表現能力に富む小さな文法について†

渡 辺 坦††

Abstract

This paper deals with the expressional capability of a programming language which is fixed in the basic format (syntax) of expressions, and free in selectiong operator names and key words. As examples, expressions of FORTAN, ALGOL, and PL/I are constructed within the extent of the basic syntax in a slightly modified form. The syntax is compact enough to be shown in a sheet of paper. The program to parse all the expressions based on this syntax is composed of 300 FORTRAN statements. The introduction of new expressions in this language can be accomplished in the same manner as writing subprograms in other languages.

This syntax makes it easy to develop problem oriented languages in many computer application fields.

1. まえがき

自然言語でも，外来語や専門用語などの新しい用語はとり入れてゆけるが，基本的な構文は固定されている．そこで，用語に対する制限はあまりないが，構文の基本パタンを固定したプログラミング言語を検討したところ，意外と表現能力に富むことがわかった．

本報告では，1ページに図示できる単純な構文を定め，その範囲内で記号の使い方を工夫することによりFORTRAN や ALGOL，および PL/I に似た表現を構成してみる．また，既存の言語にとらわれなければ，記号の選び方次第で，これらよりもさらに自然言語に近い表現がとれることを例示する．対応する構文解析ルーチンは，シラブル分けする部分を除くと，300 ステップあまりの小さい FORTRAN プログラムである．

設計やシミュレーションなどに使う大規模なアプリケーション・プログラムでは，処理の流れも，一部，入力データで指定することが多い．したがって，入力データを高レベルの問題向き言語の形で与えることができると，非常に使いやすいプログラムとなる．しかし，そのためにはコンパイラを作らなければならないので，これまで，少数の限られたプログラムでのみこの行き方がとられた．このような場合，本報告の方式を採用すると，それが容易に実現できる．

また，小型コンピュータやミニ・コンピュータでは従来，FORTRAN などが備わっていても，記憶容量が小さいため，使用上の制限が強く，あまり活用されなかった．この文法に対するコンパイラは，小さく作れるので，ミニ・コンピュータの分野でも参考になると思われる．

使える用語に自由度はあるが構文パタンが固定されている言語は，マクロ命令を定義できるアセンブラの形で，古くからある．また，ICES[2] や PLAN[3] も類似した方式をとっている．しかし，ここにあげる文法は，これらにくらべ，人間にとって非常に扱いやすい表現形式がとれることを特徴とする．

この文法は，記号の意味を記述する能力を持つプログラミング言語 EXTRAN (Expression Translator)[1] に採用した．EXTRAN の全容は別の機会に報告することにし，ここでは，この文法の表現能力と，その構文解析方法に重点をおいて述べる．

2. 構文

構文の基本パタンを定めるにあたって，二段階に分けて検討した．すなわち，はじめに論理的な扱いやすさのみを考慮して，「標準型」と呼ぶ表現形式を定めつぎにそれを書きやすく読みやすい形に変形することにした．

標準型としては，つぎの5種類の式を設ける．

† Compact Syntax Covering a Variety of Expressions, by Tan Watanabe (Central Research Laboratory, Hitachi, Ltd.)

†† 株式会社日立製作所中央研究所

定数　c
変数　x, $x(e_1, \cdots, e_n)$, $v_1, v_2, \cdots, v_n$
関数　f, $f(e_1, \cdots, e_n)$
リスト　$(e_1, \cdots, e_n)$
ブロック　DO; l_1: s_1;… l_n: s_n; END

ここで，$e_1, \cdots, e_n$ は一般の式を表わし，$v_1, \cdots, v_n$ は変数を表わす．関数は，計算した値を表わすのに使ってもよいし，操作を表わすのに使ってもよい．また，一群の宣言ステートメントをまとめ，それの名前として使うこともできる．l_i: s_i;, $(i=1, \cdots, n)$ はステートメントと呼ばれ，それは，操作を表わす式 s_i にラベルと呼ばれる記号 l_i をつけ，セミコロンで区切った記号列である．ラベル部分は必要なければ省いてよい．

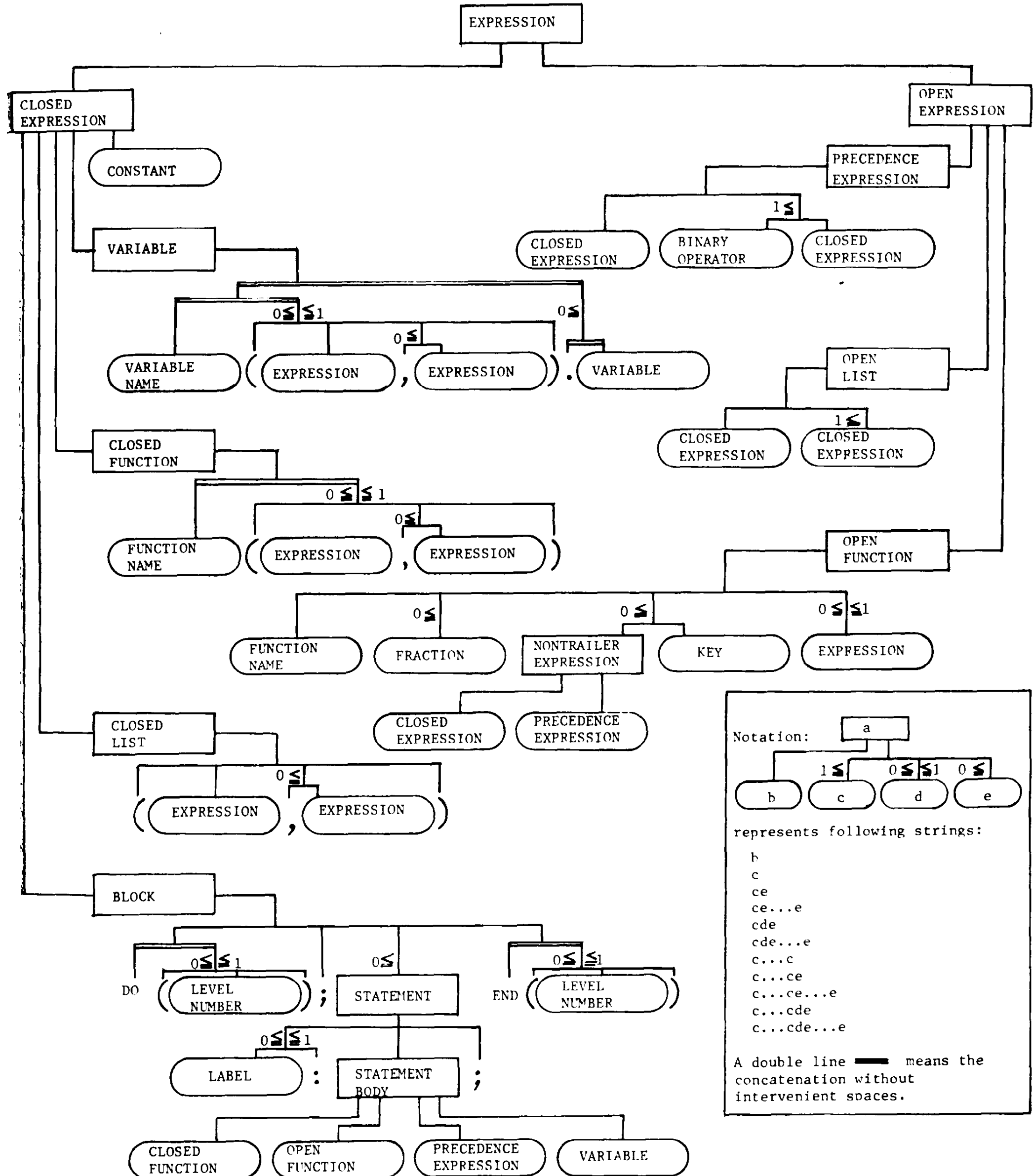

Fig. 1 Syntax Diagram

標準型の式を使いやすく変形したのが，図1に示す文法である．プログラムの構成要素を「式 (expression) と呼ぶ．プログラム自体も一つの関数型の式であり，その形は次節で説明する．図1では，次の構成要素の定義が省いてある．

（1） 定数 (constant)
1, 3.14, 1.0E6, “ABC”
などとする．

（2） 変数名 (variable name)
文字Xではじまる英数字列とする．頭文字で変数と他の記号とを区別するのは，プログラミング上の誤りを防ぐためである．構文解析上は他の種類の記号と一致しなければよい．

（3） 関数名 (function name)
X以外の英字で始まる英数字列とする．変数名も関数名も，63文字までの長さとする．

（4） 二項演算子 (binary operator)
＋，－，＊，／などの特殊文字で構成される記号列，および，（3）の関数名と同じ形の記号とする．特殊記号の列でないときは，前後にブランクをつけた形で使う．

（5） キィ (key)，フラクション (fraction)
関数名や二項演算子と同じく，X以外の英字で始まる英数字列，および特殊記号の列とする．ただし，関数名や二項演算子として使った記号はキーやフラクションにできない．ある関数のキーやフラクションを他の関数のキーやフラクションに使うことは許される．

（6） ラベル (label)
X以外の英字で始まる英数字列で，関数名やキー，フラクションと一致してはならない．文法上，ラベルも定数の一種とみなす．

（7） レベル番号 (level number)
符号のつかない整数とする．

この図だけでは，説明不十分な点もあるので，関数とリストとブロックについて，少し補足する．

関数には，閉じた関数と開いた関数，および，二項演算子型の関数がある．閉じた関数は，$e_1, \cdots, e_n$ を引数を表わす式として，f または $f(e_1, \cdots, e_n)$ の形をとる．開いた関数は，f_0 を関数名，$f_1, \cdots, f_m$ をフラクション，$k_2, \cdots, k_n$ をキーとして，

$$f_0\ f_1 \cdots f_m \qquad (1 \leq m)$$

$$f_0\ f_1 \cdots f_m\ e_1\ k_2\ e_2 \cdots k_n\ e_n \qquad (0 \leq m,\ 1 \leq n)$$

のいずれかの形をとる．開いた関数では，途中の引数 $e_1, \cdots, e_{n-1}$ は閉じた式か二項演算型の式でなければならないが，最後の引数 e_n は開いた式でもよい．たとえば，次の形の式が許される．

IF XI EQ 0 THEN GO TO L

(IF の第1引数)　(IF の第2引数)

ただし，IF と GO, EQ が関数名で，THEN がキー，TO がフラクション，Lがラベルであるとする．開いた関数は，制御や入出力，宣言などに使うステートメントを書くために設けてある．

二項演算子 $g_1, \cdots, g_n$ で結んだ式は，$a_1, \cdots, a_{n+1}$ を閉じた式として，

$$a_1\ g_1\ a_2\ g_2 \cdots g_n\ a_{n+1}$$

という形をとる．これは，次のように，算術式や代入ステートメントを表わすのに使う．

1.0－XA＊EXP(XB)
XP＝XA＊XB－(XC＋XD)

開いた関数

$$f_0\ f_1 \cdots f_m\ e_1\ k_2\ e_2 \cdots k_n\ e_n$$

を識別するとき，関数名とフラクションの列 $f_0 f_1 \cdots f_m$ 全体で一つの名前を表わすものとみる．キーは引数を分離するためのもので，識別には使わない．たとえば

GO TO L,
GO BY XI TO (L1, L2, L3)

の二つの式では，GO TO と GO BY をそれぞれの関数の名前とする．引数 $e_i, (2 \leq i \leq n)$ は，必要なければ省略して使ってもよいが，そのとき，直前のキー k_i も省略しなければならない．さらに，一つの関数が同じキーを重複して持つことは許されない．関数名の使い方をこのように制限したことと，変数名が他の記号と混同しないようにしたことにより，構文解析が非常に単純化されるが，あとでみるように，この制限内で高い表現能力を保持できる．

リストは，通常，閉じた式として，

$$(e_1, \cdots, e_n)$$

のように書くが，かっこやコンマで囲まれた所と開いた関数の最後の引数の場合には，閉じた式を並べた

$$a_1\ a_2 \cdots a_n$$

型の開いたリストにしてもよい．たとえば，

DIMESION XA(5) XB(8)

では下線部分が一つの引数として扱える．リストは，

IF (IF XI EQ 0 THEN XA ELSE XB)

```
        GT XC THEN RETURN
```

のように，開いた式を閉じた式として扱えるようにするためにも使われる．

ブロックは，処理手順を表わすために設けてある．たとえば，XI と XJ の最大公約数を求める手順を，

```
     DO;
  L: IF XI EQ XJ THEN
          RETURN(XI);
     IF XI GT XJ
          THEN XI=XI-XJ
          ELSE XJ=XJ-XI;
     GO TO L;
     END
```

のように書くのに用いる．ブロックが多重になって，END がいくつも連続する場合，その数を間違えるおそれがある．そこで，外側の DO と END の対に同じレベル番号をかっこで囲んでつければ，その内側に続く END を省略できることにする．したがって，次の二つの式は同じになる．

```
    DO(1); … DO; … END(1)
    DO; … DO; …END; END
```

3. 新しい表現の導入方法

図1の文法では，定数と変数，リスト，およびブロックの形式が固定されていて，関数の表現しかたにのみ自由度がある．その関数をどう表現するかを示すために，次の形で使われる特殊な関数 DEFINE を設ける．

DEFINE ($f(x_1, \cdots, x_n)$, e)

DEFINE ($f_0\ f_1 \cdots f_m\ x_1\ k_2\ x_2 \cdots k_n\ x_n$), e)

DEFINE ($x_1\ g\ x_2$, e, PRIORITY n)

これにより，導入する関数 $f(e_1, \cdots, e_n)$ や，$f_0\ f_1 \cdots f_m\ e_1\ k_2\ e_2 \cdots k_n\ e_n$，$e_1\ g\ e_2$ に対して，それが閉じた型，開いた型，二項演算子型のいずれの型であるかが示され，開いた型の場合，フラクションとキーの並び方も示される．二項演算子では，

```
      DEFINE (X1=X2, ASSIGN(X1, X2),
              PRIORITY 30)
```

のように，構文解析上の優先順位も合わせて指定する．

DEFINE 式の第2引数 e としては，導入する関数の意味（処理内容または宣言内容）を前節で述べた形の式で書く．そして，プログラム自体を一つの DEFINE 式とし，その中に，さらにいくつもの DEFINE 式を入れてよいことにする．このとき，一つのプログラムの中で，同じ記号に使い場所によって異なる意味を持たせてはならない．ただし，次の二つの場合は例外である．

（1） ある関数のキーやフラクションとして使った記号を他の関数のキーやフラクションとして使ってもよい．

（2） あるブロック内のステートメントにつけたラベルを他のブロックのラベルとして使ってもよい．一たんラベルとして用いた記号は，プログラム全体を通じてラベル以外の目的に使うことはできない．

このようにすると，導入する関数のシンタックスとセマンティクスを，原始プログラムの中で，同時に簡単に示すことができる．セマンティクスを正確に定めるには，四則演算や代入，選択，飛びこしなど，DEFINE 式を用いて導入する必要のない基本的な関数の表わし方とその意味を定め，また，変数やリスト，ブロックの評価のしかたを明確にしなければならない．しかし，セマンティクスについては，文献 1) と 4) で述べてあるので，省略する†．

従来のプログラミング言語では，一般のプログラマにとって，新しい機能を追加するのは困難であった．しかし，上の方式によると，サブプログラムを作るのと同じ手軽さで，新しい機能や表現形式が導入できる．

4. 表現能力

本節では，この文法の表現能力を検討するために，FORTRAN, ALGOL, および PL/I に類似した，3種類の表現形式を構成してみる．さらに，次節では，これらの既存言語にとらわれずに書くと，もっと人間にわかりやすい表現形式がとれることを例示する．

図2は FORTRAN に似せた表現であり，図3は ALGOL，図4は PL/I に対応する表現である．これらの図では，右側に既存言語での形式を示し，左側に本文法での形を示す．文字￥は，その行の残り部分が注釈であることを示す．構文の基本パタンを固定しても，関数名やフラクション，キーなどのつけ方を工夫するだけで，幅広い表現形式をとれることが察せられ

† 本報告では，文献 4) とくらべると，記法がかなり変えてある．記号の属性を宣言する式の導入も，属性指定用の基本的な式 (REAL など) を使って意味内容を書き表わした DEFINE 式による．

```
    XA=XB+XC*(XD-XE);                     ¥       XA=XB+XC*(XD-XE)
    XI=L10;                               ¥       ASSIGN 10 TO I
    CALL SUB(1,X);                        ¥       CALL SUB(1,X)
    COMMON XP XQ XR;                      ¥       COMMON XP,XQ,XR
L10: NULL;                                ¥    10 CONTINUE
    INITIAL (X,1.0 2.0 3.0);              ¥       DATA X/1.0,2.0,3.0/
    DIMENSION XP(10) XQ(20) XR(30);       ¥       DIMENSION XP(10),XQ(20),XR(30)
    FOR XJ FROM 1 TO 20 BY 2 REPEAT DO;   ¥       DO 15 J=1,20,2
      XP(XJ)=XP(XJ)+XA;                   ¥         XP(J)=XP(J)+XA
      XQ(XJ)=XQ(XJ)/XC;                   ¥         XQ(J)=XQ(J)/XC
    END;                                  ¥    15 CONTINUE
    LOCATE (XA,1, XB,2);                  ¥       EQUIVALENCE (XA,XP(1)),(XB,XP(2))
L20: FORMAT (I(10),9 F(10,3));            ¥    20 FORMAT(I10,9F10.3)
    GO TO L10;                            ¥       GO TO 10
    GO BY XJ TO (L10,L20,L30,L40);        ¥       GO TO (10,20,30,40),J
    GO TO VALUE(XI);                      ¥       GO TO I,(10,20,30,40)
    IF XA EQ XB THEN XB=XC;               ¥       IF (XA .EQ. XB) XB=XC
    INTEGERS XI XJ XK;                    ¥       INTEGER XI,XJ,XK
    GET DATA XP XQ;                       ¥       READ(5,11) XP,XQ
    REALS XP XQ XR;                       ¥       REAL XP,XQ,XR
    RETURN;                               ¥       RETURN
    REWIND 1;                             ¥       REWIND 1
    STOP;                                 ¥       STOP
    PUT EDIT (XJ,XP(XJ),                  ¥       WRITE(6,21) J,XP(J),(XQ(J),J=1,N)
      OLIST(XQ,1,XN))                     ¥
      FORM (I(10),9 F(10,3));             ¥    21 FORMAT(I10,9F10.3)
   DEFINE(FUNC(XA1,XA2), DO;              ¥       FUNCTION FUNC(XA1,XA2)
     RETURN(SQRT(XA1**2+XA2**2));         ¥       FUNC=SQRT(XA1**2+XA2**2)
                                          ¥       RETURN
     END )                                ¥       END
   DEFINE(SUBR(XA1), DO;                  ¥       SUBROUTINE SUBR(XA1)
     NULL;                                ¥       CONTINUE
     RETURN;                              ¥       RETURN
     END )                                ¥       END
```

Fig. 2　FORTRAN-like expressions

```
    XA=XB+XC*(XD-XE);                     ¥       XA:=XB+XC*(XD-XE);
    (XA,XB,XC)=0;                         ¥       XA:=XB:=XC:=0;
    XA=(IF XI EQ 0 THEN 1 ELSE 3);        ¥       XA:=IF XI=0 THEN 1 ELSE 3;
    DIMENSION XP(10) XQ(0 10,-1 1);       ¥       ARRAY XP(10),XQ(0:10,-1:1);
    DO;                                   ¥       BEGIN
      CALL SUBROUT1;                      ¥         SUBROUT1;
      CALL SUBROUT2(XP,XQ);               ¥         SUBROUT2(XP,XQ);
    END;                                  ¥       END;
    BOOLEAN XBOOL;                        ¥       BOOLEAN XBOOL;
L1: NULL;                                 ¥ L1:   ;              ¥ DUMMY
    FOR XJ FROM 1 STEP 2 UNTIL 10         ¥       FOR XJ=1 STEP 2 UNTIL 10 DO
    REPEAT DO;                            ¥       BEGIN
      XP(XJ)=XP(XJ)+XA;                   ¥       XP(XJ):=XP(XJ)+XA;
      XQ(XJ,1)=XQ(XJ,1)/XC;               ¥       XQ(XJ,1):=XQ(XJ,1)/XC;
    END;                                  ¥       END;
    CHANGE XA BY                          ¥       FOR XA =0 STEP 0.1 UNTIL 1.0,
      (0 1.0 0.1,1.2 3.0 0.2) REPEAT      ¥         1.2 STEP 0.2 UNTIL 3.0
      XS=XS+F(XA);                        ¥         DO XS:=XS+F(XA);
    GO TO L1;                             ¥       GO TO L1;
    GO TO VALUE(XSW(XI));                 ¥       GO TO XSW(XI);
    GO TO VALUE(XSW(IF XY EQ 0            ¥       GO TO XSW(IF XY=0 THEN XI
      THEN XI ELSE XI+1));                ¥                 ELSE XI+1);
    IF XA LT XEPS OR XB GT 0 THEN         ¥       IF XA< XEPS | XB > 0 THEN
      XALPHA=XALPHA*10.0;                 ¥         XALPHA:=XALPHA*10.0;
    IF XI EQ 0 THEN DO;                   ¥       IF XI=0 THEN BEGIN
      XA=XA+1.0; XB=XB+1.0; END           ¥         XA:=XA+1.0; XB:=XB+1.0; END;
    ELSE DO; XA=XA-1.0; END;              ¥       ELSE BEGIN XA:=XA-1.0; END;
    INTEGERS XI XJ;                       ¥       INTEGER XI,XJ;
    DIMENSION OF INTEGER VARIABLE         ¥       INTEGER ARRAY XE(XN:20);
      XE(XN 20);
    OWN INTEGER SYMBOL XK XL;             ¥       OWN INTEGER XK,XL;
    DEFINE(MATADD(XX,XY,XN), DO(1);       ¥       PROCEDURE MATADD(XX,XY,XN);
      DIMENSION OF REAL VARIABLE          ¥         REAL ARRAY XX,XY;
        XX XY;
      INTEGERS XN;                        ¥         INTEGER XN;
      DO;                                 ¥         BEGIN
      END;                                ¥         END;
    END(1))                               ¥       END MATADD;
    PROCEDURE(TAN(X1), DO;                ¥       REAL PROCEDURE TAN(X1);
      REALS X1;                           ¥         REAL X1;
      VALUES X1;                          ¥         VALUE X1;
      RETURN(SIN(X1)/COS(X1));            ¥         TAN:=SIN(X1)/COS(X1);
    END )                                 ¥       END TAN;
    SWITCH XSW LIST (L1,L2,L3,L4);        ¥       SWITCH XSW = L1,L2,L3,L4;
```

Fig. 3　ALGOL-like expressions

よう．

コロンやセミコロンを区切り記号として使うのが煩わしい場合は，FORTRAN のように，ラベルとステートメント本体とをそれらを書き始めるカラム位置で区別するとか，改行でステートメントの区切りを示すなどの便法を講ずるとよい．この方法は，文法上の単純な誤りを防ぐのに役立つであろう．また，コンパイル時に特別扱いする DEFINE や PRIORITY などを他の式と区別する場合にも，書き出しカラムの位置を利用すると見分けやすい．

5.　構文解析

つぎに，図1の規則に合わせて書かれた式を，第2節のはじめで述べた標準型の式に変えることを考えてみよう．開いた関数

$$f_0\ f_1 \cdots f_m\ e_1\ k_2\ e_2 \cdots k_n\ e_n$$

は，$f_0\ f_1 \cdots f_m$ を f' とおけば，標準型の関数 $f'(e_1, \cdots, e_n)$ となる．開いた関数が参照されたとき，キー $k_2, \cdots, k_n$ の並び方をその関数の定義とくらべてみれば，引数 $e_1, \cdots, e_n$ の中に省略されたものがあるかどうかわかる．省略された引数がある場合は標準型に変えるとき，空席を表わす特殊な式 NULL を挿入して，対応する部分をおきかえればよい．二項演算子で結ばれた式ではそれらの演算子の間にある優先順位を定め，順位の高い方から順番に $a_1\ g\ a_2$ のパタンを $g(a_1, a_2)$ と変換する操作をくり返せば，これもまた標準型の式になる．開いたリストも，かっことコンマを補うと，ただちに標準型となる．

通常のコンパイラでは，構文解析部分が 30〜40％ を占めている．図1の文法に従って書かれた式を標準型の式に変換することは，とりもなおさず，構文解析をすることである．その処理プログラムは，エラー処理も含めて，FORTRAN でわずか 300 ステップの大きさである．ただし，この数字は，原始プログラムを変数名や関数名，

```
    ALLOCATE XP(XM) XQ(XM,XN);          ¥      ALLOCATE XP(XM),XQ(XM,XN);
    ALLOCATE BASEDVARIABLE              ¥      ALLOCATE XI IN (XR)  XJ IN (XR);
      (XI,XR) (XJ,XR);
    XA=XB+XC*(XD-XE);                   ¥      XA=XB+XC*(XD-XE);
    (XA,XB,XC)=0;                       ¥      XA,XB,XC=0;
    XONE.XPART1=XTWO.XPART1             ¥      XONE.XPART1=XTWO.XPART1
      -2*XTHREE.XPART1;                 ¥        -2*XTHREE.XPART1;
    DO(1);                              ¥ L1:  BEGIN;
    END(1);                             ¥      END L1;
    CALL CRITICALPATH(XA,XB*XC,XD);     ¥      CALL CRITICALPATH(XA,XB*XC,XD);
    CALL PRINT(XA,XB) TASK XT2;         ¥      CALL PRINT(XA,XB) TASK(XT2);
    CLOSE FILE XTBL1 XTBL2;             ¥      CLOSE FILE(XTBL1),FILE(XTBL2);
    DECLARE SYMBOL (XPAGE FIXED         ¥      DECLARE XPAGE FIXED DECIMAL
      DECIMAL INITIAL(0));              ¥        INITIAL(0);
    DECLARE SYMBOL XP(10)               ¥      DECLARE XP(10),
      XQ(0 20,3 5);                     ¥        XQ(0:20,3:5);
    DECLARE SYMBOL                      ¥      DECLARE
     (1 XSTR BASED(XP),                 ¥        1 XSTR BASED(XP),
      2 XA, 3 XB FIXED BINARY,          ¥        2 XA, 3 XB FIXED BINARY,
            3 XC CHARACTER(20),         ¥              3 XC CHARACTER(20),
      2 XD, 3 XB FIXED BINARY,          ¥        2 XD, 3 XB FIXED BINARY
            3 XC ((0,X1) REFER          ¥              3 XC (0:X1 REFER
              XD.XB,(0,9))      );      ¥                  (XD.XB), 0:9);
    DELAY(10);                          ¥      DELAY(10);
    DELETE FILE XALPHA KEY XKEY;        ¥      DELETE FILE(XALPHA) KEY(XKEY);
    DISPLAY('END OF JOB');              ¥      DISPLAY('END OF JOB');
    DO; XA=1.0; XB=1.0; END;            ¥      DO; XA=1.0; XB=1.0; END;
    FOR NULL WHILE XA GT 0 REPEAT DO;   ¥      DO WHILE(XA > 0);
      XB=XB+FUN(XA,XB); END;            ¥        XB=XB+FUN(XA,XB); END;
    FOR XI FROM 1 TO 10 BY 2 REPEAT DO(9); ¥L9: DO XI=1 TO 10 BY 2;
      XP(XI+1)=XP(XI+1)-1.0;            ¥        XP(XI+1)=XP(XI+1)-1.0;
    CHANGE XJ BY (1 10,12 20)           ¥      DO XJ=1 TO 10,12 TO 20
      WHILE XA GT 0  REPEAT DO;         ¥        WHILE(XA > 0);
      CALL SUB(XJ,XA);                  ¥        CALL SUB(XJ,XA);
    END(9);                             ¥      END L9;
    ENTRY FUN(XA,XB) RETURNS FLOAT;     ¥ FUN:  ENTRY(XA,XB) RETURNS(FLOAT);
    EXIT;                               ¥      EXIT;
L3: FORMAT (SKIP(3),A(6),4 F(7,2));     ¥ L3:  FORMAT(SKIP(3),A(6),4 F(7,2));
    FREE XP XQ;                         ¥      FREE XP,XQ;
    GET LIST XA XB XC;                  ¥      GET LIST (XA,XB,XC);
    GET LIST                            ¥      GET LIST
      DLIST(XQ,(1,2),(XN,2));           ¥        (XQ(XI,2) DO XI=1 TO XN);
    GO TO L1;                           ¥      GO TO L1;
    GO TO VALUE(XLABEL(XI));            ¥      GO TO XLABEL(XI);
    IF XA LT XEPS OR XB GT 0 THEN       ¥      IF XA < XEPS | XB > 0 THEN
      XALPHA=XALPHA*10.0;               ¥        XALPHA=XALPHA*10.0;
    IF XI EQ 0 THEN DO;                 ¥      IF XI=0 THEN DO;
      XA=XA+1.0; XB=XB+1.0; END         ¥        XA=XA+1.0; XB=XB+1.0; END;
    ELSE DO;                            ¥      ELSE DO;
      XA=XA+XC;  XB=XB-XD;  END;        ¥        XA=XA+XC;  XB=XB-XD;  END;
    LOCATE VARIABLE XALPHA              ¥      LOCATE XALPHA FILE(XBETA);
      FILE XBETA;
L5: NULL;                               ¥ L5:   ;
    ON ENDPAGE CONDITION CALL TITLE;    ¥      ON ENDPAGE CALL TITLE;
    OPEN FILE (XLISTING,PRINT);         ¥      OPEN FILE(XLISTING) PRINT;
    DEFINE(FN(XK) (RECURSIVE,           ¥ FN:  PROCEDURE(XK) RECURSIVE
      FIXED,BINARY), DO;                ¥        RETURNS(FIXED BINARY);
      IF XK LT 0 THEN RETURN(1)         ¥        IF XK < 0 THEN RETURN(1)
      ELSE RETURN(FN(XK-1)+FN(XK-2));   ¥        ELSE RETURN(FN(XK-1)+FN(XK-2));
    END )                               ¥      END;
    PUT FILE XLIST EDIT (XA,XB,XC)      ¥      PUT FILE(XLIST) EDIT(XA,XB,XC)
      FORM (A(10),2 F(7,2)) PAGE 1;     ¥        (A(10),2 F(7,2)) PAGE;
    READ FILE XINFILE INTO XWORK;       ¥      READ FILE(XINFILE) INTO(XWORK);
    RETURN;                             ¥      RETURN;
    REVERT OVERFLOW;                    ¥      REVERT OVERFLOW;
    REWRITE FILE XSTOCK FROM XSTREC     ¥      REWRITE FILE(XSTOCK) FROM(XSTREC)
      KEY XITEM;                        ¥        KEY(XITEM);
    SIGNAL ENDPAGE(XLIST);              ¥      SIGNAL ENDPAGE(XLIST);
    STOP;                               ¥      STOP;
    UNLOCK FILE XSTOCK KEY XITEM;       ¥      UNLOCK FILE(XSTOCK) KEY(XITEM);
    WAIT XREADEVENT 1;                  ¥      WAIT(XREADEVENT) (1);
    WRITE FILE XLOANS FROM XLOAN        ¥      WRITE FILE(XLOANS) FROM(XLOAN)
      KEYFROM XLOANNUMBER;              ¥        KEYFROM(XLOANNUMBER);
```

Fig. 4 PL/I like expressions

定数，特殊記号などの要素に分解する部分を含んでいない．この文法では，関数名やキーワードをどのように選んでも，すべて同一の解析ルーチンで標準型に変換できるため，問題向き言語を作るとき，その文法として採用すれば，構文解析にほとんど労力をかけないですむ．

コンパイラでは，構文解析に先立って，原始プログラム全体を表1にあげた 20 種の構成要素に分解する．そのさい，DEFINE 式の第1引数を別扱いにし，導入された関数の名前とフラクションやキーの並び方，および意味を表わす式との対応を記録する．

構文解析ルーチンでは，定数，ラベル，変数名の三つをオペランド要素と呼び，それ以外をオペレーション・ドライバと呼ぶ．オペレーション・ドライバは，中間段階でオペレータ・スタッカと名づける所に入れ，おのおのには，表1のように，スタック順位，比較順位の2種類の優先をつける．

記号の処理順序は，つぎのようにして決める．いま，オペレーション・ドライバ a をとりだしたとき，その比較順位がオペレータ・スタッカの最上段にある記号 b のスタック順位より高いなら，a を b の上に積む．逆に，a の比較順位が b のスタック順位より高くないなら，a をそのままにしてまず b を処理する．そして，b をスタッカから消し，b の下に積まれていた記号と a とを比較する．

図5は，この文法にしたがって構文解析のおもな部分を書いたプログラムである．FORTRAN や ALGOL, PL/I にくらべ，もっと自然言語に近い表現がとれるといえよう．図では，概略のみを示すために，細かい処理操作に対する DEFINE 式の第2項をすべて NULL とした．さらに，RETURN や OR など少数の式は，もう定義する必要のない基本的な式とした．実際のプログラミングでは，NULL の代りに各式の意味する処理操作や宣言内容を書けばよい．

図5では，変数を一つも使っていない．プログラム内で記号の意味づけが自由にできることを利用して，プログラムの主要部分をデータの表現形式に依存しない形に書き表わしてある．この行き方は，アルゴリズムとデータを分離することを考えるさいに参考になる

Table 1　Elements of Programs

element		stack priority	compare priority
constant	e		
label	l		
variable name	x		
DEFINE end	)		2
DEFINE (		3	99
block end	END		6
block start	DO;	7	99
statement end	;		10
statement start		11	99
right parenthesis	)		14
closed list start	(	17	99
variable name with parenthesis	x	19	99
function name with parenthesis	$f($	19	99
comma	,	15	22
open function name	f_o	23	24
fraction	f_i	27	99
key	k	25	26
binary operator	g	30 ～ 80	30 ～ 80
qualification period	.	90	90
closed function name	f	92	91

```
DEFINE(PARSER,  DO;
        DECLARATIONS;
        INITIATION;
   L10: GET THE NEXT SYLLABLE;
        IF (OPERATION DRIVER) THEN GO TO L20;
        STACK IN THE STANDARD EXPRESSION AREA;
        INCREASE THE OPERAND COUNTER;
        GO TO L10;
   L20: IF COMPAREPRIORITY LE STACKPRIORITY THEN GO TO L30;
        STACK IN THE OPERATOR STACKER;
        IF NOT(SYMBOLS FOR DELIMITATION ONLY) THEN
          STACK IN THE STANDARD EXPRESSION AREA;
        GO TO L10;
   L30: PROCESS THE STACKED OPERATOR;
        IF PROGRAMEND THEN RETURN ELSE GO TO L20;
        END
DEFINE(STACK IN THE OPERATOR STACKER,  DO;
        IF (COMMA OR KEY) AND (MULTIPLE OPERANDS)
          THEN ENCLOSE OPERANDS IN LIST;
        STACK THE OPERATOR;
        PREPARE OPERAND COUNTER;
        IF BINARYOPERATOR THEN DIG OUT THE PREVIOUS OPERAND;
        IF KEY THEN SUPPLY OMITTED ARGUMENTS;
        END
  )
DEFINE(PROCESS THE STACKED OPERATOR,  DO;
        IF (MULTIPLE OPERANDS) THEN ENCLOSE OPERANDS IN LIST;
        IF FRACTION THEN IDENTIFY THE PRECEDING FUNCTION NAME;
        CLOSE THE EXPRESSION;
        ERAZE THE OPERAND;
        INCREASE THE OPERAND COUNTER;
        END
  )
DEFINE(DECLARATIONS,                               NULL)
DEFINE(INITIATION,                                 NULL)
DEFINE(GET THE NEXT SYLLABLE,                      NULL)
DEFINE(IF X01 THEN X02 ELSE X03,                   NULL)
DEFINE(GO TO X04,                                  NULL)
DEFINE(OPERATION DRIVER,                           NULL)
DEFINE(STACK IN THE STANDARD EXPRESSION AREA,NULL)
DEFINE(INCREASE THE OPERAND COUNTER,               NULL)
DEFINE(COMPAREPRIORITY,                            NULL)
DEFINE(STACKPRIORITY,                              NULL)
DEFINE(SYMBOLS FOR DELIMITATION ONLY,              NULL)
DEFINE(PROGRAMEND,                                 NULL)
DEFINE(COMMA,                                      NULL)
DEFINE(KEY,                                        NULL)
DEFINE(MULTIPLE OPERANDS,                          NULL)
DEFINE(ENCLOSE OPERANDS IN LIST,                   NULL)
DEFINE(FRACTION,                                   NULL)
DEFINE(IDENTIFY THE PRECEDING FUNCTION NAME,  NULL)
DEFINE(CLOSE THE EXPRESSION,                       NULL)
DEFINE(ERAZE THE OPERANDS,                         NULL)
DEFINE(STACK THE OPERATOR,                         NULL)
DEFINE(PREPARE OPERAND COUNTER,                    NULL)
DEFINE(BINARYOPERATOR,                             NULL)
DEFINE(DIG OUT THE PREVIOUS OPERAND,               NULL)
DEFINE(SUPPLY OMITTED ARGUMENTS,                   NULL)
  )
```

←Fig. 5　Syntax Analyzer

であろう．

6. む す び

いままでにあげた例から，この文法の範囲内で，さまざまな分野における各種の問題向き言語が設定できるといえよう．具体的な表現形式を定めるには，サブプログラムを書くのと同じ要領で，その形と意味を指定すればよい．

この文法を採用した EXTRAN プロセッサを 1971 年に完成し，作図用の問題向き言語などを作って実用に供した．EXTRAN プロセッサには，導入された式にその定義内容を代入するマクロ展開機能が組み込んであり，演算の実行効率を落さないよう，代入すべき式を実行に先立って代入する形をとる．さらに，このプロセッサには，式の一部または全部をインタープリティブに実行する機能，FORTRAN サブルーチンを呼び出す機能，およびマクロ展開を施した式からそれと同等の FORTRAN プログラムを生成する機能を持つ．1972 年 6 月に，処理方式の改良と頻繁に使う式を登録するライブラリ機能の拡充を行なった．現在，これを用いて，大規模な設計計算システムを開発している．

7. 謝 辞

最後に，貴重な助言をいただいた査読者，および日立製作所の吉村一馬，中田育男，新井公雄の諸氏に深く謝意を表する．

参 考 文 献

1) 渡辺坦：問題向き言語に対する意味論的メタ言語とそのコンパイラ，第12回プログラミングシンポジウム報告集，pp. B 77～88 (1971)．
2) D. Ross: ICEES System Design, MIT (1967)．
3) IBM 社：Problem Langurge Analyzer (PLAN) Program Description Manual, FORM H 20-0594 (1969)．
4) 渡辺坦：意味論的メタ言語の形をした計算機言語，情報処理，VoL. 11, No. 8 (1970, pp. 439～448)．

(昭和47年3月9日　受　付)
(昭和47年12月1日　再受付)