# パーソナルなハイブリッド処理システムSYNCの設計†

野田 松太郎†† 岩下 英俊†††*

ハイブリッド計算とは，数式処理と数値計算の効率的な融合を意味し，数値計算のみでは満足いく解を得るのが困難な問題を解決する可能性がある．このようなアルゴリズムの遂行のために，ハイブリッド計算が容易に行えるシステムの開発が必要となる．簡単なアルゴリズム開発に適するように，パソコン上でも稼働し得るようなハイブリッドシステムを提唱する．システムの記述言語は主に Prolog であり，C で作成した述語を付け加える．不必要なパターン照合操作をさけるため，数式中の変数を素数に対応させ高速な整数演算に置き換える．この新しいデータ構造を利用した基本算法とシステムの利用を容易にするため作成した PASCAL 的な固有の言語についても述べる．

## 1. 序

数式処理システムの利用が急激に普及してきている．それらの多くは今までの単なる代数計算や，特殊な微分方程式の解法への利用にとどまらず，ロボティックス，計算幾何等の現実的な問題へ広まっている．これらの利用に共通しているのは従来からの誤差を含む数値計算の結果のみでは不十分な問題へのアプローチにある．しかし，数式処理計算のみでは，問題が大規模になればいかに大型計算機を駆使しようと計算量や必要メモリ容量が膨大なものになる．そこで，数式処理が可能な範囲のみを数式処理システムで，その他を数値計算で行おうとするハイブリッド処理への要求が高まってきている．特に最近，アルゴリズムそのものをハイブリッド化することにより，新しい計算手法が生じる可能性も指摘されている[1]．

ハイブリッド計算を有効に遂行するためには，適切な計算システムの作成が不可欠な課題となる．これらの中でも，手近に計算のできる小型システムの開発を目的として，われわれもパソコン上で稼働する小型ハイブリッド処理システムを作成してきた[2]．われわれのシステムは記述言語に Prolog を採用し，将来のヒューマンインタフェースの充実などに対応しようとしている．ここでは，ハイブリッド計算による常微分方程式の求解を主眼にしている．解法としては Taylor 級数法がハイブリッド化され用いられる．Taylor 級数法は数値計算法として確立されたが数値微分の困難などにより，むしろ非能率な解法として知られている．しかし，ハイブリッド化した Taylor 級数法では数式微分の容易さなどにより手法は多くの問題に対して有効になることがわかった．しかし，システムをより高機能なものにするためには，数式，変数などに対するパターン照合操作の高速化，基本の Prolog 処理系がもつ述語の拡充等が必要となる．

そこで，本論では小型で高機能なハイブリッド処理システムの改良・開発を目指す．このために小型システムに適合するような基本のデータ構造を導入する．パターン照合操作についても，数式など大規模なものについては Prolog の長所を残し，変数への照合操作等の繁雑なものに対しては高速な整数演算を用いることを考える．このため数式中に現れる変数を素数に対応させるシステムの内部表現を考える．このように素数に対応付けられる変数には，整数，多桁整数，有理数，実数(浮動小数点数)等が含まれる．以下，ここで開発したハイブリッド処理システムを SYNC (SYmbolic and Numerical Computation) と呼ぶ．SYNC のデータはこれら変数，変数により構成される多項式，有理式さらに一般的な数式，手続きなどのように階層的な内部表現が定められる．各階層のデータ間の演算のために新しい述語が作成される．これらのいくつかはCで作成され，Prolog に付加される．また，SYNC の特色の1つに強力な数値計算とのインタフェースがある．FORTRAN への変数の入出力は COMMON 文に対応するような JOINT 文を FORTRAN のソースプログラムに加えるのみで達成される．以下，2章で変数の素数による表現について述べ，システムのもつ階層的なデータ構造と SYNC の概要を3章でまとめる．SYNC 上に実現される簡単な利用者言語について4章で述べる．5章では主に

† Portable Hybrid Computation System SYNC by MATUTAROW NODA (Department of Computer Science, Ehime University) and HIDETOSHI IWASHITA (Department of Electronics, Ehime University).

†† 愛媛大学工学部情報工学科
††† 愛媛大学工学部電子工学科
* 現在 (株)富士通研究所川崎研究所情報処理部門

FORTRAN とのインタフェースを取り上げ，いくつかの例題を示す．本論の結論とハイブリッド処理システムの今後の拡張について6章に述べる．

## 2. 素数による変数の表現

### 2.1 素数の対応付け

小型の数式処理システムの効率的な実現のための新しいデータ構造を提案する．この手法の基本は数式中に出現する変数をそのままパターン照合操作の対象とせず，変数を対応する素数に置き換え，素数間の数値演算で計算を遂行することにある．これと同じ方法はわれわれと別個に，Mawata により提案され小型数式処理システム muMATH 上で，整数の詰め込み問題に対して有効性が試されている[3]．しかし，そこではシステムのデータ構造への応用などは触れられていない．今，対象とする数式中に出現する変数を $x, y$，対応させる素数を $p_x$，$p_y$ とする．この対応を写像 *index* により，

$$index(x, y)=(p_x, p_y)$$

と表す．当然，逆写像を $index^{-1}$ と書くと

$$index^{-1}(p_x, p_y)=(x, y)$$

である．また，多変数単項式 $cx^iy^j$ を（定数係数，変数部分）の対 $(c, p_x p'^i_y{}^j)$ で表す．具体的に変数 $x$，$y$ に素数3，2を各々対応させると

$$index(x, y)=(3, 2)$$

であり，単項式 $3x^3y^2$ は

$$(3,\ 3^3 2^2)=(3,\ 108)$$

となる．逆に，変数部分108の単項式は素因数分解の一意性より次のように定められる．

$$index^{-1}(108)=index^{-1}(3^3 2^2)=x^3y^2$$

多項式は単項式の和であるので，$x^3y+xy+1$ は

$$((1, 54)\ (1, 6)\ (1, 1))$$

のように，上の *index* により表される．

### 2.2 基本アルゴリズム

上の変数の素数による表現で基本アルゴリズムがどのように表されるかを見る．簡単のための単項式 $i$ の表現を $(c_i, t_i)$ とする．単項式間の四則演算は

$$-(c, t)=(-c, t)$$
$$(c_1, t)\pm(c_2, t)=(c_1\pm c_2, t)$$
$$(c_1, t_1)\times(c_2, t_2)=(c_1c_2, t_1t_2)$$
$$(c_1, t_1)/(c_2, t_2)=(c_1/c_2, \alpha)$$

ただし，$t_1=\alpha t_2$ であり，$t_1$ が $t_2$ の倍数でないときには除法は定義されない．多項式間の演算についても素数に対応付けられた変数の積を表す整数の比較のみ

```
inputs :
 f1= ((c1(1),t1(1))(c1(2),t1(2))･･･(c1(m),t1(m)))
 f2= ((c2(1),t2(1))(c2(2),t2(2))･･･(c2(n),t2(n)))
output :
 f3= f1 + f2
   = ((c3(1),t3(1))(c3(2),t3(2))･･･(c3(p),t3(p)))

begin
  i := j := k := 1;
  repeat
    if t1(i) = t2(j) then
      begin
        if c1(i) + c2(j) ≠ 0 then
          begin
            c3(k) := c1(i) + c2(j);
            t3(k) := t1(i);
            k := k + 1
          end;
        i := i + 1; j := j + 1
      end
    else if t1(i) > t2(j) then
      begin
        (c3(k),t3(k)) := (c1(i),t1(i));
        i := i + 1; k := k + 1
      end
    else
      begin
        (c3(k),t3(k)) := (c2(j),t2(j));
        j := j + 1; k := k + 1
      end
    end
  until i > m or j > n;
  if i ≤ m then
    copy ((c1(i),t1(i))･･･(c1(m),t1(m)))
    to   ((c3(k),t3(k))･･･(c3(p),t3(p)));
  if j ≤ n then
    copy ((c2(j),t2(j))･･･(c2(n),t2(n)))
    to   ((c3(k),t3(k))･･･(c3(p),t3(p)))
end.
```

**図1** 素数の対応によるデータ構造での加算のアルゴリズム
Fig. 1 An algorithm for addition of polynomials in prime number representation.

から次のような効率的なアルゴリズムを考え得る．

2つの多項式の加算 $f_3:=f_1+f_2$

$f_1:=((c_{11}, t_{11})\ (c_{12}, t_{12})\cdots(c_{1m}, t_{1m}))$

$f_2:=((c_{21}, t_{21})\ (c_{22}, t_{22})\cdots(c_{2n}, t_{2n}))$

とする．加算のアルゴリズムは**図1**に示されるように筆算の要領で行われる．手間のかかる変数に対するパターンマッチング操作を行う必要はない．2つの多項式間の減算，乗算についても同様のアルゴリズムを示し得る．

単項式 $cx^iy^j$ の変数 $x$ に関する微分

$cx^iy^j$ を単項式1，変数 $x$ を単項式2とすると，各々の表現は $(c, t_1)$，$(1, t_2)$ である．ここで *index* により，$t_1=p_x{}^i p_y{}^j$，$t_2=p_x$ である．なお，整数 $k$ を $t_1$ が $t_2$ で割り切れる回数とする．これにより求める単項式の変数 $x$ に関する微分は

$$\mathrm{d}(c,\ t_1)/\mathrm{d}(1,\ t_2):=\begin{cases}(ck,\ t_1/t_2) & \text{for } k>0\\ 0 & \text{for } k=0\end{cases}$$

のように定められる．例えば，$\mathrm{d}(3x^3y^2)/\mathrm{d}x$ を *index* $(x, y)=(3, 2)$ の対応のもとで行うと，$\mathrm{d}(3, 108)/\mathrm{d}(1, 3)$ である．$\beta_0=103$，$\beta_1=108/3=36$，$\beta_2=36/3=12$，$\beta_3=$

12/3=4, $\beta_4=4/3$ で剰余が0とならず，割り切れない．したがって，$k=3$ となる．また $t_1/t_2=\beta_1$ である．上の公式より結果は d(3, 108)/d(1, 3)=(9, 36) となり，$9x^2y^2$ を得る．偏微分の演算を自動的に行っていることになる．なお，$x^2y^2$ を得るため変数部分 36 を素因数分解する操作は最終の出力時にのみ必要になる．

多項式の変数に対する微分 $f_2:=\mathrm{d}(f_1(x))/\mathrm{d}x$ は単項式の微分を繰り返すのみである．例えば上と同様の素数の対応で $f_1:=x^3y+xy+1$ すなわち ((1, 54) (1, 6) (1, 1)) の $x$ による微分は

① d(1, 54)/d(1, 3)=(3, 18)

② d(1, 6)/d(1, 3)=(1, 2)

③ d(1, 1)/d(1, 3)=0

を加えるのみであり，$f_2:=((3, 18) (1, 2))$ を得る．これは $f_2:=3x^2y+y$ に対応している．変数部分 18 の素因数分解はやはり最終の出力時にのみ必要になる．

次に，変数の入れ替えあるいは代入操作がこのような素数による表現のもとでいかに遂行されるかを考える．今，新しい変数 $u$ を導入し，単項式 $x^3y^2$ で，$u\leftarrow xy$ と変数の入れ替えをする．$index(x)=3$, $index(y)=2$ なので，$index^{-1}(3\cdot2)=index^{-1}(6)=u$. $x^3y^2$ の表現は前述のように 108 なので，これを6で割ることにより，$108=6^23$ すなわち

$$index^{-1}(6^23)=u^2x$$

を得る*．しかし，ここでの $u$ の対応する数6は素数ではないので，あらたに素数による対応

$$index(x, y, u)=(3, 2, 5)$$

を考え，上の $6^23$ を $5^23=75$ と与え直す．すなわち $u\leftarrow xy$ の書換えにより，$x^3y^2=xu^2$ を得る．このように，新しい変数による入れ替えや代入操作も容易に遂行することが可能である．

### 2.3 変数の順序付け

写像 *index* により変数に対応させられる素数の選び方は任意である．この対応に関連して多変数多項式に対する項の順序付けの問題がある．項の順序は単項式中の変数部分 (power product) の形によって定められ，次の条件を満たしていれば任意に選んでよい[4]．

a) $1<_T t$ for all $t\neq1$

b) if $s<_T t$ then $su<_T tu$

よく用いられる順序には，$x$, $y$ を変数とする2変数多項式について，次の2通りがある．

i) 辞書式 (purely lexicographical) 順序

$1<_T y<_T y^2<_T\cdots<_T x<_T xy<_T xy^2<_T\cdots<_T x^2<_T\cdots$

ii) 全次数式 (total degree) 順序

$1<_T y<_T x<_T y^2<_T xy<_T x^2<_T\cdots$

このような変数間の順序を保つ素数の対応を考える．今 $y<_T x$ とし，計算中に出現する $y$ の最高べきを $n_y$ とする．辞書式順序が成立するためには，

$$p_y{}^{n_y}<p_x$$

が満たされる必要がある．より一般的には

$$p_x/p_y\to\infty$$

で辞書式順序が実現される．一方，

$$p_x/p_y\to1\quad(p_x\neq p_y)$$

の選択により全次数式順序が実現されることがわかる．高次項を含む多変数多項式の計算において，辞書式順序を常に満たすことは一般に困難であるが，全次数式順序での計算は容易である．本論で述べる素数の対応による順序は辞書式順序や全次数式順序とは異なる順序となる．これは，変数間の順序が重要な問題，例えば計算機代数で注目されている Gröbner 基底の計算等で意味をもつと思われる．

## 3. システムのデータ構造

2章で述べた変数の素数による表現を小型ハイブリッド処理システム SYNC のデータ構造として用いることを考える．SYNC の数式処理部分の計算の多くは有理式の処理などの代数計算であるので，それに対応したデータ構造を用いるのが有利と思われる．既存の大型数式処理システムでも MACSYMA 等では基本のデータ構造を有理式の形にしている．文献 2) の小型ハイブリッドシステムでもこの種データ構造を使用し有効性を見た．基本とするデータ構造は次の3種である．

i) 数 (整数，多桁整数，有理数，実数)

ii) 多項式

iii) 有理式

各々の具体的な表現は，数については

整数 (*integer*)…通常の16ビット整数表現

多桁整数 (*big integer*)…標示子\による複合体 (\(*big integer* に対応する配列))

多桁整数 1684234849 の表現例

$1684234849=100\cdot256^3+99\cdot256^2+98\cdot256+97$

文字 $a, b, c, d$ の ASCII-code は 97, 98, 99, 100. よって，これを文字配列\(*abcd*) とする．

---

* この点は元吉文男博士 (電総研) に御指摘いただいた．

有理数 (*rational number*)…標示子\による複合体で2引数（多桁整数も可）のもの

(例：\(50, 13))

浮動小数点数 (*floating point*)…IEEE 形式の浮動小数点表現であり，次の表現形式をもつ．

(±)*integer*. *integer*(E±*integer*)

したがって，1.2，−3.0，2.0 E-16 等は浮動小数点数であるが，2. は許されない．

一方，多項式，有理式の表現は各々標示子で区別する．$c$ を係数とする単項式 $c\ x^{i}y^{j}$ は対応 $index(x, y)=(p_x, p_y)$ により，$(c, p_x{}^{i}p_y{}^{j})$ と書かれたが，これに標示子 **pol** を加えて多項式の表現にする．$f_1=x^3y+xy+1$ なる多項式は，$index(x, y)=(3, 2)$ で

**pol**([[1, 54], [1, 6], [1, 1]])

となる．有理式の表現は多項式表現の拡張で，標示子 **rat** を用いる．上と同じ変数の対応で，$(x^3y+xy+1)/(3x^3y^2)$ は

**rat**([[1, 54], [1, 6], [1, 1]], [[3, 108]])

となる．著名な大型数式処理システムである REDUCE や MACSYMA では多項式 $f_1=x^3y+xy+1$ の表現は各々主変数を $x$ として

REDUCE…(((x, 3), ((y, 1), 1)), ((x, 1), ((y, 1), 1)), 1)

MACSYMA…(x, 3, (y, 1, 1), 1, (y, 1, 1), 0, (y, 0, 1))

であり，有理式の表現はさらに複雑になる．ここで述べる素数を用いた表現では繁雑なリスト操作が不用で単純であることがわかる．SYNC では，これらデータ構造とそれに対する演算の組が設定されている．通常用いられる整数，実数間の演算は Prolog の命令 **is** で処理されるが，多桁整数や有理数間の演算は C で記述され，Prolog に付加された命令 **ist** でなされる．さらに，多項式および有理式間の演算のために Prolog で作られた命令 **est** がある．SYNC では，さらに通常の数式に対する演算命令（代入命令:=等）が用意されている．SYNC のデータ構造は**図 2** にまとめるように階層化されている．

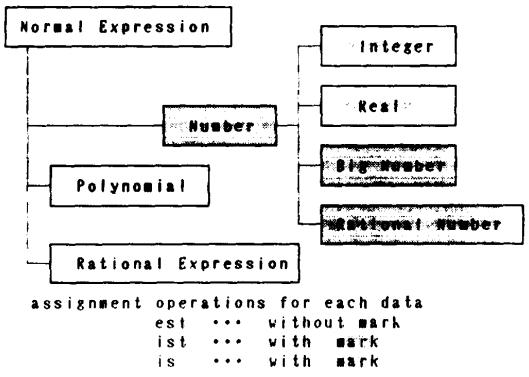


**図 2** SYNC のデータ構造
Fig. 2 Data structure and operations in SYNC.

## 4. 小型ハイブリッド処理システム SYNC

SYNC は Prolog を記述言語にしたシステムであり，パソコン IBM-PC 上で稼働する．Prolog には Arity Prolog™ を，部分的な記述のためには Lattice C™ を採用している．よく知られた多くの数式処理システムに比較すると，小型化が進み，FORTRAN との有効な結合がなされているる点に特色がある（詳細は5章）．このため，簡単なハイブリッド・アルゴリズムの開発等の研究目的に適している．SYNC のプログラムは当然，記述言語である Prolog でなされるが，より開発環境の向上をめざして独自のプログラミング機能を備えている．SYNC の利用に当たってはPASCAL 風のプログラムを作成するのみでよい．このため，数値計算になれ親しんでいる利用者や数式処理の入門者などの利用も困難でない．以下，SYNC 上のプログラム機能について簡単に述べる．

SYNC プログラミングの基本である文 (*statement*) は，数式 (*expression*)，宣言 (*declaration*)，制御文 (*control statement*) から成り立っている．文は SYNC で評価 (*evaluation*) される．評価は常に成功 (*success*) するか失敗 (*failure*) するかのいずれかである．評価が成功すると値 (*value*) をもち，失敗するとバックトラックを起こす．文の構成は**図 3** に示されている．システム動作の基本は〈**Q**〉行で入力された文に対し，〈**A**〉行で値を戻す点にある．入力された

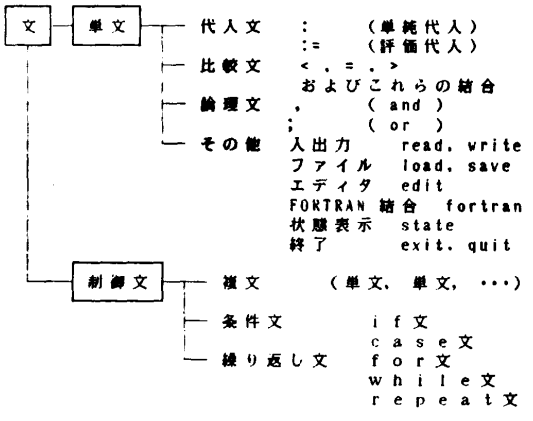


**図 3** SYNC における文
Fig. 3 Statements in SYNC.

文は評価された作業領域 (ws) に格納され，〈A〉行で出力される．直前の〈A〉行の値は ws で呼び出し得る．具体的には〈Q〉行では

〈Q〉 ws:=入力文.

なる代入操作がなされている．1入力に対し，評価が複数回成功し，複数の値が得られることはしばしば発生する．複数の値を要求する場合には〈A〉行の末尾に利用者が；を入力するが，それ以外のキー入力に対しては．が出力されバックトラックは行われない．評価がすべて失敗すると，文字列 no が〈A〉行に出力される．

図2のように，数式中の標準型 (*normal form*) には通常の数以外に多桁整数，有理数，多項式，有理式等が含まれる．これらは評価されると，それ自身を値としてもつ．いくつかの入出力例を**図4**に示す．ここで，下線部が利用者の入力である．注意が必要なのは，浮動小数点数の扱いである．入力行の末尾の．との区別のために 3.0 のように，小数点以下に1桁以上の数が必要となる．

変数は Prolog でいう *atom* であるが，SYNC では素数への対応がなされる．利用者が積極的に対応する素数を指定するには**図5**のように **var** 命令による．この命令で設定されていない場合には *default* 値が与えられる．変数名は英小文字で始まる必要がある．これに対し，英大文字で始まるパターン変数は素数への対応は行われない．パターン変数は Prolog での *variable* に相当し，ユニフィケーション操作により，任意の数式とパターン照合する．SYNC では通常の変数が大域変数として扱われ，手続きを越えて有効であるのに対し，パターン変数は手続き内でのみ有効な局所変数である．

SYNC で用いられる述語は4種類に大別される．それらは代入，比較，論理演算およびその他の入出力

```
<Q> x^3*y+x*y+1.
<A> x ^ 3 * y + x * y + 1.
<Q> 1/3 + 2/5.
<A> 11/15.
<Q> 2.1+3.0.
<A> 5.1.
<Q> ...
```

**図4** SYNC の入出力例 (数式)
Fig. 4 Examples of uses of SYNC (1).

```
<Q> var x as 3.
<A> done.
<Q> var [x,y] as [3,2].
<A> done.
```

**図5** SYNC の入出力例 (変数)
Fig. 5 Examples of uses of SYNC (2).

```
<Q> (1+!).
<A> 2.
<Q> 1+1+:(2*3).
<A> 2+2*3.
<Q> a:=1+1.
<A> 2.
<Q> p:= x^2-2*x+2.
<A> x ^ 2 - 2 * x + 2.
<Q> ...
<Q> fac(x):(f:=1.for i:=1 to x do f:=f*i.f)
<A> done.
<Q> fac(10).
<A> 3628800.
```

**図6** SYNC の入出力例 (代入文)
Fig. 6 Examples of uses of SYNC (3).

関係の述語である．

SYNC では2通りの代入文がある．述語：によるものと，:= によるものである．：は LISP の *quote* 関数のように評価は行わず，右辺の数式はそのまま左辺に代入する．これに対し := を用いると，右辺が評価されて成功した場合に結果が左辺に代入される．これらの使用例は**図6**のとおりである．：代入の他の有効な使用例は関数定義にある．階乗計算を行う関数 *fac* を定義し，それを用いる例が図6に加えられている (ただし，SYNC では階乗計算用に特別の述語！がある)．数値比較のための述語は PASCAL のように，<，=，>を組み合わすことにより多数作られる．

論理演算には，と；がある．前者は通常の AND に対応し，後者は OR に対応する．〈A〉行文末に；を入力するとバックトラックが行われ，他の値を求めようとするが，．の入力に対してはそれ以上のバックトラックは行われない．

入出力用の述語には **write**, **writeln**, **read**, **readln** 等の他，ファイルの入出力に関連する **load**, **save**, **edit**, や SYNC の終了のための **exit** (SYNC の現状を保存したままの終了)，**quit** (無条件終了) あるいはシステムの状況を見る **state** 等がある．特に，**edit** では，各種の小型エディタを SYNC 内で用い得る．

SYNC のプログラミングのための制御文は通常のプログラミング言語のもつものの主要部が含まれている．制御の基本は逐次的になされるが，繰り返し文と条件文も用いられ得る．これらの書法は PASCAL のものに似ており，概略は以下のとおりである．

基本的な文には単文と複文がある．単文は単純な命令のみであるが，複文はこれらから合成される．複文は PASCAL では *begin*, *end* で囲まれるが，SYNC では左括弧および右括弧で囲まれる．

繰り返し文には **for**, **while**, **repeat** の構文があり，これらは PASCAL におけるものと同様な機能

```
poldiv(F,G) : (
    local (m,n,f,g,q,c,d,k),
      f:=F,
      g:=G,
      q:=0,
      n:=deg(g,x),
        repeat (
          m:=deg(f,x),
          c:=coefpol(f,x,m)/coefpol(g,x,n),
          d:=m-n,
          q:=q+c*x^d,
          r:=f-c*g*x^d,
          k:=deg(r,x),
          f:=r
          )
        until n>k,
      writeln r,
      q
    ).
```

**図 7** SYNC のプログラム例 (多項式の除算)
Fig. 7 A program for polynomial-division in SYNC.

をもつ. 条件文の **if**, **case** も PASCAL 的である. これらを用いたプログラム例を**図 7** に示す. これは与えられた2つの1変数多項式 (変数 $x$) $f$, $g$ に対して多項式の除算を遂行し, 商 $q$ および剰余 $r$ を求めるものである. プログラム中の ***deg***$(f,x)$ は多項式 $f$ の $x$ に関する次数を求める述語であり, ***coefpol*** $(f, x, m)$ は多項式 $f$ の $x$ に関する次数 $m$ の係数を求めるものである.

このように, SYNC のプログラムは非常に簡明で, 理解が容易であることがわかる. SYNC では図7のようなプログラムを属性 **SYN** のファイルとし, **load** 命令で呼び出し使用する. より, SYNC の特色を用いた利用やプログラムに数値計算, 特に FORTRAN との結合がある. これは次章にまとめられる.

## 5. SYNC の FORTRAN との結合

数式処理のより高度な進度のためには, 数値計算との結合により, 有用なアルゴリズムや新しい計算法の開発が必須不可欠であると言われている. このような目的を達成するために REDUCE, MACSYMA のような著名なシステムでは FORTRAN プログラムの生成がされる. MAPLE[5] などのシステムでは記述言語が C である特徴のため, より高度な数値計算との結合も可能であるが, これらはいまだ高機能なワークステーション上でのみ稼働する. これに対し, SYNC はパソコン IBM-PC 上で稼働しかつ高度な FORTRAN との結合を達成している. 以下, この結合とそのプログラミングについて述べる.

SYNC の稼働中に同時に FORTRAN コンパイラを動作させることは現在のパソコンのメモリ容量から不可能である. 一方, SYNC の記述言語である Prolog のデータ構造は FORTRAN のものとは大きく異なっている. そこで, 両者を結合させるために, 入出力データの授受に関する考慮とプログラム変換の考え方を加える.

まず, SYNC から直接データを入出力し得る中間言語を設定する. この言語の属性を **hyb** とし, SYNC の述語 **fortran** によって呼び出す. この中間言語の入力データは, SYNC の数式あるいは数値であり, 出力データは FORTRAN による数値の計算結果である. この結果は SYNC で容易に利用することができる. このため, 中間言語には FORTRAN の COMMON 文に似た JOINT 文を加える. JOINT 文には入力に対応する /IN/ ブロックと, 出力に対応する /OUT/ ブロックが利用者によって与えられる. このようにして, SYNC のデータを /IN/ ブロックに入力された中間言語は, やはり述語 **fortran** のもとで通常の FORTRAN ソースプログラム (属性 **f 77**) にプログラム変換される. FORTRAN で計算された数値結果は, 中間言語の JOINT 文中の /OUT/ ブロックに貯えられ, SYNC から容易に使用し得る. 上に述べた SYNC と FORTRAN のデータの授受の概括が**図 8** に示させる. また, 簡単な例として1変数代数方程式をハイブリッド・ニュートン法を用いて解く問題のプログラムが**図 9～11** に与えられる. 与えられた代数方程式に対し, 記号微分により (SYNC の述語 **dif**) その導関数が求められる. さらに初期値 ($x0$), ニュートン法の打ち切り精度 (eps) が SYNC で与えられ, 利用者によって用意された newton. hyb (図9) の入力となる. この過程を管理するのが

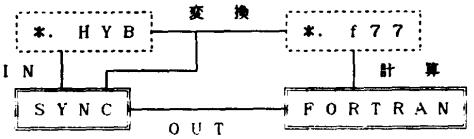


**図 8** SYNC と FORTRAN の関連
Fig. 8 The combination of SYNC and FORTRAN.

```
      DOUBLE PRECISION F,DF,X,X0,EPS,STEP
      JOINT /IN/F(X),DF(X),X0,EPS /OUT/X
      X=X0
      DO 10 I=1,20
        STEP=F(X)/DF(X)
        X=X0-STEP
        IF(ABS(STEP).LT.EPS) GOTO 20
        X0=X
   10 CONTINUE
   20 STOP
      END
```

**図 9** 中間言語 newton. hyb のプログラム
Fig. 9 Intermediate language newton. hyb.

```
      DOUBLE PRECISION F,DF,X,X0,EPS,STEP
      F(X)=X**3-17*X**2+80*X-100
      DF(X)=3*X**2-34*X+80
      X0=1.0
      EPS=0.0000001
      X=X0
      DO 10 I=1,20
        STEP=F(X)/DF(X)
        X=X0-STEP
        IF(ABS(STEP).LT.EPS) GOTO 20
        X0=X
   10 CONTINUE
   20 OPEN(255,FILE='CONT.SYN')
      WRITE(255,*) 'X:7,X,'.'
      WRITE(255,'(IH )')
      CLOSE(255)
      STOP
      END
```

**図 10** SYNC により生成された newton. f 77 プログラム

Fig. 10 Fortran program newton. f77 which is generated in SYNC.

```
<Q> f:=x^3-17*x^2+80*x-100
<A> x ^ 3 - 17 * x ^ 2 + 80 * x - 100.
<Q> df:=dif(f,x).
<A> 3 * x ^ 2 - 34 * x + 80.
<Q> x0:=1.0
<A> 1.0.
<Q> eps:=1.0E-7.
<A> 0.0000001.
<Q> fortran newton(f,df,x0,eps,y).
  *** fortran compiler messages ***
<Q> y.
<A> 2.0.
<Q> g:=x-y.
<A> x - 2.0.
<Q> load poldiv.
<A> done.
<Q> poldiv(f,g).
-2.84217094E-14
<A> x ^ 2 - 15.0 * x + 50.0.
<Q> f:=ws.
<A> x ^ 2 - 15.0 * x + 50.0.
<Q> df:=dif(f,x).
<A> 2 * x - 15.0.
<Q> fortran newton(f,df,x0,eps,y).
  *****
<Q> g:=x-y.
<A> x - 5.0.
<Q> poldiv(f,g).
7.10542736E-15
<A> x - 10.0
```

**図 11** FORTRAN との結合

Fig. 11 Combinations of symbolic and numerical FORTRAN computations.

述語 **fortran** であり，図 9 の JOINT 文のデータを通常の FORTRAN プログラムの引数のように扱う．ここでの例では代数方程式として

$f:=x^3-17*x^2+80*x-100$

を考える．このとき，SYNC によって newton. hyb からプログラム変換された newton. f 77 は図 10 のとおりである．データは自動的にファイルを経由して SYNC と FORTRAN とを往復する．図 11 では上の例を SYNC の対話的モードで計算した場合が示されている．初期値 x0=1，eps=1. E-7 ととると FORTRAN 計算の結果は $x=2.0$ となる．そこで $g:=x-2$ として図 7 の **poldiv** を用いる．剰余 −2. 84217094 E-14 の高精度で商

$x^2-15.0*x+50.0$

が得られている．これを新しく $f$ と置き直して上と同様な計算をすると $x=5.0$ が FORTRAN で求められる．**poldiv** を再び用いると剰余 7. 10542736 E-15 で，最後の商 $x-10.0$ が得られる．確かに $x=2$, 5, 10 は最初に与えた代数方程式の根になっている．このように，単純な方法で FORTRAN との結合がなされ，ハイブリッド計算が SYNC では可能である．

ある種の拡張 COMMON を用いて LISP に基づく数式処理システムと FORTRAN の結合をしようとする考えが佐々木らにより提唱されている[6]．この方法によれば FORTRAN コンパイラの改良などが要求されるが，本論のような手法は実現もきわめて容易である．特に高速性を追求しない場合のハイブリッドシステムには適切であると思われる．

## 6. ま と め

小型で利用しやすい言語を備えたハイブリッド処理システム SYNC の概要を示した．SYNC は IBM-PC などのパソコン上で稼働する．必要メモリ量は 640 KB までで十分である．SYNC は Prolog で主に記述されているが，新しい述語の作成などには C が用いられている．数式間のパターン照合操作などには Prolog は大変強力であり，十分な性能をもつ．しかし，数式中に現われる各変数間の演算をすべて Prolog の基本機能のみで行おうとすると処理速度が大きく低下する．これを防ぐため SYNC では各変数を各々素数に対応させ，変数間の演算を高速な整数演算に置き換えて処理する．この対応をより強力にするため，固有のいくつかの基本算法も考えられる．

SYNC は FORTRAN プログラムとの強力なインタフェースをもつ．利用者は最初に FORTRAN プログラムに似た中間言語を用意する (属性 **hyb**)．この言語には SYNC とのデータの授受を示す JOINT 文が追加される．この中間言語は SYNC の述語 **fortran** によりプログラム変換され，通常の FORTRAN コンパイラが受理し得る FORTRAN ソースプログラムになる (属性 **f 77**)．パソコンのような小型システムで稼働し得るためにはメモリ量の制約が大きい．こ

のため，FORTRAN プログラムの走行と SYNC とのデータの授受はファイル経由で行われる．しかし，利用者はプログラミング上で，この点を意識する必要はない．したがって，SYNC はハイブリッド・アルゴリズム開発を手近に行う場合などに非常に適したシステムと言える．ここで述べたような数式処理と数値計算プログラム（ここでは FORTRAN）の結合の方法は実現が容易でパソコン上で稼働するような小型システムには適切であると思われる．

SYNC の記述言語が Prolog であることは，当然予想されるが，データベースとの対応が取りやすい点に結びつく．基本的には SYNC 上の計算はすべてデータベースに蓄積されるため，利用者が特に必要とする数学公式などを SYNC に登録することにより，簡単にそれを利用することが可能になる．

次に，SYNC の今後の拡張と問題点について述べる．現在の SYNC の最大の問題点は依然としてその演算速度にある．Prolog のデータベースに登録された文は，それを消去する命令（SYNC では **clear**）が実行されるまでは蓄積される一方である．また，Prolog の演算は常にデータベースとのパターン照合に基づいて行われるので，プログラムの巨大化，システムとの対話の増加によって SYNC の演算速度が急激に悪化する．これを救うためには，現状では適当な時点で SYNC の再起動をするのみである．しかし，ハードウェアの進歩に加えて Prolog 処理系そのものの進化も激しいので，速度面の問題はさして重要ではないと考えられる．ただし，Prolog の制御方法を変え，データベースからの検索を高速にすることが SYNC の高速化をも含めた問題点であることに代わりはない．現在の SYNC では，実行例からも明らかなように 2 次元出力など，ヒューマンインタフェースの開発は不十分である．しかし，このような出力ルーチンの組み込みは困難ではない．

FORTRAN とのインタフェースについて詳しく述べたが，述語 **fortran** に似た述語を作成することによって他の数値計算プログラムとの結合を実現することも可能であると思われる．例えば，数値的保証を常に考慮する区間解析に基づく PASCAL-SC[7] と結合することができれば，ハイブリッド処理システムとしてはある種の究極の目標を達成することができると思われる．

**謝辞** 著者は本論作成に当たり，変わらぬ御激励をいただいた愛媛大学工学部相原恒博教授に感謝する．

## 参考文献

1) Sasaki, T. and Noda, M. T.: Approximate Square-Free Decomposition and Root-Finding of Ill-Conditioned Algebraic Equation, *J. Inf. Proc. Japan*, (to be published).
2) 野田松太郎，岩下英俊：数値・数式ハイブリッドシステムと常微分方程式，情報処理，Vol. 28, No. 7, pp. 689-696 (1987).
3) Mawata, C.: A Sparce Distributed Representation Using Prime Numbers, *Proc. SIMSAC '86*, Waterloo, Ontario, pp. 110-114 (1987).
4) Buchberger, B.: Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory, in *Mathematical System Theory* (ed. Bose, N. K.), pp. 184-232, Reidel Pub., Dordrecht, Holland (1985).
5) Char, B. W., Fee, G. J., Geddes, K. O., Gonnet, G. H. and Monagan, M. B.: Maple: A Sample Interactive Session, *Jour. Symb. Comp.*, Vol. 2, No. 2, pp. 179-200 (1986).
6) Suzuki, M., Sasaki, T., Sato, M. and Fukui, Y.: A Hybrid Algebraic-Numeric System ANS and Its Preliminary Implementation, in *Winter Institute '88* 計算数学の世界 (Yamamoto, T. ed.), pp. G1/9-10, 科研費総合研究 (A) 報告集 (1988).
7) Rall, L. B.: An Introduction to the Scientific Computing Language Pascal-SC, *Trans. Second Army Conference on Applied Mathematics and Computing*, pp. 117-148, U. S. Army Research Office, Research Triangle Park, NC. (1985).

（昭和 63 年 8 月 26 日受付）
（平成 元 年 1 月 17 日採録）

**野田松太郎**（正会員）

昭和 44 年大阪市立大学理学研究科博士課程修了．理学博士．日本学術振興会奨励研究員を経て愛媛大学工学部情報工学科に勤務．現在，同助教授．数値計算と数式処理の結合と「AI 化」およびそれに基づく新しいアルゴリズムの開発に強い関心をもっている．ACM，ソフトウェア科学会，人工知能学会，日本物理学会各会員．

**岩下 英俊**（正会員）

1963 年．1986 年愛媛大学工学部電子工学科卒業．1988 年同大学工学研究科電子工学専攻（修士課程）修了．同年(株)富士通研究所に入社．以来，並列計算機の研究・開発に従事．数式処理，並列言語処理系，並列応用等に興味をもつ．電子情報通信学会会員．