Vol. 25 No. 6　　情報処理学会論文誌　　Nov. 1984

非手続き的表現の一つとしての条件式の Pascal への導入†

渡辺勝正††　榎本好晴††　都司達夫††

プログラムを書きやすくして，ソフトウェアの生産性と信頼性を向上させる一つの方法として，非手続き的な表現を用いることが考えられる．本論文では，非手続き的表現の一つとして，「条件式」をとりあげ，それを Pascal に組み込んで実現する処理系について述べている．「条件式」は，値を求めたい変数が満たすべき条件をそのまま表現することを可能にするもので，現在のところ，1元3次の多項式，2元2次の多項式，および，初等関数を含んだ1変数の代数式で表せるものを可能とした．処理系は，条件式を含んだプログラムを解析して標準の Pascal プログラムに変換する部分と，実行時に条件式を処理して解を求める操作を行う部分とから成っている．本論文では，処理系の構成とその実現方法，条件式を解いて得られた複数個の解を有効に利用するための限定条件の導入とその処理方法，実現した処理系の大きさと評価，および，条件式の応用例について述べている．

1. まえがき

一般に，現在，高水準言語とよばれているもののほとんどでは，ある計算式中の変数が満足する新しい値を求める場合，計算の手順をあらかじめ求めておいて，一連の代入文に分解して記述しなければならない．

例1．連立方程式

$$\begin{cases}(x-2.0)(1.0+y)=-aa\\ 2.0(x+1.0)+y=5.0\end{cases}$$

を解く場合，あらかじめ

$$2.0\cdot x^2-8.0\cdot x+(8.0-aa)=0$$

$$y=3.0-2.0\cdot x$$

と変形して，代入文の系列に書き換えることでxとyの値を求める．

例1のように方程式を一連の代入文に分解すると，もとの関係式が表している意味内容がまったく失われてしまう．また，展開途中で誤る可能性がある．

一方，条件式は求めたい変数が満たすべき「条件」をそのままプログラム中に記述しようとするもので，つぎのように未知変数の指定と関係式とで与えることにする．

未知変数名：(式1　関係子　式2)

例2．例1を条件式を使用して表現する．

x, y: $((x-2.0)*(1.0+y)=-aa)$

x, y: $(2.0*(x+1.0)+y=5.0)$

これによって，プログラム上での不注意による誤りが少なくなるばかりでなく，式中の変数と定数のもつ意味が保存されるため，プログラムがわかりやすくなる．

条件式の考えは，データベース操作言語のなかに一部採用され，実現されているが，汎用の高水準言語の中に組み込まれたものは知られていない．また，外見上数式処理に関連したもののように考えられるが，条件式の処理は，数式処理の方法とは違っている．数式処理の場合には，与えられた数式に対して，式中の変数，係数，演算子を記号とみなし，数学における各手法を用いて，目的に合った式の変形が行われる．条件式の処理の場合には，与えられた代数式において，値を求めたい変数（未知変数）を識別して，値をもっている変数（既知変数）と定数の値を用いて，条件式を変形しながら標準形にまとめてゆき，未知変数の値が求められる．これは代入文を一般化したもので，変数の値をそれが満たす条件式を与えることによって指定する形でプログラムを記述するものである．

本論文では，①条件式の表現形式，②条件式を含んだプログラムを標準の Pascal プログラムに変換する処理系（プリプロセッサ）の構成，③実行時に条件式を処理し解（本論文では実数解に限定）を求める操作を行う処理系（ランタイムプロセッサ）の構成，④複数個の解を有効に利用する方法としての限定条件の導入とその処理，⑤条件式を適用した応用例，⑥実現した各処理系の大きさとその評価，について述べてゆく．

2. 条件式の表現形式

プログラム中での条件式の表現形式をつぎのように定める．

† Conditional Expression Embedded in Pascal: as a Non-procedural Representation by KATSUMASA WATANABE, YOSHIHARU ENOMOTO and TATSUO TSUJI (Faculty of Engineering, Fukui University).

†† 福井大学工学部情報工学科

表現場所：Pascal のプログラムにおいて，代入文が表現可能なすべての場所

表現書式：値を求めたい変数（未知変数）の指定と関係式とで与える

未知変数名：(式1　関係子　式2)

さらに，条件式の表現にはつぎの規則を設けている．

- 未知変数の指定は2個まで可能とする．
- 関係式は四則と等号で表された代数式とする．

(単一型条件式) 未知変数の指定が1個で，関係式が未知変数の3次までの多項式．

(連立型条件式) 未知変数の指定が2個で，二つの関係式がそれぞれの未知変数の2次までの多項式．

ただし，二つの関係式はプログラムの中で連続して与えられ，それらの間に他の文が入ることはできない．

(超越型条件式) 未知変数の指定が1個で，関係式中に Pascal で使用可能な標準算術関数 (sin, cos, arctan, sqrt, ln, exp, abs) を使用し，その引数に未知変数を含んでいる代数式．

例3．連立型条件式の適用例．

```
begin  aa:=2.0;
  x, y: ((x-2.0)*(1.0+y)=-aa);
  x, y: (2.0*(x+1.0)+y=5.0);
end.
```

例4．超越型条件式の適用例．

```
begin  read(m);
  z: (m=m*ln(z)-sin (z+1.0));
end.
```

3. 処理系の構成

3.1 プログラムの処理過程

条件式を含んでいるプログラムの処理と実行は，つぎの手順で行う（**図 1**）．

〔1〕（前処理）プリプロセッサは，条件式を含んだユーザプログラムを読み込んで，つぎのように Pascal のプログラムを生成する．

a）プログラムの構文解析を行い，条件式を識別する．

b）それぞれの条件式の型に対応する条件式解析用プロセッサにて解析された結果をもとにして，条件式を Pascal 型に変換する．

c）複数個の解を有効に利用するために導入された限定条件を Pascal 型に変換する．

d）もとのユーザプログラムで条件式以外の部分，b），c）で Pascal 型に変換・生成された部分，および，実行時に必要となるデータを編集して，出力プログラム（ファイル）を生成する．

e）実行時に超越型条件式の解を求めるための超越型条件式処理用プロセッサのプログラムを生成する．

〔2〕（翻訳・結合）生成されたプログラムをコンパイルし，実行時に必要なプログラムを結合する．

〔3〕（実行）結合・編集された目的プログラムを実行する．結合された条件式処理用プロセッサおよび解選択用プロセッサは，対応する条件式の処理および解の選択を行うとき起動される．

3.2 プリプロセッサの構成

プリプロセッサは，本体と，条件式を解析する条件式解析用プロセッサから成り立っている．

3.2.1 プログラムの構文解析

プログラムの解析は，プリプロセッサ本体が，Pascal のプログラム構造にしたがって，コンパイラが行うのと同様の方法で行う．ただし，条件式は〈実行文部〉中に存在するため，〈宣言部〉については，**program, var, procedure, begin** などの関連する

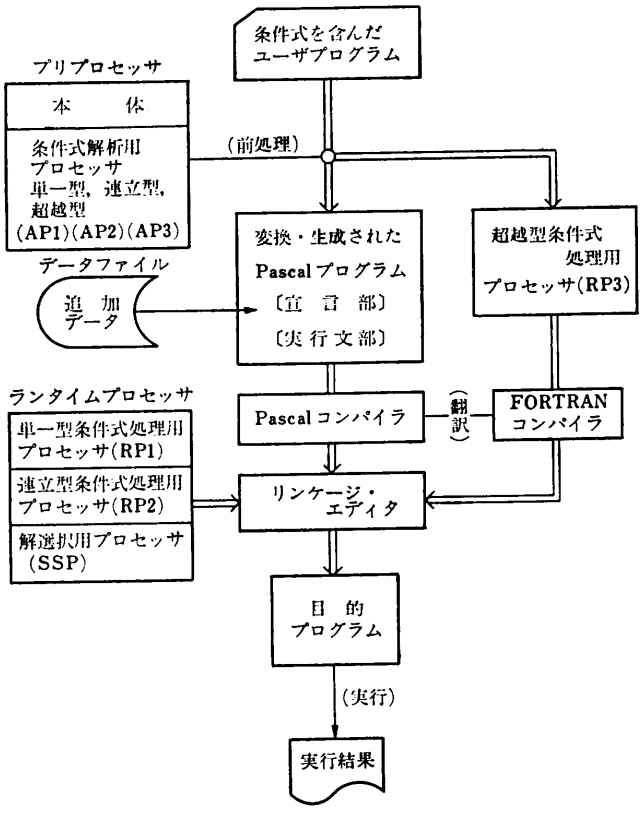

図 1　プログラムの処理と実行
Fig. 1　Processing and execution of the program.

表 1 〈実行文部〉解析用キー
Table 1 Keys for syntax analysis.

キー 1	**begin**	**repeat**	**then**	**else**
	do	:	;	
キー 2	**end**	**else**	**until**	;
	行末 (end-of-line)			
	式中の '('と')' の数が等しい			
キー 3	**case**	**end**		
キー 4	**all**	**endall**		
	some	**endsome**		

予約語をもとにして，各部分を識別する程度の解析に留めている．〈実行文部〉での構文解析では，条件式の表現場所の規則にしたがって，表1のキー1をもとにして条件式の存在する場所を検出する．実際に読み込んだ文が条件式かどうかの判断は，条件式の表現形式にもとづいて行う．条件式の終端の判定は，キー2をもとにして行う．キー3は，〈実行文部〉中での **begin** と **end**, **case** と **end** の照合を行うために使用する．

3.2.2 条件式解析用プロセッサの構成

プリプロセッサ本体は，ある文を条件式だと判定すると，条件式を表す文字列を引数として，対応する条件式解析用プロセッサに渡す．このプロセッサは，演算子，関係子などの優先順位にもとづいて条件式の構文解析を行い，その結果を木構造で表現する．さらに，条件式を Pascal 型に変換するために必要となるデータ（木構造表，変数表など）を作成し，プリプロセッサ本体に返す．

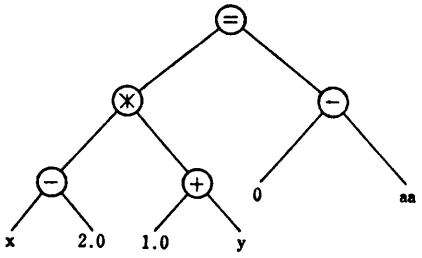

図 2 多項式の木構造
Fig. 2 Tree structure of the polynomial equation.

表 2 木構造表
Table 2 Table of the tree structure.

ノード	親のノード pa	左ノード l	オペレータノード op	右ノード r
1	3	101	5 (−)	103
2	3	104	4 (+)	102
3	5	1	8 (*)	2
4	5	100	5 (−)	105
5	0	3	3 (=)	4

表 3 変数表
Table 3 Table of variables.

番号	名前 ident	長さ lng	値 valu
100	'0'	1	0
101	'x'	1	x
102	'y'	1	y
103	'2.0'	3	2.0
104	'1.0'	3	1.0
105	'aa'	2	aa

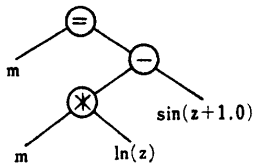

図 3 超越方程式の木構造
Fig. 3 Tree structure of the transcendental equation.

プリプロセッサには，つぎの3種類がある．

〔1〕 単一型条件式解析用プロセッサ（AP1）

〔2〕 連立型条件式解析用プロセッサ（AP2）

たとえば，例3の連立型条件式の第1式は，木構造（**図 2**）に対応して，木構造表（**表 2**）と変数表（**表 3**）が作成される．表3の中で '0' は，単項演算の部分を二項演算の形式で表現して，演算を行うために使用される．

〔3〕 超越型条件式解析用プロセッサ（AP3）

このプロセッサは，条件式中の算術関数の項をひとつの端末ノードとして扱う．

たとえば，例4の超越型条件式は，**図 3** のような木構造となり，ln(z)，sin(z+1.0) はそれぞれ一つの変数のように扱われる．

ところで，超越型条件式は式中に算術関数を含んでいるため，その解を求めるのに Mueller の方法[3]を用いている．このとき，逐次計算の初期値が必要となるが，そのため座標上のある一定範囲（検査区間）を順に解が存在する可能性があるかどうかを調べている．この検査区間は，式中の算術関数の種類と引数の構造より，関数値がオーバーフロー，アンダーフローを起こさないように決定される．この作業は，条件式の構文解析と並行して行われる．

3.2.3 条件式の Pascal 型への変換

プリプロセッサ本体では，条件式解析用プロセッサから返されたデータをもとにして，条件式の部分を対

```
procedure exdata01; {第1式の変換結果}
begin
    anyany:=1; normalf:=0; npstop:=105; te:=5;
    name[100].ident:='0'  ; name[100].lng:=1; name[100].valu:=0  ;
    name[101].ident:='x'  ; name[101].lng:=1; name[101].valu:=x  ;
    name[102].ident:='y'  ; name[102].lng:=1; name[102].valu:=y  ;
    name[103].ident:='2.0'; name[103].lng:=3; name[103].valu:=2.0;
    name[104].ident:='1.0'; name[104].lng:=3; name[104].valu:=1.0;
    name[105].ident:='aa' ; name[105].lng:=2; name[105].valu:=aa ;
    t[1].pa:=3; t[1].l:=101; t[1].op:=5; t[1].r:=103; {変数表}
    t[2].pa:=5; t[2].l:=104; t[2].op:=4; t[2].r:=102;
    t[3].pa:=5; t[3].l:=1  ; t[3].op:=8; t[3].r:=2  ;
    t[4].pa:=5; t[4].l:=100; t[4].op:=5; t[4].r:=105;
    t[5].pa:=0; t[5].l:=3  ; t[5].op:=3; t[5].r:=4  ; {木構造表}
    extsub22(name, t, npstop, te, anyany, sollnum, soll, .....);
end;                                    {条件式処理用プロセッサの呼出し}

procedure exdata02; {第2式の変換結果}
begin
    anyany:=2; normalf:=0; npstop:=105; te:=4;
        {第2式に対応した変数表と木構造表を代入文で構成}
        {第1式の変換方法と同様                        }
    extsub22 (name, t, npstop, te, anyany, sollnum, soll, .....);
    if normalf=0 then begin {限定条件の有無}
    if sollnum>0 then begin {解の個数}
        {vallimit 用のパラメータ値の設定}
        vallimit (....., sollnum, soll, .....); {解選択用プロセッサの呼出し}
        if sollnum>0 then                {解の値の設定}
          begin x:=soll[1].numbb1; y:=soll[1].numbb2 end;
                        end;
                        end;
end;
```

図 4　例3の変換結果
Fig. 4　Transfered program of Example 3.

```
(a)
    function f001(z:real):real;
    begin
        f001:=m-(m*ln(z)-sin(z+1.0));
    end;
(b)
    procedure exdata01;
    begin
        iisw:=1; normalf:=0; npstop:=103; pointflg:=3;
        name[100].valu:=0 ; name[101].valu:=z;          } (b1)
        name[102].valu:=m ; name[103].valu:=1.0;
        initdata;
        repeat                                          } (b2)
            subinit;
            repeat extlfort until pointstop=1;
        until repeatc=stoprepc;
        if normalf=0 then               {限定条件の有無}
        if trunc(stacksol[0])>0 then z:=stacksol[1];
    end;                                {解の値の設定}
```

図 5　例4の変換結果
Fig. 5　Transfered program of Example 4.

応する手続き（関数）に変換して生成する．

〔1〕 単一型条件式，連立型条件式の場合

手続きのなかには，まず変数表（name）と木構造表（t）を構成する代入文の系列が生成される（**図4**）．これらの代入文の右辺の変数の値は実行時に定まる．これらは，実行時に起動される条件式処理用プロセッサ（RP1 または RP2）へのデータ（引数値）である．

〔2〕 超越型条件式の場合

プリプロセッサ本体は，変数表をもとにして，条件式の部分を対応する手続きに変換して生成する（たとえば，図5(b)の exdata01）．さらに，木構造表も用いて，関係式を関数形式に変形して関数（たとえば，**図5**(a) の f001）を生成する．この関数（図5(a)）は，検査区間における関数値を求めるために使用される．生成された手続き（図5(b)）は，プリプロセッサで作り出された超越型条件式処理用プロセッサ（RP3）へのデータとして使用される変数・定数の値を代入文の形式で定義する部分（b1）と，検査区間の境界値を決定する手続き（initdata, subinit）と解が存在するかどうかを検査する方法が記述されている手続き（extlfort）とを呼び出す部分（b2）とから成り立っている．

3.2.4 出力ファイルの生成方法

変換された Pascal プログラムを格納する出力ファイルを生成するにあたっては，Pascal における制約—①使用可能なファイルは順編成ファイルだけである，②〈実行文部〉中で呼出しが行われる“手続き”および“関数”は特別な場合を除いて，その呼出し以前の〈手続き関数宣言部〉にて宣言しなければならない，③プログラムは全体としてブロックの入れ子構造を取りう

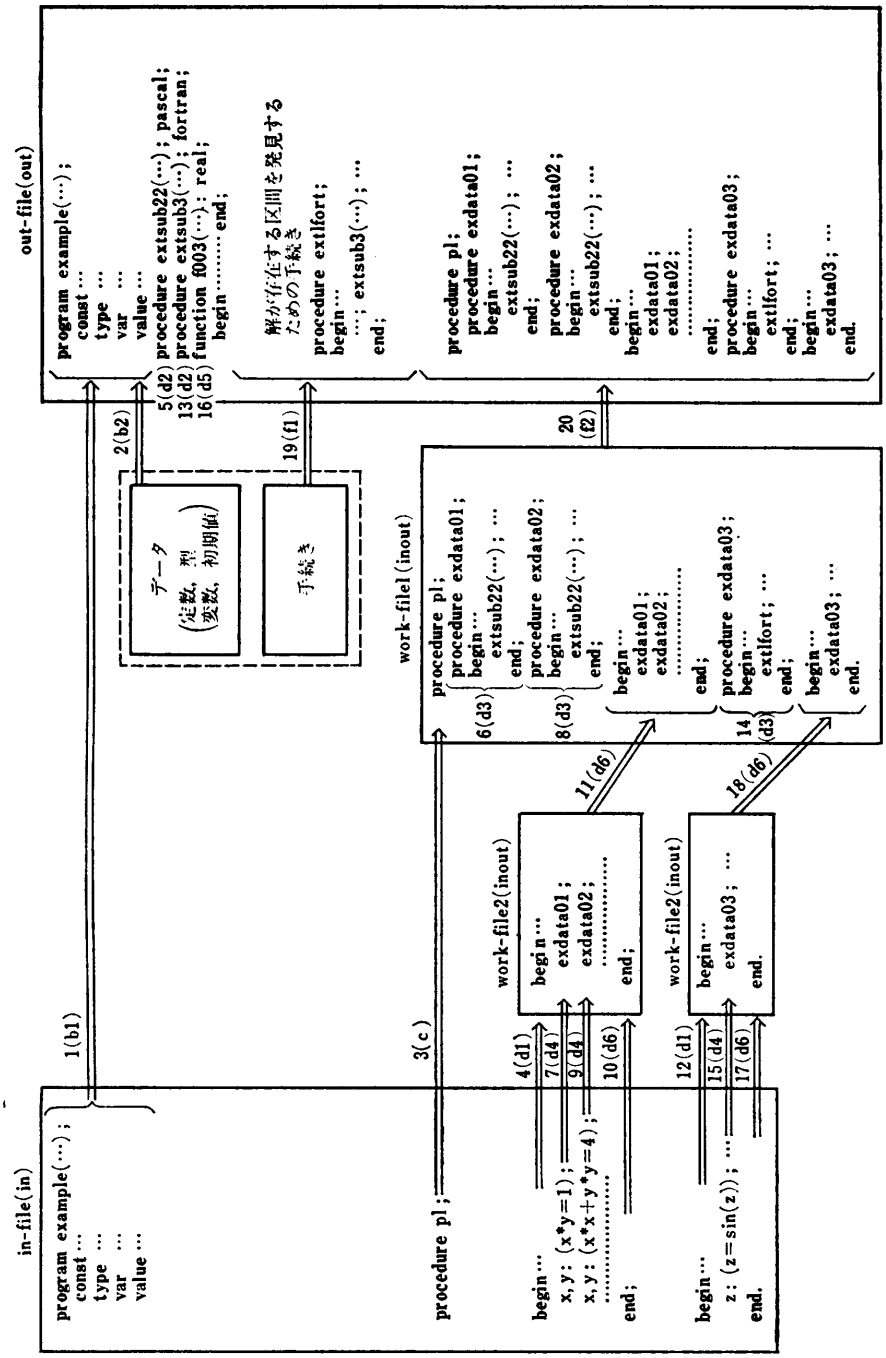

図 6 ファイルの生成
Fig. 6 Generation of out-file.

る一を満足するように工夫した.

生成は, つぎの手順にしたがって行う (図 6).

〔a〕 プリプロセッサ本体がソースファイルから1レコードずつ読み込んで処理を開始する.

〔b〕 (b1)プリプロセッサ本体によるプログラムの構文解析の結果をもとにして, グローバルな宣言・定義・設定部を出力ファイルに書き出す. (b2)条件式を Pascal 型に変換する上で必要となるデータ (定数, 型, 変数, 初期値) はあらかじめファイルの形式で準備しておき, 必要な時点で各部分に追加する.

〔c〕 〈手続き関数宣言部〉 を読み込んだ時点で, 書出しを作業用ファイル1に切り換え, **begin** を読み込むまでローカルな宣言部の内容を書き出す.

〔d〕 (d1)〈実行文部〉 の **begin** を読み込んだ

時点で，書き出しを作業用ファイル2に切り換える．(d 2)条件式が存在している場合，実行時に必要となる対応した条件式処理プロセッサ（図6の extsub 22 など）を呼び出すための外部手続き宣言部を出力ファイルに書き出す．(d 3)条件式解析用プロセッサによって生成されたデータをもとにして作成した手続き(図6の exdata 01 など）を作業用ファイル1に書き出す．(d 4)もともと条件式が存在していた部分には(d 3)での手続きを呼び出す文を作業用ファイル2に書き出す．(d 5)なお，超越型条件式の場合は，先(3.2.3項）に述べた関数宣言部（図6の f003 など）を出力ファイルに書き出す．(d 6)あるブロックの〈実行文部〉が閉じた時点で，作業用ファイル2の内容を作業用ファイル1に書き出す．

〔e〕 ソースファイルが end-of-file の状態になるまで〔c〕，〔d〕の操作を続ける．

〔f〕 ソースファイルが，end-of-file の状態となり，(f 1)ユーザプログラム中に超越型条件式が含まれていた場合，あらかじめファイルの形式で準備しておいた解の存在する可能性を検査するための各種手続き（図6の extlfort など）を出力ファイルに書き出す．(f 2)最後に，作業用ファイル1の内容を出力ファイルに書き出し，出力ファイルの生成を完了する．

3.2.5 超越型条件式処理用プロセッサの生成

超越型条件式処理用プロセッサ（RP 3）はあらかじめ用意されているのではなく，プリプロセッサ本体によって FORTRAN 副プログラムの形式で作成される．

超越型条件式の解を求める方法は，Mueller 法による逐次近似の収束計算にて解を求めている．実際には，MELCOM-COSMO 700Ⅱ の数値計算用サブルーチン・パッケージ NSP に登録されている一般の非線形方程式を解くためのサブルーチン MUELLER[3] を利用している．

この副プログラムの構成および生成はつぎのようになる．

(サブルーチン副プログラム)

主として，①サブルーチン MUELLER のパラメータ値を設定している代入文，②各超越型条件式ごとにサブルーチン MUELLER の呼出しを行っている文，③得られた値が本当に解かどうかを調べている文，から成り立っている．

(関数副プログラム)

④条件式中の関係式を関数形式に変形して生成された部分，サブルーチン MUELLER のパラメータの一つとして使用される．

とくに，①と③の部分は固定部分であるので，あらかじめデータとしてファイルの形式で準備しておき，プロセッサ（RP 3）の生成時にその都度書き出す．それ以外の②と④の部分は，各超越型条件式に対してサブルーチン MUELLER を呼び出す部分が必要なため，プリプロセッサ本体が超越型条件式を検出するたびに生成する．以上作成された部分を FORTRAN のプログラム構造にしたがって，最終的に編集して，プログラムの生成を完了する．

3.3 実行時の条件式の処理方法

3.3.1 条件式処理用プロセッサの構成

条件式処理用プロセッサは，条件式を Pascal 型に変換して生成された手続きのなかの各種情報（変数表，木構造表など）をデータとして受け取り，条件式を処理する．条件式のすべての実数解を求めて，それらを結果として生成された手続きに返す．

このプロセッサには，つぎの3種類がある．

〔1〕 単一型条件式処理用プロセッサ（RP 1）

条件式の処理過程は，つぎの連立型条件式の場合とほとんど同じであるので説明は省略する．

〔2〕 連立型条件式処理用プロセッサ（RP 2）

(値取り) 変数表 (name) と木構造表（t）をもとにして，木の端末ノードに対して，既知変数と定数は値を取り込んでスカラー型とし，未知変数は多項式の係数を用いたベクトル型とする．

未知変数が2種類の多項式

$$a_0+a_1x+a_2y+a_3x^2+a_4xy+a_5y^2+\cdots$$

は，ベクトル $[a_0, a_1, a_2, a_3, a_4, a_5, \cdots]$ で表現できる．

現時点では，x, y の2次多項式を考えているので

$$a_0+a_1x+a_2y+a_3x^2+a_4xy+a_5y^2$$

は，$[a_0, a_1, a_2, a_3, a_4, a_5]$ と表現する．たとえば，未知変数 x は，[0.0, 1.0, 0.0, 0.0, 0.0, 0.0] となる．

(刈込み) 各演算ノードは，二つの子のノードの形状（スカラー型は T，ベクトル型は X）が決定したら，その組合せに応じて演算を行い，その結果を親のノードに知らせる．この処理を刈り込みとよび，刈込み規則[1] にしたがって行う．

刈込みは，既知の値を用いて部分木を一つのノードにまとめてゆく手順で，その規則の一部はつぎのようである．

$(T)+(T)$ は，加算により，T 型ノードになる．

$(T)*[X]$ は，定数倍により，X 型ノードになる．

$[X]-[X]$ は，多項式の減算により，X 型ノードになる．

$[X]/[X]$ は，多項式の除算を行わず，その状況が N 型として親ノードに伝えられる．

$[X1]/[X2]+(T)$ は，通分により，N型ノード $[X1+T\cdot X2]/[X2]$ となる．

この適用例の一部を例5に示す．

(標準形へのまとめ) 刈込み可能なノードをすべてまとめてゆくと，条件式は最終的に

$$X=0,\quad T/X=0,\quad X_1/X_2=0$$

のいずれかの形に落ち着く．これを標準形とよぶことにする．

(標準形を解く) 2個の条件式がそれぞれ標準形にまとめられると，それらを連立方程式として解き，未知変数の値を求める (詳細は文献 1)，2) を参照)．

例5．例3の連立型条件式の処理

$x, y: ((x-2.0)*(1.0+y)=-aa)$

(値取り) $([x]-2.0)*(1.0+[y])=0-2.0$

(刈込み) $[x+-2.0]*[y+1.0]=-2.0$

$[-2.0+x+-2.0*y+x*y]-(-2.0)=0$

(標準形) $[x+-2.0*y+x*y]=0$

$x, y: (2.0*(x+1.0)+y=5.0)$

(値取り) $2.0*([x]+1.0)+[y]=5.0$

(刈込み) $2.0*[x+1.0]+[y]=5.0$

$[2.0+2.0*x+y]-(5.0)=0$

(標準形) $[-3.0+2.0*x+y]=0$

(連立方程式の解)

$$\begin{cases} x+-2.0*y+x*y=0 \\ -3.0+2.0*x+y=0 \end{cases}$$

$(x, y)=(1.0,\ 1.0),\ (3.0,\ -3.0)$

〔3〕 超越型条件式処理用プロセッサ (RP3)

このプロセッサの働きは，出力ファイル生成時に追加された解の存在する可能性を検査するための各種手続き (図6の extlfort など) によって検出された解が存在する区間と，その条件式の番号と，条件式中の変数・定数の値とをデータとして受け取り，サブルーチン MUELLER を用いて，実数解を求めて返すことである．

3.3.2 解を求める方法

〔1〕 単一型条件式

標準形にまとめられた多項式を，解の公式および Cardano の方法を用いて解き，未知変数の値を求める．

〔2〕 連立型条件式

標準形にまとめられた2個の多項式を連立にして解く．それぞれの標準形はたかだか2次までの多項式であるから，できるだけそれぞれの式を変形して，未知変数が1種類の方程式にまとめて，解の公式を用いて解く．この方法で解くことができない場合は，高次連立方程式を解くニュートン・ラフソン法を採用する．

〔3〕 超越型条件式

超越型条件式解析用プロセッサ (AP3) によって決定された検査区間を座標上に設定し，それをいくつかの小区間に分割して，その両端をサンプル点に定める．超越方程式を $f(x)=0$ (たとえば，図5(a)) として，定められた小区間を定められた方向に，各サンプル点における関数値の符号を調べてゆく．サンプル点 x_{i-1}, x_i に対して，$f(x_{i-1})\cdot f(x_i)=0$ の場合は $f(x)=0$ となっている x の値を解として採用し，$f(x_{i-1})\cdot f(x_i)<0$ の場合は区間 $x_{i-1}<x<x_i$ に解が存在するため，x_{i-1} と x_i を初期値として，サブルーチン MUELLER で解を求める．

4. 限定条件の導入とその処理方法

4.1 限定条件の意味

条件式を解いて未知変数の値を求めたとき，実数解が複数個得られる場合がある．ノイマン型の計算機では，プログラム全体で，得られた複数個の解すべてを同時に並列に使用して計算を行うことは不可能である．そこで，得られた複数個の解を有効に利用できる方法として，あらかじめ求めたい解の含まれている範囲とその解の使用方法とを限定条件の形式で条件式の前に記述することにする．

4.2 限定条件の表現形式

限定条件の書式をつぎのように定める．

{**all** / **some**} 未知変数 {**in** 〔下限‥上限〕 / 範囲の制限なし}

sat 条件式；

(**all** の場合) 条件式を解いた結果，指定された範囲を満たすすべての値を順番に解として使用する．

(**some** の場合) 条件式を解いた結果，指定された範囲を満たす解のうち，一つだけ使用する．

なお，未知変数の指定が2個の場合は，それぞれの未知変数に対して求めたい解の範囲の指定を行う．

プログラム中での限定条件の有効範囲を，**all** と

```
    ⋮
some z in [low..up] sat                    {レベル0}
    z:(m=m* ln(z)−sin(z+1.0));
    ..........
    all x in [0..5], y in [−5..0] sat      {レベル1}
        x, y:((x−2.0)*(1.0+y)=−2.0);
        x, y:(2.0*(x+1.0)+y=5.0);
        ..........
    endall (x, y) ·····
    ..........
endsome (z) ·····
    ⋮
```

図 7　限定条件のついたプログラム
Fig. 7　Program with qualifier.

endall, **some** と **endsome** の形式で設定し，この範囲内で限定条件を満足する値を解として使用する（**図 7**）．限定条件のついた条件式には，入れ子構造を最大レベル 5 の状態まで認めている．

4.3 限定条件の解析

限定条件の発見と，その有効範囲がどこまでかを識別するために，表 1 のキー 4 をもとにして〈実行文部〉の構文解析を行う．読み込んだ文が限定条件の場合，解析し，その結果限定条件を Pascal 型に変換する上で必要となるつぎのようなデータを設定する．①限定条件の有無．②**all** 型か **some** 型かの判別．③条件式のタイプ．④範囲の制限の有無．⑤指定範囲の境界値．⑥入れ子構造のレベル．

4.4 限定条件の Pascal 型への変換

前節のデータの中で，①は条件式を Pascal 型に変換して生成した手続き（図 4）中で normalf に値（無の場合 0，有の場合 1）を設定する．②から⑥は，その内容を代入文の形式で値を設定する．

次に，限定条件の有効範囲の部分を一つの“**begin** … **end**”の単位として扱う．その構成内容はつぎのようになる．①限定条件の内容を設定する代入文．②条件式に対応した部分の手続き呼出し部．③解選択用プロセッサを起動させるための手続き呼出し部（詳細は次節で述べる）．④**for** 文を使用して複数個の解を順番に設定して有効範囲内を実行する部分．⑤限定条件の有効範囲の対象（入れ子構造のレベル）を変更するための代入文．これらの内容は，もともと条件式がプログラム上で存在していた場所に対応して，いったん，作業用ファイル 2 に書き出される．

4.5 解選択用プロセッサの構成

解選択用プロセッサ（SSP）は，条件式を解いた結果得られたすべての実数解と，限定条件の内容とをデータとして受け取り，条件を満足する値を選択して，実際にプログラム上で使用できる形にまとめて，返す．

例 6．図 7 の連立型条件式の場合

連立型条件式のすべての解は

$(x, y) = (1.0, 1.0), (3.0, -3.0)$

であるが，限定条件によって，最終的には，(3.0, −3.0) を解として採用する．

5. 条件式の応用例

レバーのついた回転軸[4]（**図 8**）を考える．レバーの重力によるトルク T は，$T = w \cdot r \cdot \cos\theta$ となり，軸の終端はレバーに沿って角 θ だけ回転する．回転軸が弾性体でできていれば角 θ に関するトルクは，$T = k \cdot \theta$（ただし，k は比例定数）となる．以上より角 θ を求める超越方程式が得られる．

$$w \cdot r \cdot \cos\theta = k \cdot \theta$$

（設問）　回転軸が 1 度回転するのに必要なトルク（torque）がわかっている場合，レバーの重さ（w）と長さ（$2r$）が与えられると，回転軸が何度（theta）回転するかが計算できる．

条件式を使用したプログラムは**図 9** のようになる．

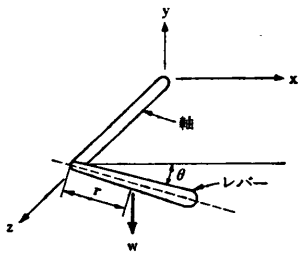

図 8　レバーのついた軸
Fig. 8　The shaft and the lever.

```
program example (input, output);
    const pai=3.1415;
    var angle  torque, k, w, r, rad, theta: real;
begin
    read (angle, torque);
    rad: =pai*angle/180.0;
    k:(torque=k*rad);
    read (w,r);
    all rad sat
        rad: (w*r* cos (rad)=k*rad);
        theta: =rad*180.0/pai;
        writeln ('  ':,'theta=', theta:10:4);
    endall (rad);
end.
```

図 9　角 θ を求めるプログラム
Fig. 9　Program to solve the angle θ.

表 4 プリプロセッサの大きさ
Table 4 Program size of the Pre-processor.

名 称	プリプロセッサ本体			条件式解析用プロセッサ		
処理内容	構文解析	変換とファイル生成	プロセッサの生成	単一型 AP 1	連立型 AP 2	超越型 AP 3
行 %	2,116行 64.2%	884行 26.8%	296行 9.0%	475行	450行	850行
サブトータル	3,296行			1,775行		
%	41.7%	17.5%	5.8%	9.4%	8.9%	16.7%
	65.0%			35.0%		
トータル	5,071行					

表 5 実行時に呼び出されるプロセッサの大きさ
Table 5 Program size of the Run-time processor.

名 称	条件式処理用プロセッサ						解選択用プロセッサ SSP		
	単 一 型 RP 1			連 立 型 RP 2					
処理内容	条件式の処理	方程式の解法	解の整理	条件式の処理	連立方程式の解法	解の整理	単一型	連立型	超越型
行	1,000 行	167 行	45 行	1,168 行	1,110 行	443 行	37 行 8.0%	320 行 69.1%	38 行 8.2%
%	82.5%	13.8%	3.7%	42.9%	40.8%	16.3%	〔共通部分〕 68行 14.7%		
トータル	1,212 行			2,721 行			463 行		

6. 処理系の大きさ

本論文では，条件式の含まれた Pascal プログラムを実行するための処理系の実現の方法について述べてきた．この処理系は Pascal と一部 FORTRAN で書かれている．プリプロセッサの大きさと実行時に必要となるプロセッサの大きさの目安として，簡単なコメント行を含めたソースプログラムの行数を表 4，5 に示す．

プリプロセッサのなかで，条件式解析用プロセッサを合計した大きさは全体の 35% を占めている．今後，条件式の種類をいろいろ拡張してゆく場合，より効率のよいプロセッサを作成する必要がある．表 5 には，超越型条件式処理用プロセッサ (RP 3) は含めなかった．これは，ユーザプログラム中での超越型条件式の個数とその構造によって大きさが変わってくるからである．連立型条件式処理用プロセッサ (RP 2) では方程式を解く部分の占める割合が，単一型 (RP 1) の場合に比べて大きくなっている．これは連立方程式を解く手順が複雑なためである．今後，代数分野に関して条件式を拡張してゆく場合，数値解析との関係をどのようにしてゆくかが問題である．

7. むすび

非手続き的表現の一つとしての条件式を Pascal 言語に組み込んだことにより，プログラムを書きやすくするだけでなく，プログラムが直観的に見やすくなり，ドキュメント性も向上されたと考えられる．

今回実現した処理系からつぎのことがいえる．

①プログラムの生成に関しては，プリプロセッサの編集・変換によって生成されたデータを必要最小限の手続きにまとめ，それをプログラムの生成時に追加し，それ以外の部分はできるだけ外部サブルーチンとして準備しておき，必要なものを実行時に結びつけるようにするとよい．②条件式を解いた結果，解が複数個得られる場合の処理方法は，限定条件を導入することで実用的な問題に対処できたと考える．③方程式の解法は初期値をいかにして見つけるかが大きな課題であるが，できるだけ既存のサブルーチン・パッケージと結びつけてゆくことが有効である．

なお，条件式は，非手続き的表現として汎用言語の中に組み込むだけでなく，パーソナルコンピュータで高級電卓のように会話型に用いることも考えられる．

本論文では，紙面の都合上，条件式を適用した例を

一つしか示さなかったが，今後，条件式による表現に適したアルゴリズムの開発が行われることを期待している．

参考文献

1) 渡辺勝正，榎本好晴，都司達夫：条件式による表現とその処理方式，福井大学工学部研究報告，Vol. 31, No. 1, pp. 95-106 (1983).
2) 渡辺勝正，榎本好晴，都司達夫：条件式の Pascal への導入とその応用例，昭和58年度電気四学会北陸支部連合大会，B-35 (1983).
3) MELCOM NSP 説明書（数値計算編），pp. 166-168，三菱電機株式会社.
4) サウスワース R. W., デロー S. L. 著，岩田倫典訳：電子計算機のための数学 I（数値計算），共立全書，第533，共立出版，東京 (1970).

（昭和59年3月12日受付）
（昭和59年5月15日採録）