属性付構文指示翻訳系の生成系 MYLANG†

山之上 卓†† 安在弘幸††

言語処理系の生成系 MYLANG とそれによって生成される言語処理系について報告する．この生成系 MYLANG はかなり広い範囲のクラスの言語，たとえば文脈依存言語，の処理系を生成することができる．MYLANG は属性付正規翻訳記法（属性付 RTF）を入力して属性付構文指示翻訳系を生成する．属性付 RTF は属性付構文指示翻訳スキームである正規翻訳記法（RTF：拡張 BNF に活動記号を付加したもの）に次の拡張を行ったものである．（1）非終端記号と活動記号に属性が付加できる．（2）活動記号の代りに，その場所に直接に述語，代入文および入出力文などを記述できる．その使用例として，文脈依存言語の認識や変換などを定義する属性付 RTF の記述，およびこれから MYLANG によって得られた系の実行結果を示す．また言語処理系の開発の例として mini-BASIC のコンパイラや LISP のインタプリタなどをとりあげる．

1. まえがき

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に，Knuth によって与えられた属性文法（attribute grammar）[1] や，Koster によって与えられたアフィックス文法（affix grammar）[2] がある．これはいま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．属性文法を用いたコンパイラ生成系には Lewi らの LILA[3] などがあり，アフィックス文法を用いたコンパイラ生成系には Koster の CDL[4] がある．Watt はこの二つの文法によって rule splitting[7] が最もわかりやすく表せることを示した．

本稿では，一種の構文指示翻訳スキームとしてわれわれがすでに与えた正規翻訳記法（Regular Translation Form, RTF：拡張 BNF に活動記号を付加したもの）[5),10),12),17] に対し，その記号に属性を付加できるように拡張した記法を与え，属性付正規翻訳法（Attributed Regular Translation Form, 属性付 RTF）と呼ぶ．属性付 RTF が定義する翻訳系を，属性付構文指示翻訳系（attributed syntax-directed translator）[11] という．

われわれは属性付 RTF を入力して，属性付構文指示翻訳系を自動的に生成する翻訳系の生成系 MYLANG を開発した（**図1**）．この生成系はかなり広い範囲のクラスの言語，たとえば文脈依存言語，の処理系を生成することができる．その原理的説明は別稿[8] に譲り，本稿ではこの生成系が広範囲に応用できることを種々の例を用いて示す．

2章では属性付 RTF について説明する．3章では文脈依存言語の認識や変換の例について，その属性付 RTF による定義と，MYLANG によって生成された翻訳系の実行について示す．4章では，MYLANG を使用して開発した言語処理系の例として，mini-BASIC コンパイラや LISP インタプリタなどを述べる．

2. 属性付 RTF

Knuth は属性文法（attributed grammar）[1] を与えた．これは文脈自由文法において，非終端記号に属性とよばれるパラメータを付加させ，さらにそれら属性の間の関係の記述を与えて，プログラム言語の意味の形式化を図ったものである．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 rule 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[6] 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指示構文解析（attribute-directed parsing）[7] といった．われわれは拡張 BNF に活動記号を導入した記法を与え正規翻訳記法（regular translation form：RTF）[5] と名づけた．本稿ではこの RTF をさらに以下のように増強し属性付 RTF と呼ぶ．

† Attributed Syntax-Directed Translator Generator MYLANG by TAKASHI YAMANOUE and HIROYUKI ANZAI (Kyushu Institute of Technology, Department of Computer Science).

†† 九州工業大学情報工学科

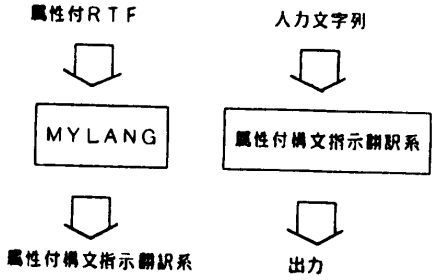

図 1 MYLANG
Fig. 1 Use of MYLANG.

(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型名に対応している．プログラム言語における変数に対応するものが属性の生起である．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．属性は次のように書いて表す．

非終端記号の場合

$<N(i_1, \cdots, i_m/s_1, \cdots, s_n)>$

活動記号の場合

$[A(i_1, \cdots, i_m/s_1, \cdots, s_n)]$, $(m, n \geqq 0)$

ここで，Nは非終端記号名，Aは活動記号名を表す．i_x $(1 \leqslant x \leqslant m)$ は相続属性の生起を表し，S_y $(1 \leqslant y \leqslant n)$ は合成属性の生起を表す．相続属性の生起と合成属性の生起は，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．非終端記号が相続属性のみもつ場合には $<N(i_1, i_2/)>$ のように書き，合成属性のみもつ場合には $<N(/s_1, s_2)>$ のように書く．属性はその書かれている順番によって識別する．各記号における各属性の位置に，対応した属性生起の名前を付加する．以後，混乱のないかぎり，属性と書けば，属性生起のことを表すことにする．

(2) 活動記号の増強を行った

a．活動記号として直接に代入文や出力文を記述できる．

例 [(X: X+Y)]

[(WRITELN(X))]

これによって翻訳系の定義をより簡潔に行えるようになる．たとえば RTF を活動ルーチンによって図2(a)のように記述していたものは，属性付 RTF によって図2(b)のように書くことができる．

b．活動記号として述語を導入した．

これによって属性付 RTF のなかで rule splitting を表すことができる．述語 $P(x_1, x_2, \cdots, x_n)$ は次のように書き活動記号として表す．

[?($P(x_1, x_2, \cdots, x_n)$)]

そして記号[α]の記号列としての意味を次のように定義する．

α='?($p(x_1, \cdots, x_n)$)' の場合

```
(*?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

```
(*?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 属性付 RTF による簡潔な記法
Fig. 2 Arrtibuted RTF makes the description simpler than RTF.

```
?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 3 算術式演算系

Fig. 3 A calculator of arithmetic expressions.

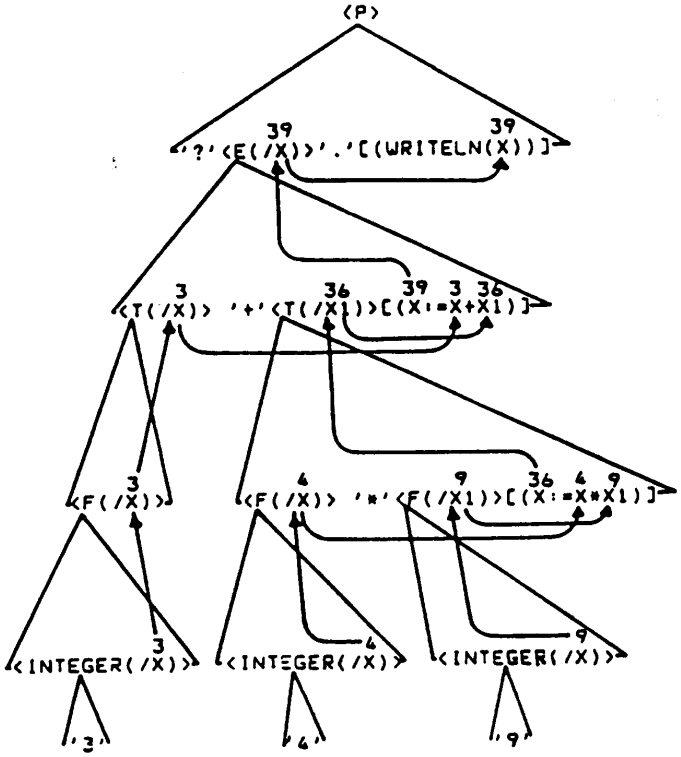

図 4 算術式の演算系に ?3+4*9 を入力したときの導出木と属性の値の流れ

Fig. 4 The derivation tree and the value flow of attributes made by the calculator for the input ?3+4*9.

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

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

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

  <DMM>=[DMM] ;                     NORMAL END

?END*)                              AAABBBC

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

(a) 定 義　　(b) 実行結果

図 5 文脈依存言語 $A^nB^nC^n$ を定義する属性付 RTF

Fig. 5 Attributed RTF which defines the context-sensitive language $A^nB^nC^n$.

$[\alpha]=\lambda$(空系列) if $P(x_1,\cdots,x_n)$ is true,

$=\phi$(空) if $P(x_1,\cdots,x_n)$ is false.

α='?$(P(x_1,\cdots,x_n))$' でない場合

$[\alpha]=\lambda$.

つまり活動記号として述語を用いることによって，その真偽により入力文字列（によるマッチング）を制御することができるわけである．たとえば

$[?(X=Y)]'a'+[?(X<>Y)]'b'$

は述語 $X=Y$ が真のときは $\lambda'a'+\phi'b'='a'$ となり，$X<>Y$ が真のときは $\phi'a'+'b'='b'$ 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3**(a)に示す．この例で活動ルーチンは数字の読取りと，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（属性付構文指元翻訳の結果）を図3(b)で示す．ここで生成された演算系に対して，

"?3+4*9."

を入力したときはできる導出木と属性の値の流れを**図4**に示す．

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 の使用例として，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 および簡単な英語の文に対して，その文脈に依存した部分を定義した属性付 RTF をそれぞれを示す．なお，これらの属性付 RTF は MYLANG に入力することによって，属性付構文指示翻訳を行う系に変換される．

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の一部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5**(a)に示す．これは文脈依存の言語として有名である．

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

    <F0(/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*)
```

図 6 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
Fig. 6 Attributed RTF and action routines which defines a translation of arithmetic and logical expressions into reverse Polish notation.

図5(a)において属性 I は 'A' の生起回数，J は 'B' の生起回数，K は 'C' の生起回数を表す．すべての生起回数を数え終わった後，属性 I, J, K が等しいかどうかを判定する．$I=J=K$ のときだけ [?(I=J)][?(I=K)] が λ になり，それ以外は ϕ になるから $A^IB^JC^K=A^nB^nC^n, n\geqq 0$ 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．同図(b)は同図(a)の属性付 RTF を MYLANG に入力することによって得られた系の実行結果である．

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.

算術式と論理式の逆ポーランド記法への変換を定義する属性付 RTF を**図6**に示す. これは次のような構文と意味をもつ.

○ 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. 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 '*' が強く, '+' が最も弱い.

この例で, 属性 T, T_1, T_2 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. このとき属性が表す型に依存して, たとえば '+' 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, $\langle F(/T)\rangle$, $\langle$PLUS $(T, T_1/)\rangle$, $\langle$MINUS $(T, T_1/)\rangle$, $\langle$MULTI$(T, T_1/)\rangle$ および $\langle$DIVIDE$(T, T_1/)\rangle$ で行われている.

たとえば

$\langle F(/T)\rangle$
$=\langle F0(/T)\rangle+'-'\langle F0(/T)\rangle$
([?$(T=0)$][CHS]+[?$(T=1)$][NOT]);

において, 単項演算子 '−' に続く因子 $\langle F0(/T)\rangle$ の型は合成属性 T で表される. $T=0$, つまり因子が整数型であれば [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. 次に,

$\langle$PLUS$(T, T_1/)\rangle$
=[?$(T=0)$][?$(T_1=0)$][PLUS]
+[?$(T=1)$][?$(T_1=1)$][OR];

は属性 T, T_1 が両方とも整数型のときに '+' を出力し, 属性 T, T_1 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, T_1 の値, つ

```
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


?X=?A/?B

ERROR IN TEXT
```

図7 図6を MYLANG 入力することによって生成された翻訳系の実行結果
Fig. 7 The result executed by the translator which defined by Fig. 6.

```
(*?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*)
```

(a) 定義

```
I LOVE YOU.

NORMAL END

I LOVES YOU.

ERROR IN TEXT

SHE LOVE YOU.

ERROR IN TEXT

SHE LOVES YOU.

NORMAL END
```

(b) 実行結果

図8 簡単な英語の文法を定義した属性付 RTF
Fig. 8 Attributed RTF which defines a simple English-like grammar.

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 という．

〈MINUS(T, T_1/)〉や〈DIVIDE(T, T_1/)〉では論理型に対する規則がない．したがって図7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．

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 では属性のも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[16]．その例として英語の文法を簡略化したものを考える．これを定義する属性付を図8(a)に示す．

図8(a)において属性Xは名詞の人称を表す．名詞を表す

〈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)〉と〈VP(X/)〉を経由して〈V(X/)〉に引き渡される．そして〈V(X/)〉で動詞の人称との一致を判定している．たとえば，

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

において，'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．

```
(*?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*)
```

(a) 定 義

```
 120
  55

NORMAL END
```

(b) 実行結果

図 9 プログラム図式としての属性付 RTF
Fig. 9 Attributed RTF as a program schema.

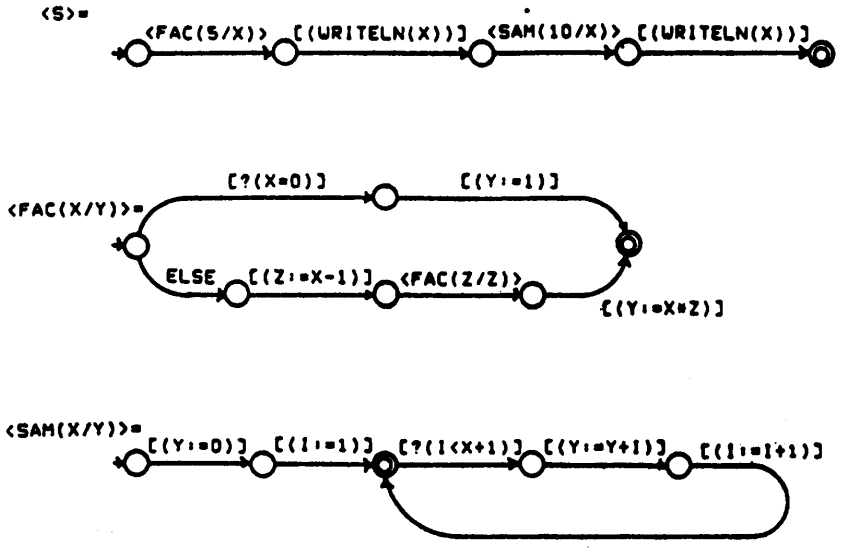

図 10 プログラム図式（◎は復帰または停止を表す）
Fig. 10 The program schema which are transformed from Fig. 9(a).

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指示翻訳を行う．4.1 節では属性付 RTF にソフトウェアモデルとしての解釈を与え，MYLANG それ自体が一つの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 のインタプリタの開発例をあげる．

4.1 ソフトウェアモデルとしての属性付 RTF

属性付 RTF は一種のソフトウェアモデルと解釈することができる．これはプログラム図式 (program schema : Manna)[9] を表している．その例として算術式の計算を定義する属性付 RTF を**図9**に示す．同図は5の階乗と，1から10までの和を求める計算系を定義する属性付 RTF であり，**図10**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．

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．この mini-BASIC コンパイラは属性付構文指示翻訳を行う．この節では IF 文を例にとって属性付構文指示翻訳を説明する．

mini-BASIC のプログラム例と実行例を**図11**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 のプログラムを三つ組 (triple) に変換する．三つ組はインタプリタによって実行される．IF 文を三つ組に変換する属性付 RTF と活動ルーチンを**図12**に示す．IF 文に対応する三つ組は**図13**のようになる．

```
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 11 mini-BASIC のプログラム例と実行例
Fig. 11 A mini-BASIC program and the executed result.

```
<IFS> ='IF'' '*<BOOL(/X)>' '*'THEN'' '*<STOD>[THENP(X/)]
                ('ELSE'' '*[ELSE1(X/Y)]<STOD>[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 12 IF 文から三つ組への変換を定義した属性付 RTF と活動ルーチン
Fig. 12 Attributed RTF and action routines which define the translation of IF-statement into triples.

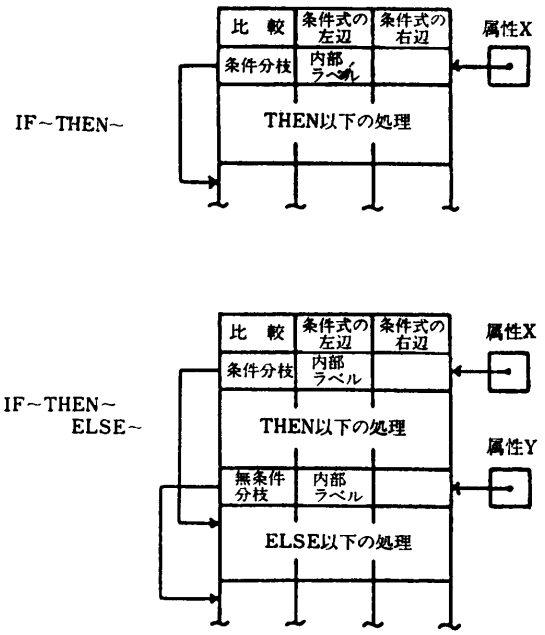

図 13 IF 文から変換された三つ組
Fig. 13 Triples which are transformed from IF-statement.

この例では，属性は前方参照を行うために用いている．三つ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．ここでは，活動ルーチン[THENP(X/)]によって，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している．

ELSE が〈STDD〉の後に続いた場合，先に決定した分岐命令の分岐先を次の番地にずらす．手続き MTXEND(27, Y, 0) によって，元の番地に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る．ELSE 以下の部分の処理が〈STDD〉で終わった後，[ELSE 2(Y/)]で無条件分岐命令の分岐先を書き込む．

このように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 インタプリタの開発

属性付 RTF と活動ルーチンで LISP インタプリタ[13)~15)]を定義し，これを MYLANG に入力することによって，LISP インタプリタを生成させることができた．**図14**は LISP インタプリタを定義している属性付 RTF の一部であり，**図15**は層性付 RTF と活動ルーチンを MYLANG に入力することによって生成された LISP インタプリタの実行結果である．

このインタプリタは，リスト領域とアトムテーブルの2種類のデータ構造をもっている．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み合わせ，積み重ねることによって，LISP インタプリタに必要な read-eval-print, eval および apply の三つの手続き（関数）を定義することができる．

```
<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 14 LISP インタプリタの属性付 RTF による定義の一部
Fig. 14 Attributed RTF which defines a LISP interpreter (a part).

```
(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 15 LISP インタプリタの実行の結果
Fig. 15 The result executed by the LISP interpreter.

図14において属性 A, X, B, Y, C および Z は，(A, X), (B, Y) そして (C, Z) がそれぞれ対になっており，それぞれの対が LISP における変数を表している．一般に LISP の変数はその値として NIL, T, 整数，アトムおよびリストをとることができる．属性 A, B および C はその種別を示す指標である．すなわち，その値が0のとき NIL，1のとき T，2のとき整数，3のときその他のアトム，そして4のときリストを表す．そして属性 X, Y および 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．

$\langle$CONS$(A, X, B, Y/C, Z)\rangle$ は LISP の基本的な関数 cons$[x; y]$ と同じ働きをもつ．属性付 RIF には“関数”はない．したがって結果の値は合成属性の対 (C, Z) に得るようにしている．

$\langle$CAR$(A, X/B, Y)\rangle$ や $\langle$CDR$(A, X/B, Y)\rangle$ はそれぞれ LISP の car$[X]$ や cdr$[X]$ と同じ働きをもつ．関数の値のかわりに，結果の値は合成属性の対 (B, Y) に得られる．LISP では

$$\text{list}[x; y]=\text{cons}[x; \text{cons}[y; \text{nil}]]$$

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．したがってこれを属性付 RTF で表せば

$$\langle\text{LIST}(A, X, B, Y/C, Z)\rangle = \langle\text{CONS}(B, Y, 0, 0/B, Y)\rangle \langle\text{CONS}(A, X, B, Y/C, Z)\rangle;$$

となる．同様にして cadr, cdar, caar, cddr などの定義も属性付 RTF に簡単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 は述語で定義できる．LISP の変数を x, y，これに対応する属性の対をそれぞれ (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 はともに4となる）．

マッカーシーの条件式：

$$[p_1 \rightarrow e_1; p_2 \rightarrow e_2; \cdots; p_n \rightarrow e_n]$$

（p_i は命題，e_i は式）

は属性付 RTF では

$$p_1e_1 + p_2e_2 + \cdots + p_ne_n$$

となる．ここで p_i は述語または非終端記号であり，e_i は活動記号または非終端記号である．e_i はなくてもかまわない．

$\langle$READ$(A, X/)\rangle$ はS式の読込みとリスト構造への変換を定義している．$\langle$NAME$(/A, X)\rangle$ はアトムを表している．アトムの値は属性の対 (A, X) として得られる．

$\langle$SEXPRESS$(/A, X)\rangle$ はS式を表している．これは再帰的に $\langle$READ$(A, X)\rangle$ を呼び出している．$\langle$RQUOTE$(/A, X)\rangle$ は QUOTE のついたS式の読込みを定義している．''' は QUOTE を表しており，ここではS式 x を読み込んで (QUOTE x) を作りその値を属性の対 (A, X) として返している．定義(3.3)がアトム 'QUOTE' を表している．

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，簡単な自然言語の構文解析，ソフトウェアモデル，手続き型言語の処理系，関数型言語の処理系などがそれぞれ定義ができることを例を挙げて示した．なお，mini-BASIC や LISP インタプリタの属性付 RTF と活動ルーチンによる定義の長さは，同じものを PASCAL だけで記述したものと比べて，1/2〜1/3 であった．

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 MYLANG は，九州工業大学情報処理教育センター，UCSD PASCAL システムおよび UNIX 上で利用可能である．現在 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，デバッキングツールなどについて研究を行っている．

謝辞 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，ならびに本システムの開発をプログラムライブラリ開発課題として認めていただいた九州大学大型計算機センターに感謝します．

参考文献

1) Knuth, D. E.: Semantics of Context-free Languages, *Math. Systems Theory*, Vol. 2, No. 2, pp. 127-145 (1968); *Correction*, Vol. 5, No. 1, pp. 95-96 (1971).
2) Koster, C. H. A.: Affix-Grammars in Peck, J. E. L.(ed.): *Algol 68 Implementation*, North-Holland, Amsterdam (1971).
3) Lewi, J. et al.: *A Programming Methodology in Compiler Construction*, Part 2, Implementation, North-Holland, Amsterdam (1982).
4) Koster, C. H. A.: Using the CDL Compiler-Compiler, in Bauer, F. and Eickel, J. (eds.): *Compiler Construction—An Advanced Course*, pp. 366-426, Springer, Berlin (1976).
5) Anzai, H.: A Theory of Recursive Descent Translator Generator, Proceedings of International Symposium, Vol. 2, pp. 1171-1182 (1980).
6) Lewis, P. M. et al.: Attributed Translations, *J. Comput. Syst. Sci.*, Vol. 9, No. 3, pp. 279-307 (1974).
7) Watt, D. A.: Rule Splitting and Attribute Directed Parsing, in *Semantic-Directed Compiler Generation, Lecture Notes in Compiler Science*, Vol. 94, pp. 363-392, Springer, Berlin (1980).
8) 安在弘幸：構文，意味および知識向き翻訳系の生成系について，情報処理学会知識工学と人工知能研究会，31-3 (1983).
9) Manna, Z.: Program Schema, in Aho, A. V. (ed.): *Currents in the Theory of Computing*, Prentice Hall, Englewood Cliffs (1973).
10) 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，第25回情報処理学会全国大会，1C-8 (1982).
11) 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，No. 47, pp. 61-68 (1983).
12) 山之上，金子，藤岡，安在：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).
13) 小林孝次郎：情報構造，サイエンス社，東京 (1977).
14) 中西正和：LISP 入門，近代科学社，東京 (1977).
15) Winston, P. and Horn, B.: *LISP*, Addison-Wesley, Reading (1981).
16) 中島秀之：Prolog，産業図書，東京 (1983).
17) 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，Vol. J 63-D, No. 9, pp. 771-778 (1980).

(昭和59年1月30日受付)
(昭和59年9月20日採録)