自然言語処理向き構文解析プログラムの自動生成システム（NLAPG）の作成

桃内佳雄　真野宏之　宮本衛市
（北海道大学　工学部）

1. まえがき

自然言語の構文解析システムでは意味情報の処理が重要であり、拡張LINGOL[1]、ATN[2]などは意味情報処理も含めた構文解析を行う。これらのシステムでは、自然言語の意味情報処理も含んだ形の文法の記述形式を与えているということもでき、それは文脈自由文法を基礎としたもので拡張文脈自由文法と呼ぶことができる。また、自然言語の構文解析システムでは、意味情報を蓄積する辞書の作成も重要であり、大規模な辞書の利用ということを考えると、辞書は文法の記述とは分離して管理されることがのぞましい。

本報告で述べる構文解析プログラム自動生成システム、NLAPG（*Natural Language Analysis Program Generator*）は、ATNに基づいて形式化された拡張文脈自由文法によって意味情報処理も含んだ文法の記述を行い、それを構文解析プログラムのための仕様として、構文解析プログラムを自動生成する。構文解析プログラムはPL/Iプログラムとして生成されるので、他のPL/Iプログラムとの結合、移植が容易である。また、辞書はVSAM（*Virtual Storage Access Method*）ファイルにVSAMエディタを用いて作成し、辞書引き処理の高速化を図っている。構文解析プログラムの構文解析法は後戻りのある再帰的下向き構文解析法であり、左再帰性の処理が問題となるが、強制的な後戻りなどにより、ある程度の制御が可能である。日本語処理向きに、入力文の自動分割、活用処理などを行う手続きも作成されている。

NLAPGは構文解析プログラム仕様の編集を行うためのエディタを含み、また、辞書、意味情報処理用の手続きをVSAMファイルに作成、編集するためのエディタも用意されており、全体として、小規模ながら構文解析プログラム作成のための一つのソフトウエア環境が構成されている。

2. システムの概略構成

NLAPGのシステム構成、およびそれをとりまく環境の概略を図1に示す。VSAM、VOS3・TSSエディタにより各種ファイルを作成し、NLAPGに構文解析プログラム仕様を入力し、構文解析プログラムを作成する。構文解析プログラムはPL/Iプログラムとして生成され、PL/I処理系により翻訳・実行させることにより入力文の構文解析を行うことができる。構文解析プログラム仕様は、その主要部分が拡張文脈自由文法で記述される文法である。ユーザ定義関数ファイルには、文法規則に含まれる意味処理部で用いられる関数を格納する。構文解析用関数ファイルには、辞書引き、自動分

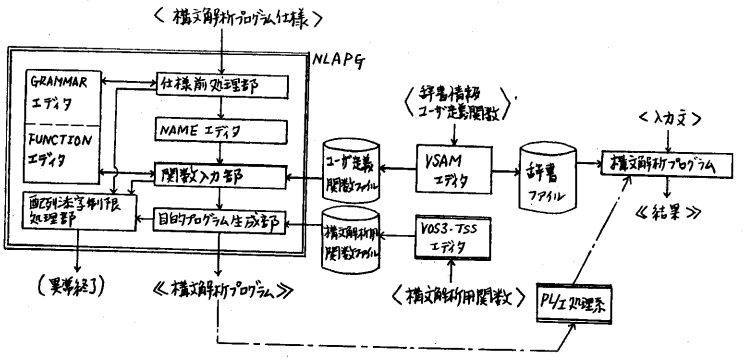

図1　システムの概略構成図

割、活用処理などのための構文解析用サブプログラムを格納する。辞書ファイルには単語辞書を格納する。

仕様前処理部では、構文解析プログラム仕様の内部表現への変換、名前の表の作成などを行い、さらに、仕様記述に構文的なエラーを発見すると GRAMMAR エディタによる編集を促す。NAME エディタは、仕様前処理部で作成された仕様の内部表現、名前の表に基づいて名前を中心とした編集処理を行う。関数入力部は、ユーザ定義関数の入力処理、FUNCTION エディタはそれに伴う編集処理を行う。目的プログラム生成部では、構文解析プログラム仕様と、ユーザ定義関数ファイルおよび構文解析用関数ファイルからの情報を利用して、目的プログラムである構文解析プログラムを生成する。配列添字制限処理部では、配列型変数の添字の値があらかじめ設定されている最大値をこえた場合に、その詳細を出力しシステムの実行を終了する。

目的プログラムである構文解析プログラムは、構文解析プログラム仕様の中でメインとして作成するか、サブとして作成するかの指示を記述することにより、そのいずれかの形の PL/I プログラムとして生成することができる。

3. 構文解析プログラム仕様

NLAPG に入力する構文解析プログラム仕様のトップレベルの構成は次のようなものである。

〈構文解析プログラム仕様〉::= 〈仕様頭部〉〈変数名定義部〉〈文法記述部〉 BYE .

〈仕様頭部〉では、NLAPG が生成する構文解析プログラムの名前と、そのプログラムをメインとして作成するかサブとして作成するかを記述する。サブとして作成する場合には、必要な引数の型の記述も同時に行う。

〈仕様頭部〉::= 〈プログラム名〉: MAIN | 〈プログラム名〉: PROC [(引数の型の並び)] .

〈変数名定義部〉は、$NAME$ 行で始まり、文法記述部の意味処理の記述の中で用いる変数名を定義する。

〈変数名定義部〉::= $NAME$ [〈型〉: 〈変数名の並び〉]* .

変数名の型としては次のようなものがある。

R (Register) : 長さに制限のない文字列の型。
C (Character) : 長さ 80 以内の文字列の型。
U (Unit) : 長さに制限のない文字列型の有限個のスロットから構成される構造の型。
I (Integer) : 整数の型。
F (Function) : 関数の型。
D (Delimiter) : 構文解析プログラムへの入力文の区切り記号の型。
E (Endmark) : 構文解析プログラムへの入力文の終止記号の型。
L (Level) : 構文解析の深さを表わす変数名の型、デフォルトは 'LV'。
IF (Input File) : 入力ファイルの型。
OF (Output File) : 出力ファイルの型。

〈文法記述部〉は、構文解析のための文法を記述する部分で、文法は ATN を参考にして形式化した拡張文脈自由文法である。拡張部分は主として意味処理の部分で、これを ATN における、*test*、*preaction*、*action* に対応するものによって記述する。

〈文法記述部〉::= $GRAM$ [〈文法規則〉]+ 〈特殊処理記述部〉 .

〈文法規則〉 ::= 〈構文規則記述部〉〈意味処理記述部〉

文法規則の簡単な1例の概略を示すと次のようになる。

<NP> => <ADJ><N> | <N>　　… 構文規則記述部

{ <test><preaction><action> } …(<ADJ>に対する)
{ <test><preaction><action> } …(<N>に対する)　} 意味処理記述部
{ <test><preaction><action> } …(<N>に対する)
{# <test>　T　　<action> } …(<NP>に対する)

<意味処理記述部>では、構文規則の各非端記号に付随する意味処理の記述を行う。test、preaction、action はATNにおけるものと同じ意味を持つ。testにはPL/Iにおける論理式、preaction、actionにはPL/Iにおける文の並び(;で区切られている)を(と)で括って書くことができる。<意味処理記述部>における、PL/Iの論理式、文で用いられる変数名は<変数名定義部>で定義されているものでなければならない。特殊な変数として次のようなものがある。＊(型R)、&(型C)、#(型U)、%(型I)。これらの変数はグローバル変数として用いられる。

<特殊処理記述部>は文法の開始記号に対して付与し、構文解析における最終的な意味処理を記述するためのものである。

<文法規則>の最初のものは、必ず開始記号に対するものでなくてはならない。また、NLAPGでは文法規則と辞書とを分離して処理することにしているので、構文規則の中には終端記号に相当するものは現われてはいけない。構文規則中の左辺に現われない非終端記号は、品詞名として取り扱われる。唯一つの終端記号としての例外は空語を表わす？である。

<意味処理記述部>のpreaction, action部で用いる関数、手続きはあらかじめユーザ定義関数ファイルに格納しておくか、FUNCTIONエディタを用いて定義する。次のような関数、手続きが作成ずみである。

COPYU : PROC(OUTU,INU); INUユニットをOUTUユニットへ複写する。
COPYR : PROC(OUTR,INR); INRレジスタをOUTRレジスタへ複写する。
SETR : PROC(OUTR,INCHAR); INCHARキャラクタをOUTRレジスタへ転送する。
GETU : PROC(UNT, NO) RETURNS(P); UNTユニットのNOスロットを複写し、それへのポインタを戻す。
PRINTR : PROC(PR); PRレジスタの内容をSYSOUTへ出力する。
PRINTI : PROC(PI); PIインテジャの内容をSYSOUTへ出力する。

<意味処理記述部>中のT(True)は、testに対しては"真"を、preaction、actionに対しては"行為なし"を意味する。

図2に示すのは、構文解析プログラム仕様の簡単な1例である。

```
JAPTET : MAIN
  $NAME$
    R: RSUB RPRED RN RT RPAT RW RV
    U: USUB UPRED UN UV
    C: CPAT CVAR
    I: ISUB IPRED INN IPAT IW IV
    F: COPYU COPYR SETR UEQ GETU APPENDR
       PRINTR PRINTC PRINTI RTOC GETPRAC
    E: .
    D: .
  $ORAM$
    <STM>=><SUBJ><PRED>
      {     T       T
        ( CALL COPYR(RSUB(LV),*);
          CALL COPYU(USUB(LV),#);
          ISUB(LV)=%;                          $) }
      {     T       T
        ( CALL COPYR(RPRED(LV),*);
          CALL COPYU(UPRED(LV),#);
          IPRED(LV)=%;
          CVAR(LV)=&;                          $) }
      {# ( UEQ(USUB(LV),2,UPRED(LV),2)='T' $)   T
        ( CALL APPENDR(*,RSYB(LV),RPRED(LV));
          %=ISUB(LV)+IPRED(LV);
          &=CVAR(LV);                          $) }
    <SUBJ>=><PN>(?!<TIT>)<PAT>
      {     T       T
        ( CALL SETR(RN(LV),&!!' ');
          CALL COPYU(UN(LV),#);
          INN(LV)=1;                           $) }
      {     T       T       T       T       }
      {     T       T
        ( CALL SETR(RT(LV),&!!' ');
          CALL APPENDR(RN(LV),RN(LV),RT(LV));
          INN(LV)=INN(LV)+1;                   $) }
      {     T       T
        ( CALL SETR(RPAT(LV),&!!' ');
          CPAT(LV)=RTOC(*);
          IPAT(LV)=1;                          $) }
      {# ( ( CPAT(LV)='WA' ! CPAT(LV)='GA' ) $)   T
        ( CALL APPENDR(*,RN(LV),RPAT(LV));
          CALL COPYU(#,UN(LV));
          %=INN(LV)+IPAT(LV);                  $) }
    <PRED>=><WHERE><V>!<V>
      {     T       T
        ( CALL COPYR(RW(LV),*);
          IW(LV)=%;                            $) }
      {     T       T
        ( CALL SETR(RV(LV),&!!' ');
          CALL APPENDR(RV(LV),RW(LV),RV(LV));
          IV(LV)=IW(LV)+1;
          CALL COPYU(UV(LV),#);
          CVAR(LV)=GETPRAC(%);                 $) }
      {     T
        ( CALL SETR(RV(LV),&!!' ');
          IV(LV)=1;
          CALL COPYU(UV(LV),#);
          CVAR(LV)=GETPRAC(%);                 $) }
      {#    T       T
        ( CALL COPYR(*,RV(LV));
          CALL COPYU(#,UV(LV));
          %=IV(LV);
          &=CVAR(LV);                          $) }
    <WHERE>=><N><PAT>
      {     T       T
        ( CALL SETR(RN(LV),&!!' ');
          INN(LV)=1;                           $) }
      {     T       T
        ( CALL SETR(RPAT(LV),&!!' ');
          CPAT(LV)=RTOC(*);
          IPAT(LV)=1;                          $) }
      {# ( ( CPAT(LV)='HE' ! CPAT(LV)='WO' ) $)  T
        ( CALL APPENDR(*,RN(LV),RPAT(LV));
          %=INN(LV)+IPAT(LV);                  $) }
      {#    T
        ( CALL PRINTR(*);
          CALL PRINTI(%);
          CALL PRINTC(&);                      $) }
  $BYE$
```

図2 構文解析プログラム仕様の例

4. 構文解析プログラム仕様の入力に伴うエラー処理と編集

構文解析プログラム仕様は図1のNLAPGの概略構成図に示される流れに沿って処理される。仕様は、TSS端末またはファイルから入力することができる。NLAPGは、仕様入力の途中で構文規則記述部をはじめにエラーを検出すると、それを修正するためにエディタに制御を移す。目的プログラム生成部での処理に入るまでに、GRAMMARエディタ、NAMEエディタ、FUNCTIONエディタにより対話型で構文解析プログラム仕様の編集を行うことができる。

GRAMMARエディタは、構文解析プログラム仕様にエラーを検出すると起動される。検出されるエラーは全部で14個で、その一部を表1に示す。test, preaction, actionの記述の中のエラーは検出しない。GRAMMARエディタは、仕様記述の先頭からエラーの発生したところまでを編集の対象とし、仕様頭部、変数名定義部、文法規則を単位として編集用バッファに取り出し編集を進める。GRAMMARエディタの実行例を図3に示す。GRAMMARエディタの編集用コマンドとして次のようなものを用意している。
H(Help)、R(Renumber)、BYE(Bye)、EL(Error Line)、BW(Backward)、L(List)、I(Insert)、C(Change)、DEL(Delete)、UP(Up)、D(Down)、S(Save)、END(End)。

エラー番号	エラーを発生した条件
20	文法規則に現われる非終端記号は7文字以内でなければならない。
30	構文規則の左辺は、くと>で括られた非終端記号でなければならない。
50	意味処理記述部は、(と)で括られたPL/Iの論理式または;で区切られた文の並び、あるいはDから構成され、その構成素の数は3でなければならない。
80	デリミター, エンドマークは1文字でなければならない。
110	構文規則は <nonterm> => という形で始まらなければならない。
120	プログラム名は7文字以内でなければならない。

表1 GRAMMARエディタが検出するエラー

BWは、エラーの直接原因となった文字列を修正するコマンドである。図3の実行例では、エラーの原因となった文字列"PATEDITOR"を"PAT"に修正している。UP, Dは、編集の単位を現在編集中の部分の前の部分、後の部分に変更するためのコマンドである。

```
?? ERROR OCCURED ERRNUM   20 ??

*** GRAMMAR EDITOR START ***
  10       <SUBJD=><PND>(?!<TIT>)<PATEDITOR>

      <SUBJD=><PND>(?!<TIT>)<PATEDITOR>
GE: BW
      <SUBJD=><PND>(?!<TIT>)<PATEDITOR>
<PATEDITOR>-->: <PAT>
<PATEDITOR>-->: *END*
  10       <SUBJD=><PND>(?!<TIT>)<PAT>
GE: END

-- SAVE OR END --
SC: SAVE

--- THIS BUFFER IS SAVED ---
*** GRAMMAR EDITOR END ***
```

図3 GRAMMARエディタによる編集例

NAMEエディタでは、仕様前処理部で作成した仕様の内部表現、名前の表に基づいて、変数名と文法規則の編集処理を行う。NAMEエディタの編集用コマンドとして次のようなものを用意している。
MG(Make Grammar)、LC(List Check)、SUBC(Sub-table Check)、NAMC(Name-table Check)、SD(Subname Delete)、ND(Name Delete)、END(End)、BYE(Bye)。
MGは、CRTから文法規則、変数名を追加するためのコマンドであり、SDは、文法規則を削除するためのコマンドである。NDは、変数名を削除するためのコマンドである。

FUNCTIONエディタは、仕様の変数名定義部で関数名として定義された名前の関数がユーザ定義関数ファイルに存在しない時に起動される。FUNCTIONエディタの編集用コマンドとして次のようなものを用意している。
H(Help)、R(Renumber)、BYE(Bye)、L(List)、I(Insert)、C(Change)、DEL(Delete)、CL(Clear Line)、END(End)。
Iによって、ファイルに格納し忘れていた関数を作成することができる。

5. 構文解析プログラムの生成

目的プログラム生成部は、構文解析プログラム仕様に対応するリスト、名前の表、意味処理用の関数を与えられて、構文解析用関数ファイルを参照しながら構文解析プログラムの生成を行う。生成される構文解析プログラムの構文解析法は後戻りのある再帰的下向き構文解析法で、その構文解析部分は、仕様における構文規則とほぼ対応した構造を持つ。構文規則の非終端記号を、ORタイプ、PRタイプ、DUタイプの3種類に分類し、各非終端記号に対してそれぞれのタイプに対応したPL/Iコードを生成する。ORタイプとPRタイプは、構文規則の⇒の左辺に現われる非終端記号に対するもので、<A>⇒<B>|<C>|… の場合 <A> はORタイプ、<G>⇒<H><I>… の場合 <G> はPRタイプである。⇒の右辺にしか現われない非終端記号はDUタイプである。ORタイプ、PRタイプの非終端記号に対しては、その非終端記号に対応する名前の手続きが生成される。構文規則の⇒の右辺に現われる非終端記号(<nonterminal>)と、それに付与されている意味処理記述のうち、<test>、<preaction>から、右のような基本パターンが生成される。<action>は、次の例で示すように遅れて生成される。

```
IF <test> DO;
    <preaction>;
    CALL <nonterminal>( );
END;
```

今、次に示すような構成を持つ文法規則が与えられたとして、対応する目的プログラムのPL/Iコードがどのような構成で生成されるかを、図4を参照しながら簡単に説明する。

```
<NP> ⇒ <ADJ><N> | <N>
        {  T          T          (ADJ.ACT)}
        {(N1.TEST)  (N1.PREA)   (N1.ACT) }
        {(N2.TEST)  (N2.PREA)   (N2.ACT) }
        {#  T          T          (NP.ACT) }
```

非終端記号 <NP> は、ORタイプとして識別されており、しかもこの <NP> は⇒の左辺に現われているので、まず、NPという名前のORタイプのPL/Iの手続きを生成し始め、Ⓐで <ADJ> に対する基本パターンを生成しようとするが、次の非終端記号が <N> であり、"|"までの部分的定義が <ADJ><N> という連結であるため、その定義式が <ADJ><N> であるような <PUN0001> というシステム生成の非終端記号に対応する、PUN0001という名前のPRタイプのPL/Iの手続きを生成し始める。<ADJ>に対して基本パターン⑤、action⑥、<N>に対して基本パターン⑦、action⑧を生成する。ここでⒶに戻り、<PUN0001>には意味処理部が付与されていないものとして①を作成する。"|"の後の <N> に対して②の基本パターン、action③を生成する。最後に④の最終部で <NP> のactionを生成する。

```
NP:PROC( ) RECURSIVE;
  DCL ----
  ----
 ┌IF WHR0011=1 THEN DO;
 │  Ⓐ----
 │ ①[CALL PUN0011( );
 │    WHR0011=STU0099(2);
 │ ③┌IF N2.TEST THEN DO;
 │  └    N2.PREA;
 │       CALL DUN0000(N);
 │  END;
 │    ----
 └END;
 ┌ELSE DO;
 │    ----
 │  ┌IF WHR0011=3 THEN DO;
 │ ④└[ N2.ACT;
 │  END;
 │  NP.ACT;
 │    ----
 └END;
  ----
END NP;
```

```
PUN0001:PROC( ) RECURSIVE;
  DCL ----
  ----
 ┌IF WHR0011<=2 THEN DO;
 │   ----
 │ ⑤┌IF WHR0011=1 THEN DO;
 │  │   ----
 │  │   CALL ADJ( );
 │  END;
 │  ┌ELSE IF WHR0011=2 THEN DO;
 │  │      ⑧[ADJ.ACT;
 │  │    ┌IF N1.TEST THEN DO;
 │  │   ⑦└    N1.PREA;
 │  │      CALL DUN0000(N);
 │  │    END;
 │  END;
 │   ----
 └END;
 ┌ELSE DO;
 │   ----
②│②[N1.ACT;
 │   ----
 └END;
  ----
END PUN0001;
```

図4 生成される構文解析プログラムの概略構成

<N>はDUタイプの非終端記号と考えており、その基本パターンのCALL文のところは、CALL N( );ではなく、CALL DUN0000(N)という手続き呼びだしになっている。手続きDUN0000は辞書引きを行う手続きで、引数としてNを持って呼びだされると、入力文からきりだしてきた文字列について辞書引きを行い、それがNという品詞であるかどうかの判定と活用処理を行う。

実際に、仕様に基づいて生成される構文解析プログラムは、上述したような方式で生成される、文法規則に対応するPL/Iコードの他に、データの宣言部、意味処理用、構文解析用の関数や手続きの宣言部、入力文の前処理部などのPL/Iコードが組み込まれたものとなる。

また、プログラムの実行の流れの制御をスタックを用いて行っており、このスタック操作のためのPL/Iコードも含まれる。図4のWHR0011という変数による判定部分も実行の制御と関連する部分である。

6. 構文解析プログラムの実行例

構文解析プログラムにおける処理の概略構成を図5に示す。前処理部では、入力文中のスペース記号に基づき入力文の分割を行う。構文解析部での構文解析は、入力文を左から右へ読み進みながら、後戻りのある再帰的下向き構文解析法によって行われる。文法規則に対応して生成されるPL/Iコードに沿っての実行過程の詳細な説明は省略する。

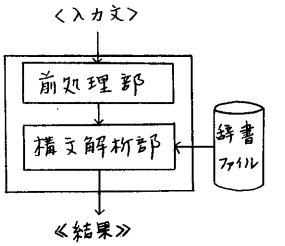

図5 構文解析プログラムの処理の概略

日本語処理向きということを考えて、入力文の自動分割、活用処理を行うための手続きを構文解析プログラムに自動的に付加するようにしている。自動分割の手続きは、辞書引きの手続きから呼びだされ、活用処理の手続きは、自動分割の手続きから呼びだされる。これらの処理手続きにおいても後戻り処理が行われる。

自動分割は、辞書を参照しつつ文字列の左からの最長一致によって行われる。活用処理は、同じく辞書情報を参照しながら、活用する単語の語幹の後に続く活用語尾をきりだすことによって行われる。

図6は、図2の構文解析プログラム仕様から生成された構文解析プログラムの実行例である。入力文は、「花子は走る。」に対応するベタ書きのローマ字の文、「HANAKOWAHASHIRU.」である。辞書は、7.の図7に示すような簡単な構造のものであるが、入力文の分割、活用処理を行っている。①の部分で、品詞列として"PN PAT V"という形に入力文を分割し構文解析に成功し、動詞の活用形は"SHUSHI"(終止形)としている。ここで、強制的に後戻りがおこり、次の解析の可能性を探しに行き、品詞列の分割は①と同じであるが、動詞の活用が"RENTAI"(連体形)の場合も成功としている。「走る」は、終止形と連体形が同じであり、これは活用処理の部分がうまく働いていることを示している。動詞が文末にきて、助動詞、助詞が付属しない場合には、その活用形は連体形であるという処理を意味処理記述部につけ加えることはそうむずかしいことではない。

```
*** INSENTENCE FILE OR CRT  ***
SL: CRT

INS: HANAKOWAHASHIRU.

INPUT SENTENCE IS ACCEPTED !
-- REGISTER PRINT --
PN PAT V
-- INTEGER PRINT --
     3
-- CHARACTER PRINT --
SHUSHI
INPUT SENTENCE IS ACCEPTED !
-- REGISTER PRINT --
PN PAT V
-- INTEGER PRINT --
     3
-- CHARACTER PRINT --
RENTAI
```

①

②

図6 構文解析プログラムの実行例

7. 辞書

NLAPGにおける辞書の構造は次のようなユニット構造である。

- <単語> : <単語>（文字列）
- HIN : <単語>の品詞を入れるスロット。
- HEN : <単語>の活用の有無、活用形を入れるスロット。
- STAT : <単語>の現在の活用の状態を入れるスロット。
- slot1 ⋮ slotn

<単語>をユニット名とすると、3個の意味付けの決まっている特別なスロットと、任意個の意味付けの自由なスロットから構成され、各スロットは長さに制限のない文字列型の入れ物となっている。slot1 以下の構成をどうするかは、ユーザの自由である。

構文解析プログラムにおける辞書情報の抽出は、辞書引きの手続き DUN0000 が成功の状態で戻る時、＊レジスタに入力単語、＆キャラクタにその品詞、＃ユニットに slot1 ～ slotn を入れて戻ることにより行われ、再び DUN0000 が呼びだされて、これらのグローバル変数の内容が更新されない限り、以後の構文解析処理でこれらの情報を用いることができる。

```
*..HANAKO(1)                          *
*                                     *
*                  HIN: PN            *
*                                     *
*                        UID: HAN     *
*AKO                                  *
*                                 PO: *
*F: HITO                              *
*                                     *
*.:HASHI(1)                           *
*                                     *
*                  HIN: V             *
*                                     *
*                        HEN: GODAN   *
* RU                                  *
*                                UID: *
* HASHI                               *
*                                     *
*  POF: HITO                          *
*                                     *
*.KWA(1)                              *
*                                     *
*                  HIN: PAT           *
*                                     *
*                        UID: WA      *
*                                     *
*                                     *
```

図7 VSAMファイル中の辞書の一部

図7に、VSAMファイルに格納されている辞書の例の1部を示す。

8. あとがき

文法規則の記述能力は、他の拡張文脈自由文法に比べてそれほど劣るものではないと思われるが、構文解析プログラムの後戻りのある再帰的下向き構文解析法には、やはり左再帰性の処理が問題として残る。同じような枠組の中で、上向き構文解析法、あるいは、拡張LINGOLのような融合型の解析法による構文解析プログラムの自動生成システムについても考えて行きたい。(3)

構文解析プログラム仕様の意味処理記述部のエラーの検出も含めて、生成されたPL/Iプログラムのデバッグのための良いツールの開発も残された問題の1つである。問題点は残るが、このシステムでかなりの程度の有用な構文解析プログラムを作ることができるか検討中である。なお、システムは HITAC M-200H VOS3 上の PL/I で書かれている。

<謝辞> 本研究に関して、貴重な御助言を頂いた、本学、竹村伸一教授に感謝致します。

9. 参考文献

(1) 田中穂積：計算機による自然言語の意味処理に関する研究
電総研研究報告 第797号 (1979)

(2) Woods, W.A.: Transition Network Grammars for Natural Language Analysis
CACM, Vol.13, No.10 (1970)

(3) 松本裕治，田中穂積：Prologに埋めこまれたbottom-up parser：BUP
情報処理学会自然言語処理研究会資料34-6 (1982)

(4) Levi, J. et.al. : A Programming Methodology in Compiler Construction : Part1 : Concepts.
North-Holland (1979)

〈付録〉図6の実行例のトレース

構文解析プログラムの実行においてどのように構文解析が進んでいるかトレースすることができる。これはTRACEコマンドを入力することによって行うことができる。図8は、図6の実行例をトレースモードで実行した時の出力である。

図6の実行例の構文解析プログラムの仕様は図2であり、この構文規則の部分だけを取りだして書くと次のようになる。

〈STM〉⇒〈SUBJ〉〈PRED〉 ①

〈SUBJ〉⇒〈PN〉(?|〈TIT〉)〈PAT〉 ㊁

〈PRED〉⇒〈WHERE〉〈V〉|〈V〉 ㊂

〈WHERE〉⇒〈N〉〈PAT〉 ㊃

5.で簡単に説明したように、この構文規則に対応して構文解析プログラムには、次のような手続きが生成される。

STM、SUBJ、PRED、WHERE、

DUN0001、PUN0001

DUN0001は㊁の規則から、〈DUN0001〉⇒?|〈TIT〉に対応して、PUN0001は㊂の規則から、〈PUN0001〉⇒〈WHERE〉〈V〉に対応して生成される。この他に、構文解析のトップレベルの手続き、STT0000、辞書引きの手続きDUN0000、分割の手続きDIV0000、活用処理の手続きCNJ0099がある。上の構文規則から生成される構文解析プログラムにおける手続きの呼びだし関係を示すと次のようになる。

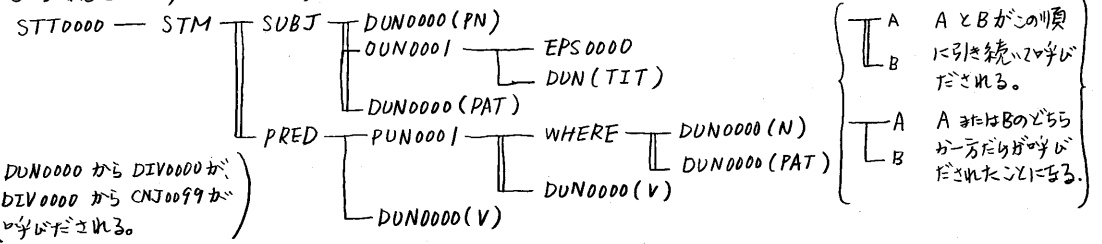

図8の右側の出力情報から、どのような手続きが呼びだされているかを知ることができる。①のところで、"HASHIRU"に対して、〈WHERE〉の〈N〉の解析に行き、失敗し後戻りして、〈PRED〉の"|"の後の〈V〉の解析へ進んでいる。また、一度構文解析に成功した後の強制的後戻り後のトレース情報が②以下に出力されている。