ソフトウェア工学　78－5
(1991. 3. 8)

関数型プログラムのグラフ還元
—サイクル構造の扱いを中心に—

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　関数型プログラムの組合せ論理コードをグラフ還元で実行する方式は、関数型言語の一処理方式として定着している。Turnerがこの方式の有効性を示す際に、データ構造としてサイクル構造を活用できることを挙げたが、その具体的な方法には言及していない。その主張の可否をサイクル構造問題と称することにすれば、筆者は先に再帰呼出しの場合については然るべき対応によりTurnerの枠組みでグラフ還元が可能であることを示した。本稿では、その続報として、不動点コンビネータYに関するサイクル構造問題を検討し、同様の結論を得た。

On Graph Reduction of Functional Program
—especially on treatment of cycle structure—

Yoshio SUGITO
Computer Language Section, Computer Science Division
ELECTROTECHNICAL LABORATORY
1-1-4 Umezono, Tsukuba-shi, Ibaraki-ken, 305, JAPAN

Abstract　A method of using graph reduction based on the combinatory logic in order to evaluate a combinatory logic code translated from a given functional program is fully established in the domain of processing a functional language. D.A.Turner showed the effectiveness of the method, and claimed as one of its merits an availability of a cycle structure for a recursive call and for a fixed-point combinator Y. Unfortunately he did't give us its concrete explanation, therefore it remained some doubt about its validity. The author calls the doubt 'a cycle structure problem', and in his former report he investigated the problem in case of a recursive call in the Turner's frame of a special stack and a graph representation. The conclusion is that if we carefully decide related graph rewriting rules so as to keep its redex properly, Turner's assertion will hold. In this report, the author tackles the problem in case of a fixed point combinator Y in the same Turner's frame, and obtains the same conclusion as before.

1 はじめに

関数型言語で記述したプログラム (以降では関数型プログラムと称する) の処理系を還元 (reduction) 方式で実現することを考える場合、特にいわゆるグラフ還元 (graph reduction)[1] 利用の方式におけるサイクル構造の取扱いに焦点をあてて検討する。

関数型プログラムの処理系 [2] の実現を目指すとき、“関数”という概念と直接対峙してきた研究分野である組合せ論理 (combinatory logic)[3] やラムダ計算 (λ-calculus)[4] の成果を反映させるのは極めて正統的な接近法であろう。

本稿で取扱う処理方式はこのような流れに沿うもので、Turner[5] が先駆的に行なったものである。彼は、所与の関数型プログラム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) したあと、組合せ論理に基づく還元により評価 (“実行”に相当) するという処理方式の枠組みで、目的コード長の最適化や正規グラフ還元 (normal graph reduction) 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実証した。

ここで還元とは、入力記号列に対して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施しつつ変換していく操作あるいは過程のことであり、適用された書き換え規則の時系列として定義される。

還元の一形態として、書き換え規則が入力記号列のデータ構造を積極的に反映および利用している方式であるグラフ還元と称するものがある。上述の Turner[5] に登場する正規グラフ還元は、勿論その好例である。

ところで、Turner は文献 [5] において、グラフ還元の長所として再帰呼出しに対するサイクル構造の活用という主張を掲げているが、同文献を見る限りではその具体的な対処法が曖昧である。

筆者は先に、同文献の階乗プログラムの例題において再帰呼出しをプログラムの木構造の根への有向辺で表現してある場合について、その具体的なグラフ還元の適用法を発表した [6]。

本稿は、その続編として、やはり Turner[5] において再帰呼出しの実現に必須の不動点コンビネータ Y(Fixed-point combinator) が自己ループ有向辺という最小のサイクル構造で表現できることを図示するのみでその実現法が言及されていない事柄を採り上げ、その具体的なグラフ還元の適用法を見出したので紹介する。

以降では、組合せ論理の還元による関数型プログラムの処理方式を駆足で見たあと、Turner[5] が概観的に述べている特殊スタック併用のグラフ還元方式を説明する。そのあと、前述の不動点コンビネータの自己ループ有向辺表現による階乗プログラムを例題として、グラフ還元の具体的な適用法を図解入りで述べる。

2 組合せ論理の還元による関数型プログラムの処理

組合せ論理は、組合せ項 (combinatory term) を基本構成要素とする記号系列として表現され、特にコンビネータと称する特定の記号アトムから成る組合せ項が“演算子”あるいは“関数”の役割を果している。コンビネータは、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、記号系列を書き換える (即ち、還元する) ことにより、ラムダ計算における λ 変換と同様の作用を、ラムダ計算独特の束縛変数という (やや扱いにくい) 概念を導入することなく実現する。

代表的なコンビネータの書き換え規則を以下に例示する。コンビネータ Y は、前章で述べた不動点コンビネータであり本稿の主役でもある。

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“引数”との並び [これをリデックス (redex) と称する] に関するコンビネータの作用の結果である。

```
S f g x  ==>  f x(g x)
K f g    ==>  f
I f      ==>  f
B f g x  ==>  f(g x)
C f g x  ==>  f x g
Y f      ==>  f(Y f)
```

注意すべき点は、上記のような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 sub 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、これらの演算をも含めて還元形式だけで実行可能とするには、各演算に関する書き換え規則を追加しておく必要がある。

例えば、後述の階乗プログラムの例題に登場する諸演算は、次のような書き換え規則を想定している。

```
sub   f   g   ==>  (f - g) という減算の実行値
tim   f   g   ==>  (f * g) という乗算の実行値
eq    f   g   ==>  “true”値          f = g   のとき
                  ==>  “false”値         f ≠ g   のとき
```

```
if  f  g  x   ==>   g          f = " true " のとき
              ==>   x          f = " false " のとき
```

関数型プログラム (の 2 項表現形) を組合せ論理コードに " コンパイル " する具体的な手順は [5] 等を参照していただくことにして、ここでは与えられた整数の引数 n の階乗を求める関数型プログラム例と、それの組合せ論理コードへのコンパイル結果 (目的コード長の最適化も施してあるもの) とを示しておくだけにする。但し、コンパイル結果に関しては、引数 n について abstraction したもの、および、更に関数名 fac についても abstraction したものを載せる。後者の版は不動点コンビネータ Y の支援を必要とするものである。

階乗の関数型プログラム (原始コード) は例えば次のように記述できる。

$$\mathrm{Fac}(n) \equiv \text{if } n = 0 \text{ then } 1 \text{ else } n * \mathrm{Fac}(n-1)$$

これを組合せ論理コードにコンパイルするには、2 項関係が明示的になる 2 項単位の括弧対表現が望ましいので、その形式による表現も記しておく。

$$\mathrm{Fac}(n) \equiv (((\mathrm{if}((\mathrm{eq}\ n)0))1)((\mathrm{tim}\ n)(\mathrm{Fac}((\mathrm{sub}\ n)1))))$$

上記プログラムを引数 n に関する abstraction でコンパイルすると、得られる結果である組合せ論理コード (目的コード) は次のようになる。

$$\mathrm{Fac} \equiv \mathrm{S}(\mathrm{C}(\mathrm{B}\ \mathrm{if}(\mathrm{eq}\ 0))1)(\mathrm{S}\ \mathrm{tim}(\mathrm{B}\ \mathrm{Fac}(\mathrm{C}\ \mathrm{sub}\ 1)))$$

この目的コードには (引数 n に関する abstraction ゆえ) 引数 n に相当する項が登場していないことに注目されたい。この目的コードを " 実行 " するには、引数の具体値を目的コード本体に後置させる適用形 (application form) を構成したあと、組合せ論理／演算評価系に基づく還元を施せばよい。

上記の目的コードの右辺には当然ながら再帰呼出し "Fac" が含まれている。この再帰呼出しが登場しないような目的コードを得るには、上記目的コードを関数名 Fac に関する
abstraction でコンパイルすればよい。その結果を仮に h と称することにすれば h は次のようになる。

$$\mathrm{h} \equiv (\mathrm{B}(\mathrm{S}(\mathrm{C}(\mathrm{B}\ \mathrm{if}(\mathrm{eq}\ 0))1))(\mathrm{B}(\mathrm{S}\ \mathrm{tim})(\mathrm{C}\ \mathrm{B}(\mathrm{C}\ \mathrm{sub}\ 1))))$$

目的コード h には実際に引数 n や関数名 Fac が一切登場していないことに留意されたい。この目的コードを " 実行 " するには、再帰呼出しを実現するための不動点コンビネータ Y を前置し、整数引数を後置すればよい。例えば、8 の階乗を求めるには (Y h 8) とすればよい。

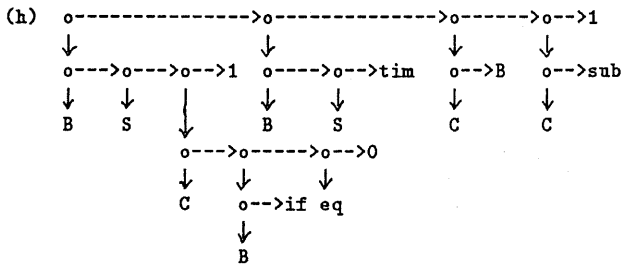
3 組合せ論理のグラフ還元

組合せ論理コードに関する還元の実行形態としては、本章で扱ういわゆるグラフ還元を用いることが少なくない。しかし、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ゆえ、以降では、ある特定のデータ構造 (一般にグラフ) で記号列を表現する状況を想定し、その記号列に施されるべき各書き換え規則が当該データ構造の特徴を反映して設定されている場合の還元形態を " グラフ還元 " と称することで議論を進めて行く。

組合せ論理のような 2 項関係の並びで構成される記号列を表現するデータ構造としては、ごく自然に 2 進木 (binary tree) の利用が考えられるが、その場合でも各項の配置の仕方は一意的には定まらない。ここではグラフ還元関連の文献で通常利用されている次のものを採用する。

即ち、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、左部分木→右部分木の順序で終端点に配置しながら部分木を構成しつつ上昇していく方式である。

例えば、前章の階乗計算の関数型プログラムの組合せ論理コード h を表現すれば次のようになる。

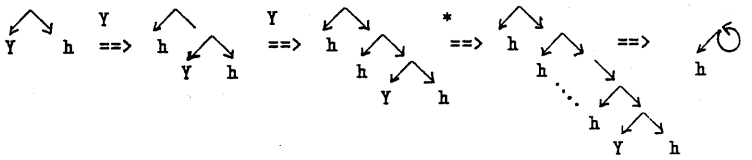


この目的コードの木表現自身には幸いにもサイクル構造が含まれていない。然るに引数 n に関する abstraction でコンパイルした目的コードには関数名 Fac が右辺にも登場し、それを Turner[5] は目的コードの木表現において根への有向辺というサイクル構造で表現したのである。(もっとも、サイクル構造を許した瞬間から、木表現ではなくなるが、この際は厳密さを欠く言葉遣いを許していただく。) その場合のグラフ還元における対処法を述べたのが [6] である。

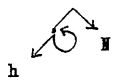
今回、上図の h の木表現にはサイクルが含まれていないのに何が問題になるかと言えば、

この目的コードを“実行”する際に前置する不動点コンビネータ Y の取扱いについてである。

Turner[5] は、Y の書き換え規則のグラフ表現として、次のような図を与えている。



上図の最右側にあるのは、紛れもなく自己ループ有向辺であり、従って (Y h M) という目的コードをグラフ表現すれば、右図のようなサイクル構造が出現することになり、新たなサイクル構造問題が浮上する。



Turner[5] はこの Y コンビネータのサイクル構造図の具体的な取扱い法について、前述の関数呼出しのサイクル構造の場合と同様に、一切言及していない。

従って、その図が、単なる等価表現の論理的説明のため [7] なのか、あるいは彼が実際にそのような構造を採用していることの表明のためなのかが曖昧である。

このようなサイクル構造を実際に設定してグラフ還元を (素朴に) 施すと、直観的には不測の事態となり得るかもしれない筈であるが、そのあたりの疑問に応える対処法が文献 [5] では読取れないのである。

それでは先ず、Turner が 2 進木上でグラフ還元を施すために左祖先スタック LAS(left ancestor stack) を用意し、その支援のもとに評価を進めている方法について簡単に述べる。

根から左部分木を下降して辿りつつ、点がセル［非終端点］の場合には、その点へのポインタを LAS に積み込んで更に下降する。

点が終端点［アトム］の場合、コンビネータか関数名のいずれかの筈であり、そのアトムを LAS に積み込むと共に、そのコンビネータ／関数が必要とする引数の個数分だけ遡って LAS の内容を調べる。LAS の各段はセルへのポインタだから、その右部分木の内容 (即ち、ポインタの指示するセル内の右項) を見ながらコンビネータ／関数としての評価を試みる。

必要な引数の個数分だけ LAS 内を遡ることができさえすれば、コンビネータの場合には常に評価が可能であるが、関数の場合には例えば四則計算のようにアトム引数を要求することがあるので必ずしもその時点では評価が可能とは限らない。

いずれにせよ評価が可能ならば書き換え規則に従って変換を行ない、その結果へのポインタが LAS に格納される。

評価が不可能の場合には、未評価のまま右部分木に移動して、更に続行する。

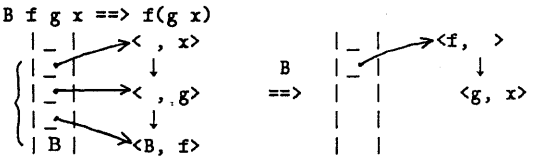
以上の操作がすべて不可能となる時点で評価が終了する。

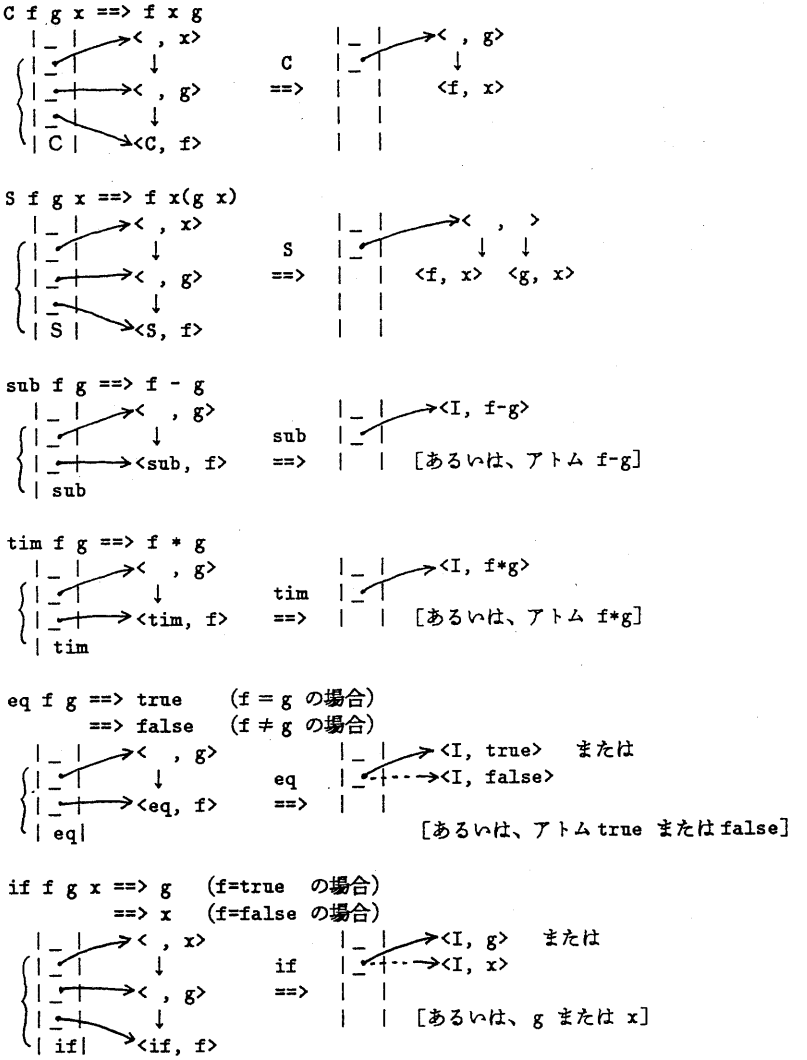
以下に示すのは、このような LAS を併用させた方式での書き換え規則の例である。

梯形は上方をスタックの底とする LAS である。角括弧対は 2 項セルを表わし、角括弧対内の空白項と下向き矢印で別の角括弧対へのポインタを示してある。

角括弧対内の左項にコンビネータ I を置く機会が生れるのは、四則計算のように結果が単一アトムとなる際に形式的に角括弧対表現を存続するための便法である場合や、if 演算のように結果が部分木となり得る際の根の間接点化のための場合がある。

f, g, x は部分木 (の根) を表わし、状況に応じてアトムの場合もありうる。





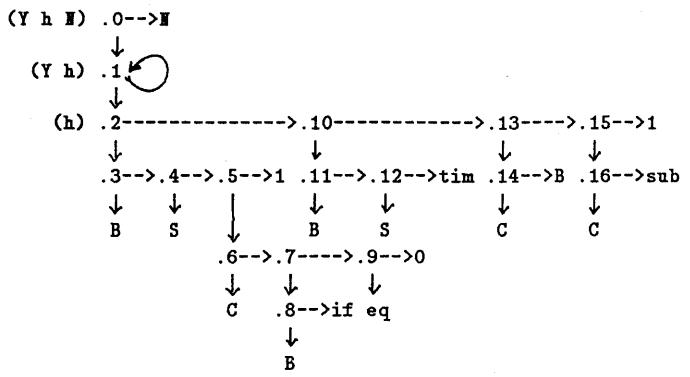
4 グラフ還元の実行

以上の準備のもとに、(Fac N) に相当する (Y h N) を評価する問題を考え、次のような2進木表現を設定する。(Y h N) 木は、勿論、((Y h) N) 木であるから、前章で図示した (Y h) に関する自己ループ有向辺が登場し、具体的には次のような図となる。

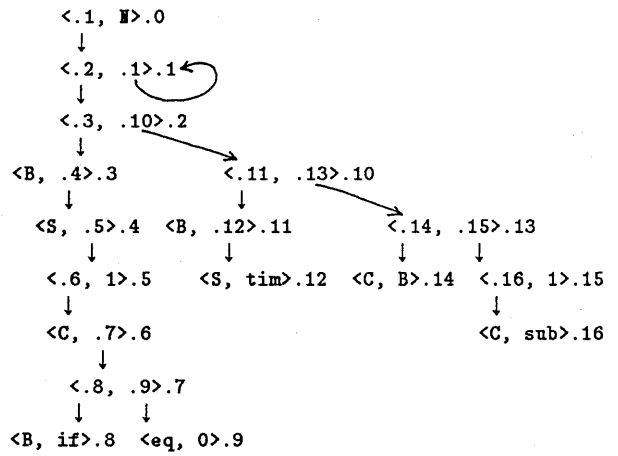
ここで記述を明確にするため、非終端点には preorder 順序［木を根→左部分木→右部分木と辿る順序］で 0 を出発値として昇順で番号付けしていき、k 番目の非終端点は“.k”として記すことにする。

この非終端点番号は、いわば当該非終端点の絶対番地に対応しているので、角括弧対で (他／自己の) セルを参照する場合のポインタの宛先表現としても利用できる。そこで、とくに出発状態のグラフに含まれる非終端点がグラフ還元の過程で存続しているか否かを明示する意味あいからも、この記述法を採用する。

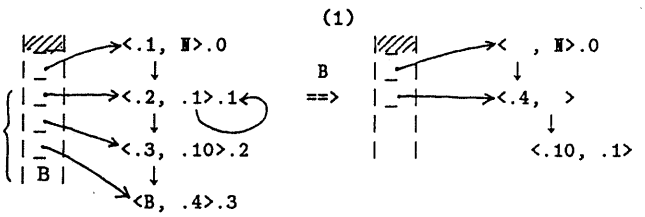
ただし、非終端点番号が角括弧対の右脇に添えてある場合は、その角括弧対自身（もちろん非終端点）の番地であり、角括弧対内部にある場合は、ポインタの宛先である。また、角括弧対の右脇が空白の場合には、そのセルが新設されたことを意味する。

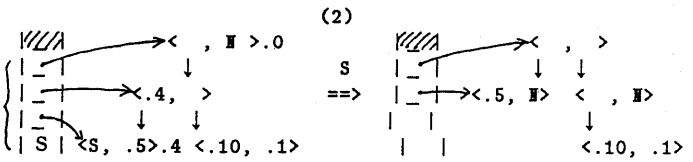


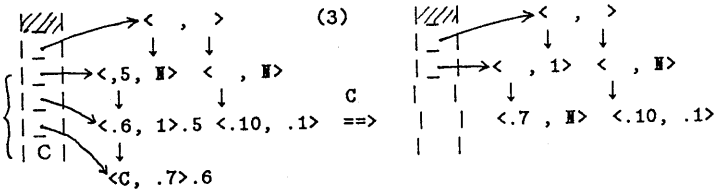
以上の便法を利用して、上図のグラフを角括弧対表現で図示すると次に示すような図が得られる。上から２つめの角括弧対には自己参照、即ち、自己ループ有向辺が登場していることに注意されたい。

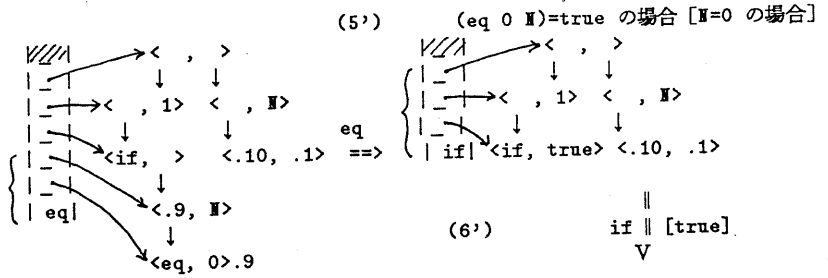


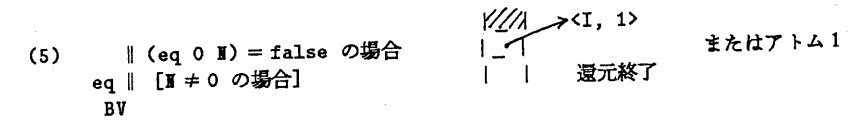
それでは、(Y h N) を評価するグラフ還元を、上述の書き換え規則およびデータ構造を用いて実施する過程を以下に示す。

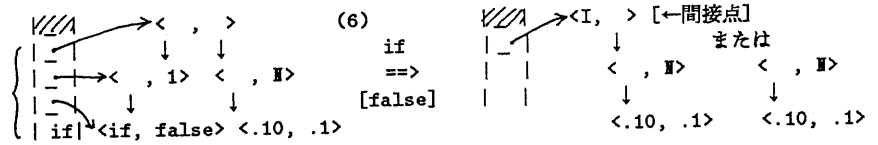


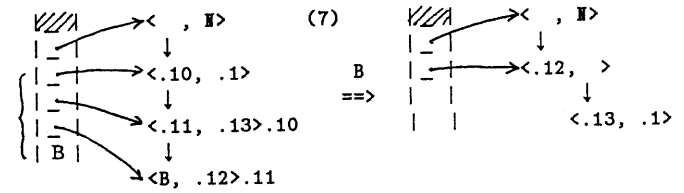


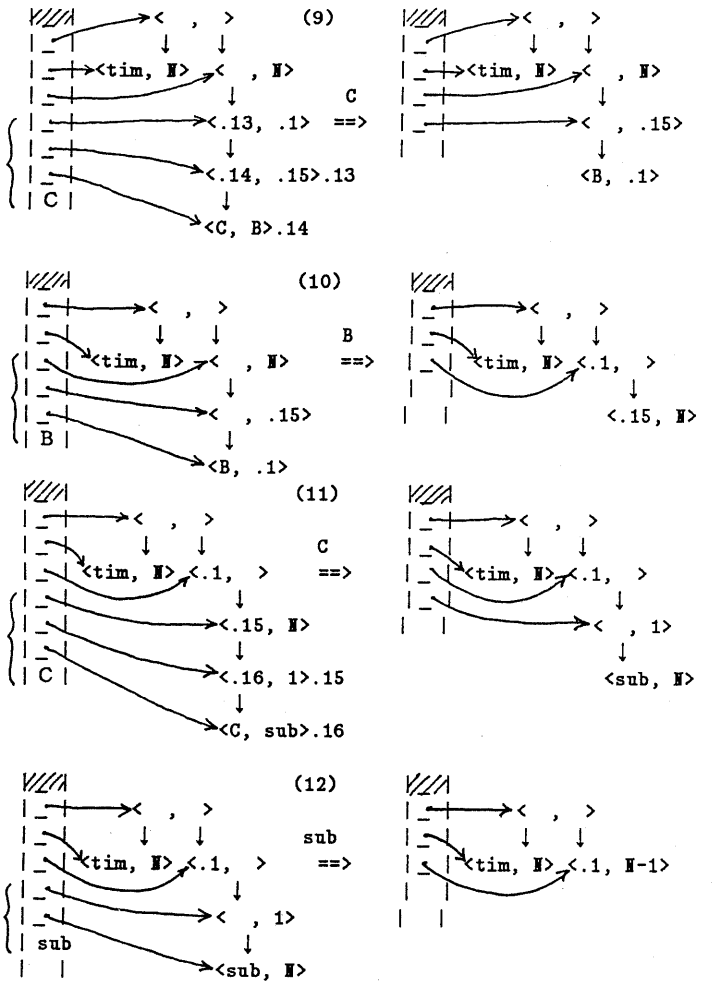

(3)
C
==>
<.6, 1>.5 <.10, .1>
<C, .7>.6
(4)
B
==>
<.7, N>
<.10, .1>
<.8, .9>.7
<B, if>.8
<if, >
<.9, N>


(5')
eq
==>
<if, >
<.10, .1>
<.9, N>
<eq, 0>.9
(eq 0 N)=true の場合 [N=0 の場合]
<if, true> <.10, .1>
(6')
if ‖ [true]
V


(5)
‖ (eq 0 N) = false の場合
eq ‖ [N ≠ 0 の場合]
BV
<I, 1>
還元終了
またはアトム 1


(6)
if
==>
[false]
<if, false> <.10, .1>
<I, > [←間接点]
または
<, N>
<.10, .1>

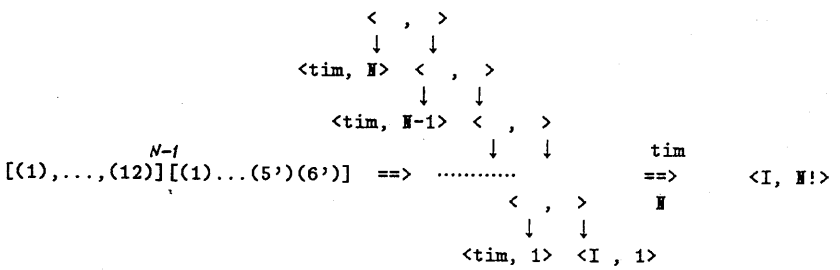

(7)
B
==>
<.10, .1>
<.11, .13>.10
<B, .12>.11
<.12, >
<.13, .1>
(8)
S
==>
<S, tim>.12 <.13, .1>
<tim, N>



ここでFacの再帰呼出しに相当する".1"への回帰という、不測の事態になり得る箇所に到達することになる。この".1"への回帰は、段階(1)での自己ループ有向辺というサイクル構造の対処法が深く関与している筈であるが、文献[5]では具体的に言及されていないのである。

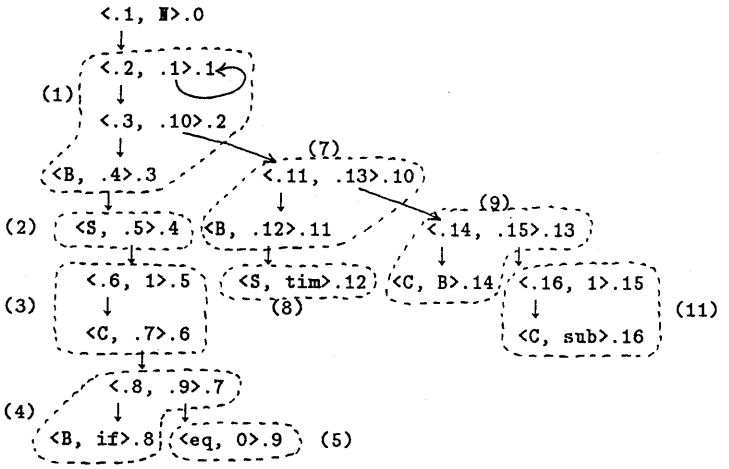
5 サイクル構造への対処法

前章の".1"への回帰という問題が何らかの方法により解決されるならば、階乗プログラムのグラフ還元は次のように進行していく筈である。



この問題の解決の鍵は、前章のグラフ還元の実行過程の段階 (1) に存在していたのである。即ち、段階 (1) の書き換え図の左辺にある自己ループ有向辺を有するセル (つまり ".1" 番地のセル) を根とする部分木が、書き換えにより同図の右辺では消去されている訳であるが、このとき当該部分木を廃棄せずに保存しておけばよいのである。なぜならば、その部分木は根の番地 (今の場合は ".1") さえ指定すればいつでも復元できるからである。

あとは、同様な要領で前章のグラフ還元過程の各段階の左辺について、保存すべき部分木か否かを検討していけば、出発時の木構造が次の図のような過程で復元できることを容易に導ける。



ここで (n) として囲まれている部分は、前章の還元過程の n 段階における書き換えに際して保存すべき左辺の部分木 (場合によっては単一セル) である。上図から明らかなように、6(あるいは 6')、10、12 の各段階における左辺は出発時の木構造の再現に寄与していないので、廃棄可能であることが分る。

6 段階の左辺には (if, false) が、6' 段階の左辺には (if, true) が、12 段階の左辺には (sub, N) が、それぞれ含まれているが、false/true や N の値は演算環境 (即ち、還元過程の進行状況) に応じて変化し得るものである。このことから一般に、左辺の中に、アトムとして評価される値が演算環境に応じて可変であり得るものを含む場合には、それらは廃棄可能であることが推定される。ただし、10 段階に関しては、この説明では意義付けがあてはまらないようである。

結局のところ、書き換えの左辺 (即ち、リデックス) に対応する部分木が、還元の出発時に存在していたセル (換言すれば、番地を付けられたセル) を含む場合には、書き換えのあとでも保存する必要がある、という極めて当然の判断基準が存在していることだけは事実である。

この ".1" セルを根とする部分木の再現が不要となるのは、前章のグラフ還元過程が 5' 段階に到達するときであり、このとき初めて ".1" セルを根とする部分木全体を廃棄してもよいことになる。これ以後のグラフ還元過程は本章冒頭の図の中央にある乗算の木を対象として進行する。そして、いずれも書き換えの際にリデックスを保存する必要のないものばかりであり、最終的には N! という値のアトムだけが残される。

6 おわりに

関数型プログラムを組合せ論理の還元依存で処理する方式においてグラフ還元を利用する場合、Turner[5] が図示するだけで殆ど言及しなかった組合せ論理コードのグラフ表現上のサイクル構造について、その対処法を不動点コンビネータ Y に的を絞って検討した。

その結果、彼の提案した特殊スタック LAS をグラフ表現と併用する枠組みでは、グラフ還元過程の各段階のリデックスを適宜保存すれば、サイクル構造を維持しつつ再帰関数を評価できることが判明した。しかも、それは以前に筆者が [6] で述べた再帰呼出しのサイクル構造の場合と同様の対処法である。

グラフ還元の利点として挙げられているものに、サイクル構造と並んで共有構造がある。階乗プログラムの例題では、共有構造特有の問題を孕む唯一の可能性がある S コンビネータが、アトム (整数 N) の共有の場合だけで終っているので、[6] と同様に共有構造とサイクル構造とが絡み得る問題については議論していない。

この問題を検討することや、上述の LAS のような特殊スタック類を一切用いずにグラフ表現だけという枠組みでサイクル構造問題を検討することは今後の課題である。

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、およびご助言や研究環境等で日頃ご支援いただく当研究室各位をはじめ関連諸氏に対して深謝する。

References

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

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

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

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

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

[6] 杉藤：“関数型プログラムとグラフ還元”、情報処理学会ソフトウェア基礎論研究会、36-2(1990年9月21日).

[7] W.M.Farmer,J.D.Ramsdell,R.J.Watro:"A Correctness Proof for Combinator Reduction with Cycles", ACM Transaction on Programming Languages and Systems, Vol.12, No.1, pp.123-134, January 1990.