知識工学と人工知能 31－2
（１９８３．９．２７）

属性付構文向き翻訳系の生成系
ＭＹＬＡＮＧ

山之上　卓　　安在　弘幸
（九州工業大学）

１．　まえがき

Knuth によって与えられた属性文法(attributed grammer)[1]は、言語処理系の生成システムの道具として最も有望なものの一つである。これは今までの構文解析の技術をそのまま使って言語の構文と意味の仕様を与えることができるからである。

本報告では、一種の構文向き翻訳スキームである正規翻訳記法（Regular Translation Form, ＲＴＦ：拡張ＢＮＦに活動記号を付加したもの）を拡張して、属性を加えた属性付正規翻訳記法(Attributed Regular Translation Form,属性付ＲＴＦ）を与える。属性付RTF は、属性付構文向き翻訳系(attributed syntax- directed translation) を定義する。

MYLANGは属性付RTF を入力して、属性付構文向き翻訳系を自動的に生成するものである。

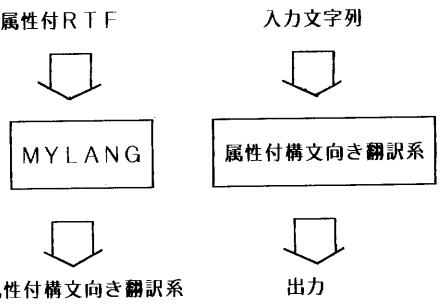


図１　ＭＹＬＡＮＧ

属性付RTF は言語の構文と意味を表すが、意味の情報を使ってBNF や拡張BNF などの記法では定義できなかった広い範囲の文法も定義できるようになる。

２章で属性付RTF について説明を行なう。３章で文脈依存言語の認識や変換の例について、その属性付RTF とそれをMYLANGに入力することによって生成された翻訳系の実行について述べる。４章ではMYLANGを使用して開発した言語処理系の例としてＬＩＳＰのインタープリタ etc.をとりあげる。

２．　属性付ＲＴＦ

Knuth は属性文法(attributed grammar)を与えた[1]。これは文脈自由文法において、プログラム言語の意味を記述するために、非終端記号に属性とよばれるパラメータを付加させ、属性の関係の記述を与えて、意味の形式化を計ったものである。属性には相続属性と合成属性がある。導出木において、相続属性は根から枝の方向へ属性の値が移動することを表し、合成属性は枝から根の方向へ属性の値が移動することを表す。属性の値はsemantic rules によって定められる。Lewis らは文脈自由文法に活動記号を加えてtranslation grammar を定義し、これに属性を加えてattributed translation grammar[2]とした。Wattは属性の値によって、すなわち文脈に依存して生成規則の適用を変えることができることを示した。これをrule splittingといい、それによる構文解析の方法を属性向き構文解析(attribute-directed parsing)と言う[3]。

属性付正規翻訳記法(attributed regular translation form:属性付RTF)は属性付構文向き翻訳系(attributed syntax-directed translator) を定義する。我々は拡張BNF に活動記号を導入して正規翻訳記法(regular translation form:RTF)を与えた。属性付RTF はRTF を以下のように増強したものである。

(1) RTF に属性を付加した。

ここで属性は通常のプログラム言語における変数に対応している。属性は非終端記号と活動記号に付加できる。属性には相続属性と合成属性の２種類があるが、これはサブルーチンの引数における値引数と変数引数にそれぞれ対応している。属性は次のように書いて表す。

非終端記号の場合.

＜Ｎ（i1, …,im/s1, …,sn ）＞

活動記号の場合.

［Ａ（i1, …,im/s1, …,sn ）］,(m,n ≧０)

ここで、ｉx (1≦x≦m) は相続属性を表し、ｓy (1≦y≦n) は合成属性を表す。非終端記号が相続属性しか持たない場合には <N(i1,i2/)>のように書き、合成属性しか持たない場合には <N(/s1,s2)>のように書く。

(2) 活動記号の増強を行なった。

a.活動記号の中で直接に代入文や出力文を記述できる。

例.　　［（Ｘ：＝Ｘ＋Ｙ）］

［（ＷＲＩＴＥＬＮ（Ｘ））］

このことによって簡潔に翻訳系の定義を行えるようになる。例えば今まで図-2.1(a) のように記述していたものは、図-2.1(b) のように書けばよい。

```
(*?RTF
                    :
   <S>= ......[PLUS]......;
                    :
?END*)
(*?ACTION*)

   PROCEDURE(*[PLUS]=*)PLUS;
      BEGIN
        X:=X+Y
      END;

(*?END*)
```

(a) ＲＴＦ

```
(*?RTF
                    :
   <S>= ......[(X:=X+Y)]......;
                    :
?END*)
```

(b) 属性付ＲＴＦ

図２．１　属性付ＲＴＦによる簡潔な記法

b.活動記号のなかに述語を導入した。このことによって属性付RTF の中でrule splittingを表すことができる。述語Ｐ(x1,x2, …,xn)は次のように書いて活動記号の中に入れる。

[？(Ｐ(x1,x2, …,xn))]

活動記号[α]の記号列としての意味を次のように定義する。

α＝´？(P(x1, …,xn))´の場合

$$[\alpha] = \begin{cases} \lambda & \text{if } P(x1,\cdots,xn) \\ \phi & \text{if not } P(x1,\cdots,xn) \end{cases}$$

α＝´？(P(x1, …,xn))´でない場合

[α]＝λ ．

つまり活動記号の中に述語を入れることによって、その真偽により入力文字列を制御できる訳である。例えば

[?(X=Y)]´ａ´＋ [?(X<>Y)]´ｂ´

は述語 X=Yが真のときはλ'a' + φ'b' ='a'となり、 X<>Y が真のときはφ'a' + λ'b' ='b'となる。

属性付RTF の例として、算術式の演算系を定義した属性付RTF と活動ルーチンを図-2.2(a) に示す。この例で活動ルーチンは数字の読み取りと、整数への変換を行っている。この属性付RTF をMYLANGに入力することによって生成された演算系の実行結果（属性付構文向き翻訳の結果）を図-2.2(b) で示す。ここで生成された演算系に対して、

？３＋４＊９．

を入力したときにできる導出木と属性の値の流れを図-2.3に示す。この例では述語は使っていないが、これは次の章で例をあげて示す。

```
(*?CALICU*)
(*?RTF

   <P>=('?'<E(/X)>'.'[(WRITELN(X))])*;

   <E(/X)>=<T(/X)>('+'<T(/X1)>[(X:=X+X1)]
                 +  '-'<T(/X1)>[(X:=X-X1)] )* ;

   <T(/X)>=<F(/X)>('*'<F(/X1)>[(X:=X*X1)]
                 +  '/'<F(/X1)>[(X:=X/X1)] )* ;

   <F(/X)>=<INTEGER(/X)>+'('<E(/X)>')' ;

   <INTEGER(/X)>=
       [CL][NUM][CON]([NUM][CON])*[NVAR] ;

?END*)

(*?ACTION*)
PROCEDURE EXECUTE; VAR NAME:SYMBOL;
PROCEDURE(*[CL]=*)CL; BEGIN NAME:='' END;
PROCEDURE(*[NUM]=*)NUM; BEGIN TNUMBER END;
PROCEDURE(*[CON]=*)CON;
 BEGIN NAME:=CONCAT(NAME,CH) END;
PROCEDURE(*[NVAR]=*)ANVAR;
 BEGIN STACK1[STP1]:=NVAR(NAME) END;
(*?END*)
```

(a) 属性付RTF と活動記号による算術式演算系の定義

```
?1+2+3+4+5+6+7+8+9+10.
55

?3+4*(9-3)/3+5.
16

NORMAL END
```

(b) (a) をMYLANGに入力することによって生成された演算系の実行結果

図２．２　算術式演算系

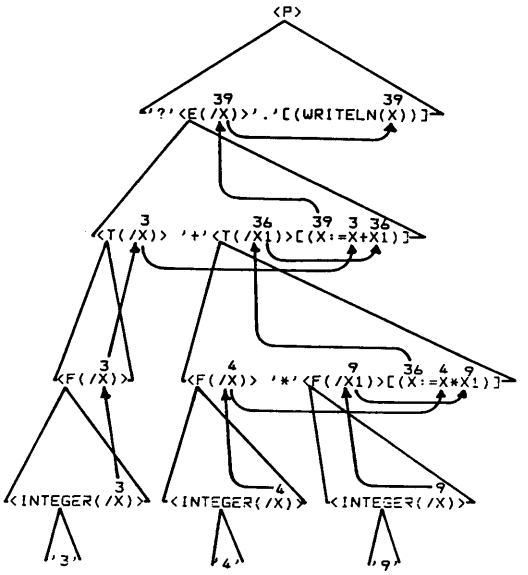


図２．３　算術式の演算系に?3+4*9を入力したときの導出木と属性の流れ

3．　文脈依存言語の認識や変換を定義する属性付ＲＴＦ．

この章では属性付RTF の例として、文脈依存言語 $A^nB^nC^n(n\geq 0)$ を定義した属性付RTF 、算術式と論理式の逆ポーランド記法への変換を定義した属性付RTF 及び簡単な英語の文脈に依存した部分を属性付RTF で定義した例をあげる。なお、これらの属性付RTF はMYLANGに入力することによって、属性付構文向き翻訳を行う系に変換される。この系は実際に実行することができる。

```
(*?SENS*)
(*?RTF

   <S>=[(I:=0)]('A'[(I:=I+1)])*
       [(J:=0)]('B'[(J:=J+1)])*
       [(K:=0)]('C'[(K:=K+1)])*
       <T(I,J,K/)> ;

   <T(I,J,K/)>=[?(I=J)][?(I=K)] ;

   <DMM>=[DMM] ;

  ?END*)

  (*?ACTION*)
  PROCEDURE EXECUTE;
  PROCEDURE(*[DMM]=*)DMM; BEGIN WRITELN END;
  (*?END*)

AABBCC

NORMAL END

AAAABBBBCCCC

NORMAL END

AAABBBC

ERROR IN TEXT
```

図３．１　文脈依存言語$A^nB^nC^n$を定義する属性付ＲＴＦ

3.1 文脈依存言語$A^nB^nC^n$を定義する属性付ＲＴＦ

BNF 記法や拡張BNF 記法では文脈自由の範囲までを定義することができる。これにたいして、属性付RTF では文脈依存の範囲にある言語も定義することができる。例として$A^nB^nC^n$, $n\geq 0$ を定義する属性付RTF を図-3.1に示す。これは文脈依存の言語として有名である。

図-3.1において属性Ｉは＇Ａ＇の回数、Ｊは＇Ｂ＇の回数、Ｋは＇Ｃ＇の回数を表す。すべての回数を数え終った後、これらの属性を相続属性として<T(I,J,K/)> に引き渡す。<T(I,J,K/)> では属性I,J,K が等しいかどうかを判定している。I=J=K のときだけ[?(I=J)][?(I=K)]が1になり、それ以外は0になるから$A^IB^JC^K=A^nB^nC^n$, $n\geq 0$ が定義されていることになる。<DMM> と活動ルーチンはダミーである。

3.2 算術式と論理式の逆ポーランド記法への変換を定義する属性付ＲＴＦ

この節では属性付RTF によるrule splittingを紹介する。

算術式と論理式を逆ポーランド記法に変換する属性付RTF を図-3.2に示す。これは次のような構文と意味を持つ。

○　型には整数型と論理型がある。整数型の変数と論理型の変数は論理型の変数に'?' をつけることによって区別する。Ｘ，Ａ，Ｂなどは整数型の変数であり、？Ｘ，？Ａ，？Ｂなどは論理型の変数である。論理型の定数は真値を！Ｔ、偽値を！Ｆで表す。

○　論理式において＇＋＇はＯＲを表し、　＇＊＇はＡＮＤを表すことにする。　　単項演算子＇－＇はＮＯＴを表す。演算順位は単項演算子＇－＇が最も強く、次に＇＊＇、＇＋＇が最も弱いとする。

この例で、属性Ｔ，Ｔ１，Ｔ２は型を表している。属性が０のときは整数型を表し、１のときは論理型を表す。

属性が表す型に依存して、例えば＇＋＇が算術和を表すか、または論理和を表すかが、rule splittingによって識別される。rule splittingは、<F(/T)>,<PLUS(T,T1/)>,<MINUS(T,T1/)>,<MULTI(T,T1/)>,<DIVIDE(T,T1/)> で行われている。

```
<F(/T)> = <F0(/T)> + '-'<F0(/T)>
                ([?(T=0)][CHS]+[?(T=1)][NOT]) ;
```

において、単項演算子'-' の後に因子<F0(/T)>の型は合成属性Ｔで表される。Ｔ＝０、つまり因子が論理型であれば[CHS] で'$CHS'を出力する。Ｔ＝１、つまり因子が論理型であれば[NOT] で'.NOT.' を出力する。ここでは合成属性によりrule splittingを行っている。これをsynthesized rule splittingという。

```
<PLUS(T,T1/)>=  [?(T=0)][?(T1=0)][PLUS]
              +  [?(T=1)][?(T1=1)][OR] ;
```

は属性T,T1が両方とも整数型のときに'+' を出力し、属性T,T1が両方とも論理型のときは'.OR.'を出力することを表している。属性T,T1の値、つまり型が異なっていればエラーとなる。ここでは相続属性によってrule splitting を行っている。これをinherited rule splittingと言う。

<MINUS(T,T1/)>や<DIVIDE(T,T1/)> では論理型に対する規則がない。従って図-3.2(b) のように?A/?B という入力があればエラーとなる。

以上のように属性付RTF を用いることによってattribute-directed parsing を行うことができる。

```
(*?RTF

    <A>=          <V(/T1)>'='<E(/T2)>[?(T1=T2)][ASSIGN] ;

    <E(/T)>=      <T(/T)>('+'<T(/T1)><PLUS(T,T1/)> +
                          '-'<T(/T1)><MINUS(T,T1/)> )* ;

    <T(/T)>=      <F(/T)>('*'<F(/T1)><MULTI(T,T1/)> +
                          '/'<F(/T1)><DIVIDE(T,T1/)> )* ;

    <F(/T)>=      <FO(/T)> + '-'<FO(/T)>( [?(T=0)][CHS] + [?(T=1)][NOT] ) ;

    <FO(/T)>=     <V(/T)> + <C(/T)> + '('<E(/T)>')' ;

    <V(/T)>=      <NAME>[(T:=0)][INAME] + '?'<NAME>[(T:=1)][BNAME] ;

    <NAME>=       [CLN][ALP][CON]( ([ALP]+[NUM])[CON] )* ;

    <C(/T)>=      <INTEGER>[(T:=0)][INT] + '!'('T'[T] + 'F'[F])[(T:=1)] ;

    <INTEGER>=    [CLN][NUM][CON]( [NUM][CON] )* ;

    <PLUS(T,T1/)>  =[?(T=0)][?(T1=0)][PLUS] +
                    [?(T=1)][?(T1=1)][OR]    ;

    <MINUS(T,T1/)> =[?(T=0)][?(T1=0)][MINUS] ;

    <MULTI(T,T1/)> =[?(T=0)][?(T1=0)][MULTI] +
                    [?(T=1)][?(T1=1)][AND]    ;

    <DIVIDE(T,T1/)>=[?(T=0)][?(T1=0)][DIVIDE] ;

?END*)

(*?ACTION*)
 PROCEDURE EXECUTE;
 VAR NAME:SYMBOL;
 PROCEDURE(*[ASSIGN]=*)ASSIGN; BEGIN WRITELN(' :=')    END;
 PROCEDURE(*[NOT]=*)PNOT;      BEGIN WRITE(' .NOT.')   END;
 PROCEDURE(*[CHS]=*)PCHS;      BEGIN WRITE(' $CHS')    END;
 PROCEDURE(*[T]=*)T;           BEGIN WRITE(' !T')      END;
 PROCEDURE(*[F]=*)F;           BEGIN WRITE(' !F')      END;
 PROCEDURE(*[PLUS]=*)PLUS;     BEGIN WRITE(' +')       END;
 PROCEDURE(*[OR]=*)POR;        BEGIN WRITE(' .OR.')    END;
 PROCEDURE(*[MINUS]=*)MINUS;   BEGIN WRITE(' -')       END;
 PROCEDURE(*[MULTI]=*)MULTI;   BEGIN WRITE(' *')       END;
 PROCEDURE(*[AND]=*)PAND;      BEGIN WRITE(' .AND.')   END;
 PROCEDURE(*[DIVIDE]=*)DIVIDE; BEGIN WRITE(' /')       END;
 PROCEDURE(*[CLN]=*)CLN;       BEGIN NAME:=''          END;
 PROCEDURE(*[ALP]=*)ALP;       BEGIN TALPHA            END;
 PROCEDURE(*[NUM]=*)NUM;       BEGIN TNUMBER           END;
 PROCEDURE(*[CON]=*)CON;       BEGIN NAME:=CONCAT(NAME,CH) END;
 PROCEDURE(*[INT]=*)INT;       BEGIN WRITE(' ',NVAR(NAME)) END;
 PROCEDURE(*[INAME]=*)INAME;   BEGIN WRITE(' ',NAME)   END;
 PROCEDURE(*[BNAME]=*)BNAME;   BEGIN WRITE(' ?',NAME) END;
(*?END*)
```

(a) 算術式と論理式の逆ポーランド記法への変換を定義した属性付RTF と活動ルーチン

```
X=A+B*(C-D)
X A B C D - * + :=

NORMAL END


?X=?A+?B*(?C+-?D)
?X ?A ?B ?C ?D .NOT. .OR. .AND. .OR. :=

NORMAL END
```

(b) (a) をMYLANGに入力することによって生成された翻訳系の実行結果

図3．2　算術式と論理式の逆ポーランド記法への変換系

3.3 簡単な英語の構文解析

英語の文法はＢＮＦで記述されることがあるが、ＢＮＦでは文脈に依存した部分は扱うことができない。属性付ＲＴＦでは属性の持つ情報を利用して文脈に依存した部分の定義を与えることができる。例として英語の文法を簡略化したものを考えた。これを定義する属性付RTF を図-3.3に示す。

図-3.3において属性Ｘは名詞の人称を表す。

```
<N(/X)>=
    'I'[(X:=1)] + 'YOU'[(X:=2)] + 'SHE'[(X:=3)] ;
```

の部分で名詞の人称が与えられる。最初に決定した名詞の人称は主語の人称として<NP(/X)>,<VP(X/)> を経由して<V(X/)> に引渡される。<V(X/)> で動詞の人称との一致を判定している。

```
<V(X/)>='LOVES'[?(X=3)] + ELSE'LOVE'[?(X<=2)] ;
```

は'LOVES' の主語は三人称で、'LOVE'の主語は一人称または二人称であることを表している。人称が一致しないときは述語が偽となってエラーになる。

```
(*?LOVE*)
(*?RTF
  <S>       = <NP(/X)>' '<VP(X/)>'.' ;

  <NP(/X)> = <N(/X)> ;

  <VP(X/)> = <V(X/)>' '<NP(/X)> ;

  <N(/X)>  =    'I'  [(X:=1)]
              + 'YOU'[(X:=2)]
              + 'SHE'[(X:=3)] ;

  <V(X/)>  =        'LOVES'[?(X=3)]
              + ELSE'LOVE' [?(X<=2)] ;


  <DMM>     = [DMM] ;
 ?END*)
(*?ACTION*)
PROCEDURE EXECUTE;
PROCEDURE(*[DMM]=*)DMM; BEGIN END;
(*?END*)
```

```
I LOVE YOU.
NORMAL END

I LOVES YOU.
ERROR IN TEXT

SHE LOVE YOU.
ERROR IN TEXT

SHE LOVES YOU.
NORMAL END
```

図３．３　属性付RTF による簡単な英語の文法の定義

４．　MYLANGによる言語処理系の開発

この章ではMYLANGによる言語処理系の開発の例をあげる。開発された処理系は属性付構文向き翻訳を行なう。4.1 節では属性付RTF にソフトウエアモデルとしての解釈を与え、MYLANGそれ自体が言語処理系であることを示す。4.2 節ではmini-BASICコンパイラの開発について、IF文を例にとって説明する。4.3 節ではLISPのインタープリタの開発例をあげる。

4.1 ソフトウエアモデルとしての属性付RTF

属性付RTF は一種のソフトウエアモデルとして解釈できる。これはプログラム図式(program schema:Mnna)[4]を表している。例として算術式の計算を定義する属性付RTF を示す。図-4.1は５の階乗と、１から１０までの和を求める属性付RTF であり、図-4.2はこれに対応するプログラム図式である。ここで<FAC(X/Y)>はX の階乗をY で返すことを表し、<SAM(X/Y)>は１からX までの和をY で返すことを表している。

```
(*?FAC*)
(*?RTF
 <S>= <FAC(5/X)>[(WRITELN(X))]
      <SAM(10/X)>[(WRITELN(X))] ;

 <FAC(X/Y)>=   [?(X=0)][(Y:=1)]
             + ELSE[(Z:=X-1)]<FAC(Z/Z)>[(Y:=X*Z)] ;

 <SAM(X/Y)>= [(Y:=0)][(I:=1)]
             ([?(I<=X)][(Y:=Y+I)][(I:=I+1)])* ;

 <DMM>=[DMM] ;

 ?END*)

(*?ACTION*)
PROCEDURE EXECUTE;
PROCEDURE(*[DMM]=*)DMM; BEGIN WRITELN END;
(*?END*)
```

```
 120
  55
NORMAL END
```

図４．１　プログラム図式としての属性付RTF

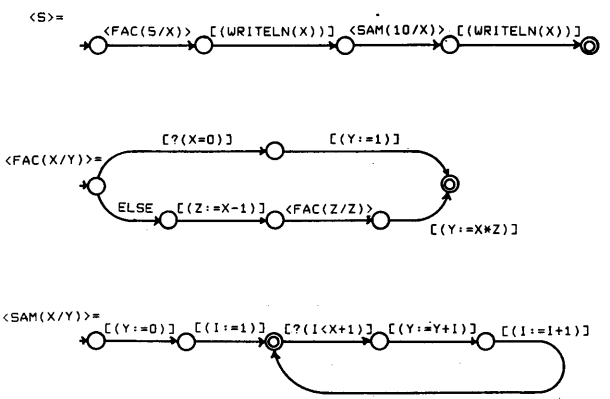


図４．２　プログラム図式（◎は復帰または停止を表す）

```
5 GO TO 110 ;
10 REM  SUBPROGRAM FOR FINDING GCM ;
20 IF A>=B THEN GO TO 60 ;
30 LET W=1;
40 LET A=B ;
50 LET B=W ;
60 IF B=0 THEN RETURN ;
70 LET X=A-A/B*B ;
80 LET A=B ;
90 LET B=X ;
100 GO TO 60 ;
110 REM
120 REM  TEST OF MIMIBASIC COMPILER ;
130 LET A=128 ;
140 LET B=48 ;
150 PRINT A,B ;
160 IF A>0 THEN IF B>0 THEN GOSUB10 ;
170 PRINT A ;
END RUN.

COMPLETED  RUNNING...
        128
         48
         16
NORMAL END


       RESET
>  COMPLETED

>  MYLANG SYSTEM END.
```

図４．３　mini-BASICのプログラム例と実行例

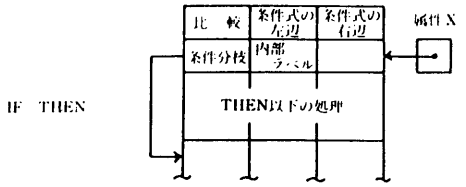
```
<IFS> ='IF'' '*<BOOL(/X)>' '*'THEN'' '*<STDD>|THENP(X/)|
                    ('ELSE'' '*|ELSE1(X/Y)|<STDD>|ELSE2(Y/)|)/ ;


(*|THENP(X/)|*)
 BEGIN LBL(.X.).ATAI:=MTXPTR END;

(*|ELSE1(X/Y)|*)
 BEGIN LBL(.LBLPTR.).NAME:='SYSLBL     ';
       LBL(.LBLPTR.).ATAI:=0 ; Y:=LBLPTR ;
       LBL(.X.).ATAI:=LBL(.X.).ATAI+1 ;
       MTXEND(27,Y,0) ;
       LBLPTR:=LBLPTR+1
 END;

(*|ELSE2(X/)|*)
 BEGIN LBL(.X.).ATAI:=MTXPTR END;
```

図４．４　IF文から3-組への変換を定義した
属性付RTF と活動ルーチン



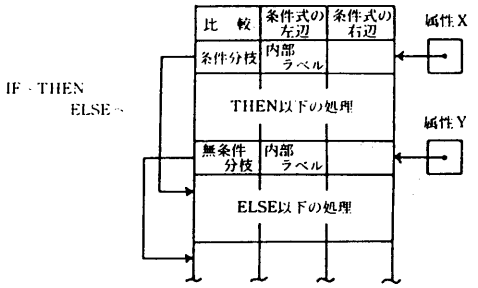


図4.5 IF文から変換された3－組

4.2 MYLANGによるmini-BASICコンパイラの開発

属性付RTF と活動ルーチンでmini-BASICコンパイラを定義し、これをMYLANGに入力することによって、mini-BASIC コンパイラを生成させることができた。mini-BASIC コンパイラは属性付構文向き翻訳を行なう。この節ではIF文を例にとって属性付構文向き翻訳を説明する。

mini-BASICのプログラム例と実行例を図-4.3に示す。これは128 と48の最大公約数を求めるプログラムである。

mini-BASICコンパイラは、mini-BASICのプログラムを3-組(Triple)に変換する。3-組はインタープリタによって実行される。IF文を3-組に変換する属性付RTF と活動ルーチンを図-4.4に示す。IF文に対応する3-組は図-4.5のようになる。

この例では、属性は前方参照を行なうために用いている。 3-組における比較命令と条件分岐命令は、属性付RTF の<BOOL(/X)>の部分で書き込まれる。しかし条件分岐命令の分岐先は、<BOOL(/X)>の部分では決定できない。THEN以下の部分がどのくらいの長さになるかわかっていないからである。そこでこの分岐命令のある場所を属性X に記憶させておく。分岐先はTHEN以下の部分の処理、つまり<STDD>の部分の処理が終わったときに決定する。ここで、活動ルーチン|THENP(X/)| でこの番地を属性Xの指す場所の命令の番地部、つまり分岐命令の番地部に代入する。

ELSEが<STDD>の後に続いた場合、|ELSE1(X/Y)|においてTHENP(X/)| で決定した分岐命令の分岐先を次の番地にずらす。前の番地に手続きMTXEND(27,Y,0)によって無条件分岐命令を書き込む。この無条件分岐命令も、分岐先がまだ決定できないので属性Y にこの番地を記憶させ、ELSE以下の処理が<STDD>で終わった後、|ELSE2(Y/)| で分岐先を書込む。

このようにして属性を使うことによって前方参照を行うことができる。

4.3 MYLANGによるLISPインタープリタの開発

属性付RTF と活動ルーチンでLISPインタープリタを定義し、これをMYLANGに入力することによって、LISPインタープリタを生成させることができた。図-4.6はLISPインタープリタを定義している属性付RTF の一部であり、図-4.7は属性付RTF ＋活動ルーチンをMYLANGに入力することによって生成されたLISPインタープリタの実行結果である。

このインタープリタはデータ構造として、リスト領域とアトムテーブルを持っている。活動ルーチンではこの二つのデータ構造を扱うCONS,CAR,CDRなどの基本的な操作を定義している。これらの基本的な操作を組合せ、積重ることによって、LISPインタープリタに必要なR-E-P,EVAL,APPLYの三つの手続き（関数）を定義することができる。これが図-4.1のように属性付RTF で行なわれている訳である。ここで属性A,X,B,Y,C,Z は、 AとX, BとY, CとZ がそれぞれペアになっており、一つのペアがLISPにおける変数を表している。A,B,C はNIL,T,アトム，リストの区別を表している。X,Y,Z はアトムテーブルまたはリスト領域を指すポインタである。属性のペアが0,0 であれば、これはNIL を表す。

<CONS(A,X,B,Y/C,Z)> はLISPの基本的な関数cons[x;y] と同じ働きを持つ。属性付RTF には”関数”はないから合成属性C,Z として結果の値を得るようにしている。

<CAR(A,X/B,Y)>,<CDR(A,X/B,Y)> はそれぞれLISPのcar[x], cdr[x]と同じ働きを持つ。関数の値のかわりとして結果は合成属性B,Y として得られる。

<LIST(A,X,B,Y/C,Z)> はLISPの関数list[x;y] と同じ働きを持つ。結果は合成属性C,Z として得られる。

LISPでは

list[x;y]=cons[x;cons[y;nil]]

であるが、これはy とnil のconsを求め、次にx と先に求めた値のconsを求めたものである。従ってこれを属性付RTF で表すと

<LIST(A,X,B,Y/C,Z)>=
<CONS(B,Y,0,0/B,Y)><CONS(A,X,B,Y/C,Z)> ;

となる。これと同様にしてcadr,cdar,caar,cddr などを簡単に属性付RTF に書き換えることができる。

LISPの五つの基本関数の中のcons,car,cdrは活動ルーチンによって定義しているが、eq,atom については述語で記述できる。LISPの変数をx,y 、これに対応する属性のペアをそれぞれA,X B,Y としたとき、eq[x;y] は[?(A=B)][?(X=Y)]であり、atom[x] は[?(A<>4)] である（変数がリストであるとき、属性A,B は４になる）。

マッカーシーの条件式

[p1→e1;p2→e2; …;pn→en]（piは命題、eiは式）

は属性付RTF では

p1e1+p2e2+…+pnen

となる。ここでpiは述語または非終端記号であり、eiは活動記号または非終端記号である。eiはなくてもかまわない。

<READ(A,X/)>はＳ式の読込みとリスト構造への変換を定義している。<NAME(/A,X)>はアトムを表している。アトムの値は属性A,X として得られる。<SEXPRESS(/A,X)>はＳ式を表している。これは再帰的に<READ(/A,X)>を呼出している。<RQUOTE(/A,X)>は''' で始まるＳ式を表している。''' はquote を表しており、ここではＳ式 x を読込んで(QUOTE x) を作りその値をA,X として返している。定数3,3 がアトム'QUOTE' を表している。

```
<LIST(A,X,B,Y/C,Z)>   =  <CONS(B,Y,0,0/B,Y)><CONS(A,X,B,Y/C,Z)> ;
<CADR(A,X/B,Y)>       =  <CDR(A,X/B,Y)><CAR(B,Y/B,Y)> ;
<CDAR(A,X/B,Y)>       =  <CAR(A,X/B,Y)><CDR(B,Y/B,Y)> ;
<CAAR(A,X/B,Y)>       =  <CAR(A,X/B,Y)><CAR(B,Y/B,Y)> ;
<CDDR(A,X/B,Y)>       =  <CDR(A,X/B,Y)><CDR(B,Y/B,Y)> ;

<READ(/A,X)>          =  <NAME(/A,X)> + <SEXPRESS(/A,X)> + <RQUOTE(/A,X)> ;
  <SEXPRESS(/A,X)>    =
        '('<RACON(/A,X)> (<RACON(/B,Y)><NCONC(A,X,B,Y/)>)* ')' ;
  <RACON(/A,X)>       =  <READ(/A,X)><CONS(A,X,0,0/A,X)>' '* ;
  <RQUOTE(/A,X)>      =  '''<RACON(/A,X)><CONS(3,3,A,X/A,X)> ;
```

図４．６　LISPインタープリタの属性付RTF による定義の一部

```
(CAR '(A B))
A

(CDR '(A B))
(B)

(CONS 'A '(B))
(A B)

(ATOM 'A)
T

(ATOM '(A))
NIL

(DEFUN APPEND (L1 L2)
  (COND ((NULL L1) L2)
        (T (CONS (CAR L1)
                 (APPEND (CDR L1)
                         L2
                 )
           )
        )
   )
)
APPEND

(APPEND '(A B) '(C D))
(A B C D)

(DEFUN MEMBER (X Y)
   (COND ((NULL Y) NIL)
         ((EQUAL X (CAR Y)) T)
         (T (MEMBER X (CDR Y)))
   )
)
MEMBER

(MEMBER 'A3 '(Z ABC (B) A3 (QY)))
T

(MEMBER 'X2 '(A (X2 B) C (B)))
NIL

(DEFUN UNION (X Y)
   (COND ((NULL X) Y)
         ((MEMBER (CAR X) Y)
           (UNION (CDR X) Y))
         (T (CONS (CAR X)
                  (UNION (CDR X) Y)))
   )
)
UNION

(UNION '(A B C D) '(B X C Y))
(A D B X C Y)
```

図4．7　LISPインタープリタの実行の結果

5．　あとがき

本報告では属性付RTF と活動ルーチンによって、文脈に依存した文法の定義、簡単な自然言語の構文解析の定義、ソフトウエアモデルの表現、手続き型言語の処理系の定義及び関数型言語の処理系の定義ができることを示した。

ここで用いている属性付翻訳文法はL-attributedである。本システムは九州大学大型計算機センターのプログラムライブラリ開発課題である。また、九州工業大学情報処理教育センター及び UCSD PASCALシステムで利用可能である。現在MYLANGを用いてPROLOGの処理系を開発中である。またMYLANG自体の拡張として、属性の型の拡張、コード生成系の生成系、エラー回復、LRパーサー、デバッキングツールetc.について研究を行なっている。

最後にMYLANGの開発において多大な援助をいただいた卒論生の諸君や関係者各位に深謝の意を表します。

参考文献

[1] Knuth,D.E.:"Semantics of Context- free Languages",Mathematical Systems Theory 2,pp127-145 (1968)

[2] Lewis,P.M.et al:"Attributed Translations", J.Computer and System Science 9, pp279-307(1974)

[3] Watt,D.A.:"Rule splitting and Attribute Directed Parsing",In:Semantic-Directed Compiler Generation, Lecture Notes in Compiler Science,94, Spring-Verlag,pp363-392(1980)

[4] Manna,Z.:"Program Schema",In:Aho,A.V.(ed): Currents In The Theory of Computing,Prentice-Hall (1973)

[5] 安在、藤岡、山之上：『オートマトン自動生成システムとそれを用いた言語処理系の開発』、情学全大'82後

[6] 山之上、安在：『属性付正規翻訳記法と属性付構文向き翻訳』、九工大研究報告、No47(1983) (予定)

[7] 山之上他：『記号処理系のオートマタシステムによる実現』、九工大研究報告、No.44 pp.83-90(1982)

[8] 小林孝次郎：『情報構造』、サイエンス社(1977)

[9] 中西正和：『LISP入門』、近代科学社(1977)

[10]Winston,P.H.:"LISP",Addison-Wesley Publishing Company,Inc.(1981)

[11]中島秀之：『Prolog』、産業図書(1983)

[12]安在、潮崎：『再帰降下順序変換機系とその生成機械』、電子通信学会論文誌、J63-D,9,pp.771-778(1980)

[13]安在他：『半線形代数』、第１報～第８報、電子通信学会技術報告、AL80-60,61,62(1980),AL81-4,5,38,39,74(1981)