北海道大学工学部研究報告
第119号（昭和59年）

Bulletin of the Faculty of Engineering,
Hokkaido University, No. 119 (1984)

自然言語処理向き構文解析プログラム自動生成システム（NLAPG）

桃内　佳雄　真野　宏之*　宮本　衛市　竹村　伸一

（昭和58年9月30日受理）

An Automatic Parser Generator (NLAPG) for Natural Language Processing

Yoshio MOMOUCHI, Hiroyuki MANO, Eiichi MIYAMOTO and Shin-ichi TAKEMURA

(Received September 30, 1983)

Abstract

An automatic parser generator, NLAPG (Natural Language Analysis Program Generator), generates the PL/I program for parsing of natural language. The specification of the program given to NLAPG is described by the augmented context free grammar based on ATN (Augmented Transition Network). Parsing is done with the recursive top-down parsing algorithm using backtracking.

The PL/I program generated by NLAPG is highly portable and can be easily combined with other PL/I programs. A dictionary for parsing is constructed as VSAM (Virtual Storage Access Method) file. It facilitates the fast access to the words in the dictionary.

1．まえがき

計算機の高速化・小型化が進み，計算機の利用分野および利用者の範囲が拡大してくると，計算機と人間の間の柔軟なインタフェースの実現が一つの重大な課題となってくる。その実現のためにいくつかの方法が考えられるが，中でも，現在，計算機と人間の間の実用的な対話のために用いられている計算機専用の人工言語をわれわれ人間が日常用いている自然言語におきかえるということが可能になるならば，人間にとって計算機はさらに使い易いものとなるであろう。しかし，そのためには，計算機に自然言語の生成と理解のための機構をプログラムあるいは装置として組みこんでやらなければならない。自然言語の理解ということに限って考えるとしても，そのような機構の実現はそれほど簡単なことではない。計算機による高度な自然言語理解システムを実現するためには，計算機科学だけでなく，言語学，心理学，あるいは論理学などの研究成果をとりこみ，さらに計算機科学の立場からの自然言語の構造の解明，および自然言語理解における情報処理過程の解明と具体的実現についての考察が必要となる。

本報告は，語彙解析，構文解析，意味解析，および談話解析などの，自然言語理解システムが行わなければならない処理の中で，特に，構文解析処理の部分に関する考察である。すなわち，構文解析プログラムの仕様を与えることによって，その仕様に基づいて構文解析を行う構文解析プログラムを自動生成するシステムの一つの構成について報告する。

情報工学専攻　情報システム工学講座
*現在，（株）日立製作所

自然言語の構文解析においては，自然言語の多様な現象を処理するために，構文解析の過程で意味情報を利用することが有用である。ATN[1] や拡張 LINGOL[2] などは，このような点を考慮して開発された構文解析システムである。これらのシステムでは，構文解析プログラムの仕様として，文脈自由文法を基盤とし，それに意味情報処理を付加した拡張文脈自由文法を用いており，その文法を入力すると，システムそのものがその文法によって規定される構文解析プログラムとして機能するようになる。このような構成は，自然言語理解システムの他の部分との結合や他のシステムへの移植の容易さなどの点に問題が残る。また，自然言語の構文解析システムでは，語彙に関する情報を蓄積する辞書の作成も重要な課題であり，従来のシステムでは，それを文法の記述に含めて構成することが多かったが，大規模な辞書の作成と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することが望ましい。さらに，自然言語として日本語を対象とする場合には，分かち書きや活用などの処理も行えるようにしなければならない。

本報告におけ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，意味情報処理も含む構文解析プログラムの仕様の記述を行い，それを入力として，構文解析プログラムを生成する。構文解析プログラムは，PL/I プログラムとして出力されるので，他の PL/I プログラムとの結合が容易であり，また，他のシステムへの移植も容易である。辞書は構文解析プログラム仕様の記述とは分離して，ファイルとして作成する。構文解析プログラムは，辞書引きと活用処理の手続きをも組み込んだ形で生成される。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ある。

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，辞書と意味情報処理用の手続や関数をファイルに作成および編集するためのエディタも用意しており，全体として，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を構成している。

2．システムの構成

NLAPG のシステム構成とその環境の概略を図 1 に示す。VSAM エディタと VOS 3・TSS エディタを用いて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すると，構文解析プログラムをその出力として得る。PL/ I プログラムとして生成される構文解析プログラムを 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処理を行う。

ユーザ定義関数ファイルには，構文解析プログラム仕様中の意味情報処理記述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，分かち書き，および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している。辞書ファイルには，語彙に関する各種情報を格納する。辞書ファイルは，生成された構文解析プログラムの実行時に用いられる。

仕様前処理部では，構文解析プログラム仕様のその内部表現への変換や名前の表の作成などを行い，さらに，仕様記述中に構文的なエラーを検出すると，GRAMMAR エディタを起動し，編集を促す。NAME エディタは，仕様の内部表現と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を行い，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基に，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，目的プログラム中に生成される配列型変数の添字の値を監視する。NLAPGは，HITAC M-280H VOS 3 の下で，PL/Iで書かれている。ユーザ定義関数ファイルと辞書ファイルはVSAMファイル，構文解析用関数ファイルは順編成ファイルとして構

成されている。

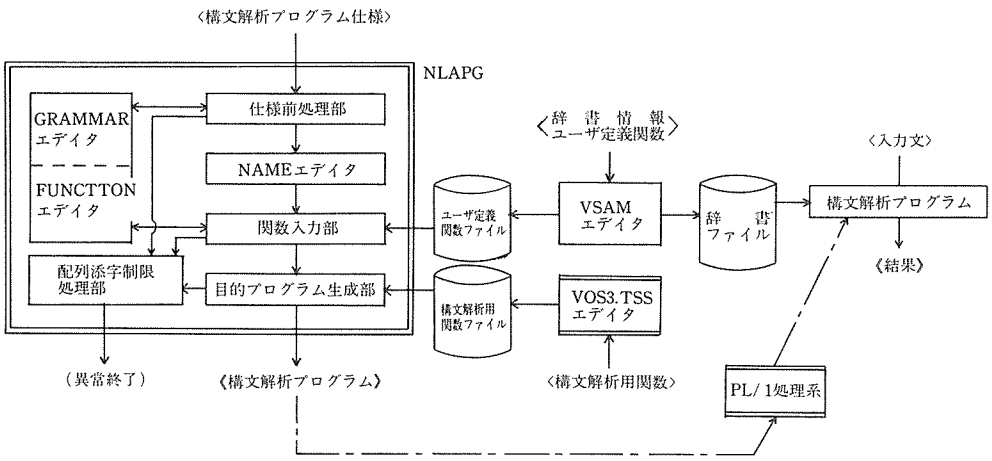


図-1　システム構成

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例を図2に示す。これは，簡単な日本語の文の構文解析プログラムの仕様を与えるものである。仕様のトップレベルは，〈仕様頭部〉と〈変数定義部〉と〈文法記述部〉とから構成される。拡張BNF記法でその構成を示すと次のようになる。

〈構文解析プログラム仕様〉::＝〈仕様頭部〉〈変数定義部〉〈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を記述する。〈プログラム名〉は英字で始まる7文字以内の文字列でなければならない。

〈仕様頭部〉::＝〈プログラム名〉：MAIN.

〈変数定義部〉は，$NAME$行で始め，〈文法記述部〉の意味情報処理の記述中で用いる変数を定義する。

〈変数定義部〉::=$NAME$ [〈型〉：〈変数の並び〉]*.（[X]* はXの0個以上の並び）

変数の型としては，R (Register)，C (Character)，U (Unit)，I (Integer)，F (Function)，D (Delimiter)，E (End mark)，L (Level)，IF (Input File)，およびOF (Output File) がある。

〈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情報処理を記述する部分で，これをATNにおける，test，preaction，および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〉.（[X]+ はXの1個以上の並び）

〈文法規則〉 ::＝〈構文規則〉〈意味情報処理記述部〉.

〈文法規則の一例の骨組を次に示す。

〈STM〉 ⇒ 〈SUBJ〉〈PRED〉　　：構文規則
{〈test-1〉〈preaction-1〉〈action-1〉}：〈SUBJ〉に対する意味情報処理記述
{〈test-2〉〈preaction-2〉〈action-2〉}：〈PRED〉に対する意味情報処理記述
{# 〈test-3〉　T　〈action-3〉}：〈STM〉に対する意味情報処理記述

```
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: .
  $GRAM$
    <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(*,RSUB(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
        ( 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      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      T
        ( CALL PRINTR(*);
          CALL PRINTI(%);
          CALL PRINTC(&);                      $) }
  $BYE$
```

図-2　構文解析プログラム仕様例

〈構文規則〉では，〝〈〟と〝〉〟で括られた文字列は非終端記号を意味し，さらに，〝⇒〟（この記号の左辺が右辺により定義される）や〝！〟（〝あるいは〟を意味する）などのメタ記号を用いることができる。

〈意味情報処理記述部〉では，構文規則の各非終端記号に付随する意味情報処理の記述を行う。testとpreactionとactionはATNにおけるものと同じ意味を持つ。testにはPL/Iにおける論理式を，preactionとactionにはPL/Iにおける文の並び（〝；〟で区切る）を〝(〟と〝$)〟で括って書くことができる。そこで用いられる変数は一部を除いて〈変数定義部〉で定義されているものでなければならない。定義せずに用いることのできる特別なグローバル変数として，＊(型R)，&（型C)，#（型U)，および%（型I）がある。また，preactionとactionで用いる関数や手続きは，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。次のような手続きがあらかじめ構文解析用関数ファイルに格納されている。

COPYU：PROC（OUTU，INU）［INU（unit）をOUTU（unit）へ複写する］

COPYR：PROC（OUTR，INR）［INR（register）をOUTR（register）へ複写する］

SETR：PROC（OUTR，INCHAR）［INCHAR（character）をOUTR（register）へ転送する］

APPENDR：PROC(R1，R2，R3)［R1(register)とR2(register)をappendしてR3(register)へ転送する］

PRINTR：PROC（PR）［PR（register）を出力する］

PRINTI：PROC（PI）［PI（integer）を出力する］

PRINTC：PROC（PC）［PC（character）を出力する］

この他にもいくつかの組み込みの手続きおよび関数を用意している。

〈意味情報処理記述部〉の中の〝T〟は，testに対しては〝真（True)〟を，preactionとactionに対しては〝行為（action）なし〟を意味する。また，変数は，構文解析プログラムの実行時においては，構文解析過程のレベルごとに異なる配列として具体化されるので，そのレベルを区別するためのパラメータのために必ずその引数としてレベルを示す変数を持たなければならない。たとえば，〝RSUB（LV)〟の〝LV〟はレベル変数である。

〈特殊処理記述部〉は，文法の開始記号に対して付与するもので，構文解析における最終的な意味情報処理の記述を行う。図2の例では，一番最後の〝{#〟と〝}〟で括られた部分がそれで，構文解析の結果である諸情報の出力の記述を行っている。

〈文法規則〉の最初の構文規則は，必ず開始記号に対するものでなければならない。また，NLAPGでは，文法規則と辞書とを分離して管理することにしているので，構文規則中に終端記号に相当するものが現われてはいけない。ただし，唯一つの終端記号としての例外は，空語を意味する〝？〟である。構文規則中の右辺に現われて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。

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のシステム構成に示される流れに沿って処理される。仕様はTSS端末またファイルから入力する。目的プログラム生成部での処理に入るまでに，GRAMMARエディタとNAMEエディタと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処理を行うことができる。

GRAMMARエディタは，仕様前処理部で構文解析プログラム仕様中に構文的なエラーを検出すると自動的に起動される。検出されるエラーは，非終端記号の文字列長さの制限（7文字以内）

の違反など14種類である。test, preaction, およびactionの記述中のエラーは検出しない。仕様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，変数定義部，または文法規則を単位として編集用バッファに取りだし，処理を進める。GRAMMARエディタの実行例を図3に示す。編集用コマンドとして，H (Help)，R (Renumber)，BYE，EL (Error Line)，BW(Backward)，L(List)，I(Insert)，C (Change)，DEL (Delete)，UP，D (Down)，S (Save)，およびE (End) を用意している。BWは，エラーの直接原因となった文字列を修正するためのコマンドである。図3の実行例では，エラーの原因となった文字列 "PARTICLE" を "PAT" に修正している。UPとDは，それぞれ，編集の単位を現在編集中の単位の前の単位，後の単位に変更するためのコマンドである。

```
      ?? ERROR OCCURED ERRNUM   20 ??
*** GRAMMAR EDITOR START ***
  10       <SUBJ>=><PN>(?!<TIT>)<PARTICLE>
GE: BW

<PARTICLE>->: <PAT>

  10       <SUBJ>=><PN>(?!<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と名前の表に基づいて，変数と文法規則の編集を行う。編集用コマンドとして，MG (Make Grammar)，LC (List Check)，SUBC (Sub-table Check)，NAMC (Name-table Check)，SD (Sub-name Delete)，ND (Name Delete)，END，およびBYEを用意している。MGはCRTから文法規則や変数を追加するためのコマンドである。SDは文法規則を削除するためのコマンドで，NDは変数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。編集用コマンドとして，H(Help)，R(Renumber)，BYE，L (List)，C (Change)，DEL (Delete)，CL (Clear Line)，およびENDを用意している。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の内部表現，名前の表，および意味情報処理用関数を与えられて，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にほぼ対応した構造を持つ。構文規則の非終端記号を，ORタイプとPRタイプと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あり，〈A〉⇒〈B〉!〈C〉!…のようにその右辺が"!"で結合されている非終端記号はORタイプ，〈A〉⇒〈B〉〈C〉…のようにその右辺が連結で結合されている非終端記号はPRタイプである。また，"⇒"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タイプと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，DUタイプの非終端記号に対しては，その非終端記号を品詞名として，それをパラメータとして持つ辞書引き手続きの呼びだしのためのPL/Iコードが生成される。また，たとえば次のような構文規則，〈A〉⇒〈B〉(〈C〉!〈D〉)〈E〉は，システムによって生成される新しい非終端記号を用いて，次の2つの構文規則に変換された形でPL/Iコードの生成が行われ

る。〈A〉 ⇒ 〈B〉〈OUN0001〉〈E〉，〈OUN0001〉 ⇒ 〈C〉！〈D〉。ここで，〈OUN0001〉がシステムにより生成された非終端記号で，〈A〉はPRタイプ，〈OUN0001〉はORタイプとして，対応するPL/Ⅰコードが生成される。また，〈A〉 ⇒ 〈B〉〈C〉…とか 〈A〉 ⇒ 〈B〉！〈C〉！…などの構文規則に対応して生成される手続きA中での〈B〉や〈C〉に対応するPL/Ⅰコードの基本的なパターンは次のようなものである。Nontは〈B〉や〈C〉などの非終端記号に，そして，Testと Preactionは非終端記号に付随している意味情報処理記述の中のtestとpreactionに対応するものとする。

```
[testが"T"でない場合]          [testが"T"の場合]
    IF Test DO;
        Preaction;                  Preaction;
        CALL Nont;                  CALL Nont;
        END;
```

すなわち，Testが"真(True)"の場合に，Preactionを実行し，手続きNontを呼びだす。actionに対応するPL/Ⅰコードは，上のPL/Ⅰコードに引き続いて，手続き全体の構成を考えながら適切な場所に埋め込まれる。

次に示すような構成の文法規則に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のかその概略について説明する。

〈A〉 ⇒ 〈B〉〈C〉！〈D〉

{	T	T	(B. ACTION) }
{	(C. TEST)	(C. PREACTION)	(C. ACTION) }
{	(D. TEST)	(D. PREACTION)	(D. ACTION) }
{#	T	T	(A. ACTION) }

まず，この構文規則を次のような2つの構文規則に変換する。〈PUN0001〉はシステムが生成する非終端記号である。

〈A〉 ⇒ 〈PUN0001〉！〈D〉，〈PUN0001〉 ⇒ 〈B〉〈C〉

従って，〈A〉はORタイプ，〈PUN0001〉はPRタイプである。〈B〉と〈C〉はORあるいはPRタイプで，〈D〉はDUタイプであるとする。〈A〉および〈PUN0001〉に対応して生成される手

```
A:PROC( ) RECURSIVE;
  DCL -----
  ---------
  IF WHR0011=1 THEN DO;
  ⑦  -----
  ②  CALL PUN0001;
     -----
  ③ ┌IF D.TEST THEN DO;
    │   D.PREACTION;
    │   CALL DUN0000(D);
    └END;
     -----
  END;
  ELSE DO;
     -----
     IF WHR0011=3 THEN DO;
     ④  D.ACTION;
     END;
  ⑤  A.ACTION;
     -----
     CALL -----;
  END;
  -----
END A;
```

```
PUN0001:PROC( ) RECURSIVE;
  DCL -----
  -----
  IF WHR0011<=2 THEN DO;
     -----
     IF WHR0011=1 THEN DO;
        -----
     ⑥  CALL B;
     END;
     ELSE IF WHR0011=2 THEN DO;
          ⑦  B.ACTION;
          ⑧ ┌IF C.TEST THEN DO;
            │   C.PREACTION;
            │   CALL C;
            └END;
     END;
     -----
  END;
  ELSE DO;
     -----
  ⑦  C.ACTION;
     -----
     CALL -----;
  END;
  -----
END PUN0001;
```

図-4　生成されるPL/Ⅰコードの概略構成例

続きの概略構成を図4に示す。まず，〈A〉に対応する，Aという名前の手続きを生成し始め，①で，〈PUN0001〉に対応する，PUN0001という名前の手続きの作成に取りかかる。〈B〉に対して基本パターン⑥とaction⑦，〈C〉に対して基本パターン⑧とaction⑨を生成し，手続きPUN0001の作成を修了すると①に戻る。〈PUN0001〉には意味情報処理の記述が付与されていないものとして，手続きAから手続きPUN0001の呼びだしである②を生成する。〈D〉に対して基本パターン③とaction④を生成し，最後に⑤で〈A〉に対するactionを生成する。〈D〉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③のCALL文での呼びだしは，CALL D；ではなく，CALL DUN0000 (D)；となっている。手続きDUN0000は，辞書引きを行う手続きで，引数としてDを持って呼びだされると，入力文から切り出してきた文字列について辞書引きを行い，それがDという品詞であるかどうかの判定と活用処理を行う。〈D〉はDUタイプであるから，Dは品詞名として処理されるのである。

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，上述のような形で生成される，文法規則に対応するPL/Iコードの他に，データの宣言部，意味情報処理と構文解析用の関数や手続きの宣言部，および入力文の前処理部などが組み込まれたものとなる。また，後戻りなどのプログラムの実行の制御は，PL/Iの再帰的呼び出し機能とスタックを用いて行っており，そのためのPL/Iコードも生成する。図4のWHR0011という変数による判定部分などがこの実行の制御と関連する部分である。上述のPL/Iコードの生成はこの実行の制御の構造を大きな枠組として行われ，その構造はORタイプとPRタイプとでは異なったものとなる。

6．構文解析プログラムの実行例

生成された構文解析プログラムにおける処理の概略を図5に示す。前処理部では，入力文中の空白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，入力文を左から右に読み進みながら行われる。その実行の制御は，5章でも述べたように，PL/Iにおける手続きの再帰的呼び出し機能やスタック操作などによって行われるが，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についての説明は省略する。

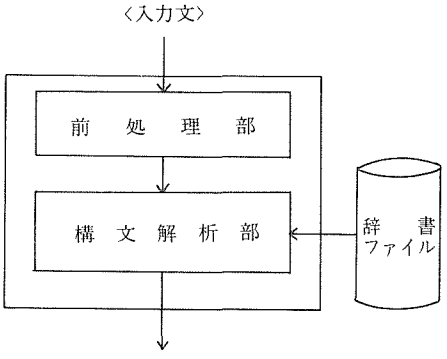


図-5　構文解析プログラムの構成

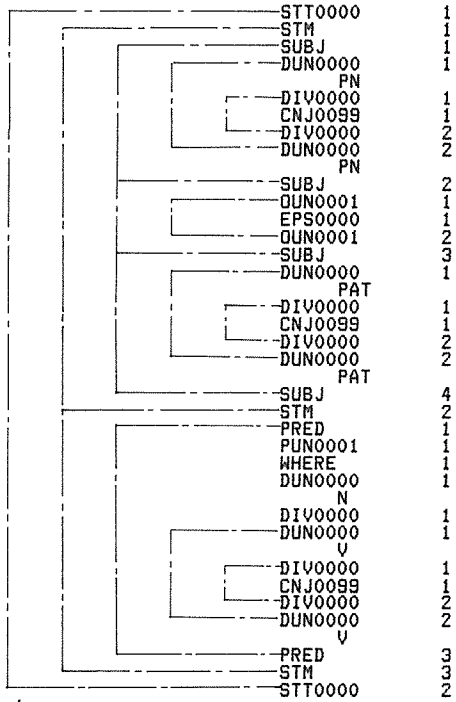
日本語処理向きということを考えて，入力文の自動分かち書きと活用処理を行うための手続きを構文解析プログラムに自動的に付加している。自動分かち書きの手続きは辞書引きの手続きから呼び出され，活用処理の手続きは自動分かち書きの手続きから呼び出される。これらの処理手続きにおいても後戻り処理が行われる。自動分かち書きは，辞書を参照しつつ，文字列の左からの最長一致によって行われる。活用処理は，同じく辞書情報を参照しながら，活用する語の語幹に続く活用語尾を切り出すことによって行われる。活用する語は，辞書にその語幹が格納されているものとしている。

図6は，図2の仕様例に対して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。」に対応するローマ字表記「HANAKOWAHASHIRU.」である。構文解析プログラムにはトレース機能も自動的に組み込まれるようになっており，この実行例では，そのトレース機能をコマンド〝TRACE〟を入力することにより働らかせている。このトレース機能により，構文

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

INS: TRACE

                                                  ----- TRACE MODE -----
INS: HANAKOWAHASHIRU.
```




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

図-6　構文解析プログラムの実行例

解析プログラムの実行過程の概略を追跡することができる。構文解析の結果，この入力文は正しい文であるとして受理し（"INPUT SENTENCE IS ACCEPTED!"），品詞列として「固有名詞・助詞・動詞」（"PN PAT V"）という形に入力文を分割し，動詞の活用形は終止形（"SHUSHI"）であると出力している。入力文は，CRT あるいはファイルから入力することができるが，この例では CRT から入力している。

トレース出力情報に関連して説明を加える。まず，この実行例の構文解析プログラム仕様は図2であり，この仕様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〉　　㋑

〈SUBJ〉 ⇒ 〈PN〉 (?！〈TIT〉) 〈PAT〉　　㋺

〈PRED〉 ⇒ 〈WHERE〉 〈V〉！〈V〉　　㋩

〈WHERE〉 ⇒ 〈N〉 〈PAT〉　　㋥

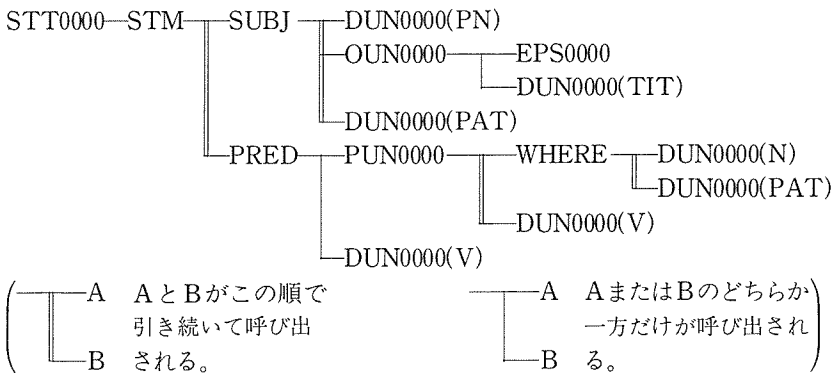
この構文規則に対応して，構文解析プログラムには，STM, SUBJ, PRED, WHERE, OUN0001, および PUN0001 という名前の手続きが生成される。OUN0001 と PUN0001 は，㋺と㋩を次のよ

うに変換する過程で生成される非終端記号に対応している。

㋥ { ⟨SUBJ⟩ ⇒ ⟨PN⟩ ⟨OUN0001⟩ ⟨PAT⟩ ㋭
　　⟨OUN0001⟩ ⇒ ?! ⟨TIT⟩ ㋬

㋦ { ⟨PRED⟩ ⇒ ⟨PUN0001⟩ ! ⟨V⟩ ㋮
　　⟨PUN0001⟩ ⇒ ⟨WHERE⟩ ⟨V⟩ ㋯

この他に，構文解析のトップレベルの手続き STT0000，自動分かち書きの手続き DIV0000，活用処理の手続き CNJ0099，および辞書引きの手続き DUN0000 が含まれる。これらの手続きの呼び出し関係を示すと次のようになる。なお，EPS0000 は空語 (?) の処理を行う。



DUN0000 から DIV0000 が，DIV0000 から CNJ0099 が呼び出される。図 6 のトレース出力は，これらの手続きの呼び出しを出力しており，どの手続きが呼び出されて処理が進められているかが分かる。図中，一点鎖線部分は，後で計算機からの出力の上に書き加えたものであるが，文法規則による構文構造と手続き呼び出しの間の関係を判り易く示すためのものである。

7．辞　書

NLAPG における辞書は，次のようなユニットと呼ばれるフレーム構造の集合として構成する。

⟨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
slotl ：
　…　　} その他の各種情報を入れるスロット
slotn ：

⟨単語⟩ をユニット名として，このユニットは，3 個の意味づけの決っている特別なスロットと任意個の意味づ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。slot 1 以下の構成をどのようにするかは辞書作成者の自由である。

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，＊（register）に入力語，＆（character）にその品詞，＃（unit）に slot 1～slot n を入れて戻ることにより行われ，再び DUN0000 が呼び出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。

また，辞書はVSAMエディタを用いて，VSAMファイルとして作成し，辞書引き処理の高速化を図っている。

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，左再帰性の処理が問題として残る。NLAPGでは，文法規則の意味情報処理記述の中に左再帰性の処理の記述を行うよう要求することになる。今後，同様の枠組の中で，上向き構文解析法あるいは拡張LINGOLのような融合型の構文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ゆきたい[3,4]。

構文解析プログラム仕様中の意味情報処理記述部のエラーの検出も含めて，生成された構文解析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一つである。

現在，日本語の部分的な文法の記述とその構文解析プログラムの自動生成を進めており，NLAPGは柔軟な自然言語理解システムの作成のための一つの有用なツールとして機能することを確かめつつある。

参考文献

1）Woods, W. A. : Communications of ACM, Vol. 13, No. 10, pp. 591-606 (1970).
2）田中穂積：電子技術総合研究所研究報告第797号（1979）.
3）松本裕治，田中穂積：情報処理学会自然言語処理研究会資料 34-6， pp. 33-38 (1982).
4）Winograd, T. : Language as a Cognitive Process, Vol. 1 Syntax, Addison-Wesley (1983).
5）Levi, J. et al.: A Programming Methodology in Compiler Construction : Part 1 : Concepts, North-Holland (1979).