数値・数式混合処理システムのPrologによる開発

野田松太郎，　武能芳弘
（愛媛大・工）

1．緒言

最近、数式処理システムへの関心が深まり色々な視点からの研究がなされている。また、REDUCE－3、MACSYMA等の著名なシステムが各種大型計算機上で稼動している。数式処理は全般には、もはや「実験研究」の段階から、「実用」の時代へ突入しているように思われる。これらシステムでは，数式を入力することにより、その簡素化，微分、積分あるいは微分方程式の求解等を数式のまま遂行する点に特色がある[1]。

しかし、良く知られているように、工学分野に出現する大半の問題――その多くは微分方程式によるモデル化により表現される――では、厳密な解析的計算のみで目的が達っせられるものは皆無といってもいい過ぎではない。必然的に近似計算そして数値計算に頼ることになる。この種の要求に答えるべく、上記のREDUCE等の数式処理システムでも技術計算用のFORTRANプログラムの発生も行ないはするが、十分に利用者の要求を満しているとはいえない。この点に着目し、微分方程式を対象にし、数値計算プログラムを自動的に生成するようなシステムの構築に関する研究もなされつつある[2]。しかし、この種のシステムでは、たしかに微分方程式をそのままの形で計算機に入力するものの、数式処理を行なわないため、色々の困難がつきまとう。すなわち、入力される微分方程式がうまく整理されていなければ、やたら複雑なプログラムが生成されるし、解析的に解ける場合も数値計算を遂行することになる。

そこで、本研究では、上に述べた2つの面を統合することを考える。我々の究極の目的は、利用者が数式のまま入力する微分方程式等について、

1）式の簡素化、統合等の処理により標準的な形に変形し、
2）厳密解（解析解）の存在を検討し、
3）解析解の存在しない場合には、対応する数値計算プログラムの発生を行なう。

というシステムの構築である。さらに、これを多様な利用者の要求に対応すべく、小型計算機上に実現しようとする。

本稿では、上記の目的を達成するための第一段階について報告する。すなわち、このシステムの基礎となる処理系の作成であり、それによる数式の簡素化さらに数値・数式混合処理に対してである。

基礎となる処理系として一般的に考えられるのは、大半の数式処理システムが依っているLISP言語によるもの、高速に数式処理が遂行し得るといわれるSMP[3]が依っているC言語あるいは、微分計算に限ってではあるものの数式処理と数値計算の結合に対し効果的な[4]PASCAL言語などがある。一方、Bundy等は[5]、数式パターンの処理に有効なProlog言語を用いて、簡単な数式の統合あるいは、大学入試程度の問題の解等について十分な成果を発表している。我々の場合，数式の処理に関しては確かにLISP、Prolog系の言語が、数値・数式混合処理系ではむしろ、C，PASCAL等の言語が有力であることに注意する必要がある。また、色々な利用者が小型計算機上でも稼動さすことを考えると、約

植性の良い言語の採用が好まれる。以上の諸点を踏まえ、「Cにより記述されたProlog」を採用することにした。ただし、特に数値・数式混合処理部等を考えると、上のようなPrologを始めから作り上げることが好ましいように思われる。もちろん、今後の数値計算プログラムのデータベース化等を考えると、関係データベースそのものともいえるPrologの使用が好ましいことは明らかである[6]。

以上に述べた、Prolog処理系の特色を、Prolog言語そのものの概要とともに2にまとめ、これによる数式処理の流れを3に述べる。数値計算との簡単な結合例は4に、また今後に残された問題点等はちにまとめる。

2．Prolog処理系について

PrologはKowalski，Colmerauerにより提案された[7]、一階述語論理を基礎にした言語である。最近では、第5世代計算機のソフトウェア技術の中心となる核言語に採用され、その使用の便のため急速に普及している。多くの解説書も出版されているが[8]、以下本稿の一貫性のためにその簡単な概要と、本システム構築のために作成したProlog処理系の特色についてまとめる。

2－1．Prologの概要

Prologは次のような特徴を持っている。

1）パターンマッチング操作による述語の呼び出し
2）パターンマッチング操作によるリスト処理
3）非決定的処理（バックトラッキング）
4）入力変数と出力変数の区別をしない

以下の便のため、次のプログラムを考えてみる。

```
append((),$X,$X) ←
append(($A|$X),$Y,($A|$Z)) ← append($X,$Y,$Z)
```

このPrologプログラムは2個のリストの結合を表わしており、2つの文 —— これをホーン節という —— よりなっている。ホーン節を構成する各語を述語という。第1のホーン節は、述語appendにより空リスト（）と$Xより$Xが生成されることを主張し、第2のものでは、第1要素が$A、残りが$Xのリストと$Yをappendすると第1要素が$A、残りが$Zのリストが生成されることを主張している。ここで、$P, Q_1, Q_2, \cdots, Q_N$を述語とすると、一般にホーン節は次の3種類のいずれかの形式をとる。

a） $P \leftarrow Q_1, Q_2, \cdots, Q_n$

Pを実行するためには、$Q_1, Q_2, \cdots, Q_n$を実行すればよい。上のプログラムの2番目のホーン節は、$n=1$の場合の例である。

b） $P \leftarrow$

無条件にPが成立している。上のプログラムの1番目のホーン節に相当する。

C） $?-Q_1, Q_2, \cdots, Q_n$

$Q_1, \cdots, Q_n$を実行せよ。または、$Q_1, \cdots, Q_n$を同時に満足する解を求めよという処理系への実行命令、または、質問である。上のappendのプログラムに対し、

```
?-append((a,b),(c,d),$X)
```

により解を求める。$Xは（a，b，c，d）というリストになる。

ここで、上に述べたPrologの特徴をまとめ直すと次のように言える。

1）Prologにおける述語は、パターンマッチング操作（以下、これをユニフィケーションと呼ぶ）によって呼び出される。また、変数とその値を結合する働きをユニフィケーションは持っており、述語中の変数に適当な値を代入することによって2つの述語を同一にすることを目的としている。たとえば、

```
color(apple,$X)
```

と

```
color($Y,red)
```

は、ユニフィケーションにより

```
color(apple,red)
```

という1つの述語になる。

2）PrologはＬＩＳＰで扱うような2進木リストを扱うことができ、そのユニフィケーションは、リストの第1要素と残りの部分に分割することによって行なわれる。ＬＩＳＰの関数 car，cdr のProlog による定義は、次の通りである。

```
car(($A|$D),$A) ←
cdr(($A|$D),$D) ←
```

ここで、（$Ａ｜$Ｄ）はリストを表わしており、その第1要素が$A、残りの部分が$Dである。

3）処理系がユニフィケーションを行なう場合に、可能性のある述語のどれを選択するかという条件を明確に記述せず、どれかを選択するというように抽象的に記述する非決定的処理が行なわれる。そのために、処理系は、バックトラッキングと呼ばれる手法を持っている。プログラムの実行中にユニフィケーションが不可能になると、後戻り（バックトラック）を行ない、別の述語のユニフィケーションが試みられる。

4）Prologにおける述語の引数が変数である場合、処理系は、それが入力変数であるか出力変数であるかの区別をしない。したがって、どちらの変数としても用いることができ、いわゆる逆関数としてプログラムを実行しうる。たとえば、リストの要素を逆順に並び換える次のプログラムでは、変数$Lと$Rのどちらを入力変数としても同じ結果になる。

```
reverse($L,$R) ← rev($L,(),$R)
rev((),$R,$R) ←
rev(($A|$B),$S,$R)←rev($B,($A|$S),$R)
?-reverse((1,2,3),$R)
(3,2,1)
?-reverse($L,(1,2,3))
(3,2,1)
```

以上のような特徴により、Prologは従来の人工知能向き言語では得られなかった明確さと効率を持ち、広く活用されている。また、Prologの基本動作は、パターンマッチング操作であるため、数式の処理はきわめて容易に行なえることにな

る。さらに処理に適した述語（命令）を組み込むことにより高速性を保証することができる。

2—2．作成したProlog処理系

1でも述べたように、数値・数式混合処理にともなう困難の克服，他機種への移植性の向上を図るため．Cにより小型Prolog処理系を開発，作成した。Cはシステム記述性に優れた言語であり、近代的な流れ制御機構を持っている。そのため、処理系の構造的な特徴をより明らかにすることが可能となり、処理系の高速化に対する障害部分を明確にし得る。さらに開発される言語（ここではProlog）の仕様や処理方式の変更，拡張に対して細かく柔軟に対処することができる。これらが．PrologをCで作成した理由であるが．以下にこのProlog処理系の持つ特色、制限等をまとめる。

1）述語は述語名と引数のリストから構成される。
例：city(ehime,matsuyama)

2）述語を構成する引数は次の4種である。
a）変数。記号$を加えて表現する。
例：$X,$Y
b）定数。特定の名前であり．アトムと数のいずれかである。
例：ehime,1984,-1.5
c）リスト。ある順序にならぶ変数または定数の組。
例：(a,b,c),($X|$Y)
d）複合項。
例：color(apple,red)

3）プログラムはホーン節の集合である。

4）各ホーン節中の述語のユニフィケーションは左から右へとなされ、プログラム中の各ホーン節の述語へのユニフィケーションは上から下へとなされる。

5）エディタとして簡単な行編集機能を持ち、行番号を用いたプログラムの修正を行なうことができる。

6）トレーサとして、処理系がユニフィケーションを行なう過程を必要に応じて表示する機能がある。

7）組み込み述語は、Table 1に示した通りである。

Table 1. Built-in predicates and their functions in our Prolog.

operator	symbol	functions
cut	!	freeze certain choices in "Backtracking"
relational	lt	less than
	le	less than or equal
	eq	equal
	ne	not equal
	ge	greater than or equal
	gt	greater than

arithmetic	add	X+Y=Z add(X,Y,Z)
	sub	X-Y=Z sub(X,Y,Z)
	mul	X*Y=Z mul(X,Y,Z)
	div	X/Y=Z div(X,Y,Z)
	mod	XmodY=Z mod(X,Y,Z)
others	read / write	input and output
	atom,car,cdr	list processing
	convert,simpl, etc.	algebraic manipulation

これらを用いて、たとえば否定を表わすプログラムは次のように書ける。

```
not($X) ← $X,!,false
not($X) ←
```

8) ちり集め（garbage collection）。バックトラッキングが行なわれる時，多くのメモリ領域内に不要な情報が残されたままとなる。これら不要部分を処理し、以後の計算のためにメモリ領域を確保する。

なお、このProlog処理系は、Cが稼動し得る計算機上への移植は容易である。現在は、DEC Micro / PDP-11、富士通FM-11等の小型計算機上で稼動し、推論速度は50 ～ 100 Lips（logical inference per second）である。

3．Prologによる数式処理

Prologによる数式処理システムとしては、Edinburgh大学でBundy等により開発されたPRESSが知られている[5]。PRESSの現在での機能は代数方程式の解、不等式の解あるいは、多項式、三角関数、対数関数等を含む式に対する色々な代数学的処理に限られている。PRESSでは、数式の評価を、isolation、collection、attractionの3段階に分けている。第1段階のisolationでは、方程式等に出現する未知数を抽出する（例：$x-a=b \longrightarrow x=a+b$、$\log_a x=b \longrightarrow x=a^b$ ---）。collectionの段階は未知数の式中での出現回数を1回にする（例：$x\cdot a+x\cdot b \longrightarrow x(a+b)$、$2\sin x\cdot\cos x \longrightarrow \sin 2x$ ---）。さらに第3のattractionの段階でなされることは、第2のcollectionと形式的には良く似ているが演算の「近い」未知数の出現回数を減らしてゆくものである。これら3段階は、具体的な算術演算を行なうことのない方程式に対するある種の演算である。そこで「オブジェクト・レベル」の演算である通常の算術演算と区別して「メタ・レベル」の演算（あるいは推論）と呼ぶ。PRESSにおける数式の処理の制御は、この「メタ・レベル」の推論によっている。この方式を採用することにより、制御方式に階層を持たせることができ、システム全体の構造化を進めることが容易となる。

我々のシステムにも、このPRESSの特徴を取り入れて構造化することにより、たとえメモリ領域の小さな計算機上でも、目的別にシステムを実現しうることを目指している。以下、全体の流れの中、記号微分を実行する部分につ

```
diff($E,$X,$N)(-atom($E),atomdiff($E,$X,$N)

    atomdiff($X,$X,1)(-
    atomdiff($E,$X,1)(-

diff(+($E1,$E2),$X,+($N1,$N2))(-diff($E1,$X,$N1),diff($E2,$X,$N2)

diff(-($E1,$E2),$X,-($N1,$N2))(-diff($E1,$X,$N1),diff($E2,$X,$N2)

diff(-($E),$X,-($N))(-diff($E,$X,$N)

diff(*($E1,$E2),$X,+(*($E2,$N1),*($E1,$N2)))
                    (-diff($E1,$X,$N1),diff($E2,$X,$N2)

diff(/($E1,$E2),$X,/(-(*($E2,$N1),*($E1,$N2)),^($E2,2)))
                    (-diff($E1,$X,$N1),diff($E2,$X,$N2)

diff(^($E1,$E2),$X,+(*($E2,*(^($E1,-($E2,1)),$N1)),*(^($E1,$E2),
*(ln($E1),$N2))))
                    (-diff($E1,$X,$N1),diff($E2,$X,$N2)

diff(exp($E),$X,*($N,exp($E)))(-diff($E,$X,$N)

diff(ln($E),$X,/($N,$E))(-diff($E,$X,$N)

diff(log($E1,$E2),$X,/(*($N,log($E1,e)),$E2))(-diff($E2,$X,$N)

diff(sin($E),$X,*($N,cos($E)))(-diff($E,$X,$N)

diff(cos($E),$X,-(*($N,sin($E))))(-diff($E,$X,$N)

diff(tan($E),$X,*($N,^(sec($E),2)))(-diff($E,$X,$N)

diff(cosec($E),$X,-(*($N,*(cot($E),cosec($E)))))(-diff($E,$X,$N)

diff(sec($E),$X,*($N,*(tan($E),sec($E))))(-diff($E,$X,$N)

diff(cot($E),$X,-(*($N,^(cosec($E),2))))(-diff($E,$X,$N)
```

Fig.1. Prolog program for the symbolic differentiation.

いて具体的に述べる。なお，Fig.1に示された記号微分のための述語diffは文献8に示されたものを改良したものである。演算可能の範囲は，通常の多項式，三角関数，指数関数，対数関数等である。しかし，この術語diffの入力，出力の数式は表現形式として前置形式を採用している。したがって，通常の形式（中置形式）で数式の入・出をするためには，この種の形式間の変換を行なう述語（我々のシステムではconvert）が必要となる。この他，出力時の数式の簡素化（simple）等の術語が考えられる。これらの術語の一連の処理の流れが，一段「高い」レベルで制御される必要がある。実際に数式

```
a*x^2 + b*x + c
```

をxで微分する場合の制御の流れと各時点での中間結果がFig.2に示されている。ただし，入力式でのx^2はx²を表わす。これをPrologの質問形式で書くと，

```
?-convert(a*x^2+b*x+c,$T1),diff($T1,x,$T2),
  simpl($T2,$T3),convert($ANS,$T3),write($ANS)
a*2*x+b
```

である。

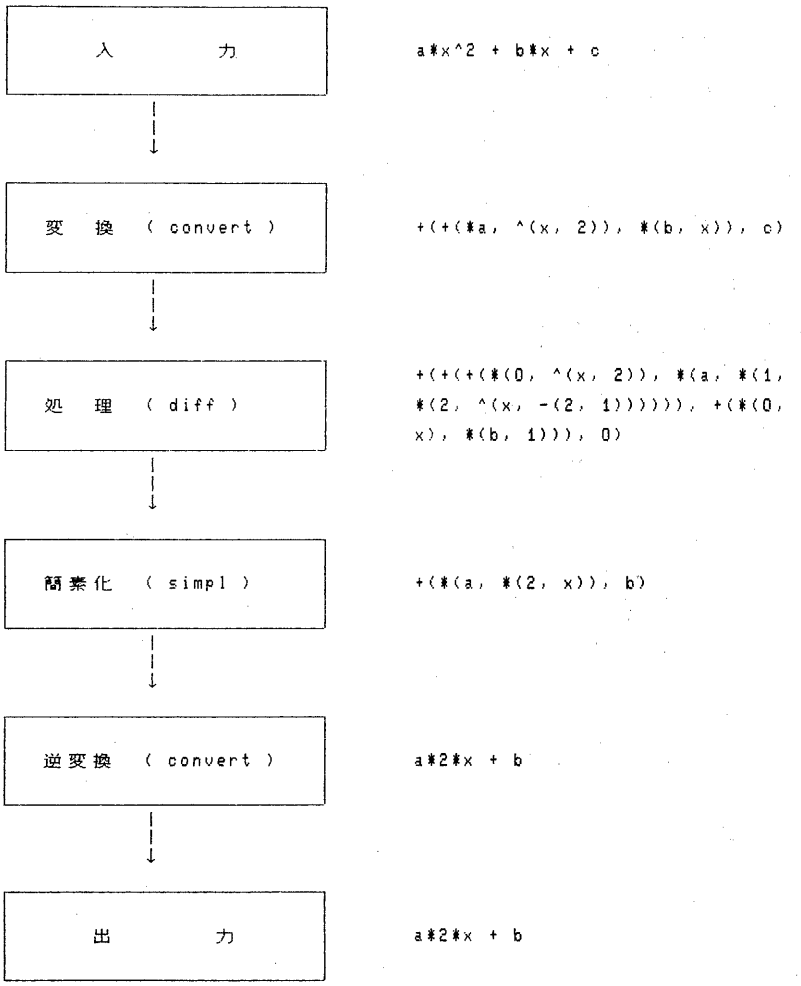

Fig.2. Control flow of the symbolic calculation.

数値結果を得るためには述語 eval を用いる。係数a＝1，b＝2でx＝3での値を得ようとすると、

```
?-convert(a*x^2+b*x+c,$T1),diff($T1,x,$T2),
  simpl($T2,$T3),convert($T4,$T3),
  eval($T4,(a=1,b=2,x=3),$ANS),write($ANS)
```

により、結果＝8を得ることが出来る。なお、次の2点は特に重要であるので注意しておく。

1) 簡素化（simpl）を最終結果のみについて行なおうとすると、中間結果の膨張により、たちまちメモリが不足してしまう。これを防ぐために、2－2で述べた「ちら集め」機能を活用するとともに、各処理段階での簡素化を有効に用いる必要がある。ここでいう簡素化は、多項式に関する係数の統一、三角関数、指数関数、対数関数等について考えられている。

2) 数値結果を求める場合に、eval で代入する数値は上の例のような整数にとどまることはない。負数、実数（浮動小数点数）についても可能である。なお、実数に対するより高度な取り扱いと問題点は4．に述べる。

4．数値・数式混合処理

3．で述べたような数式処理を有効に使用することにより、本来は数値解しか求めることの出来ないような問題にも誤差の少ない計算を行なうことが出来る。微分演算を取り上げてみても、数値微分のアルゴリズムの不安定性のためにあまり重要視されなかったようなアルゴリズムが有効になる可能性がある。

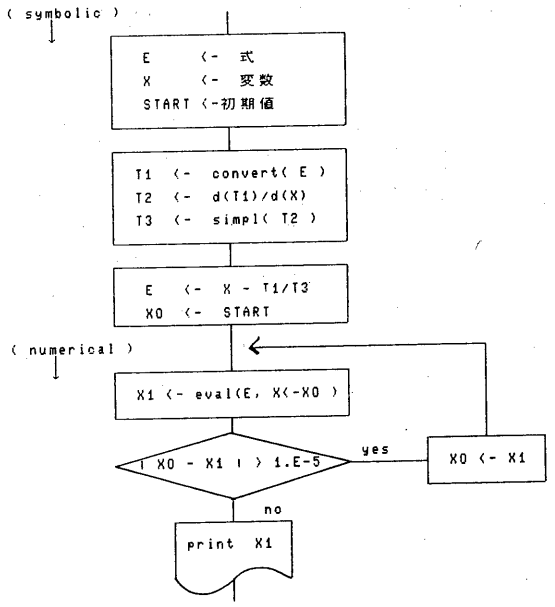

Fig.3. An example of the symbolic and numerical "interface" -- a case of the Newton's method Xl=X0-E(X0)/E'(X0) --

たとえば、常微分方程式に対する「リー級数法」[9]である。しかし、このような手法の実用化のために残された問題として、数式処理と数値計算のインターフェースの問題がある。同様の問題は式の簡素化の過程においても出現する。ここで開発したシステムでは、このような簡素化の処理として、多項式、三角関数、指数関数、対数関数等を含み、かつ浮動小数点数を扱い得るような述語をもうけている。ただし、浮動小数点数に対するパターンマッチング操作は丸め誤差の存在のため十分な精度で行なうことは出来ない。

次に、簡単な数値・数式混合処理の例として、多項式の零点を求めるニュートン法についてFig 3に示した。対象となる多項式とその変数およびニュートン法の初期値が各々、E, X, STARTに代入され、数式微分を受ける。以後は制御が数値計算に委ねられることになる。Xの旧値X0に対してのニュートン反復で、新値X1を生じ、さらにこの値を旧値とし直し、次の反復を行なう。数値が求められる反復式（E）は数式で得られているので、解は良い精度を少ない反復で得ることが出来る。この種の技法は、より広い範囲の問題に拡張することが出来る。

さらに多項式の因数分解の述語を作成すると、よりすぐれた数値計算と数式処理とのインターフェースが構成されることが容易にわかる。

5. 結言

この研究では、現在その使用が広まりつつある数式処理システムの持つ問題点の一つである数値計算との結合を強めることを目的とした数値・数式混合処理システムの開発を行なった。我々の究極の目的である「知的」な数式処理および数値計算支援システムの完成への段階として、以下の点が確立された。

1) システムの記述をProlog言語で行なうことの妥当性。
2) 上の目的を遂行するため、移植性・拡張性に富むProlog処理系のC言語による開発。
3) 簡単な、数値・数式混合処理システムの作成。

このように作成されたシステムは、現在、小型あるいは超小型計算機上で稼動しており、目的としている「パーソナルな使用」に十分対応している。

今後、早急に為されるべき点は以下の諸点である。第1に4.で述べたことだが、より強力な数値・数式インターフェース完成のための浮動小数点数の表現とそのパターンマッチングがある。さらに、入力された偏微分方程式等のある種の標準形への書き換え規則の導入と解析解存在のための検討、および数値計算用アルゴリズムの自動選択がある。これらをシステムに導入する場合には、データベースの有効利用が考えられ、本システムの特徴（Prologによるシステムの記述）がより十分に表われるであろう。

引用文献

1) J.A.von Hulzen and J.Calmet : Computer Algebra Systems , in "Computer Algebra Symbolic and Algebraic Computation" ed. B. Buchkerger et al. , pp.221-243 , Springer-Verlag , 1982.

2) 梅谷他：数値シミュレーション用プログラム言語ＤＥＱＳＯＬ，情報処理学会数値解析研究会資料５－２，１９８３ その他.

3) C.A.Cole and S.Wolfram : SMP - A Symbolic Manipulation Program , SYMSAC 1981 , pp.20-22 , 1981.

4) L.B.Rall : Differentiation in Pascal-SC : Type GRADIENT , ACM Trans. Math. Softw. 10 , 2 , pp.161-184 , June , 1984.

5) A.Bundy and B.Welham : Using Meta-level Inference for Selective Application of Multiple Rewrite Rule Sets in Algebraic Manipulation , Artificial Intelligence , 16 , 3 , pp.189-212, July ,1981.
A.Bundy : "The Computer Modelling of Mathematical Reasoning" , Academic Press , 1983.

6) R.A.Kowalski : Algorithm = Logic + Control , Comm. ACM , 22 , 1979.

7) A.Colmerauer , H.Kanoui , R.Pasero and P.Roussel : Un systeme de communication homme-machine en francais , Rapport preliminaire, Groupe de Res. en Intell. Artif. , U. d'Aix-Marseille , 1972.
R.A.Kowalski : Predicate Logic as Programming Language , Proc. IFIP 74 , North- Holland Pub Co. , 1974.

8) W.F.Clocksin and C.S.Mellish : "Programming in Prolog" , Springer-Verlag , 1981.（邦訳あり : 『プログ・プログラミング』 （日本コンピュータ協会））その他.

9) H.Knapp and G.Wanner : Numerical Solution of Ordinary Differential Equation by Groebner's Method of Lie-Series , MRC Report #830 ,1968.