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†

山之上 卓†† 安在弘幸††† 吉田 将†††† 杉尾俊之††††† 武内 惇††††† 椎野 努†††††

NBSG/PD は，日本語をベースにした仕様記述言語 NBSG に対して，そこで抽出された機能要素を処理手続きとして記述するために開発された日本語プログラム言語である．NBSG/PD は，限定された語いを持ち，平易で理解しやすい構文を実現している．本論文は，言語処理系の生成系 MYLANG によって，NBSG/PD プログラムをCプログラムに変換する NBSG/PD プリコンパイラを試作したこと，およびその際行った MYLANG の機能の増強について述べている．過去の言語処理系の生成系の多くは，その入力である言語の構文と意味の記述に，異なる記法を用いていた．これに対して，機能を増強した MYLANG に入力される言語の構文と意味は，同一の記法によって記述される．NBSG/PD プリコンパイラの試作に必要な労力は，3人週間程度であった．

1. まえがき

近年，計算機の日本語処理技術の進歩により，わかりやすいプログラムを作るために，日本語を使用する試みが進められている．しかし，それらの多くは既存の高水準プログラム言語の制御構造をそのまま踏襲しており，日本語を使用しているものの，わかりやすいプログラム言語を設計するという点では，今後さらに改善の余地がある．我々は，設計者が処理手続きを構成する際の思考過程に適合した制御構造を与える表現法を提供し，日本語によるわかりやすいプログラムモジュールの設計を実現するという観点に立って，日本語による手続き記述言語 NBSG/PD[1]* を開発した．NBSG/PD は，限定された語いを持ち，平易で理解しやすい構文を実現している．

Watt の論文[8]によると．「属性文法やアフィックス文法は，コンパイラ記述システムの設計において，現在もっとも有望な道具」である．しかしながら，属性文法を採用している過去のコンパイラ生成システムの多くは，属性文法で用いる意味関数を記述するために，LISP[9]，PASCAL[10),11)] や C[12)] などを用いていた．したがって，これらのコンパイラ生成システムのユーザは，字句解析，構文解析ならびに意味解析などを記述するために，異なった複数の記法を用いることになる．例えば UNIX で使うことのできる言語処理系の生成系として有名な YACC[13)]では，字句解析系生成部 lex の入力は正規表現，構文解析系生成部 YACC の入力は BNF，意味解析部はCでそれぞれ記述される．我々は，一種の属性文法を採用した，言語処理系の生成系 MYLANG を開発している．MYLANG を拡張することによって，上の複数の記法を統一することができる．

我々は言語処理系の生成系 MYLANG によって，NBSG/PD プログラムをCプログラムに変換する NBSG/PD プリコンパイラの試作を行った．これと同時に，MYLANG そのものの拡張も行った．

機能を増強した MYLANG に入力される言語の構文と意味は，一種の属性文法に基づいた，拡張属性付正規翻訳記法 (Extended Attributed Regular Translation Forms: EARTF) と呼ばれる記法によって記述される．MYLANG によって生成される言語処理系は，拡張属性付構文指示翻訳系 (Extended Attributed Syntax-Directed Translators: EASDT) と呼ぶ．これは，ATN (Augmented Transition Networks)[7] と等価な機構であり，下向き処理を行う．

拡張属性付正規翻訳記法によって，NBSG/PD プリコンパイラのすべての部分を定義できた．NBSG/PD プリコンパイラの試作に必要な労力は，3人週間程度

† Experimental Generation of an NBSG/PD Pre-Compiler by the Language Processor Generator MYLANG by TAKASHI YAMANOUE (Interdisciplinary Graduate School of Engineering Science, Kyushu University), HIROYUKI ANZAI (Faculty of Engineering, Kyushu Institute of Technology), SHO YOSHIDA (Establish Preparation Room of Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology), TOSHIYUKI SUGIO, ATSUSHI TAKEUCHI and TSUTOMU SHIINO (Oki Electric Industry, Co. Ltd.).

†† 九州大学大学院総合理工学研究科
††† 九州工業大学工学部情報工学科
†††† 九州工業大学情報工学部創設準備室
††††† 沖電気工業(株)

* NBSG/PD は Nihongo Base Siyo-kijutsu Gengo/Procedure Description の省略形である．

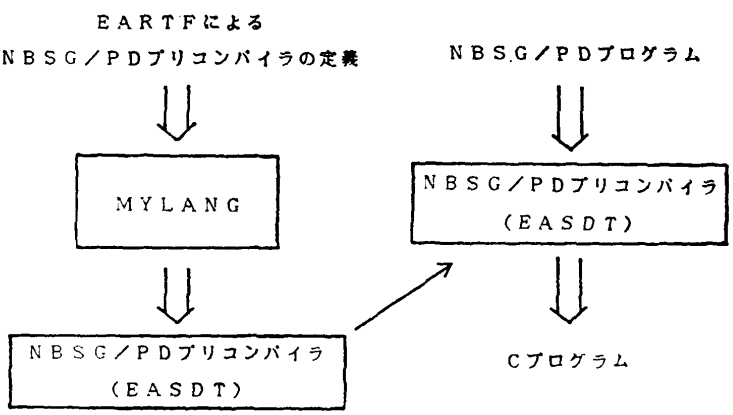


図 1 MYLANG と NBSG/PD プリコンパイラ
Fig. 1 A view of MYLANG and an NBSG/PD pre-compiler.

であった．**図 1** に今回の NBSG/PD プリコンパイラの開発におけるシステム全体の構成を示す．

2 章では NBSG/PD の構文について説明する．3 章では拡張属性付正規翻訳記法について説明する．4 章では NBSG/PD プリコンパイラについて述べる．5 章では，NBSG/PD プリコンパイラの開発過程について述べる．

2. NBSG/PD[1]

ソフトウェアの要求仕様は一般に，非制限的な（構文が規定されていない）自然言語で記述され，その書式も明確に示されてなく，したがって仕様記述のあいまいさ，冗長，矛盾，脱漏を避けるのはかなり難しい．これらの問題を解決するために，仕様の記述方法にある程度の制限を設け，これによって計算機によるサポートを可能とする方法が考えられる．NBSG[2] はそのような方法を実現すべく開発された日本語をベースとする仕様記述言語で，設計ガイダンスおよび静的解析機能などの設計支援機能を備えている．

NBSG/PD は，上述の NBSG によって抽出された機能要素の処理手続きを記述するために開発された日本語プログラム言語である．NBSG/PD は，限定された語いを持ち，平易で理解しやすい構文を実現している．NBSG/PD プログラムの制御構造は，プログラム方法論として著名な Jackson 法[6]の場合と同じように，シーケンス，分岐および繰り返しの三つの基本構造を採用し，それぞれ 1 入口 1 出口のブロック構造で表現される．**図 2** に拡張 BNF による NBSG/PD の文法の一部を示す．以下に NBSG/PD プログラムの制御構造を，正規表現を用いて説明する．

（1） 基 本 表 現

基本表現は，これ以上分割できない代入文や関数呼び出しなどの表現である．基本表現は正規表現における字母系（アルファベット）の要素に相当する．図 2 の非終端記号〈基本表現〉によって基本表現の文法を示す．

（2） シーケンス表現

a_i $(1 \leqq i \leqq n)$ が表現であれば，

$a_1\ a_2\ \cdots\ a_n$

```
<ブロック>=´「´<変数宣言の列>/ <シーケンス表現>´」´ :
<シーケンス表現>=<表現><表現>* :
<表現>=<分岐表現>+<ブロック>+<基本表現>(´。´+<繰り返し表現>) :
<分岐表現>=´ここで.´<のとき対><のとき対>*
                ´その他のとき.´<表現>/
          ´以上。´ :
<のとき対>=<基本表現>´のとき.´<表現> :
<繰り返し表現>=
          (´の間.´+´となるまで.´) ´次のことを繰り返す。´
                    <シーケンス表現>
          ´繰り返し終り。´ :
<基本表現>=<関係式>(<論理演算子><関係式>)* :
<関係式>  =<算術式>(<関係演算子><算術式>)/ :
<算術式>  =<算術項>((´+´ + ´-´)<算術項>)* :
<算術項>  =<算術因子>((´*´ + ´/´)<算術因子>)* :
<算術因子>=<名前>(´=´<算術式>+<実行>)/ + ´(´<基本表現>´)´ :
<実行>    =´(´<引き数の列>/´)´ (を実行する´+´をする´)/ :
```

図 2 拡張 BNF による NBSG/PD の文法の一部
（"=" の代わりに "≔" を，"|" の代わりに "+" をそれぞれ用いる．
"α/" は "λ（空系列）$+\alpha$" を表す．）
Fig. 2 A part of an NBSG/PD grammar defined by extended BNF.
("≔" and "+" are adopted instead of "=" and "|".
"α/" indicates "λ (an empty string) $+\alpha$.")

はシーケンス表現である．これは正規表現

$a_1\ a_2\ \cdots\ a_n$

に相当する．ここで表現とは，基本表現，分岐表現，繰り返し表現またはブロックのいずれかである．分岐表現と繰り返し表現については，次の(3)と(4)にそれぞれ示す．ブロックとは変数宣言の列（省略可）の後ろにシーケンス表現を連結し，これを左括弧「と右括弧」で囲んだものである．図2の非終端記号〈ブロック〉，〈シーケンス表現〉および〈表現〉によってブロック，シーケンス表現および表現の文法をそれぞれ示す．

(3) 分岐表現

a_i $(1 \leqq i \leqq n-1)$ が NBSG/PDの基本表現で，b_j $(1 \leqq j \leqq n)$ が NBSG/PD の表現であれば，

ここで，a_1 のとき，b_1
a_2 のとき，b_2
……
その他のとき，b_n
以上．

は分岐表現である．これは正規表現

$a_1\ b_1 + a_2\ b_2 + \ldots + \text{else}\ b_n$

に相当する．ここで＋は，「または」を意味する．b_n は省略可能である．なお else は$\sim(a_1 \cup a_2 \cup \ldots \cup a_{n-1})$を意味する．図2の非終端記号〈分岐表現〉によって分岐表現の文法を示す．

(4) 繰り返し表現

a が NBSG/PD の基本表現で，b_i $(1 \leqq i \leqq n)$ が NBSG/PD の表現であれば，

aの間，次のことを繰り返す．
$b_1\ b_2\ \ldots\ b_n$
繰り返し終わり．

と

aとなるまで，次のことを繰り返す．
$b_1\ b_2\ \ldots\ b_n$
繰り返し終わり．

は繰り返し表現である．これは正規表現

$(a\ b_1\ b_2\ \ldots\ b_n)^*$

と

$(\sim a\ b_1\ b_2\ \ldots\ b_n)^*$

にそれぞれ相当する．ここで＊は0回以上の繰り返しを意味する．図2の非終端記号〈繰り返し表現〉によって繰り返し表現の文法を示す．

図3はソートを行う NBSG/PD プログラムである．

3. 拡張属性付正規翻訳記法（EARTF）

我々は前報告で属性付翻訳記法 (Attributed Regular Translation Forms: ARTF) を与えた[3),4)]．これは正規翻訳記法 (RTF: 拡張 BNF に出力を表す活動

```
ﾓｼﾞｭｰﾙ main()
「
    INT ボックス ;
    INT 配列ポインタ, 比較ポインタ ;
    STATIC INT 配列[20]={100, 43, 60, 37, 60, 93, 20000, 22,
                         20, 12453, 9870, 2, 77, 221, 546, 37,
                         9999, 42, 38, 711} ;
    配列ポインタ = 0.
    (配列ポインタ != 20)の間，次のことを繰り返す。
    「
        比較ポインタ = 配列ポインタ.
        (比較ポインタ != 20)の間，次のことを繰り返す。
        「
            比較ポインタ = 比較ポインタ + 1.
            ここで，
                配列[配列ポインタ] > 配列[比較ポインタ]のとき，
                「
                    ボックス = 配列[配列ポインタ].
                    配列[配列ポインタ] = 配列[比較ポインタ].
                    配列[比較ポインタ] = ボックス.
                」
                その他のとき，
            以上.
        」
        繰り返し終り.
        書式出力("%d¥n", 配列[配列ポインタ])を実行する。
        配列ポインタ = 配列ポインタ + 1.
    」
    繰り返し終り.
」
//
```

図3 SORT を行う NBSG/PD プログラム
Fig. 3 A sorting program in NBSG/PD.

記号を付加したもの)[5]に以下の拡張を行ったもので，一種の属性文法スキームである．

（1） 非終端記号と活動記号に属性が付加できる．

（2） 活動記号の代わりに，その場所に直接に述語や代入文などを記述できる．属性文法における意味関数は，これらの代入文や述語または活動記号に対応付けられた活動ルーチンによって表す．述語を用いることにより，文脈に依存した言語の認識などを行うことができる．

（3） 各記号の属性生起に局所変数を対応付けることによって，コピールールを省略する．ここでコピールールを一意に定めるため，解釈の上でL-attributedの制限を加える．

図4に簡単な算術式の演算系を定義したARTFの例を示す．ここで，任意の記号列Sに対して，〈s〉，'s'，[s] はそれらがそれぞれ非終端記号，入力記号*および出力記号であることを示す．また，二つの超記号 = と + はそれぞれBNFにおける ::= と | に対応し，同じ役割を果たす．さらに四つの超記号 +，*，(,) はそれぞれ正規表現におけるそれぞれに対応し同じ役割で使われる．超記号 ; は ARTF 定義式の区切りを示す．〈e(/x)〉 や [lookup (n/x)] などの (/x) や (n/x) は，これらの記号が属性を持つことを示している．ここで丸括弧内の x や n は属性生起に対応付けられた局所変数である．丸括弧内の / は相続属性の生起と合成属性の生起を区別する．相続属性の生起は / の左側に，合成属性の生起は / の右側に並べる．属性は属性生起の位置によって識別される．[(x:=x+x1)] や [(x:=x*x 1)] が活動記号の位置に直接に記述された代入文である．〈e(/x)〉，〈t(/x)〉 および 〈f(/x)〉 は算術式，算術項および算術因子をそれぞれ認識してその計算を行い，その結果を x に与えることを表す．〈id (/n)〉 と [lookup (n/x)] は別の場所ですでに定義されていると仮定している．〈id (/n)〉 は名前を認識して，これを n に与えることを表す．[lookup (n/x)] は，名前表を名前 n で探索し，これに格納されている数値を x に与えることを表す．

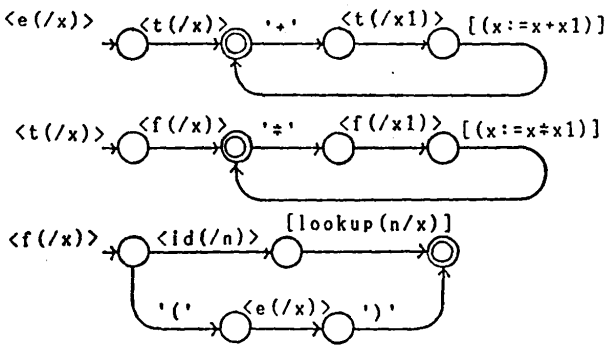


図5 図4から変換されたASDT
Fig. 5 The ASDT which is transformed from Fig. 4.

この例では使われていないが，述語（現在は関係式のみ）は，[?(x=y)] のように記述する．[α] は，α が ?(述語) でかつ述語が真であれば入力記号列として空系列 λ，α が ?(述語) でかつ述語が偽であれば入力記号列として空 ϕ となる．これ以外の [α] は入力記号列として常に λ である．(ω を任意の文字列とすると，$\lambda\omega=\omega\lambda=\omega$，$\phi\omega=\omega\phi=\phi$ である．)

ARTFは，言語処理系の生成系MYLANGによって属性付構文指示翻訳系 (Attributed Syntax-Directed Translators : ASDT) に変換される．ASDTは再帰的状態遷移網に局所変数と変数値引き渡し機構などを加えたものである．**図5**にMYLANGによって図4から変換されたASDTを示す．ここで，図4の一つの定義式が図5の一つのオートマトンに対応している．→○は初期状態，○は状態そして◎は終了状態を表す．非終端記号による状態遷移は，この位置にその非終端記号の名前を持つオートマトンが埋め込まれることを表す．入力記号による状態遷移は，入力文字列の先頭とその入力記号が一致した場合に，状態遷移が行われる．活動記号による状態遷移はその活動記号が指示する動作を行う．なお述語による状態遷移があった場合，これはその述語が真であるときに状態遷移を行う．

Mannaは，自由プログラム図式の同型問題を証明するために，正規表現を用いた[14]．Jacksonは，Jackson法として著名なソフトウェア設計法を提案した[6]．この方法の中で，プログラムの構造は正規表現によって表される．Shawは，flow expressionsを提案した[15]．この表現法は，制御の流れを正規表現で表し，並行プロセスを表すために，Shuffle productを導入

* ARTFの中で記述される 's' は，実際に入力される記号は sであることを示す．

```
<e(/x)> ::= <t(/x)> ( '+' <t(/x1)> [(x:=x+x1)] )* ;
<t(/x)> ::= <f(/x)> ( '*' <f(/x1)> [(x:=x*x1)] )* ;
<f(/x)> ::= <id(/n)>[lookup(n/x)] + '(' <e(/x)> ')' ;
```

図4 簡単な算術式の演算系を定義したARTFの例
Fig. 4 A definition of a simple arithmetic calculator by ARTF.

```
<活動記号>-'['(<標準活動記号>+<述語>+<算術式代入文の列>)']' :
<述語>     -'?('<関係式>')' :
<関係式>   -<算術式><関係演算子><算術式> :
<算術式代入文の列>-'('<算術式代入文>('.'<算術式代入文>)*')' :
<算術式代入文>-<左辺>':='<算術式> :
<左辺>     -<変数名>+'['<算術式>']' :
<算術式>   -<算術項>(('+' + '-')<算術項>)* :
<算術項>   -<算術因子>(('*' + '/')<算術因子>)* :
<算術因子>-<変数名>+<整数>+'#'<定数名>+'''<文字列>'''
           +'['<算術式>']'+'('<算術式>')' :
```

図 6 拡張 BNF による活動記号の文法
(記法は図 1 と同じ.)
Fig. 6 Action symbols grammar defined by extended BNF.
(Notations are as same as in Fig. 1.)

したものである．片山は，属性文法が計算モデルとして有用であることを示した[16]．

ARTF では正規表現に属性文法を導入している．このことによって，ARTF はプログラム図式や計算モデルなどにも利用できる．拡張属性付正規翻訳記法 (EARTF) は，ARTF に RAM (Random Access Memory) を操作する関数の記法を加えたものである．RAM を操作する関数は ARTF において活動記号の位置に以下の形式で直接に記述することができる．

ロード命令：変数名:=[算術式]

ストア命令：[算術式]:=算術式

ここで [算術式] は，RAM において算術式の値が示す番地の内容を表す．

拡張された MYLANG は，定数マクロの機能も用意している．EARTF の前に

名前 整数;

と書いて定数の定義を行う．EARTF 内の算術式に出現した算術因子

名前

は，定義された整数で置き換わる．また，算術因子

'文字列'

はこの文字列と 1 対 1 に対応する整数に置き換わる．この整数は文字列が格納されている RAM の番地を表す．

図 6 に拡張 BNF による活動記号の文法を示す．非終端記号〈活動記号〉は，活動記号が標準活動記号，述語または算術式代入文の列を，左ブラケット [と右ブラケット] で囲んだものであることを示している．標準活動記号については，後で説明を行う．〈述語〉は，述語が ?(関係式) として表されることを示している．〈関係式〉は，関係式が関係演算子を算術式で囲んだものであることを示している．〈算術式代入文の列〉は，算術式代入文の列が一つの算術式代入文か，または複数の算術式代入文をカンマで区切ったものであることを示している．〈算術式代入文〉は，算術式代入文が := を左辺と算術式で囲んだものであることを示している．〈左辺〉は，左辺が変数名または [算術式] であることを示している．〈算術因子〉は，算術因子が変数名，整数，#定数名，'文字列'，[算術式] または (算術式) のどれかであることを示している．

図 7 は基本的なリスト処理関数を EARTF で定義したものである．ここで，RAM 上のアトムとリストの配置は**図 8** のようになっている．〈cons($x, y/z$)〉は，x と y の内容を新しいセルの A と D にそれぞれ代入し，このセルが格納されている番地を z に与え，その後 #list (2000) 番地の内容に 2 を加えることによって，次の新しいセルを準備することを表している．〈car (x/x)〉は，まず相続属性生起に対応した変数 x の内容が，セルへのポインタ（つまりリスト）であることを確認し，次にそのセルの A の内容を合成属性生起に対応した変数 x に与えることを表している．〈cdr (x/x)〉は〈car (x/x)〉における A の代わりに，D を与えることを表している．〈atom ($x/$)〉は x の内容がアトム領域へのポインタであれば λ，そうでなければ ϕ

```
#list 2000: /* list area */

<cons(x,y/z)>-[([[#list]]:-x,[[#list]+1]:-y,z:-[#list],
                [#list]:-[#list]+2)] :
<car(x/x)>-[?(x>#list)][(x:-[x])] :
<cdr(x/x)>-[?(x>#list)][(x:-[x+1])] :
<atom(x/)>-[?(x<#list)] :
<null(x/)>-[?(x='nil')] :
```

図 7 EARTF による基本的なリスト処理関数の定義
(2000 番地の内容は新しいセルへのポインタである．アトムは 2000 番地未満に格納される.)
Fig. 7 Basic list manipulation functions defined by EARTF.
(A content of the address 2000 is the pointer to a new cell. Atoms are stored in memories whose addresses are smaller than 2000.)

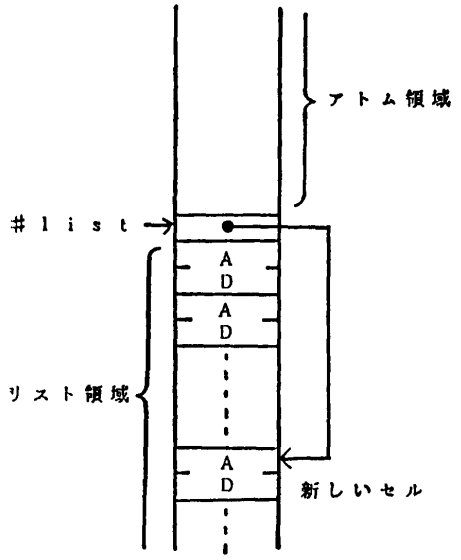


図 8 図7における RAM 上のアトムとリストの配置
Fig. 8 Locations of atoms and lists of Fig. 7 on RAM.

となる．〈null (x/)〉は x の内容がアトム nil であれば λ, そうでなければ ϕ となる．

言語処理系において，リスト処理は重要な役割を果たす．NBSG/PD プリコンパイラの意味解析部は，この基本的なリスト処理関数を組み立てることによって構成されている．

MYLANG は，入力文字列の2文字以上の先読みを行ったり，入力文字列の後戻りを行う機能なども備えている．また，シーケンシャルファイルの操作を行うこともできる．これらの機能は標準活動記号と名付けた活動記号によって呼び出すことができる．標準活動記号は記号 [. α (i/s)] で表す．ここで，α は各機能に付けられた名前である．i は相続属性生起の列，s は合成属性生起の列を示す．以下に各標準活動記号の意味を示す．

[. tch (x/y)]

読み残された入力文字列の，先頭から x 個先の文字を y とする．

[. gch (x/)]

読み残された入力文字列の，先頭から x 個の文字を読み捨てる．

[. gbp (/w)]

読み残された入力文字列を w に記憶する．

[. back (w/)]

w に記憶された文字列を新しい入力文字列とする．この記号と [. gbp (/w)] と組み合わせることによって，ユーザは入力文字列の後戻りを定義できる．

[. open (fid, rw/fn)]

入力または出力ファイル fn をオープンする．fid は，ファイル名，rw は入力または出力の区別を表す．

[. close (fn/)]

ファイル fn をクローズする．

[. cinf (fn/)]

入力文字列をファイル fn の内容とする．

[. coutf (fn/)]

出力をファイル fn に行うよう切り替える．

[. tapeout (x/)]

文字列 x を出力する．

実際には，w は入力バッファへのポインタであり，fid, rw, x は RAM に書かれた文字列の先頭番地である．各文字列の最後は0で終わる．

4. NBSG/PD プリコンパイラ

NBSG/PD プリコンパイラは，NBSG/PD プログラムをCプログラムに変換する翻訳系であり，字句解析部と構文意味解析部の二つの部分からなる2パスコンパイラである．4.1 節では字句解析部について，4.2 節では構文意味解析部についてそれぞれ説明する．

4.1 字句解析部

NBSG/PD プリコンパイラの字句解析部は，NBSG/PD プログラムを中間プログラムに変換する．この字句解析部は単に予約語と他の名前を区別しているだけである．予約語はそのまま出力される．他の名前は，n (名前) の形に変換されて出力される．**図9**は EARTF によって定義された字句解析部の一部である．非終端記号〈lex〉は字句解析部の主ルーチンを定義している．〈inilex〉は初期化を定義している．〈tokens〉は字句解析を定義している．〈wln〉は改行の出力を行う．〈neof (b/)〉は，入力ファイルが終わりでなければ λ, 終わりになれば ϕ となる．〈read (/a,b)〉は予約語が現れるまで入力ファイルを読み込む．ここで a は予約語でない名前を表し，b は予約語を表す．〈wtokena (a/)〉は，予約語でない名前を n (名前) の形にして出力する．ここで a は空文字列であればなにも出力しない．〈tapeoutb (b/)〉は予約語 b をそのまま出力する．〈inistr (#w/)〉と〈inistr (#b/)〉は，文字列 #w と #b を初期化する．#w は予約語でない名前を表し，#b は予約語を表す．〈strconst〉は，文字

```
<lex>-<inilex><tokens><wln><wln>  :

<tokens>-[(b:-#eof)]  ( <neof(b/)><read(/a.b)><wtokena(a/)><tapeoutb(b/)> )* :

<read(/#w.#b)>-<inistr(#w/)><inistr(#b/)> (<strconst>+else<nx>*) <re> :

<nx>-[.tcb(0/x)]<nresc><chrcat(#w.x/)>[.gcb(1/)] :

<nresc>-[.gbp(/x)] (<resc>[.back(x/)][?(1-2)])/ :

<re>-<cln><resc><con>[.nametape(#b/)] :

<resc>-<symx>+else(<kstr>+<astr>) :
<symx>-<symx1>+else<symx2> :
<symx1>-<symx12>+<symx13>+<symx14>+<symx15> :
    <symx12>-' ' + '.' + '.' + ',' :
    <symx13>-':' + ':' + ''' + '"' :
```

図 9 EARTF によって定義された字句解析部の一部
Fig. 9 A part of the lexical analyzer which is defined by EARTF.

```
ﾓｼﾞｭｰﾙ n(main)()「    INT  n(ボックス) :
    INT  n(配列ポインタ), n(比較ポインタ) :
    STATIC INT  n(配列)[20]-{100.43.60.37.60.93.20000.22.
    20.12453.9870.2.77.221.546.37.9999.42.38.711} :
    n(配列ポインタ) - 0.
    (n(配列ポインタ)  !- 20)の間，次のことを繰り返す。
    「      n(比較ポインタ) - n(配列ポインタ).
        (n(比較ポインタ) !- 20)の間，次のことを繰り返す。
        「      n(比較ポインタ) - n(比較ポインタ) + 1.
            ここで，          n(配列)[n(配列ポインタ)] > n(配列)[n(
比較ポインタ)]のとき，          「          n(ボックス) - n
(配列)[n(配列ポインタ)].
                    n(配列)[n(配列ポインタ)] - n(配列)[n(比較ポインタ)].
                    n(配列)[n(比較ポインタ)] - n(ボックス).
                」
                その他のとき，          以上.
        」
        繰り返し終り.
        n(書式出力)("%dYn".n(配列)[n(配列ポインタ)])を実行する。
        n(配列ポインタ) - n(配列ポインタ) + 1.
    」
    繰り返し終り.
」
//
```

図 10 字句解析部によって図3から変換された中間プログラム
Fig. 10 An intermediate program which is transformed from Fig. 3.

```
<branch(/x)> -<ここで><ten><notokilis(/x)>
                'その他のとき'<sonota(x/)>
              '以上'<maru> :
    <ここで>-'ここで' '+'ここで':
  <notokilis(/x)>-<notoki(/x)>(<notoki2(/x1)><nconc(x.x1/)>)* :
  <notoki(/x)>-  <eexpg(/a)>'のとき'<ten><stmt(/b)><pea(a.b/x)> :
  <notoki2(/x)>- <notoki(/y)><list('else '.' '/x)><nconc(x.y/)> :
  <sonota(x/)>-  <ten>
      (<stmt(/y)><cons('else '.y/z)><nconc(x.z/)>)/ :
  <pea(x.y/z)>-<sand('if('.x.')'/z)><nconc(z.y/)> :

  <sand(a.x.b/x)>-
      <cons(a.x/x)><cons(b.'nil'/b)><nconc(x.b/)> :
```

図 11 EARTFによって定義された構文意味解析部の一部
Fig. 11 A part of the syntax-semantics analyzer which is defined by EARTF.

列定数を認識し，これを出力する．〈nx〉は予約語でない名前の1文字を認識し，これを #w の後尾に連結する．〈re〉は予約語を認識し，これを #b に書き込む．〈nresc〉は，まだ読み込んでいない入力ファイルの先頭が，予約語であれば ϕ，そうでなければ λ となる．〈resc〉は予約語を認識する．

図10 はこの字句解析部によって図3から変換された中間プログラムである．なお字句解析部全体の行数は124行である．

```
main(){int n50;
int n52,n54;
static int n56[20]={100,43,60,37,60,93,20000,22,20,12453,9870,2,77,221,546,37,99
99,42,38,711}
;
n52=0;
while((n52!=20)){{n54=n52;
while((n54!=20)){{n54=n54+1;
if(n56[n52]>n56[n54]){n50=n56[n52];
n56[n52]=n56[n54];
n56[n54]=n50;
}
}
}
printf("%d¥n",n56[n52]);
n52=n52+1;
}
}
}
```

図 12 構文意味解析部によって図10から変換されたCプログラム
Fig. 12 The C program which is transformed from Fig. 10.

4.2 構文意味解析部

NBSG/PD プリコンパイラの構文意味解析部は，中間プログラムをCプログラムに変換する．**図 11** はEARTF によって定義された構文意味解析部の一部である．非終端記号〈branch (/x)〉は，NBSG/PDの分岐表現からのCの if 文への変換を定義している．変換された if 文は，リスト構造として x に与えられる．〈ten〉は読点，〈maru〉は句点の認識を行う．〈notokilis (/x)〉は，図2の〈分岐表現〉の定義の一部である〈のとき対〉〈のとき対〉*の部分から，これに対応したCの if 文の列への変換を定義している．〈notoki (/x)〉は，図2の〈のとき対〉の部分から，これに対応したCの if 文への変換を定義している．〈notoki 2 (/x)〉は2回目以降の〈のとき対〉の出現に対する処理を定義している．〈sonota (/x)〉は予約語「その他のとき」の後に出現する表現の処理を定義している．〈pea (x, y/z)〉は，if (x) y を作りこれを z に与えることを定義している．ここで x はNBSG/PDの基本表現をCの式に変換したものであり，y はNBSG/PDの表現をCの文に変換したものである．〈sand (a, x, b/x)〉は x をaとbで囲み，これを x に与えることを定義している．〈list (x, y/z)〉はLISPの関数 (LIST X Y) と同じ機能を持つ．LISPの関数値に対して，z に結果が与えられる．〈nconc (x, y/)〉はLISPの関数 (NCONC XY) と同じ機能を持つ．〈eexpg (/a)〉はNBSG/PDの基本表現を認識し，これをCの関数に変換し，リスト構造としてaに与える．〈stmt (/b)〉はNBSG/PDの表現を認識し，これをCの文に変換し，リスト構造としてbに与える．**図 12** はこの構文意味解析部によって図10から変換されたCプログラムである．なお構文意味解析部全体の行数は263行である．

5. NBSG/PD プリコンパイラ開発過程

NBSG/PD プリコンパイラは，1984年の春から，以下のような手順をへて，現在まで開発を続けている．

(1) 九州大学大型計算機上の MYLANG によって，強い制限を加えた，簡単な NBSG/PD コンパイラを作成した．同時に，NBSG/PD の構文の見直しも行った[17]．ここでは，NBSG/PD コンパイラの意味関数の一部として，オートマトン合成関数を使用した．プログラムの制御構造を表すシーケンス表現，分岐表現および繰り返し表現などは，この関数によってオートマトンに変換された．この簡単な NBSG/PD コンパイラの開発は，1984年5月に，2週間程度の時間をかけて行われた．

(2) MYLANG をパーソナルコンピュータ if 800 model 50 上に移植し，この環境上で NBSG/PD プリコンパイラの試作を行った[18]．NBSG/PD 構文の再見直しや MYLANG 自体の拡張も同時に行った．MYLANG の移植から NBSG/PD プリコンパイラ試作品の稼働までを，1985年4月から6月までの3か月で行う計画を立て，この計画期間内に試作品が稼働した．この中で，NBSG/PD プリコンパイラの字句解析部の開発に1週間，構文解析意味解析部の開発に2週間程度の時間がかかった．意味関数として用いるリスト処理関数をテストするために，LISP インタプリタの開発も行った．これは2日間で終了した．

(3) 以後，NBSG/PD プリコンパイラと MYLANG のデバッグおよびバージョンアップを逐次行

っている．1回のバージョンアップに必要な時間は1日～1週間である．MYLANG のバージョンアップは，ブートストラップの技法を用いている．言語仕様の設計を除く，NBSG/PD プリコンパイラの試作および MYLANG の拡張は，すべて1人で行われた．

6. あとがき

本稿では NBSG/PD プリコンパイラの MYLANG による試作について述べた．意味関数を含む NBSG/PD プリコンパイラのすべての部分が，MYLANG の入力である EARTF によって定義できた．

試作された NBSG/PD プリコンパイラおよび MYLANG はパーソナルコンピュータ if 800 model 50/60 上にインプリメントされている．

MYLANG によって出力される EASDT は，局所変数や RAM をもっており，副作用が生じる．また EASDT は，インタプリタによって解釈実行されているため実行速度に問題がある．これらの問題を解決するために，現在 MYLANG のバージョンアップを行っている．またこれと並行して，NBSG/PD プリコンパイラの拡張も行っている．

謝辞 MYLANG の開発において多大な援助をいただいた九州工業大学の修論生，卒論生や関係者各位，ならびにいつも貴重な御意見をいただく筑波大学佐々政孝先生に感謝します．

参考文献

1) 椎野，武内，杉尾：日本語をベースにした仕様記述言語 NBSG における手続記述について，情報処理学会ソフトウェア工学研究会資料，34-13 (1984).
2) 椎野，武内，杉尾：日本語をベースにした仕様記述言語 NBSG，情報処理学会ソフトウェア工学研究会資料，30-2 (1983).
3) 山之上，安在：属性付構文指示翻訳系の生成系 MYLANG，情報処理学会論文誌，Vol. 26, No. 1, pp. 195-204 (1985).
4) 安在，山之上：言語処理系の生成系 MYLANG の基礎概念，電子通信学会論文誌 (D), Vol. J 69-D, No. 2, pp. 117-127 (1986).
5) 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 (D), Vol. J 63-D, No. 9, pp. 771-778 (1980).
6) Jackson, M. A.: *Principles of Program Design*, Academic Press, New York (1975).
7) Woods, W. A.: Transition Network Grammars for Natural Language Analysis, *CACM*, Vol. 13, No. 10, pp. 591-606 (1970).
8) Watt, D. A.: Rule Selitting and Attribute Directed Parsing, in *Semantic-Directed Compiler Generation, Lecture Notes in Computer Science*, Vol. 94, pp. 363-392, Springer, Berlin (1980).
9) 松田裕幸：拡張属性文法に基づく言語記述言語 LEAG の概要とその応用，情報処理学会ソフトウェア基礎論研究会資料，10-2 (1984).
10) Lewi, J. et al.: *A Programming Methodology in Compiler Construction*, Part 2, *Implementation*, North-Holland, Amsterdam (1982).
11) 高山，安在：翻訳系の生成系の自己生成について，情報処理学会プログラミング言語研究会資料，1-1 (1985).
12) 石塚，佐々：属性文法によるコンパイラ生成系，情報処理学会第 26 回プログラミング・シンポジウム報告集，pp. 69-80 (1985).
13) Johnson, S. C.: YACC-Yet Another Compiler-Compiler, CSTR 32, Bell Laboratories (1975).
14) Manna, Z.: Program Schema, in Aho (ed.), *Currents in the Theory of Computing*, Prentice Hall, Englewood Cliffs (1973).
15) Shaw, A. C.: Software Specification Languages Based on Regular Expressions, Technical Report, ETH Zurich (1979).
16) 片山卓也：属性文法型計算モデル，情報処理，Vol. 24, No. 2, pp. 147-155 (1983).
17) 山之上，安在，武内，椎野：NBSG のコード生成について，第 29 回情報処理学会全国大会論文集，1 R-9 (1984).
18) 山之上，安在，杉尾，武内，椎野：言語処理系の生成系 MYLANG を用いた NBSG/PD コンパイラの開発，第 31 回情報処理学会全国大会論文集，4 E-3 (1985).

(昭和61年6月2日受付)
(昭和61年10月8日採録)

山之上 卓（正会員）

昭和34年生．昭和55年呉工業高等専門学校電気工学科卒業．昭和57年九州工業大学情報工学科卒業．昭和59年同大学院修士課程修了．現在九州大学大学院博士課程総合理工学研究科在学中．言語処理系の生成系，自然言語処理，人工知能などの研究に従事．電子通信学会，日本ソフトウェア科学会，ACM, IEEE 各会員．

安在 弘幸（正会員）

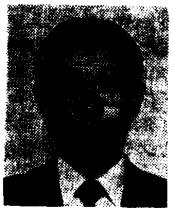
昭和7年生．昭和39年九州大学工学部電子工学科卒業．昭和41年同大学院修士課程修了．同年九州産業大学工学部講師，昭和42年九州大学中央計数施設講師，昭和47年九州工業大学情報工学科助教授，現在同学科教授．工学博士．オートマトン理論および言語処理系の生成系の研究に従事．電子通信学会，日本ソフトウェア科学会，人工知能学会，ACM 各会員．

吉田 将（正会員）

昭和8年生．昭和33年九州工業大学電気工学科卒業．昭和35年九州大学大学院工学研究科修士課程修了．工学博士．昭和37年九州大学工学部講師，その後，九州工業大学教授，九州大学工学部教授を経て，昭和61年10月九州工業大学情報工学部長．この間，九州工業大学および九州大学情報処理教育センタ長，九州大学大型計算機センタ長を歴任．機械翻訳，自然言語処理，人工知能などの研究に従事．電子通信学会，日本認知科学会，米国 ACL 各会員．

杉尾 俊之（正会員）

昭和35年生．昭和57年熊本大学工学部電子工学科卒業．同年沖電気工業(株)入社．現在，同社研究開発本部総合システム研究所オフィスシステム研究部勤務．ソフトウェア開発支援エキスパートシステムの研究開発に従事．特にソフトウェア生産技術，ソフトウェア設計自動化システムに興味をもつ．電子通信学会，IEEE 各会員．



武内 惇

昭和21年生．昭和45年日本大学理工学部電気工学科卒業．昭和47年北海道大学大学院工学研究科電子工学専攻修士修了．同年沖電気工業(株)入社．現在情報処理事業本部ソフトウェアセンタ生産技術開発部勤務．ソフトウェア開発支援システムの開発に従事．特にソフトウェア設計支援システムに興味をもつ．電子通信学会，人工知能学会，ACM，IEEE 各会員．

椎野 努（正会員）

昭和16年生．昭和39年名古屋大学工学部電気工学科卒業．同年沖電気工業(株)入社．マイクロ波通信システムの研究，無線伝送システム自動設計の研究，データ通信方式の研究，ソフトウェア CAD，仕様記述言語等の研究を経て，現在 AI ワークステーション，自然言語処理，エキスパートシステム等，人工知能関連の研究開発に従事．工学博士．電子通信学会，人工知能学会，IEEE，日本心理学会各会員．