属性の値として非終端記号を持つことができる属性文法

5N-5

山之上 卓+ 前田博基++ 安在弘幸+
+九州工業大学
++九州松下電器株式会社

1．まえがき

属性文法はコンパイラ記述システムにおいて、現在もっとも有望な道具の１つである。

ＭＹＬＡＮＧは属性文法を採用したコンパイラーコンパイラの一つである。これまでに、LISP,PROLOGなどのインタープリタ、プログラムの構造（の一部）をインデントで表現する言語ＯＳＣＡＬのプリプロセッサ、日本語プログラミング言語ＮＢＳＧ／ＰＤのプリプロセッサ、簡略化された中国語文の構文解析システムなど、種々の言語処理系がＭＹＬＡＮＧを用いることによって実現されている。

今回、ＭＹＬＡＮＧの大幅なバージョンアップを行った。このなかで、ＭＹＬＡＮＧの入力記法である属性付正規翻訳記法（ＡＲＴＦ）を、その属性の値として非終端記号を使用することができるよう拡張した。このことによってプログラム言語の文法や意味などを、従来のＡＲＴＦと比較してより簡潔に記述できるようになった。

2．ＭＹＬＡＮＧの概要

ＭＹＬＡＮＧは、その入力記法（ＡＲＴＦ）で記述された言語処理系の定義から、目的の言語処理系を自動的に生成するものである。

ＭＹＬＡＮＧの一つの特徴は、属性文法で用いる意味関数の記述記法が、構文の記述記法と一致していることである。従ってＭＹＬＡＮＧのユーザーは、ただ１つの記法を習得することによってＭＹＬＡＮＧを使いこなせるようになる。（ＹＡＣＣの場合では、字句解析部生成系Ｌｅｘの入力記法である正規表現，構文解析部生成系ＹＡＣＣの入力記法であるＢＮＦ，意味解析部でＣ言語の合計３つの記法が必要となる。）

なおＡＲＴＦは副作用を持つため、厳密な意味での属性文法ではない。図１にメモリー付き電卓プログラムを定義したＡＲＴＦを示す。

```
/*?calm.rtf*/
/*?rtf
% stdio.h;
int a,i,x,y;
int $変数の値[30];
int *表ポインタ;
byte $変数表[30][80];
[s] =
    [inistdio][表の初期化]<入力>*;
[表の初期化]=
    [(*表ポインタ:=0,i:=0)]
    ([?(i<30)][($変数の値[i]:=0,i:=i+1)])*;
<入力>=
    ([ws('>')/)]('?'<出力> + '.'<終了>
                        +else<代入式>))*;
<出力>=
      <式(/x)><bln>[ws('      ')/)][wi(x/)][wlf];
<代入式> =
      <aname>[nametape($変数表[*表ポインタ]/)]
      '='[参照(/x)]
      <式(/y)><bln>[($変数の値[x]:=y)];
<式(/x)>=
      <項(/x)> ('+'<項(/y)>[(x:=x+y)]
               + '-'<項(/y)>[(x:=x-y)])*;
<項(/x)>=
      <因子(/x)>('*'<因子(/y)>[(x:=x*y)]
               + '/'<因子(/y)>[(x:=x/y)])*;
<因子(/x)>=
      <ri(/x)> + '('<式(/x)>')' + else<値(/x)>;
<値(/a)> =
  <aname>[nametape($変数表[*表ポインタ]/)]
  [参照(/x)][(a:=$変数の値[x])];
[参照(/y)]=
  [(y:=0)]
  ( !<eqstr($変数表[y],$変数表[*表ポインタ]/)>
    [(y:=y+1)]  )*
  ( ( [?(y=*表ポインタ)]
      [(*表ポインタ:=*表ポインタ+1)] )/ );
<終了>=[?(1=2)];
?end*/
```

図１．ＡＲＴＦで記述したメモリー付き電卓の定義

3．非終端記号を値として持つことができる属性

拡張ＢＮＦなどでプログラミング言語の文法を定義する場合、類似した文法パターンが数多く出現する場合がある。特に

＜～の並び＞＝＜～＞（’，’＜～＞）＊　；

のようなパターンは頻繁に出現する。

上の例で、もし～の部分を変数にできれば、文法の記述量の削減を期待できる。

ＡＲＴＦは変数を持つ。今回バージョンアップを行ったＭＹＬＡＮＧに入力されるＡＲＴＦは、変数の値として非終端記号を持つことができる。また、ＡＲＴＦの右辺の非終端記号を変数の値によって定めることができる。具体的には以下のように記述する。

- ○ 非終端記号を変数に代入する場合
  [（ｘ：＝＿＜非終端記号名＞）]
- ○ 非終端記号Ａを非終端記号Ｂの相続属性の値として引き渡す場合
  ＜Ｂ（＿＜Ａ＞／）＞
- ○ 右辺の非終端記号を変数ｎの値によって定める場合
  ＜＿ｎ＞

以上のような拡張によって、従来

```
<...>  = ...  <変数の並び> ... ;
<変数の並び> = <変数>(','<変数>)* ;
<...>  = ...  <記号の並び> ... ;
<記号の並び> = <記号>(','<記号>)* ;
```

のように記述していた文法を

```
<...>  = ...  <並び(_<変数>/)> ... ;
<...>  = ...  <並び(_<記号>/)> ... ;
<並び(n/)> = <_n>( ',' <_n>)* ;
```

のように書き換えることができる。

An Attributed Grammar
which Attributes may have Non-Terminal Symbols as its value
Takashi YAMANOUE+, Hiroki MAEDA++, Hiroyuki ANZAI+
+Kyushu Institute of Technology, ++Kyushu Matsushita Electric CO..LTD.

4．Ｏｓｃａｌにおける事例

Ｏｓｃａｌはインデントによってプログラムの構造（の一部）を決定する言語である。これはＰａｓｃａｌのｂｅｇｉｎ－ｅｎｄの代わりに、インデントの深さによってネストの深さや対応を表す言語である。図２にＯｓｃａｌで記述したソートプログラムを示す。

昨年ＭＹＬＡＮＧの旧バージョンによって、Ｏｓｃａｌプログラムを Ｐａｓｃａｌに変換するＯｓｃａｌトランスレータが作られた。今回、ＭＹＬＡＮＧの新バージョンを使ってＯｓｃａｌトランスレータを書き換えたので、その比較を行う。

```
OSCAL baka_sort
  CONST
     max=20
  TYPE
     tab_rec=RECORD
           key:INTEGER
           info:STRING[80]
  VAR
     i,j  : INTEGER
     work : tab_rec
     table: ARRAY[1..max] OF tab_rec
  PROC read_tab
     VAR
        i  :  INTEGER
     BEGIN read_rec
         FOR i:=1 to max
             readln(table[i].key)
             readln(table[i].info)
  BEGIN baka_sort
      read_tab
      i:=1
      WHILE i < max
          j:=i
          WHILE j < max
              j:=j+1
              IF table[i].key > table[j].key
                    work:=table[i]
                    table[i]:=table[j]
                    table[j]:=work
          write(table[i].key,' ')
          writeln(table[i].info)
          i:=i+1
```

図2．Ｏｓｃａｌで記述したソートプログラム

Ｏｓｃａｌはネストの対応などをインデントによって決定する。従来のＡＲＴＦでは、インデントの深さが同じかどうかを判定する部分を、定義の繰り返し、フィールドの並び、実行文の列などの定義に対して別々につけ加えなければならなかった（図３）。これに対して今回拡張を行ったＡＲＴＦは、同じようなパターンを持った規則を１つにまとめることができるため簡潔に記述できる（図４）。ＯＳＣＡＬの処理系全体での比較を表１に示す。

5．あとがき

本稿は、属性文法の属性の値として非終端記号を利用することを提案した。また、このことによって同じようなパターンを持った生成規則をまとめることができ、言語処理系の定義の量が、実際に減少することを示した。

今後ＭＹＬＡＮＧによるより多くの処理系の実現と、ＭＹＬＡＮＧへの抽象的データ型の導入などを行う予定である。

```
<変数定義(n/)>=
    'VAR'' '*<bln><ws('var'/)><wlf>
    <最初の変数行(n/m)><残りの変数行(m/)>* ;
<最初の変数行(n/m)>=
    <gbp(/b)><f_indent(n/m)>
   (<変数行(m/)>+else[.back(b/)][?(1=2)]) ;
<残りの変数行(m/)>=
    <gbp(/b)><indent(m/)>
   (<変数行(m/)>+else[.back(b/)][?(1=2)]) ;

<定数定義(n/)>=
   'CONST'' '*<bln><ws('const'/)><wlf>
   <最初の行(n/m)><残りの行(m/)>* ;
<最初の行(n/m)>=
   <gbp(/b)><f_indent(n/m)>
   (<その他の文(m/)>+else[.back(b/)][?(1=2)]) ;
<残りの行(m/)>=
   <gbp(/b)><indent(m/)>
   (<その他の文(m/)>+else[.back(b/)][?(1=2)]) ;
```

図3．旧ＡＲＴＦによるＯＳＣＡＬの定義の一部

```
<変数定義(n/)> =
   'VAR'' '* <bln> [ws('var'/)] [wlf]
   <繰り返し(_<変数行>,n/m)> ;

<定数定義(n/)> =
   'CONST'' '* <bln> [ws('const'/)] [wlf]
   <繰り返し(_<その他の文>,n/m)> ;

<繰り返し(nt,n/m)> =
   <最初の(nt,n/m)> <残りの(nt,m/)>* ;
<最初の(nt,n/m)>=
   [gbp2(/ip)] <f__indent(n/m)>
   (<_nt(m/)> + else [back(ip/)] [?(1=2)]) ;
<残りの(nt,m/)>=
   [gbp2(/ip)] <indent(m/)>
   (<_nt(m/)> + else [back(ip/)] [?(1=2)]) ;
```

図4．新ＡＲＴＦによるＯｓｃａｌの定義の一部

表１．旧ＡＲＴＦと新ＡＲＴＦによるＯｓｃａｌトランスレータの定義の比較

	旧ＡＲＴＦ	新ＡＲＴＦ
定義式の数	９１個	７３個
定義全体のサイズ	9302 byte	7968 byte

参考文献

[1]山之上、安在：言語処理系の生成系MYLANG、情報処理学会論文誌、Vol.26,No.1,pp.195-204 (1985).

[2]山之上、安在、吉田、杉尾、武内、椎野：言語処理系の生成系MYLANGによるNBSG/PDプリコンパイラの試作、情報処理学会論文誌、Vol.28,No.1,pp.64-73(1987).

[3]山之上、紀太、相沢：パソコン通信を利用したソフトウェアの共同開発、情報処理学会第３７回（昭和６３年度後期）全国大会3L-9 (1988).