ソフトウェア基礎論　17－5
(1986. 7. 3)

属性文法に基づく言語PANDAとDCGへの変換

Programming Language PANDA and its Translation into DCG

杉山　裕二*　　馮　安*
Yuji SUGIYAMA　　An FENG
藤井　護**　鳥居　宏次*　前野　芳史***
Mamoru FUJII　Koji TORII　Yoshifumi MAENO

*大阪大学基礎工学部情報工学科
Faculty of Engineering Science, Osaka University
**大阪大学大型計算機センター
Computation Center, Osaka University
***沖電気工業株式会社
OKI Electric Corporation

あらまし　言語やコンパイラの記述を目的とする属性文法に基づく言語PANDAを提案する。PANDAでは、コピー規則を少なくするために、略記法として“共通属性宣言”を導入しているPANDAで書いたコンパイラには、コピー規則は10%しかなく、非常に読み易いものとなっている。そして、PROLOGのDCGへのトランスレターについても述べる。

ABSTRACT　This paper proposes a programming language PANDA based on Attribute grammars, which has been developed to describe programming languages and compilers. As a basic concept,"common attribute" is introduced to get feasible descriptions. We will also discuss about the strategy for translating programs in PANDA into DCG in　PROLOG.

1. はじめに

論理型言語PROLOGには、DCG (Definite Clause Grammar)と呼ばれる表記法があり、これを用いれば、文法に基づいた処理を比較簡単に書くことができる。そのため、DCGはその種の問題のプロトタイピングに適している。しかし、DCGを種々の問題に応用する場合、構文規則だけの記述では事足りず、非終端記号に対応する述語に引数を持たせ、その値の評価のためPROLOG手続き (extra condition)を挿入する必要が起きる。その結果、DCGプログラムは、構文規則と引数やPROLOG手続きが混在したものとなり、繁雑で、読み易さにかける。これに対し、KNUTH[1]が文脈自由言語の意味を定義するための方法として提案した属性文法は、構文規則と意味規則がはっきり分離しており、その点では見やすくなっている。しかし、実用的なコンパイラなどを属性文法で記述しようとすると、属性値のコピーを行うだけの意味規則（コピー規則と呼ぶ）が非常に多くなり、記述が繁雑になるという問題がある。

本報告では、属性文法におけるコピー規則の省略に関する略記法、およびその略記法を採用した属性文法記述言語PANDA(Programming.lANguagefor Describing Attribute grammars)を提案し、VAX-11上に作成されているPANDA プログラムをDCGプログラムへ変換する方法について述べる。

属性文法におけるコピー規則の問題について、“単なるコピー規則は省略可”というような略記法を導入したものもあるが[2]、ここでは、それを一歩押し進め、通常の手続き言語の大域変数のような感覚で使うことができる略記法“共通属性宣言”を提案する。共通属性宣言では、属性識別を共通属性と宣言することにより、すべての非終端記号がその識別子に関連した相続属性と合成属性を持ち、これらに値を明示的に与えない限り、“左から右へ”の意味規則の評価順序に従って、値がコピーされるような意味規則が追加される。意味規則の評価順序を“左から右へ”としたのは、属性文法を書くうえで、それが一般的と判断したからである。

変換プログラムでは、共通属性宣言がなされた一連の属性に関して、変換されたDCGプログラムにおける引数間の不必要なコピーを省略する工夫がなされており、実行効率の向上が図られている。

2. 共通属性宣言の導入

2.1 属性文法における問題

コンパイラなどのような実用的プログラムを属性文法で書くと、コピー規則が非常に多く現われる。例えば、図1(a)のプログラムを入力したとき、図1(b)のよ

```
VAR   I , K;
BEGIN
   READ(K) ;
   I = K + 1 ;
   J := I * 2 ;
   WRITE(I)
END.
```

(a)

```
4 I = K + 1 ;
    ^ERROR: := is expected.
5 J := I * 2 ;
  ^ERROR:undefined variable.

There are 2 errors in your program.
```

(b)

図　1

うに、エラーの発生した場所とエラーの種類を示し、最後にエラーの総数を表示するようなコンパイラを作成するとしよう。

このような形でエラー・メッセージを出力するためには、変数名、関数名などのような識別子に関するデータやエラーの発生した場所、種類、個数などを記憶しておく必要がある。通常の手続言語では、大域変数に記憶され、モジュール間の値の受渡しは陽に書かなくてもよいが、属性文法では、属性値として、非終端記号から非終端記号へと値を渡していかなければならない。

[例1]　図1の処理を行うプログラムを属性文法で書けば、例えば、図2のようになる。

図2では、属性は次のように表わされている：

X_D_ATTR　または　X_ATTR

ここで、Xは非終端記号、Dは相続属性(I)または合成属性(S)の区別、ATTRは属性値の持つ意味を表わす。非終端記号の略記文字列および属性の意味は以下のとおりである。

sl:	<state_list>の略記文字列
st:	<statement>の略記文字列
blk:	<blank*>の略記文字列
lineno:	現在行の番号
line:	現在行の内容
pos:	エラーが起こった場所
err_list:	エラー・メッセージのリスト
errs_inline:	現在行にあるエラー数
errs:	プログラム中のエラー総数
const_set:	使える定数識別子の集合

```
<state_list> ::= <statement> <blank*>
        ";" <blank*> <state_list>
   { sl1_I_lineno -> st_I_lineno ;
     sl1_I_line -> st_I_line ;
     sl1_I_pos -> st_I_pos ;
     sl1_I_err_list -> st_I_err_list ;
     sl1_I_errs_inline ->
                    st_I_err_inline ;
     sl1_I_errs -> st_I_errs ;
     sl1_const_set -> st_const_set ;
     sl1_var_set -> st_var_set ;
     sl1_pro_set -> st_pro_set ;
     sl1_func_set -> st_func_set ;
     ......
     blk1_S_lineno -> blk2_I_lineno ;
     concat(blk1_S_line,";") ->
                    blk2_I_line ;
     blk1_S_pos -> blk2_I_pos ;
     blk1_S_err_list -> blk2_I_err_list ;
     blk1_S_errs_inline ->
                  blk2_I_err_inline;
     blk1_S_errs -> blk2_I_errs ;
     blk1_const_set -> blk2_const_set ;
     blk1_var_set -> blk2_var_set ;
     blk1_pro_set -> blk2_pro_set ;
     blk1_func_set -> blk2_func_set ;
     ......
     sl1_S_lineno := blk2_S_lineno ;
     sl1_S_line := blk2_S_line ;
     sl1_S_pos := blk2_S_pos ;
     sl1_S_err_list := blk2_S_err_list ;
     sl1_S_errs_inline :=
                    blk2_S_err_inline ;
     sl1_S_errs := blk2_S_errs ;
     sl1_code   :=
              concat(st_code,nl,sl2_code)
   }
```

図　2

var_set:	使える変数の集合
pro_set:	使える手続きの集合
func_set:	使える関数の集合

組込み関数concat(x1,x2,...,xn)は、文字列型の引数x1,x2,...,xnを順に連接した文字列を返す関数で、"->"、":="、"nl"の意味は以下のとおりである。

->: 相続属性への代入

:=: 合成属性への代入

nl: 改行符

非終端記号の略記文字列のあとにつく数字は構文規則

の右側にその非終端記号がいくつかあるとき、その中の何番目の非終端記号の属性かを示す(例えば、s12は2番目の<state_list>)。

図2で、意味規則のほとんどは同じ属性のコピーである。それらは、記述を繁雑にし、読みやすさを損なっている。そこで、そのようなコピー規則を少なくするような表記法を導入する。

2.2 共通属性

2.2.1 属性文法

属性文法は三つの要素からなる：

①文脈自由文法

$G = (V_T, V_N, P, S)$

但し、V_TおよびV_Nは、それぞれ終端記号および非終端記号の集合、Pは構文規則の集合、$S \in V_T$はGの開始記号である。 $V = V_T \cup V_N$ とおく。

②属性の集合

各非終端記号は合成属性／相続属性をもつ。非終端記号Xの合成属性、相続属性の集合をそれぞれSA(X), IA(X)と書き、$A(X) = SA(X) \cup IA(X)$とおく。但し、$IA(X) \cap SA(X) = \phi$，且つ互いに異なる 非終端記号X,Yに対して、

$$A(X) \cap A(Y) = \phi$$

各終端記号は合成属性／相続属性を持たないとする。

また、 $a \in A(X)$のとき、X = owner(a)と定義する。

③意味規則の集合

構文規則ごとに意味規則があり、そこでの属性値を評価する。構文規則の左辺の非終端記号のすべての合成属性、およびすべての右辺の非終端記号の相続属性に対しては、明示的に値を与えなければならない。

構文規則

$$X_0 ::= X_1\ X_2\ \ldots\ X_n \qquad (n \geq 1)$$

における（意味規則の）評価位置 i $(0 \leq i \leq n)$とは、i = 0 のとき、"::="とX_1の間、i = n のとき、X_nの後、$1 \leq i \leq n-1$のとき、X_{i-1}とX_iの間の場所を意味する。特に、評価位置0をfirst、n をlastと呼ぶ。

2.2.2 共通属性の置換

共通属性attrが宣言されると、各非終端記号Xは合成属性X_S_attrおよび相続属性X_I_attrをもつ。

構文規則の意味規則に、共通属性attrが複数回現われているとき、attrのk回目の出現を共通属性attrの具体例$attr_k$という。

意味規則中に現われる共通属性の具体例$attr_k$は図3に示すアルゴリズムCHANGEに従って、上記のいずれかの属性に置き換えられる。構文規則pにおいて、評価位置kの左側（または右側）で、評価位置kに一番近い非終端記号の添字(0からnまで)をleft_nont(p,k)（またはright_nont(p,k)）と書く。但し、X_nの右側の非

[アルゴリズム CHANGE]

構文規則 $X_0 ::= X_1\ X_2\ \ldots\ X_n (n \geq 1)$ の評価位置kの意味規則中に現われる共通属性の具体例$attr_k$は以下の属性に置き換えられる。

(1) $attr_k$に値が代入される場合、

(a)right_nont(p,k) = 0ならば、

X_0_S_attr.

(b)そうでなければ、

$X_{right_nont(p,k)}$_I_attr.

(2)そうでない場合（attrが引数として使われる）、

(a)left_nont(p,k) = 0ならば、

X_0_I_attr.

(b)そうでなければ、

$X_{left_nont(p,k)}$_S_attr.

図 3

終端記号はX_0であるとする。

[例2] 次の構文規則と意味規則を考える。

$Y_0 ::= Y_1\ Y_2\ \ldots\ Y_n \qquad (n \geq 1)$

{ B := f(A) }

但し、$Y_i (0 \leq i \leq n)$はすべて非終端記号、A,Bは共通属性、この意味規則の評価位置をkとする。CHANGEによりこの意味規則は次のように変換される：

(1) k = 0 のとき、

Y_1_I_B := f(Y_0_I_A)

(2) k = n のとき、

Y_0_S_B := f(Y_0_S_A)

(3)$1 \leq k \leq n$ のとき、

Y_{k+1}_I_B := f(Y_k_S_A)

2.2.3 コピー規則の追加

共通属性の値の設定・変更が明示的になされている場合、それがどのように合成属性／相続属性のみの意味規則に書き換えられるかを2.2.2で述べた。ここで、共通属性の値が非終端記号から非終端記号へと渡されていくために追加されるコピー規則について述べる。

図4に示すアルゴリズムADDに従って、各構文規則に意味規則が追加される。各意味規則中の共通属性はアルゴリズムCHANGEにより、相続属性または合成属性に変換されているものとする。

共通属性を持つ属性文法Gをアルゴリズム CHANGEおよびADDで変換して得られる（共通属性を持たない）属性文法をE(G)と書くことにする。

2.2.4 記述例

共通属性の導入は属性文法のクラスなどを拡張するでなく、記述をより簡潔、より短くするためである。例えば、もしline,...,func_setを共通属性として定

[アルゴリズムADD]
各構文規則p:
$X_0 ::= X_1\ X_2\ \ldots\ X_n \quad (n \geq 1)$
について、以下のコピー規則がpの意味規則として、
①$X_i \in V_N$ 且つ X_i_I_attrへ値を代入する意味規則が存在しないような各X_i $(1 \leq i \leq n)$, および各共通属性attrについて、
j = left_nont(p,i)とおいたとき、
(a) j = 0 ならば、意味規則
"X_i_I_attr <- X_0_I_attr"を
(b) j ≠ 0 ならば、意味規則
"X_i_I_attr <- X_j_S_attr "
を追加する。
②j = left_nont(p,n) とおく。X_0_S_attrへ値を代入する意味規則が存在しないような各共通属性attrについて、
(a) j = 0 ならば、意味規則
"X_0_S_attr := X_0_I_attr"を
(b) j ≠ 0 ならば、意味規則
"X_0_S_attr := X_j_S_attr "
を追加する。

図 4

義すれば、図2は図5のように簡単になる。
図5で、第3行目の ":"の前の数字3はその意味規則の評価位置で、意味規則 "concat(line,";")->line;" を評価位置3で、すなわち2行目の構文規則中の ";"と <blank*>の間で評価することを表わしている。共通属性 line は二つの具体例$line_1$,$line_2$があり、すなわち "concat($line_1$, ";") -> $line_2$; "である。

```
<state_list> ::= <statement> <blank*>
                 ";" <blank*> <state_list>
{ 3:  concat(line,";") -> line ;
  sll_code := concat(st_code,nl,sl2_code)
}
```

図 5

2.2.5 条件LCAG
これから、構文規則
p: $X_0 ::= X_1\ X_2\ \ldots\ X_n (n \geq 1)$
の評価位置k$(0 \leq k \leq n)$で明示的に評価される共通属性の具体例の全体の集合を
$CA(X_{right\_nont(k,p)})$
と書く。例えば、図 5 の構文規則では、$CA(X_4)$ = {$line_2$},$CA(X_i) = \phi\ (i=0,1,2,5)$。
属性文法のサブクラスに、評価順序が "左から右へ" 1パスですべての属性値が評価できるクラスL-AG[3]がある。共通属性をもつ属性文法GCが与えられるとき、GCが以下の条件LCAGを満たすとき、GCを展開したE(GC)はL-AGに属す。共通属性全体の集合をCAと書く。
[条件LCAG] すべてのp ∈ P :
$X_0 ::= X_1\ X_2\ \ldots\ X_n \quad (n \geq 1)$
(1)任意の$a \in IA(X_K)(1 \leq k \leq n)$を評価するとき、その規則にある引数は
$$IA(X_0) \cup CA \cup \bigcup_{i=1}^{k-1} IA(X_i) \cup \bigcup_{i=1}^{k-1} SA(X_i)$$
に属さなければならない。
(2)任意の$a \in SA(X_0)$を評価するとき、その規則にある引数は
$$IA(X_0) \cup CA \cup \bigcup_{i=1}^{n} IA(X_i) \cup \bigcup_{i=1}^{n} SA(X_i)$$
に属さなければならない。
(3) 任意a ∈ CAを評価するとき、
①その規則の評価位置が明示的に与えていれば、それをkとすると、その規則にある引数は
$$IA(X_0) \cup CA \cup \bigcup_{i=1}^{k-1} IA(X_i) \cup \bigcup_{i=1}^{k-1} SA(X_i)$$
に属さなければならない。
②そうでなければ、その規則の引数の中で$SA(X_0)$に属する属性があってはいけない。
条件LCAGは、GCが "左から右" への1パスで評価できるための十分条件であり、それを満たしていれば、図6のアルゴリズムでGCの評価を行うことができる。

```
ALGORITHM EVALUATE_SUBTREE(X0)
  FOR K = 1  TO n  DO
     IF XK ∈ VN THEN{
       適当な意味規則を用いて、
         attr∈IA(XK)∪CA(XK)を評価する ;
       CALL EVALUATE_SUBTREE(XK)
       }
  適当な意味規則を用いて、
     attr ∈ SA(X0)∪CA(X0)を評価する
```

図 6

3. PANDA

PANDAは共通属性をもつ属性文法を記述するための言語である。ここでは、その概要を紹介し、PL/0+コ

ンパイラの記述を例として上げる。

PL/0+ とは、PL/0[4]にWRITE文、READ文を追加した言語である。PANDAで書いたPL/0+コンパイラの記述の抜粋は図7に示す（但し、"..."は省略された部分を表わす）。

PANDAプログラム（PANDA言語による記述をそう呼ぶ）は次の五つの部分からなる：

```
定数宣言部
  共通属性宣言部
  関数定義部
  メイン部
  プログラム本体
```

（Ⅰ） 定数宣言部

定数宣言部は、プログラムを読みやすくしたり頻繁に使うような定数を宣言する。図7 では 1 行目から 3 行目までが定数宣言部にあたり、NO,MAX_NUM_LENをそれぞれ0､10で置き換えることを表わしている。

（Ⅱ）共通属性宣言部

共通属性宣言部は以下で用いる共通属性を宣言する。図7では、4行目から7行目までが共通属性宣言部で、11個の共通属性を宣言している。

（Ⅲ）関数定義部

関数定義部で利用者が関数、手続きを手続的に定義する。図7では、8行目から18行目までが関数定義部にあたり、output_err,error等を定義する。
関数定義部で定義できるのは関数と手続きの二種類がある。

手続きは、共通属性に値を与えたり、入出力を行ったりするが、手続き自身は値を持たない。手続き内部で宣言される変数は手続きに局部的である。引数は“call by reference”で値が渡される。引数および変数のデータタイプはint,bool, char,string, setのいずれかである。手続き定義で許される文は、if文、case文、write文、read文、代入文および手続きである。

一方、関数は、副作用がなくて、関数値として一つの値を返す。引数は“call by value”で値を渡す。手続き定義で許される文は、 if文、case文、代入文およびreturn文である。

（Ⅳ）メイン部

メイン部はプログラムの開始点を表わす部分で、次のようになっている：

<プログラム名(MAIN)> ::= < 開始記号 >
　　{ $文_1$;$文_2$;...;$文_n$}.

```
1 CONST NO = 0;
2       ...
3       MAX_NUM_LEN = 10;
4 GLOBAL var_set,pro_set,
5            func_set,const_set : SET;
6       err_list,line,pos : STRING ;
7       errors,err_inline,lineno,level: INT;
8 DEF  output_err(err : INT)
9 {        ...              }
10 DEF  error(errno : INT)
11 {   ...
12 CASE errno of
13  1:err_list:=concat(err_list,
14                'Use = instead of :=',nl);
15        ...
16  26:err_list:=concat(err_list,
17                'Not complete program',nl)
18  end  }
19 <pl(main)> ::= <pl0>
20  { head: [ err_list <- '';
21            ...
22            '' -> pro_set];
23  end:IF errors!=0 THEN output_err(errors)
24            ELSE      write(pl0_code) }.
25 <pl0> ::= <blank*><block> <blank*>
26                    "." <blank*>
27  { pl0_code <- bck_code }.
28    ...
29 <number> ::= <digit+>
30  { num_img := d+_img ;
31   IF d+_len>MAX_NUM_LEN THEN error(25) }.
32 <digit+> ::= <digit> <digit+>
33  { d+_img := concat(d_img,d+_img);
34    d+_len := d+_len + 1             }.
35 <digit+> ::= <digit>
36  { d+_img := d_img;
37    d+_len := 1            }.
38 <digit> ::= [c]
39       WHEN ("0"≦c)and(c ≦ "9")
40       { d_img := chr(c);
41        end: [line := concat(line,chr(c));
42              pos := concat(pos,' ')]}.
43 <empty> ::= "".
```

図 7

但し、n > 0。図7では、19行目から24行目までがメイン部で、プログラム名はpl、開始記号はpl0である。$文_i$(1≦i≦n)は意味規則((Ⅴ)の(c)参照) で、開始記号の属性を評価したり、入出力を行う。共通属性の初

期値は、評価位置0の意味規則でを与えるとよい。例えば、図7の20行目から22行目までは共通属性に初期化があり、23行目から24行目までは共通属性errorsと合成属性p10_codeの処理である。

(V) プログラム本体

プログラム本体は構文規則と文脈条件、意味規則の並びである（図7の25行目以降）。

(a)構文規則

構文規則は次のように書く：

X_0 ::= X_1 X_2 . . . X_n $(n \geq 1)$

ここで、X_0は非終端記号、$X_i(1 \leq i \leq n)$は非終端記号または終端記号で、非終端記号は“<"と“>"で囲み（例えば、<number>,<digit+>）、終端記号の文字列は引用符で囲む（例えば、"BEGIN","."）。

すべての文字 c について、

<char> ::= "c"

という構文規則が必要なとき、一般に文字の種類だけ上記のような構文規則を書かねばならないが、PANDAでは、“任意の一文字”を表わす終端記号が用意されており、識別子を“["と“]"で囲んで表わす（図7の38行目参照）。

(b) 文脈条件

文脈条件は、構文規則が適用されるための条件を表わし、次ぎのように書かれる：

WHEN 条件式

例えば、図7の39行目で、一文字 c が

"0"≦c≦"9"

を満たすとき、すなわちcが数字のとき、38行目の構文規則が適用される。

(c) 意味規則

次のようになる：

{文$_1$;文$_2$; ...文$_m$}

但し、m ≧ 0。文$_i(1 \leq i \leq m)$は意味規則で、if文、case文、write文、read文、代入文、手続きが許される。

PANDA では、三種類の代入文がある：

a_0 := f(a_1,a_2,...,a_k) (1)

b_0 <- f(b_1,b_2,...,b_k) (2)

f(b_1,b_2,...,b_k) -> b_0 (3)

但し、$k \geq 0$,$a_0 \in$ CAUSA,$b_0 \in$ CAUIA。すなわち、合成属性に代入を行うとき、かならず(1)のように書く(例えば、図7の33行目)。相続属性に代入を行うとき、かならず(2)または(3)のように書く。共通属性に代入を行うとき、自由にどちらを使ってもよい(例えば、図7の20,22,41行目)。共通属性宣言と代入文の書き方から、代入される属性が合成属性、相続属性、共通属性のいずれかであるかが分かる。例えば、図7で、num_img,p10_codeが合成属性で(書き方により)、err_list,pro_setが共通属性(4,6行目の宣言により)である。

(VI)略記文字列

属性は、X_Aで表わされる。但し、Xは非終端記号の識別子、Aは属性の識別子である。構文規則の読みやすさのために非終端記号として長い名前を用いることがしばしばである。そこで、非終端記号の識別子として次の条件を満たす略記文字列を許している：

$\alpha_1\alpha_2 \ldots \alpha_n$を非終端記号とするとき($\alpha_i$は文字、$1 \leq i \leq n$)、略記文字列は$\alpha_1^{m1}\alpha_2^{m2} \ldots \alpha_n^{mn}$。

$mk(1 \leq k \leq n)$は0または1（但し、$\sum_{k=1}^{n} mk > 0$）で、

$mk = 0$のとき、α_kは省略されることを表わす。

略記文字列の表わす非終端記号は次のにより決定される：

①さきの条件を満たす非終端記号を選ぶ。そのような非終端記号がなければ、その略記文字列に対応する非終端記号は存在しない(エラーである)。それが一つなら決定、二つ以上あるときは②へ進む。

②略記文字列と非終端記号との一致した範囲の長さが一番長いものを選ぶ。それが一つなら決定、二つ以上あるときは③へ進む。

③略記文字列と非終端記号とが（左から右へ比較していき）最初に一致した文字の位置が最も左のものを選ぶ。それが一つなら決定。二つ以上あるときは④へ進む。

④略記文字列と非終端記号とが最後に一致した位置が最も右のものをを選ぶ。それが二つ以上あるとき、その略記文字列に対応する非終端記号は存在しない（エラーである）とする。

例えば、図7の33行目にある d_img、d+_img がそれぞれ digit_img、digit+_img を表わす。

(VII)組み込み関数

PANDAでは、以下の組み込み関数を用意している：

(1)手続き

① write(引数$_1$, 引数$_2$, ... , 引数$_n$)
引数の値を順に出力する。

② read(引数$_1$, 引数$_2$, ... , 引数$_n$)
順に引数に値を入力する。

(2)関数

① concat(引数$_1$, 引数$_2$, ... , 引数$_n$)
文字列型である引数を順に連結した文字列を返す。

② union(引数$_1$, 引数$_2$, ... , 引数$_n$)
集合である引数から次のような集合を返す：
{x | x∈引数$_1$,...,または x∈引数$_n$}

③ intersection(引数$_1$, 引数$_2$, ... , 引数$_n$)
集合である引数から次のような集合を返す：
{x | x∈引数$_1$,...,かつx∈引数$_n$}

④ difference(引数$_1$, 引数$_2$)

集合である引数から次のような集合を返す：
{x | xは引数$_1$に属し、かつ引数$_2$に属さない}

⑤ insert(引数$_1$, 引数$_2$)
集合である引数$_1$に引数$_2$の値加えた集合を返す。

⑥ member(引数$_1$, 引数$_2$)
引数$_1$が集合である引数$_2$の要素であればtrue、そうでなければfalseを返す。

4. DCGへの変換

PANDAの実行系として、ここではPANDAプログラムを条件LCAGを満たすもののみに限定し、それをDCGプログラムに変換し、PROLOGにより実行する方法を採用した。

まず、簡単にDCGについて述べ、その後、PANDAプログラムをDCGに変換する方法の概略を示す。

4.1 DCG

一般に、DCGの構文規則は、次の形をしている。

LHS --> RHS.

・文法規則の左辺LHSは、非終端記号を表わす述語である。
・文法規則の右辺RHSは、非終端記号を表わす述語または終端記号を ',' または ';' で区切った並びである。','、';'はそれぞれ“かつ（and)”、“または（or)”という意味を持つ。
・終端記号は、任意のPROLOGの項である。文字列であるときは、PROLOGのリストで表わす。
・PROLOG手続呼出しである補強項（extra condition）は、右辺の任意の位置に、括弧 '{ 'および '}'で囲んで挿入できる。

4.2 変換方法

PANDAでの記述を次の手順でDCGへ変換する：

(1)定数宣言部で定義され定数名は、その実際の値と置換える。

(2)関数定義部では、関数と手続きに分けて考える：

(a)関数の場合、次ぎのように変換する：

'関数名'(引数$_1$,..., 引数$_n$, リターン引数):-
文の変換結果

文の変換については、意味規則の変換（(6)参照）に順ずるが、return文に関しては式をリターン引数に代入する文に変換する。

(b)手続きの場合、次ぎのように変換する：

'手続き名'(引数$_1$,..., 引数$_n$):-
文の変換結果

(3)メインコントロール部は次のように変換する：

'プログラム名'(Input) :-
get_inp(Input,List),!,
{位置 0 の属性評価式の並び},
'開始記号'(a_1,...,a_n,b_1,...,b_k,List,_),
{位置 1の属性評価式の並び}

ここでget_inp(Input,List)は、ファイルInputより文字列を読込み、Listにリスト展開する手続きで、a_1,...,a_nは意味規則で評価する開始記号の合成属性／相続属性で、b_1,...,b_kは共通属性に対応する。また、位置 0,1 の属性評価式の並びは、それぞれ評価位置 0,1 のメイン部の文（意味規則）をカンマで区切って並べたものである。意味規則の評価位置の決め方は(6)で述べる。

(4)各非終端記号を','で囲んでDCGの項とする。

(5)各非終端記号の属性は、対応する項の引数とする。そのとき、属性名の先頭の文字列が小文字であれば、prologの変数にするため、その前に下線“_”をつける。また構文規則に同じ非終端記号が現われるときは、属性（変数）名の後に数字を付けて区別する。

共通属性についても、値が変わり得る場合には、他の属性と同じように属性名の後に数字を付けて区別するが、値が同じ場合には、同じ変数を使い、値のコピーだけの補強項はできるだけ少なくする。例えば、共通属性aについて、構文規則

<A> ::= <B> <C>

でaの値を定める規則がなければ、

'A'(_a0,_a2) --> 'B'(_a0,_a1),'C'(_a1,_a2).

に変換される。

(6) 意味規則については、まず以下の変換をする：

・代入式、出力式の時、EXP を EXP' に
・条件式の時、if COND then EXP_1 else EXP_2
を ((COND',EXP_1');EXP_2') に
・条件式の時、
EXP_1 or EXP_2 を (EXP_1';EXP_2') に
EXP_1 and EXP_2 を (EXP_1',EXP_2') に

但し、COND',EXP',EXP_1',EXP_2'はそれぞれCOND,EXP,EXP_1,EXP_2をprolog記述に変換したものを表わす。

変換した意味規則を以下の方法で決定される位置に補強項として挿入する。

構文規則を

X_0 ::= X_1 X_2 ... X_n ($n \geq 1$)

とすると、

①意味規則の評価位置が明示的に与えられてあれば、示されたところへ挿入する。そうでなければ、②へ進む。

②相続属性の評価であれば、その属性のownerはX_i($0 \leq i \leq n$)であれば、評価位置 i-1 へ挿入する。そうでなければ、③へ進む。

③合成属性の評価（そのownerがX_0でなければならない）であれば、評価位置 n へ挿入する。そうでなければ、④へ進む。

④属性a_1,...,a_kを引数として持つ式であれば、以下の(a),(b)により決定される評価位置へ挿入する：

(a)まず、$a_1,\ldots,a_k$から共通属性を取除いたものを$b_1,\ldots,b_m(0\leq m\leq k)$とする。m = 0ならば、評価位置は n で、そうでなければ、(b)へ進む。

(b)owner(b_i)($1\leq i\leq m$)の添字の最大値をevalpとする。$b_1,\ldots,b_m$の中で、owner(b_i) = X_{evalp}であるような属性の中で合成属性があれば、その評価位置はevalpで、そうでなければ、その評価位置はevalp-1である。

PANDAプログラム（属性記述）が条件LCAGを満たしていれば、上記の①～④で位置が決定される。

(7)文脈条件は、(5)(6)に準ずる。

(8)関数の変換

次の形に変換する：

'関数名'(引数$_1,\ldots,$引数$_n$,リターン引数)

(9)手続き

次の形に変換する：

'手続き名'(引数$_1,\ldots,$引数$_n,a_1,a_2,\ldots,a_k$)

ここで、$a_1,a_2,\ldots,a_k$は共通属性に対応する。

例えば、図7で32～34行目は図8のように変換される。ここで注意してもらいたいのは、変換されたDCGでもコピーだけを行う部分はほとんどないことである。従って、共通属性の導入は記述を短くするだけでなく、実行効率も向上できる。

```
'digit+'(_L_dP_img,_L_dP_len,_var_set0,
    _var_set2,_pro_set0,_pro_set2,_func_set0,
    _func_set2,_cons_set0,_cons_set2,_lineno2,
    _err_lis0,_err_lis2,_errors0,_errors2,
    _blanks0,_blanks2) -->
 'digit'(_d_img,_var_set0,_var_set1,_pro_set0,
    _pro_set1,_func_set0,_func_set1,_cons_set0,
    _cons_set1,_err_inline0,_err_inline1,_line0,
    _line1,_lineno0,_lineno1,_err_lis0,
    _err_lis1,_errors0,_errors1,_blanks0,
    _blanks1),
 'digit+'(_R_dP_img,_R_dP_len,_var_set1,
    _var_set2,_pro_set1,_pro_set2,_func_set1,
    _func_set2,_cons_set1,_cons_set2,
    _err_inline1,_err_inline2,_line1,_line2,
    _lineno1,_lineno2,_err_lis1,_err_lis2,
    _errors1,_errors2,_blanks1,_blanks2),
  {concat(_d_img,_R_dP_img,_R1),
   _L_dP_img = _R1,
   _L_dP_len is _L_dP_len + 1 }.
```

図 8

5. まとめ

共通属性などの導入により、PANDAプログラムは非常に簡潔かつ読み易いものとなる。実際、PL/0+言語のコンパイラを、2.で一部述べたようなエラー処理を行うように、PANDAで記述してみたが、400行程度ですんだ。もし共通属性宣言を用いなければ、4000行程度になる。一般に、属性文法によるコンパイラの記述の50-70%はコピー規則であると言われているが[2]、PANDA によるPL/0+コンパイラの記述には、コピー規則が38個(約10%)しかなかった。DCGで同様のPL/0+コンパイラを記述してみても、行数は3000行以上になり、これと比較してもPANDAプログラムの簡潔さが分かる。

PANDAプログラムをDCGプログラムへ変換するプログラムはVAX-11上で、C言語で書かれている、約8000行で、4か人月かかった。PANDAによるPL/0+コンパイラの記述をDCGプログラムへの変換時間は約2分である。

PANDA実行系としては、現時点では、その対象になるプログラムとしてはL-AGに属するものしか考えていないが、より大きなクラス（例えば、m-APAG,OAG[7]）への拡張や、カット演算子の挿入など、変換されたDCGプログラムの最適化を検討している。また、構造エディタなど PANDAのプログラミング環境についても検討し、順次作成していく予定である。

参考文献：

[1]Knuth,D.E: Semantics of Context-free Languages,Math. Syst. Th.,Vol. 2,No.2 (1968),pp.127-145.

[2]Farrow,R.:Generating a Pratical Compiler from an Attribute Grammar,IEEE software,Vol.1, No.4(Oct.1984),pp.77-93.

[3]Bochmann,G.V.:Semantic Evaluation from Left to Right,CACM,Vol.19,No.2(Feb.1976), pp.55-62.

[4]N.Wirth: Algorithm + Data structure = Programs, Prentice-Hall(1976).

[5]Pereira,F.,Warren,D.:Definite Clause Grammar for Language Analysis,AI 13(1980),pp.231-278.

[6]Yellin,D.M. & Mueckstein,E.M.: Two-Way translator Based on Attribute Grammar Inversion,IEEE Proceeding of the 8th International Conference on Software Engineering,pp.36-42(1985).

[7]Kastens,U.: Oredered Attributed Grammars,Acta Informatica,Vol.13,1980.

[8]前野：属性文法記述からPROLOGプログラムの生成に関する研究，大阪大学基礎工学研究科修士論文。

[8]前野，杉山，藤井，鳥居：属性文法記述からDCGの生成，61年度電子通信学会総合全国大会。