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。特に規則Bの矢印の右辺に存在する括弧対が、ここではどのように表現されているかに注意されたい。



以上の準備のもとで、BC(KI)Ixy 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する)させると、一例として以下の経過を辿りながら前章と同じ結果を得る。(“一例として”と述べたのは、段階(1)ではリデックスとしてCとKの2つが存在し、ここではKを選択しているからである。)



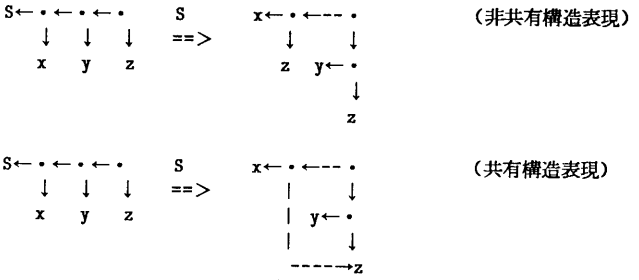
この図解から、前章の還元過程では登場していた規則()が、この還元過程では一切出現していないことが分る。これは一種のグラフ還元効果(より正確には上昇型データ構造効果)というべきものであろう。

4. 共有構造の登場

それでは今まで意図的に避けてきた第2章の別の例題、即ち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本稿の主題である共有構造の問題が浮上するのである。

規則Sの矢印の右辺を見ると、コンビネータの第3“引数”zが2箇所に出現していることが分る。そこで、規則Sをデータ構造で表現する場合には、このことをどのように扱うかで2通りの考え方が存在する。それは言うまでもなく、共有構造を採用するか否かである。(同様に、規則Wや規則Yにも共有構造の問題がある。)

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のは、まさに共有構造の積極的な利用においてであり、実際、グラフ還元にまつわる文献でも通常、規則Sは共有構造で表現されている。

共有構造の主な利点をあげると次のようになろう。

(1)部分記号列をポインタ等で共有するので記憶資源の節約になる。

(2)共有部分内で閉じている未評価記号列が存在する場合、一旦そこで評価(還元)が実行されると、還元演算の特性により書き換えが実施されるため、次回以降では再評価の手間が不要(実は再評価したくてもできない!)となり、時間資源の節約になる。

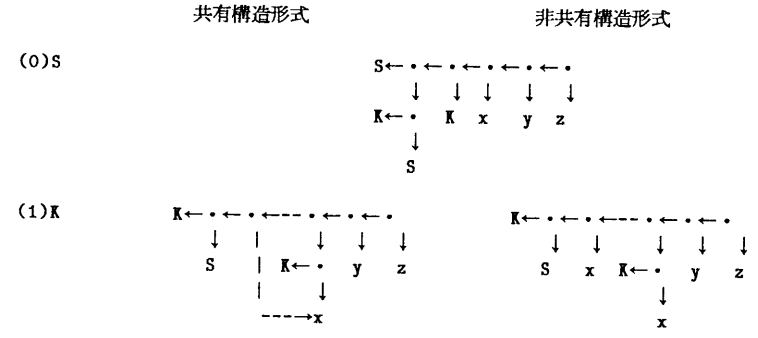
非共有構造の主な利点をあげると次のようなことが考えられる。

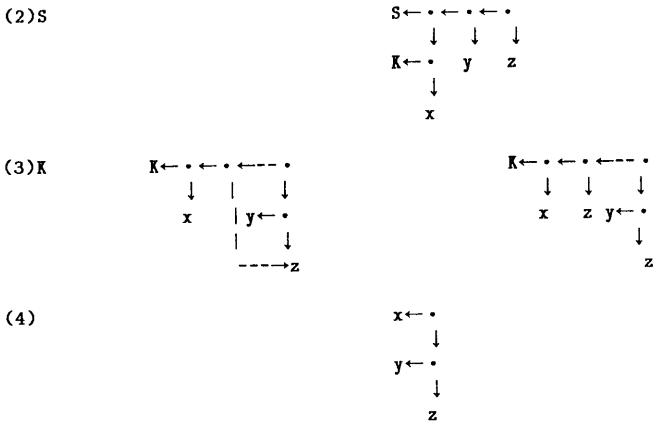
(1)どの部分記号列の評価でも安心して独占的に実行できるので演算機構が簡素化する。

(2)同時期に複数のリデックスが存在する場合、並列評価の可能性が高い。

各構造の欠点に関しては、上記の利点の裏側あるいは反面を考えれば凡その見当がつき、例えば次のように指摘できる。共有構造の主な欠点は、消去する場合には細心の注意が必要なこと、並列評価が困難なこと、演算機構が重いこと、等である。一方、非共有構造の主な欠点は、複製(コピー)する手間と記憶資源が余分に必要なこと、共有されていれば1回で済む評価が個別に必要なこと、等である。

ここで第2章の例2に関して、共有構造形式と、非共有構造形式とを並べて還元過程を図解してみよう。





以上の図解から何が読取れるかと言えば、この程度の共有化(わずかに1文字アトムのみ)ならば殆ど全く問題がないということである。強いて注意事項を述べるならば、上記の段階(1)や(3)で規則Kを適用する際には、共有構造形式の場合、第2“引数”zを消去するときにそれが共有されていることに配慮して対処すべきである、ということである。

5. 共有構造の破綻または限界

それでは、組合せ論理でのグラフ還元が共有構造の利点を享受するべく、その還元過程において共有構造を維持し続けられるであろうかという問題を考えるために、事例研究をしてみよう。本章では、今までの無味乾燥な組合せ論理の記号列に代って、それなりに意義のある組合せ論理表現を材料とする。

即ち、組合せ論理が関数を対象としている以上、いわゆる関数型言語との相性が良いことは充分に予想されるが、実際、関数型言語で記述されたプログラム(これを関数型プログラムと呼ぼう)を組合せ論理で許容される“形”(コンビネータを含む組合せ項表現)のコードに変換したあと、このコードに2章で述べた組合せ論理の還元を施すことにより値を求めることで関数型プログラムの実行とみなすような関数型言語の処理系が存在する。

そこで、関数型プログラムから組合せ論理表現に変換(この変換を[3]では“コンパイル”と呼ぶ)したものを実験材料として選択する。但し、本稿ではあくまでも組合せ論理での還元に主眼があるので、関数型言語表現から組合せ論理表現へのコンパイル方法等については割愛する。詳細は文献[3]等を参照されたい。

今、次のような関数型プログラムを考える。

```
twice twice suc 0   where    twice f x = f(f x)
                             suc x = x + 1
```

これは、昇順での“次の値”を求める関数suc、与えられた関数を2回適用する関数twiceを、引数0に対して上記の順序で評価するという簡単な演算のプログラムである。

今の場合、“次の値”操作を“→”で表現すれば、最終的には ((((0→)→)→)→) となるので結果は4の筈である。

このプログラムを組合せ論理コードにコンパイルし、その目的コードに引数0をデータとして与えた形で記述すると次のようになる。ここで、plus は2つの引数を加える関数である。

S B I (S B I) (plus 1) 0

先ず、参考のためにこの組合せ論理表現に対する還元過程の一例を以下に示すことにする。

```
                    S                                  I                                B,()
SBI(SBI)(plus 1)0 ==> B(SBI)(I(SBI))(plus 1)0 ==> B(SBI)(SBI)(plus 1)0 ==>
                    S                                 I,()
SBI(SBI(plus 1))0 ==> SBI(B(plus 1)(I(plus 1)))0 ==>
                      S                                                          I,()
SBI(B(plus 1)(plus 1))0 ==> B(B(plus 1)(plus 1))(I(B(plus 1)(plus 1)))0 ==>
                                          B
B(B(plus 1)(plus 1))(B(plus 1)(plus 1))0 ==>
                                         (),B
(B(plus 1)(plus 1))((B(plus 1)(plus 1))0) ==>
                                        (),B
B(plus 1)(plus 1)((plus 1)((plus 1) 0)) ==>
                                       ()
(plus 1)((plus 1)(plus 1 (plus 1 0)) ==> plus 1(plus 1(plus 1 (plus 1 0)))
```

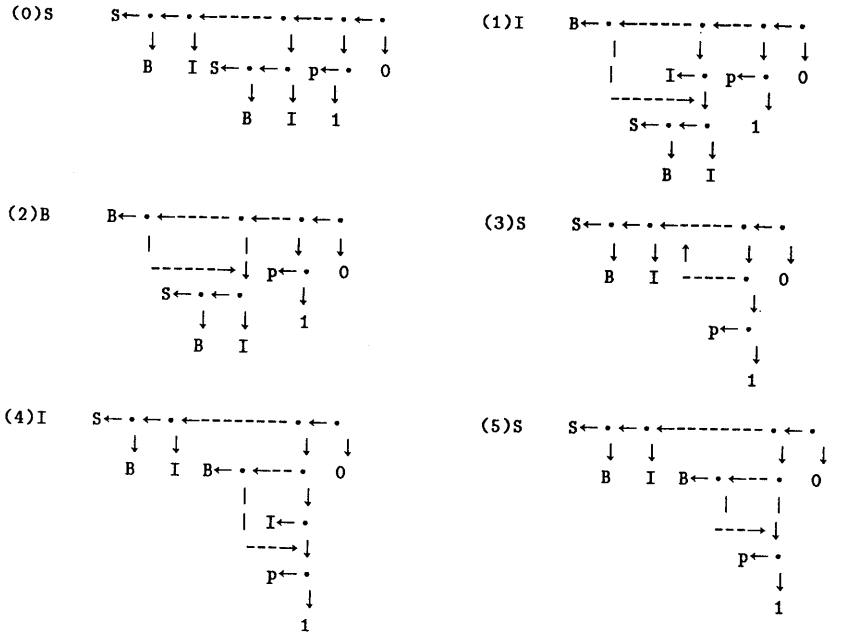
上記還元過程の最終結果 plus 1(plus 1(plus 1 (plus 1 0))) に、関数plusを適用すれば所期の答である4が得られる。

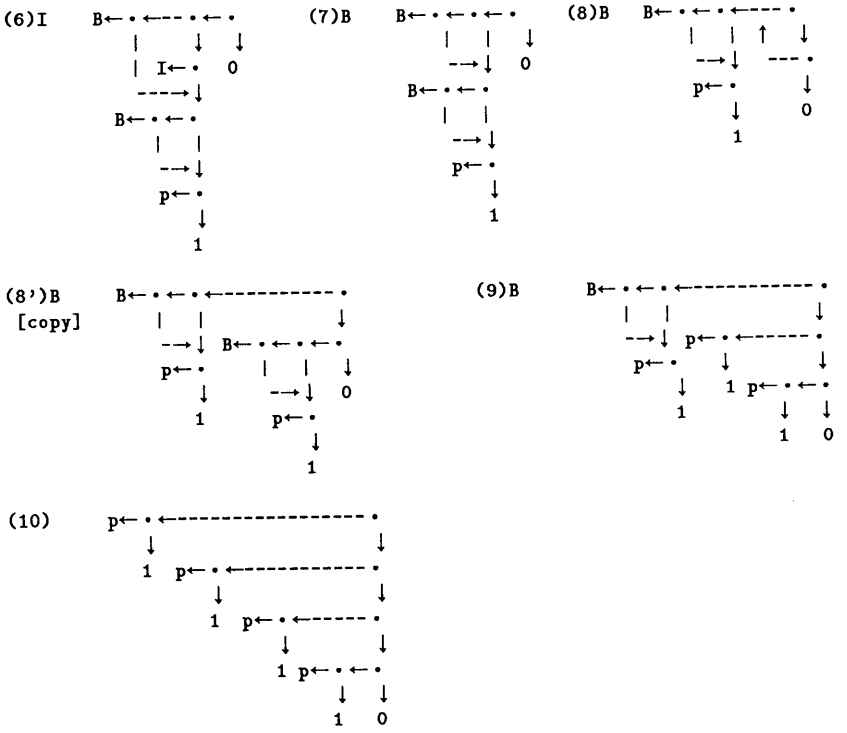
当然ながら関数 plus も次のような書き換え規則として考えられるので、やはり還元の一種とみなすことができる。

plus e1 e2 ==> e1およびe2が数値アトムならば (e1 + e2) という数値アトム

結局、組合せ論理の還元と関数の還元とを統合した還元系を用意すれば、関数型プログラムの評価系が得られることになる。これが本章の冒頭で述べた関数型言語の処理系の基本部に相当するものである。

それでは次に、上記の組合せ論理表現を共有構造形式でのグラフ還元として図解することに話を進めることにしよう。ここでは簡単のために plus を p と表記する。





この還元過程で注目すべき特徴は、共有部分の大きさと、共有構造を維持する取扱いの困難さである。

先ず、共有部分の大きさに関しては、前章のグラフ還元の図解では、共有部分はわずかに1文字アトムだけというささやかさであった。それに対して本章の上記の還元過程の図解では、例えば段階(1)では (SBI) 、段階(4)では (p 1) 、段階(6)では実に (B (p 1) (p 1)) というような共有部分の大きさである。特に段階(6)では共有部分自体に別の共有部分 (p 1) を含む二重構造から成っている。

ところで、段階(7)で規則Bを適用する際のリデックスは、勿論、2つあるBの内の上に位置する方であるが、これにより書き換えられた結果である段階(8)はかなり複雑な共有形態である。これが次の話題である「共有構造を維持する取扱いの困難さ」に結びつく。要するに共有構造の利点を享受したくても段階(8)の形のままで規則Bを適用することは絶望的であり、段階(8')に示すように共有部分を複製して主要な共有構造を解除する必要があるという事実である。このことは共有構造をデータ構造に許していても、場合によってはその利用を断念せざるを得ないことがあるということである。実は図示していないが、段階(8)と類似の事態が既に段階(3)で発生しており、規則Sを適用するにはやはり共有部分を複製する必要がある。

そして、上記の図解をよく眺めてみると、たとえ共有構造が大きくてもコンビネータの書き換え規則のリデックスが含まれているものは見当らず、前章で述べた共有構造の利点の内の“再評価不要”論が達成できていない。この現象がどの程度まであてはまるのか定かではないが、気掛りなことではある。

6. おわりに

本稿で述べたグラフ還元と共有構造に関する話題は、表題の立派さほどには内容が煮詰まっておらず混沌状態なので、羊頭狗肉のそしりを免れないことを充分に自覚している。ただし、事例研究を通して得られた共有構造に関する建前と実態との乖離を意味する事実の落ち穂拾い程度の責任は果せたのではと考えている。

最初の構想では、再帰プログラムのグラフ還元が真の意味での共有形式で(即ち、還元過程の途中で共有部分を複製する操作が一切登場せずに)実現可能か否かの問題にいささかなりとも寄与することであったが、残念ながらここでは別ルートに踏込んだようである。

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。

7. 参考文献

[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]A.Diller: "Compiling Functional Languages'', John Wiley & Sons LTD, 1988
[3]D.A.Turner: "New Implementation Techniques for Applicative Languages", Software-Practice and Experience, Vol.9, pp.31-49, 1979
[4]J.H.Fasel,R.M.Keller(Eds.): "Graph Reduction", Lecture Notes in Computer Science, No.279,Springer-Verlag, 1987
[5]杉藤："コンビネータとグラフ還元"、電子情報通信学会ソフトウェアサイエンス研究会資料SS87-29 (1988.2.12)
[6]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資料SS88-44 (1989.3.10)

盛光印刷所