数理解析研究所講究録
第 581 巻 1986 年 146-153


# 国産数式処理システムGALにおけるパターンマッチング

SA SA KI TATEAKI　　MOTOYOSHI FUMIO
佐々木建昭，　元吉文男
（理　研）　　（電総研）

## 1. はじめに

現在開発中の国産数式処理システムGALにおいては、パターン・マッチングを非常に重視している。パターン・マッチングは数式処理の応用において非常に多用され，利用価値が高い機能であることはREDUCEのLET文が示すところであるが、将来実用化されるであろう数学公式データベースの運用を考慮すると、その重要性は我々の予想をはるかに越えるものだと言わなければならない。筆者らを含め、ほとんどの数式処理システム開発者は、パターン・マッチングを数式処理の基本的な機能の一つであると認識している。したがって、パターン・マッチングそのものを目的とすることも当然あるが，それよりも，パターン・マッチングを利用して不定積分の計算や微分方程式の求解を実行することをより重視するのである。数式処理システムにおけるパターン・マッチャーのこの位置づけには留意する必要がある。

数式のパターン・マッチングで我々が最もよくなじんでいるのはREDUCE [1] のLET文であろうが，REDUCEに限らず，多くの数式処理システムがパターン・マッチング機能を有している。これまでに最も高級なパターン・マッチング機能をサポートしたのは恐らくMACSYMA [2] であり、それに近い機能を有したシステムにSMP [3] がある。GALのパターン・マッチャーはMACSYMAのそれと同程度のものである（ただし、きめ細かさの点でMACSYMAが，ユーザの使い易さの点でGALが，それぞれ優る）と考えている。

本稿では、数式パターン・マッチングの（大雑把な）一般論，GALにおけるパターン・マッチャーの紹介に加えて、パターン・マッチングによる数式の簡単化が応用上実に有用であることを示す。

## 2. 数式パターンとパターン変数

数式パターンとは，変数として実変数以外に“パターン変数”を含み得る点を除けば、通常の数式と同様である（ただし、システム・インプリメンテイションの都合上，扱いうるパターンに制限を加えるシステムが多い）。パターン変数とは一般に任意の数式あるいは部分式とマッチし得る変数である。ただし、パターン変数に条件が付けられているときには，その条件を満足する数式としかマッチしない。パターン変数は，通常の変数と区別するため，陽に指定する必要がある。たとえばMACSYAではA，B，Xがパターン変数であることを

MATCHDECLARE([A, B, X], pred1, pred2, ...)

と指定する。ここで，pred1，pred2，...はA，B，Xを限定する条件文である。この方式ではパターン変数がグローバルに定義されることになる。一方、SMPではパターン変数は頭に“$”あるいは“$$”をつけることにより指定する。たとえば$X，$$Xがそうである。ここで，$Xは一つの項とのみマッチし、$$Xは複数個の項ともマッチし得る。$Xや$$Xに条件が付けられることはMACSYMAと同様であるが、方式は異なる（以下をみよ）。

GALにおけるパターン変数の指定法はSMPにおけるそれと類似である。すなわち"@"を頭につけた変数がパターン変数である。ただし、SMPのように@X と@@X を区別せず、どちらも(条件があればそれを満たす)任意の数式とマッチするものとして扱う。SMP方式とGAL方式のどちらがよいかは異論のある所であろうが、たとえば

exp(X+Y+Z) ⟶ exp(X)・exp(Y)・exp(Z)

の変形が、GALにおいては

exp(@X +@Y) ⟶ exp(@X)・exp(@Y)

なる公式で実行されるが，SMPでは注意深く

exp($X +$$Y) ⟶ exp($X)・exp($$Y)

としなければならない。この複雑さの代償にSMPではもつき変数のマッチングがきわめて効率よく実行される。

GALでは数式パターンとして，集合式，関係式，論理式，配列、およびベクトルや行列等の構造体数式は許していないが、それ以外の普通に我々が数式と考えるものは何でも許している。さらに、パターン変数は関数名であっても(そのときは実関数名とのみマッチする)，変数や関数の添字であっても、またベクトルや行列のインデックスであってもよい。

パターン変数には一般に条件がつけられ、MACSYMAではMATCHDECLAREで条件を指定することを述べた。SMPでは数式パターン中に現れたパターン変数の直後に(必要ならば)条件を付ける。たとえば

Eul[$n_= Natp[$n]]

は任意変数(=パターン変数)$n の関数Eul[$n]を与えているが，変数$nのあとに続くNatp[$n]が条件である。一方，GALでは条件を

<u>pattern</u> WHERE <u>Conditions</u>

の形で，数式パターンとは分離して与える。ここで，WHEREのかわりにWITHを用いてもよく，条件はANDとORでつなげばいくつ与えてもよい。たとえば，SMPに対する上例をGALで与えると

Eul(@N) WHERE ?INT(@N) AND @N >=0

と表現できる。SMP方式では2個以上のパターン変数にまたがる条件は表現しえないが、GALおよびMACSYMAの方式ではそれが可能であることを指摘したい。なお、MACSYMAと異なりGALおよびSMPでは，パターン変数の仕様は局所的である，すなわちパターン変数は一つの数式パターン内でのみ同一視される。したがって，別個のパターン中で同一名のパターン変数が使われても、それらには何の関係もない。

以上を総括すると、MACSYMAとSMPおよびGALで数式パターンの仕様に差はあるものの，本質的な機能には大差ないと言えよう。

## 3. パターン・マッチングによる簡単化

数式パターン・マッチングの詳細に入る前に，パターン・マッチングを用いた数式の簡単化を述べておくことは，パターン・マッチャーの有用性のみならず，パターン・マッチャーがどういうものかを理解するのに役立とう。

GALでは数式の簡単化（simplification）とは，left ⟶ right なる書き換え規則による数式の変形であると割り切っている。書き換え規則（ルール）には

i) システム組込みのルール，
ii) ユーザ定義の open ルール，
iii) ユーザ定義の boxed（packed）ルール，

の3種があるが，ここでは後2者を説明しよう。

open ルールが入力されると，それ以後そのルールが消去されるまでに計算されるあらゆる数式（ただし，計算の最終結果式のみ）にそのルールが適用される（したがって，open ルールが多数あると計算速度はガタ落ちする可能性がある）。各ルールは

<u>left</u> ——> <u>right</u>;　あるいは
<u>left</u> ——> <u>right</u>　WHERE　<u>conditions</u>;

なる形式で与えられ，RULE 文により

RULE　**rule1, ..., rulen**;

と定義される。ルールを消去するには UNRULE 文が使える。

一方，boxed ルールとは名前をつけられた一群のルールであり，その名前が呼ばれて初めて有効となる。boxed ルールは RULEBOX 文により

RULEBOX　sin = { **rule1, ..., rulen** };

のように定義される。上例では"sin"が boxed ルールの名前であるが，例が示すように名前は関数名，あるいは変数名と同じでもよい。このルールは

SIMP(expr, sin, ...);

のようにして起動される。ここで，expr は簡単化の対象となる数式であり，上例では boxed ルール sin, ... を起動している。boxed ルールは必要な場合にのみ起動されるので，open ルールを用いるよりも効率的であることは言うまでもなかろう。その代償に，ユーザは簡単化のたびに SIMP コマンドを発行しなければならない。

ルールが最も多用されるのは特殊関数を含む数式の計算であろうが，しかし単なる多項式の簡単化においても絶大な威力を発揮するのである。次頁の例1を見よう。この例では，入力形がそのまま保存されていれば式は簡単だが，展開された形式では元の式の明解さは完全に失われてしまっている。応用分野の計算では例1の出力形のような不明解な式が頻繁に出現する。例1の出力形を入力形のように簡単化したいのであるが，因数分解を試みても無駄である。人間がよくやる方法は，うまくまとまりそうな形に当りをつけてそれを試すことである。そこで，X+Y−Z でまとまりそうだと当りをつけたとしよう。このとき，GAL では X+Y−Z ——> W なるルールを定義すればよい。例2は GAL でそれを実行したも

のである(2次元出力ルーチンはまだプログラムされていない。ルールにおいて"—>"は"—>"でも"==>"でも"=>"でもよい)。

```
    A2 := (X+Y-Z)*(X**2 + Y**2 - Z**2) + X*Y*Z ;

 A2 := X**3 + X**2*Y - X**2*Z + X*Y**2 + X*Y*Z - X*Z**2 + Y**3 - Y**2*Z
       - Y*Z**2 + Z**3

    A4 := (X+Y-Z)**2*(X**4 + Y**4 - Z**4) + A2 - 2*X*Y*Z ;

 A4 := X**6 + 2*X**5*Y - 2*X**5*Z + X**4*Y**2 - 2*X**4*Y*Z + X**4*Z**2
       + X**3 + X**2*Y**4 + X**2*Y - X**2*Z**4 - X**2*Z + 2*X*Y**5 - 2*X*Y**4*Z
       + X*Y**2 - 2*X*Y*Z**4 - X*Y*Z + 2*X*Z**5 - X*Z**2 + Y**6 - 2*Y**5*Z
       + Y**4*Z**2 + Y**3 - Y**2*Z**4 - Y**2*Z + 2*Y*Z**5 - Y*Z**2 - Z**6 + Z**3
```

例1. 簡単化の対象となる多項式 A2 と A4 (展開形)

```
    RULE X+Y-Z ==> W ;        (ルールの定義)

 "RULE DEFINED"

    A2;       (A2の値をみる)

 W*X**2 + W*Y**2 - W*Z**2 + X*Y*Z

    A4;       (A4の値をみる)

 W**2*X**4 + W**2*Y**4 - W**2*Z**4 + W*X**2 + W*Y**2 - W*Z**2 - X*Y*Z
```

例2. ルールで簡単化した結果(オーダリング: W > X > Y > Z)

見てわかるように, A2 と A4 の入力形がほぼ再現された(入力形を完全に再現するにはWについてまとめた形式に変換するだけでよい)。

別の例を見よう(現在入力ルーチンとしてREDUCEのそれを借用中なので@の前に!が必要。近く自前の入力ルーチンを組むが、そうすれば!は不要):

```
       RULEBOX sin = sin(!@X)**2 + cos(!@X)**2 -> 1 ;     (ルールボックスの定義)

    "RULEBOX DEFINED"

       B3 := (sin(X) + cos(X))**3 ;

    B3 := sin(X)**3 + 3*sin(X)**2*cos(X) + 3*sin(X)*cos(X)**2 + cos(X)**3

       B4 := (sin(X) + cos(X))**4 ;

    B4 := sin(X)**4 + 4*sin(X)**3*cos(X) + 6*sin(X)**2*cos(X)**2 + 4*sin(X)
          *cos(X)**3 + cos(X)**4

       SIMP(B3,sin);      (B3を簡単化)

    2*sin(X)**2*cos(X) + 2*sin(X)*cos(X)**2 + sin(X) + cos(X)

       SIMP(B4,sin);      (B4を簡単化)

    1 + 4*sin(X)**2*cos(X)**2 + 4*sin(X)*cos(X)
```

例3. boxed ルールによる簡単化の例

この例では B3 と B4 を sin と cos に関して対称的な数式とした。ルールも同様なので、簡単化された結果も sin と cos に関して対称的になっていることに注意されたい。物理や工学の計算、特にベクトル量を含む計算では、対称性を保持し

つつ計算すれば式がうまくまとまり、結果の物理的・工学的意味が理解し易いが，そうでなければみじめな結果に終ることはよくある。この点で，GRLの簡単化は多項式などの数式計算においても強力な武器になるものと考えられる。

## 4. 数式パターン・マッチング概論

前節で述べたルールによる簡単化では

①まず left のパターンとマッチする部分式を探す，

②left に条件が付与されていればその条件をチェックする，

③マッチする部分式が見つかればそれを right で置き換える，

という手順を繰り返している。したがって、その本質はパターン・マッチングであると言える。

パターン・マッチャーには，どの程度の機能をサポートするかによって種々のレベルが存在する。最も簡単なものはシンタクティカル・マッチングであろう。すなわち、数式を表現しているデータ（リストで表現されることが大部分である）を先頭からスキャンして，パターン変数を適当な部分式で置き換えさえすれば同一のデータになるときのみ、マッチングが成立するとみなすものである。この方式は多くの項書き換えシステムで採用されているものであり、Prolog などでユニフィケイションと名付けられているものでもある。SMP では＄変数のマッチングはこの方式で実行する。

しかしながら、実用的な数式処理システムにおいては，上述の簡単なパターン・マッチャーでは明らかに不十分である。少なくとも項や因子の可換性，省略された因子1と指数0，を考慮したセマンティカル・マッチングが不可欠であり、さらに内部表現の非一意性なども考慮する必要がある。これらの詳細と対策は次節で述べる。

数式パターン・マッチングを一般的に論じる場合、次の2点が重要である。

(i) 数式パターン・マッチングは一意的ではない（たいてい無限通りのマッチングが可能）。このことは次の等式から自明であろう：

$$A + B = (A+C) + (B-C),$$
$$A * B = (A*C) * (B/C).$$

(ii) 数式パターン・マッチングは単純ではない（多くの場合，方程式を解くことに帰着）。たとえば次の例がそうである：

$$\frac{d^2}{dx^2} @F(x) + @F(x) \Leftarrow \mathrm{match} \Rightarrow 5.$$
$$\therefore\ @F(x) = a \cdot \sin(x) + b \cdot \cos(x) + 5.$$

一般的な数式パターン・マッチングにおける代表的な問題点を筆者なりにまとめてみた（以下のもの以外にもあるかも知れない）。

(a) combinational パターン・マッチング（複数個の項または因子を同時に考えて初めてうまく決まるマッチング）。

例4 $X^{@N} Y^{@N} @Z \Leftarrow \mathrm{match} \Rightarrow X^7 Y^5$,

答：$\{@N=5,\ @Z=X^2\}$

例5　$@X^4\ @Y\ + @X^2 \Longleftarrow \mathrm{match} \Longrightarrow X^{12}Y^2 + X^4$
　　答：$\{@X = X^2,\ @Y = X^4Y^2\}$.

例4において、左から式をスキャンして@N=7としたのではマッチングは失敗し、同様に例5において @X=X^3 としてはマッチングは失敗することに注意されたい。上例のようなマッチングを実行するには、たとえば例4では@Nに対する不等式を作り、数式を全部スキャンし終えた後にそれを解くか、あるいはバックトラッキングにより一度決めたマッチングを修正することが必要である。これらを一般的にサポートするのは容易ではない。

(b) group パターン・マッチング（いくつかの項あるいは因子をまとめて初めてうまく決まるマッチング）。

例6　$2(X+1)\cdot \exp(X^2)\cdot \exp(2X) \Longleftarrow \mathrm{match} \Longrightarrow @F(X)\cdot @A$　WHERE　?FREEOF(@A, exp)
　　答 $\{@F(X) = \exp(X^2)\cdot \exp(2X),\ @A = 2(X+1)\}$.

この例のようなマッチングを完全に実行するには、項あるいは因子のあらゆる組合せを考慮する必要があり、マッチングに要する時間が大巾に増加する。特に、対象数式が因子に分解されていない場合には因数分解しなければならないが、因数分解は非常に高価な演算である。

(c) equationalパターン・マッチング（方程式を解くことにより初めてうまく決まるマッチング）。

例7　$\sin(@X)\cdot DF(\sin(@X)) \Longleftarrow \mathrm{match} \Longrightarrow \sin(X)\cdot \cos(X)$,
例8　$2\sin(@X)\cdot @X \Longleftarrow \mathrm{match} \Longrightarrow \pi$,
　　答：$\{@X = \pm\pi/2\}$.

例8のようなマッチングをサポートするには高級な求解ルーチンが必要である。
　以上の難しいマッチングをどの程度サポートするかは、システムにおけるパターン・マッチャーの位置付けによるが、「パターン・マッチャーは高度であればあるほど良い」という訳でないことを指摘したい。実際、例8のように超越方程式を解いてマッチング結果を出されても、ユーザはとまどうだけであろう。

## 5. GALにおけるパターン・マッチャー

　1節に述べたごとく、GALにおいてはパターン・マッチャーは主としてアルゴリズム・インプリメンテイションのための基礎的機能と位置づけられているが、そのためには次の性質が要求されるため機能が制限される：
(イ) 実際的計算時間でパターン・マッチングが実行できること，
(ロ) 唯一のマッチング結果を出すこと，
(ハ) 人間が出すであろう結果とできる限り一致すること。
これらの要求は相互に矛盾することもあるから、パターン・マッチャーがこれで一意に規定される訳ではないが、プログラミングの指針にはなる。
　上述の制約のため、前節に述べた難しいマッチングはそのほとんどが排除される。すなわち、難点(a)は(イ)に反するからサポートしない。難点(b)は(ハ)の観点から部分的にはサポートするものの，それは(イ)と(ロ)の観点からごく単純なもの（たとえば除算で分離できる因子とのマッチング）に限られる。難点(c)

は整数のべき乗根因子を計算することを除いてサポートしない。数式パターンでは一般に多数通りのマッチングが存在することを述べたが、(ロ)と(ハ)の観点からそのうち最も妥当らしいものを一つだけ出力することになる。したがって，GALのパターン・マッチャーにはシステム・プログラマの恣意性が不可避的に入り込むことになる。しかしながら、上述の制限を加えることにより、項数$m$の多項パターンと項数$n$の数式のマッチングに要する時間は$O(m \times n)$に押えられている。これは，定数因子を除けば、多項式の乗算と同程度の計算量であり、GALのパターン・マッチャーは十分効率的と言えよう。

上記(ハ)について説明を追加しておこう。たとえばユーザが @X * @Y * @Z なるパターンを与えたとしよう。これと項$X^2$の間には幾通りものマッチングが考えられるが，GALは @X = $X^2$ なるマッチングをまず求める。そうすると @Y = @Z = 1 となる。これを答としてもよいが，GALは"FAIL"との答を返す。なぜなら，ユーザが @X * @Y * @Z なるパターンを入力したことは，「ユーザは3個以上の因子をもった項を探している」と解釈するのが最も自然だからである。因子のうち1個は数因子であってもよいから、GALは$X^2$を @X * @Y とはマッチさせる(@X = $X^2$, @Y = 1)が、@X * @Y * @Z とはマッチさせないのである。同じように、@X / @Y は分数形の数式とのみマッチさせる。

GALにおけるパターン・マッチャーは，全体パターン・マッチャーと部分パターン・マッチャーに大別され，それぞれ次のように起動される：

(A) MATCH(expr, pattn),
(B) PARTMATCH(expr, pattn).

ここで、expr はマッチングの対象となる数式であり、pattn は(一般にパターン変数を含む)パターン数式である。パターンには

<u>pattern-expr</u> WHERE <u>conditions</u>

のように条件を付けることができる。MATCH は pattn が expr 全体とマッチするときのみ、PARTMATCH は pattn が expr の一部あるいは全体とマッチするときに、各パターン変数とマッチする数式の対の集合を答として返す。マッチングが失敗すれば"FAIL"を答として返す。

<u>例9</u> MATCH($X^2+2X+1$, @X + @Y);
答 {@X = $X^2$, @Y = $2X+1$},

PARTMATCH($X^2+2X+1$, @X + @Y);
答 {@X = $X^2$, @Y = $2X$}.

パターンはシステム内部で単項パターンと多項パターンに大別される。単項パターンは基本的には単項の数式とマッチさせるが，パターンが @X のように単変数のときには数式全体とマッチさせることもある。単項数式と単項パターンをマッチさせるには，パターンの中のパターン変数を含まない因子を対象数式から除き，しかる後にパターン中の残った因子を対象数式の因子と左から順にマッチさせていく。すなわち，項は

TERM = <left-FACTOR> * <rest-FACTORs>

と分割することを基本とする。多項パターンは基本的には多項の数式とマッチさせるが、単項数式中の因子に多項のものがあればそれとマッチさせることもある。また、TERM + @X の形の2項パターンは @X=0 として単項式とマッチさせることもある。多項数式と多項パターンをマッチさせるには、これらの数式を左の項から順にマッチさせていく。すなわち、数式は

数式 = <left-TERM> + <rest-TERMs>

と分割することを基本とする。項中の因子は

因子 = <BASE>**<EXPONENT>

と分解して、底部と指数部を個別にマッチさせる。その際，底部のマッチングを優先することはもちろんである。

最後に、多項パターンのマッチングによる簡単化について説明しておこう。例として，RULE 文，あるいは RULEBOX 文により

sin(@X)**2 + cos(@X)** → 1

が入力され，次式を簡単化する場合を考えよう：

(A) $2\sin^3(X) + \sin(X)\cdot\cos^2(X) + \sin^2(X)\cdot\cos(X)$.

多項パターンがルールとして入力されると、システムはルールの両辺にシステム用パターン変数である@EXTRA@ を乗じる：

(B) $\sin(@X)^2\cdot@EXTRA@ + \cos(@X)^2\cdot@EXTRA@ \rightarrow @EXTRA@$.

次に(A)と(B)の部分のマッチングを行うが，@EXTRA@ をやや特別に扱う。まず、(A)と(B)の第一項どうしのマッチングにより $\{@X = X,\ @EXTRA@ = 2\sin(X)\}$ が得られる。次に、この答を(B)の第二項に代入すると $2\sin(X)\cdot\cos^2(X)$ が得られるから、この項に数因子を除いて一致する項を(A)から探し、第2項を見い出すと @EXTRA@ の数因子を1に修正する。かくして、(A)は

(A') $\sin^3(X) + \sin(X) + \sin^2(X)\cdot\cos(X)$

に書き換えられる。(A')と(B)はもはやマッチしないから(A')が答となる。

参考文献

[1] Hearn, A. C., "REDUCE user's manual", Version 3.0, The Rand Corporation, 1983.
[2] The MATHLAB Group, "MACSYMA reference manual", Version 9, Lab. Computer Science, MIT, 1977.
[3] Cole, C. A., Wolfram, S., et al., "SMP handbook", version 1, CALTEC, 1981.