Vol. 24 No. 4 情報処理学会論文誌 Jul. 1983

先読みと予測機能をもつ述語論理型構文解析プログラム：PAMPS†

上原邦昭†† 豊田順一†††

本報告は，述語論理に基づく自然言語の構文解析プログラム PAMPS について述べたものである．述語論理型構文解析プログラムは，文法の明晰性，モジュラ性をもつこと，文法と解析アルゴリズムを完全に分離できることなどの特徴をもっている．PAMPS の文法は，書換え規則の非終端記号に引数を与えた拡張文脈自由文法として定義される．引数は文脈情報や解析する文の深層構造を表現するために用いられ，これらの情報は書換え規則間でパターンマッチングにより引数を通じて互いに受け渡される．PAMPS の構文解析アルゴリズムは上昇並行型で，“予測情報”と，1語先読みすることによって得られる“先読み情報”とを利用しながら解析するものである．そのため無駄な解析を極力避けることができ，効率の良いアルゴリズムとなっている．また規則の適用条件や適用時の動作手続きを，書換え規則内に自然な形で埋め込むために，局所 Horn 構成という概念を導入している．局所 Horn 構成内での手続きは，プログラム言語 Prolog と同様に，Horn 節集合として定義され，下降縦型探索によって実行される．したがって PAMPS は構文解析プログラムと Prolog を融合したものと考えることができる．このほか PAMPS による構文解析例，および他の述語論理型構文解析プログラム DCG との比較などについても言及している．

1. まえがき

計算機による自然言語処理において，述語計算の定理証明問題を構文解析に応用する研究が，最近注目を浴びている[4),11)~13),15),16)]．これは，Kowalski[8),9)]らによって提案されたもので，扱う述語論理式をある種の制限された形（Horn 節[10)]という）に限定すると，文脈自由文法 (CFG) による構文解析と，Horn 節による定理証明が密接に関連していることに基づいている．たとえば CFG の書換え規則は Horn 節に，構文解析アルゴリズムは定理証明戦略に対応している．さらに定理証明におけるパターンマッチング (unification[10)]) により，CFG に欠けている文脈依存性を導入することができる．

Warren らは，彼らの提案に従いプログラム言語 Prolog[3)] で述語論理型構文解析プログラム DCG (Definite Clause Grammar)[12)] を実現した．DCG は文法全体の見通しがよいこと，文法の記述と文法を適用するアルゴリズムが完全に分離していることなどの利点をもっている．しかし，DCG の構文解析は下降逐次型で行われ，解析結果の優先順位付けが困難なこと，また入力文の長さに対して処理時間が指数関数的に増大することなど，検討すべき課題もいくつか残されている．

著者らは，解析過程で得られる解析情報を有効に利用し，無駄な解析をできるだけ避け，効率よく実行するアルゴリズムを設定すること，構文解析に必要な規則適用条件や適用時の動作手続きを，文法内に自然な形で埋め込むことの2点を目標として，新しい述語論理型構文解析プログラム PAMPS (Pattern Matching Parsing System) を開発した[19)]．

本報告では，PAMPS の文法形式，構文解析アルゴリズム，および構文解析例について述べる．PAMPS の文法は，CFG の非終端記号に引数を与えた拡張 CFG として定義される．書換え規則間では，すべて引数を通じて文脈情報が受け渡される．PAMPS のアルゴリズムは上昇並行型で，次にどのような語が期待できるかという“予測情報”と，先読み語から得られる“先読み情報”とを利用しながら解析するものである．また複雑な手続きを書換え規則内に埋め込むために，局所 Horn 構成という概念を導入する．手続きは Horn 節集合として定義する．さらに DCG との比較，実験結果，および問題点を言及する．

2. PAMPS の構文解析アルゴリズム

本章では，簡単のために CFG を用いて PAMPS のアルゴリズムを説明する．

† Using Lookahead and Predictive Information in a Predicate Logic Oriented Parsing Program: PAMPS by KUNIAKI UEHARA (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University) and JUNICHI TOYODA (I. S. I. R., Osaka University)

†† 大阪大学基礎工学部情報工学科
††† 大阪大学産業科学研究所

2.1 準　備

まず最初に CFG を定義する.

〔定義 1〕 CFG は 4 字組 $\langle V_N, V_T, P, S\rangle$ で表される. V_T および V_N は互いに素な有限集合で, それぞれ終端記号, 非終端記号の集合, S は V_N の元で開始記号である. P は $A\to\alpha(A\in V_N, \alpha\in(V_N\cup V_T)^*)$ なる形の書換え規則の有限集合である. 非終端記号, 終端記号はそれぞれ自然言語の文法カテゴリー, 単語に対応する. また導出, 導出過程, 言語などに関する定義は文献[1] に従うものとする.

PAMPS の書換え規則は, 以下の 2 形式からなる.

1) 文法規則: $A\to\alpha$ $(A\in V_N, \alpha\in V_N^+)$

2) 辞書規則: $A\to a$ $(A\in V_N, a\in V_T)$

なお, このように書換え規則を制限しても, CFG の能力は変わらない[1]. さらに P に $S'\to ST$, $T\to\dashv$ という書換え規則を付け加える. ただし S' は新しい開始記号, $\dashv$ は文の終了を示す終端記号, T は $\dashv$ を導く非終端記号で, S' は P のいかなる書換え規則にも出現しない. また解析したい文を $a_1\cdots a_n\dashv\dashv$ $(a_i\in V_T, 1\le i\le n)$ とする.

構文解析アルゴリズムは通常上昇型 (bottom-up) と下降型 (top-down) の二つに分けられる. 上昇型の欠点は, 入力文に従って可能な書換え規則を無制限に適用して構文木を成長させるため, 最終的な構文木と関係ない無駄な部分構文木を作りがちなことである. この欠点は, 下降型の特徴である. 次にどんな非終端記号が期待できるかという "予測" を利用すれば, ある程度解消できる. またある時点で適用可能な書換え規則が複数個存在しても, その時点で解析中の終端記号に続く記号を前もって "先読み" すれば, 適用可能な書換え規則の数を小さく絞り込むことができる. これら "予測" と "先読み" を PAMPS に導入するために, 以下の関数 first, follow を定義する.

〔定義 2〕 関数 first とは,

$$\text{first}(A)=\{B\mid A\overset{*}{\Rightarrow}B\alpha\ \alpha\in V_N^*\ A,\ B\in V_N\}$$

である.

〔定義 3〕 関数 follow とは,

$$\text{follow}(A)=\{B\mid S'\overset{*}{\Rightarrow}\alpha AB\beta\quad \alpha\in V_N^*\quad A\in V_N\quad B\in V_N\cup\{T\}\quad \text{ただし } B\in V_N \text{ のとき } \beta=\gamma T,\ \gamma\in V_N^*,\ B=T \text{ のとき } \beta=\lambda\ (\lambda \text{ は空系列})\}$$

である. とくに follow $(S')=\{T\}$ とする.

PAMPS は, 解析の途中結果を保存するために, "状態" (state) と呼ぶ解析情報を組み立て, この "状態" を参照しながら解析を進める.

〔定義 4〕 "状態" とは 4 字組 $\langle\beta, i, j, A\rangle$ である. ただし終了を示す終端記号を除く解析文の長さを n とするとき, $0\le i\le j\le n+1$ である. $A\to\alpha\beta$ $(A\in V_N, \alpha\in V_N^+, \beta\in V_N^*)$ を書換え規則とするとき, $\alpha\overset{*}{\Rightarrow}a_{i+1}\cdots a_j$, $S'\overset{*}{\Rightarrow}\gamma A\delta T$, $\gamma\overset{*}{\Rightarrow}a_1\cdots a_i$, $\delta\in V_N^*$ である. また β の先頭非終端記号を予測記号と呼ぶ. とくに $\beta=\lambda$ のとき "状態" をフレーズと, $\beta\neq\lambda$ のときゴールと呼ぶことがある.

最後にバッファの概念を導入する. バッファは 2 個のセルからなり, 先頭のセルには現在解析中の終端記号に対応する辞書規則の左辺非終端記号が, 次のセルにはその直接右隣りにある終端記号 (先読み語と呼ぶ) に対応する非終端記号が入る.

2.2 構文解析アルゴリズム

以上で準備を完了したので, PAMPS のアルゴリズムで説明する. 解析は左から右へ 1 語ずつ上昇並行型で進める. 初期設定としてゴール $\langle$ST, 0, 0, $S'\rangle$ を生成する.

いま解析は a_{j-1} まで進み, 続く終端記号 a_j, a_{j+1} に対して, それぞれ適用可能な辞書規則 $D_1\to a_j$, $D_2\to a_{j+1}$ があるものとする. また以下では英大文字は非終端記号を, ギリシャ文字は空系列を含む非終端記号列を表すものとする.

try-W: 先頭のバッファセル buffer$_1$ には, 解析しようとしている終端記号 a_j に対応する非終端記号 D_1を, 次のバッファセル buffer$_2$ には先読み語 a_{j+1} に対応する非終端記号 D_2 を入れる. すでにゴール $\langle B\gamma, k, j-1, A\rangle$ が設定されており, $D_1\in$first (B) が成り立ち (予測検査), かつ $D_2\in$follow (D_1) が成り立つならば (先読み検査), フレーズ $\langle\lambda, j-1, j, D_1\rangle$ を生成する.

生成されたフレーズ $\langle\lambda, i, j, B\rangle$ のそれぞれについて, 以下の動作をいずれも実行できなくなるまで繰り返す.

try-C: ゴール $\langle BC\alpha, k, i, A\rangle$ がすでに設定されている場合, もし新しく設定されるゴールの予測記号 C から, $C\overset{*}{\Rightarrow}D_2\gamma\Rightarrow a_{j+1}\gamma$ となる導出が可能ならば (先読み検査), ゴール $\langle C\alpha, k, j, A\rangle$ を設定する. これは D_2 が first (C) に含まれるかどうかにより決定できる. もし含まれなければ, このゴールは将来とも予測情報として参照されることはないので, 設定されない.

try-R: ゴール $\langle B, k, i, A\rangle$ がすでに設定されている場合, もし開始記号 S' から $S'\overset{*}{\Rightarrow}\gamma AD_2\delta\Rightarrow$

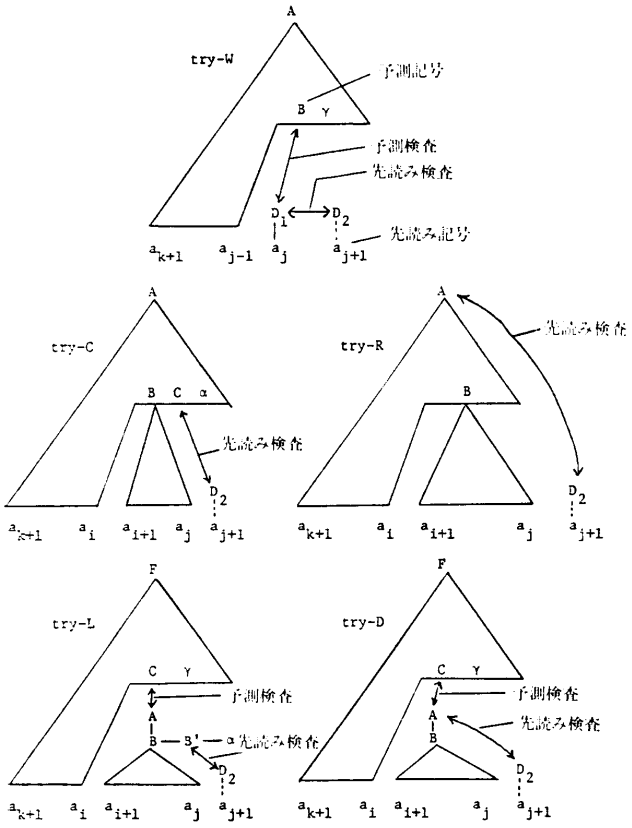

図 1 PAMPS の動作説明図
Fig. 1 Pictorial representation of PAMPS algorithm.

$\gamma Aa_{j+1}\delta$ となる導出が可能ならば（先読み検査），フレーズ $\langle \lambda, k, j, A\rangle$ を生成する．この判定は，D_2 が follow (A) に含まれるかどうかにより決定できる．もし含まれないならば，生成されるフレーズの非終端記号 A が，最終的な構文木の節点として出現する可能性はないので，このフレーズは生成されない．

try-D：ゴール $\langle C\gamma, k, i, F\rangle$ かつ $A\in$first (C) なる規則 $A\to B$ があれば(予測検査)，フレーズ $\langle \lambda, i, j, A\rangle$ を生成する，このとき try-R と同様に D_2 が follow (A) に含まれないならば(先読み検査)，このフレーズは生成されない．

try-L：ゴール $\langle C\gamma, k, i, F\rangle$ かつ $A\in$first (C) となる規則 $A\to BB'\alpha$ があれば(予測検査)，ゴール $\langle B'\alpha, i, j, A\rangle$ を設定する．try-C と同様に D_2 が first (B') に含まれないならば(先読み検査)，このゴールは設定されない．

以上の動作のいずれもが実行できなくなれば，入力を 1 記号進めて try-W にもどる．フレーズ $\langle \lambda, 0, n+1, S'\rangle$ が生成された時点で解析を終了する．

PAMPS のアルゴリズムの正当性の証明は，「$S'\overset{*}{\Rightarrow}a_1\cdots a_n\dashv$ となるのは，"状態" $\langle \lambda, 0, n+1, S'\rangle$ が生成されるときのみである．」という命題を証明することに等しい．これは文献1)での定理 4.9，あるいは文献2)での補題 1，補題 2，および定理 1 と同様に，帰納法を用いて容易に証明できるが，紙面の都合上省略する．

2.3 他のアルゴリズムとの比較

本節では，PAMPS のアルゴリズムの有効性を示すために，従来からよく知られている Earley のアルゴリズム[5]，構文解析プログラム LINGOL のアルゴリズム[14),17)]，Bouckaert らのアルゴリズム[2)] と比較する．これらはいずれも上昇並行型で解析を行い，LINGOL 以外はすべて先読み機能をもつ．また Earley と Bouckaert らのアルゴリズムは任意の CFG を，LINGOL は Chomsky の標準形[1)] と $A\to B$ の形をした単一規則からなる CFG を扱うことができる．それぞれのアルゴリズムに，同一文とその文を解析するための文法を与え，解析中に生成される "状態" の個数によりアルゴリズムを比較する*．なお LINGOL と PAMPS は，任意の CFG を扱うことができない

表 1 他のアルゴリズムとの状態生成数の比較
Table 1 Comparison with other algorithms for the number of generated states.

文法1) A->AA|x

	状態生成数			
入力文	Earley	Bouckaert(M_0')	LINGOL	PAMPS
x	15	7	6	5
xx	28	14	12	8
xxx	47	26	21	14
xxxx	74	44	34	24

文法2) S->AB A->a|SC B->b|DB C->c D->d

adbcddb	39	26	23
$ad^3bcbcd^3bcd^4b$	97	63	56
$adbcd^2bcd^5bcd^3b$	103	66	59
$a(bc)^3d^3(bcd)^2dbcd^4b$	121	87	74
$a(bcd)^2dbcd^3bcb$	81	59	50

文法3) E->T|E+T T->P|T*P P->a
PAMPS: E->T|EAT T->P|TMP P->a A->+ M->*
LINGOL: E->T|EC C->AT T->P|TD D->MP P->a A->+ M->*

a+a*a	71	29	30	18

文法4) K-> |KJ J->F|I F->x I->x
PAMPS, LINGOL: K->KJ|J J->F|I F->x I->x

xx	61	18	15
xxx	84	26	21
xxxx	107	34	27
xxxxx	129	42	33

* LINGOL の場合，ゴールと PPT (部分構文木) の和を状態生成数とする．

ので，適当に文法を変換している場合もある．この状態生成数による比較は，文献 2),5)でも採用されており，アルゴリズムの効率を議論するうえで，十分に妥当なものと考えられる．実験結果を**表1**に示す．

1) Earley のアルゴリズムとの比較

Earley のアルゴリズムは以下に示す predictor 操作を含んでいる．

〈predictor 操作〉 ゴール $\langle B\beta, i, j, A\rangle$ がすでに設定され，かつ書換え規則 $B\rightarrow\gamma$ ($\gamma\in V_N^+$) が存在するならば，ゴール $\langle\gamma, j, j, B\rangle$ を設定する．この操作を新たなゴールが生成されなくなるまで繰り返す．

predictor 操作は，ゴールの予測記号 B と，書換え規則の集合のみに依存している．PAMPS では書換え規則集合が与えられた時点で，predictor 操作によって得られる情報と同等のものを関数 first の形式で前もって求めているため，解析時にはこの操作による“状態”生成を省くことができる．

また Earley のアルゴリズムは，predictor 操作時にあらかじめすべての可能な先読み記号を計算し，先読み記号のそれぞれについてゴールを作り出す．そして書換え規則の適用が終了しフレーズが生成された時点で，その計算された先読み記号と実際の終端記号とを比較し，不用なフレーズを消去する．このような後処理的な先読み機能は，不用な“状態”を生成しやすく，状態生成数という観点からすれば，無駄の多いものである．逆に PAMPS は，先にも述べたように，先読み機能を前処理的に用いるため，不用な状態の生成を抑えることができる．以上の2点のために，PAMPS の状態生成数は Earley のそれに比べて大幅に少なくなっている．

2) LINGOL との比較

LINGOL は先読み機能をもたず，また書換え規則の形式が PAMPS に比べて制限されているために，状態生成数は PAMPS よりも多くなっている．

3) Bouckaert らのアルゴリズムとの比較

彼らは先読み機能の違いによって4種類のアルゴリズム (M_j^i $0\leq i, j\leq 1$) を提案しているが，そのうちで最も状態生成数の少ない M_0^1 と比較する．M_0^1 は Earley のアルゴリズムを改良したもので，PAMPS と同様の先読み機能をもっている．したがって彼らのアルゴリズムも predictor 操作を含み，PAMPS よりも多くの“状態”を生成している．

3. PAMPS の文法

本章では書換え規則の各非終端記号に引数を与えた拡張 CFG を PAMPS の文法として定義する．引数は文の深層構造や文脈情報を表現するためのデータ構造として扱われる．またこれらのデータ構造に対する操作はパターンマッチングによって行われる．さらに拡張 CFG への変更に伴い，2章のアルゴリズムにはパターンマッチング操作を付け加える．

3.1 PAMPS のデータ構造

非終端記号の引数として項〈term〉が許される．項は変数〈variable〉，数〈integer〉，アトム〈atom〉，または複合項〈compound term〉からなる．複合項はたとえば構文解析で得られる文の深層構造など，構造をもつデータを表し，項名〈functor〉とそれに続くいく

```
<term> ::= <functor>(<terms>)
          |<atom>|<integer>|<variable>
<terms> ::= <term>|<term>,<terms>
<functor> ::= <identifier>
<atom> ::= <identifier>
<variable> ::= *<identifier>
```

図 2 項の式的定義
Fig. 2 Formal definition of term.

```
変 数  *number, *x

アトム  sing, plural

数     1, 2596

複合項  .(sing,.(plural,.(mass,nil)))
```

図 3 項の例
Fig. 3 Example of term.

a) 文法規則

```
<grammatical-rule> ::= <lhs> -> <rhs>.
     |<lhs> -> <rhs>;Priority(<integer>).
<lhs> ::= <non-terminal>
<rhs> ::= <non-terminal>,<rhs1>|<non-terminal>
<rhs1> ::= <non-terminal>|{<goals>},<rhs>|<rhs>,{<goals>}
<goals> ::= <literal>,<goals>|<literal>
<literal> ::= <predicate>(<arguments>)
<non-terminal> ::= <non-terminal-symbol>(<arguments>)
<arguments> ::= |<terms>
<predicate> ::= <identifier>
<non-terminal-symbol> ::= <identifier>
```

b) 辞書規則

```
<lexical-rule> ::= <lhs> -> [<terminal>].
     |<lhs> -> [<terminal>];<features>.
     |<lhs> -> [<terminal>];<features>;Priority(<integer>).
<terminal> ::= <identifier>
<features> ::= <feature>|<feature>,<features>
<feature> ::= <feature-type>(<feature-value>)
<feature-type> ::= <identifier>
<feature-value> ::= <term>
```

c) 局所 Horn 節

```
<local-horn-clause> ::= <head>.| <head> <- <body>.
<head> ::= <literal>
<body> ::= <literal>|<literal>,<body>
```

図 4 文法の形式的定義
Fig. 4 Formal definition of PAMPS grammar.

つかの引数からなる．この引数としてまた項が許される．項の定義を**図2**に示す．この定義に従う項の例を**図3**に示す．

3.2 文法の表現

PAMPS の文法は文法規則〈grammatical-rule〉，辞書規則〈lexical-rule〉，局所 Horn 節〈local-Horn-clause〉からなる．

文法規則の定義を**図4** a）に示す．優先順位（priority）はその規則の優先度を示す数値で，文法に曖昧性がある場合に使用する．中カッコで囲まれたゴール列〈goals〉は，局所 Horn 節で定義される手続きを呼び出すために使われる．局所 Horn 節については 3.4 節で述べる．

次に辞書規則の定義を図4 b）に示す．特徴素列〈features〉は，その語のもつ人称，数，性などの統語的情報や，格文法での格などの意味的情報を表すために用いる．特徴素〈feature〉は，特徴素タイプ〈feature-type〉と特徴素値〈feature-value〉からなる．

この定義に従う簡単な文法を**図5**に示す．この文法は冠詞と名詞，主語と述語の数の一致を調べ，文の統語的構造を組み立てるものである．たとえば文法規則(2)は，“SUBJ が構造 $*x$ を，PRED が構造 $*y$ をもち，SUBJ の数 *num と PRED の数 *num が一致するならば．S は構造 $s(*x, *y)$ をもつ.” と読むことができる．5章ではこの文法による解析例を示す．なお変数の有効範囲はその変数の出現する規則内のみで，他の規則で同一変数名が使用されても，それらはまったく別のものとみなされる．

3.3 パターンマッチング

文脈情報や文の深層構造を表現する項は，パターンマッチングにより操作される．パターンマッチングとは，同一名をもつ二つの非終端記号* の引数同士を，それぞれ構造的に照合する操作である．パターンマッチングする二つの引数のうち，1) 少なくとも一方が変数の場合，その変数を他方の項で置換(substitution)する．2) 一方がアトムか数の場合，他方が同一のアトムか数でなければ，パターンマッチングは失敗す

```
文法規則
(1) SENTENCE()->S(*x),{OUT(*x)}.
(2) S(s(*x,*y))->SUBJ(*x,*num),PRED(*y,*num).
(3) SUBJ(subj(*x),*num)->NP(*x,*num).
(4) NP(np(noun(*x),art(*y),mod(*z),nbr(*num2)),*num2)
      ->DET(*y),{F-GET(*y,num,*num1)},
        ADJ(*z),
        NOUN(*x),{F-GET(*x,num,*num2),MEMBER(*num2,*num1)}.
(5) NP(np(propn(*y)),*num)->PROPN(*y).
(6) PRED(pred(verb(*x),obj(*y)),*num1)
      ->VP(*x,*num1),NP(*y,*num2).
(7) VP(vp(*x),*num)->V(*x,*y),{F-GET(*y,num,*num)}.

辞書規則
(8) DET(the)->[the];num(.(sing,.(plural,.(mass,nil)))).
(9) ADJ(tall)->[tall].
(10) NOUN(man)->[man];num(sing).
(11) V(reach,reaches)->[reaches];num(sing).
(12) PROPN(new-york)->[new-york].

局所 Horn 節
(13) MEMBER(*x,.(*x,*y)).
(14) MEMBER(*x,.(*y,*z))<-MEMBER(*x,*z).

非終端記号の表す文法カテゴリ
S: 文          SUBJ: 主語      PRED: 述語      NP: 名詞句
DET: 冠詞      ADJ: 形容詞     NOUN: 名詞      PROPN: 固有名詞
VP: 動詞句     V: 動詞         T: 終了記号

関数 first, followの要素に含まれる非終端記号
<non-terminal>  <first>                          <follow>
SENTENCE        SENTENCE,S,SUBJ,NP,DET,PROPN     T
S               S,SUBJ,NP,DET,PROPN              T
SUBJ            SUBJ,NP,DET,PROPN                PRED,VP,V
PRED            PRED,VP,V                        T
NP              NP,DET,PROPN                     PRED,VP,V,T
DET             DET                              ADJ
ADJ             ADJ                              NOUN
NOUN            NOUN                             PRED,VP,V,T
PROPN           PROPN                            PRED,VP,V,T
VP              VP,V                             NP,DET,PROPN
V               V                                NP,DET,PROPN
```

図5 文法例
Fig. 5 Example of PAMPS grammar.

```
<non-terminal1>          <non-terminal2>          <result of pattern matching>

ADJ(*x)                  ADJ(tall)                        <success>

                                                  ADJ(tall)
                                                  *x=tall<------<substitution>

S(np(noun),pred(vp(*p))) S(*x,pred(vp(v(reach)))) <success>

                                                  S(np(noun),pred(vp(v(reach))))
                                                  *p=v(reach)
                                                  *x=np(noun)  <------<substitution>

NP(noun(man),nbr(sing))  NP(noun(man),nbr(plural)) <failure>
```

図6 パターンマッチングの例
Fig. 6 Illustration of pattern matching.

* 以後，図4で定義した引数つきの非終端記号をたんに非終端記号と呼ぶ．

る．3) 一方が複合項の場合，他方が同一の項名をもつ複合項でなければ，パターンマッチングは失敗する．もし項名が等しければ，それらの引数に対してパターンマッチングを再帰的に繰り返す．このうち1) と3) は文の深層構造を組み立てる際に，あるいは書換え規則間で文脈情報を伝達する手段として，2) は書換え規則の適用条件の判定として利用できる．**図6**にパターンマッチングの例を示す．

パターンマッチングの導入に伴い，2章で示したアルゴリズムを修正する．構文解析中“状態”が生成されるときに，その親となるゴールの予測記号とフレーズの非終端記号，あるいは文法規則の右辺最左非終端記号とフレーズの非終端記号間で，引数同士をパターンマッチングするという操作を新たに付け加える．もしパターンマッチングが失敗すれば，“状態”の生成は阻止される．**図7**に修正したアルゴリズムを示す．

3.4 局所 Horn 構成[6]

自然言語のように例外が多く，構造的に複雑な言語を解析する場合，単純な非終端記号間での値の受け渡しのみではうまくいかないことが多い．たとえば文法カテゴリー間での意味的整合性を調べるための条件を文法規則内に埋め込み，文法的には適格でも意味的に不適格な文を排除する必要がある．また解析で得られる構文木に意味的な処理を施し，意味的内部構造を抽出する場合には，そのための手続きを文法規則が呼び出せるようにしなければならない．これらの要求を満たすために，文献6)で提案された局所 Horn 構成という概念を導入する．

たとえば，図5の文法規則(4)の右辺には，中カッコで囲まれた部分が2か所ある．これらのゴール列は，局所 Horn 節集合で定義される手続き，あるいはあらかじめ PAMPS に組み込まれているシステム組込み述語を呼び出すために使われる．このように，中カッコ内での手続き呼び出し，および呼び出される手続き集合を総称して局所 Horn 構成という．

局所 Horn 節は，Prolog のプログラムを構成する Horn 節[3]とまったく同様である．この局所 Horn 節は手続き的にも宣言的にも解釈できる．局所 Horn 節 **P←Q, R.** を手続き的に解釈すれば，頭部〈head〉**P** は手続きの入口，体部〈body〉は手続きの呼び出し **Q, R** からなる手続き本体と考えられる．また体部のない局所 Horn 節 **P.** は事実の主張と考えることができる．たとえば図5の局所 Horn 節集合 (13), (14) で定義されている MEMBER は，第1引数の項が第2引数の複合項に含まれるかどうかを調べる手続きである．中カッコ内のゴール列* は，左から右へと1個ずつ各局所 Horn 節の頭部とパターンマッチングを試み，成功した局所 Horn 節を起動する．起動をかけられた局所 Horn 節は，さらに体部をサブゴール列として新たな局所 Horn 節とパターンマッチングを試みる．この実行は下降縦型探索で行われる．もしパターンマッチングが失敗すれば，バックトラックによってやり直す．最終的にすべてのパターンマッチングが成功すれば，局所 Horn 構成内の実行は終了し，また構文解析の処理にもどる．もし失敗すれば，その書換え規則の適用を中止する．なおバックトラックは中カッコ

```
parse: generate state<S(...)T(...),0,0,S'(...)>;
       buffer2:=set-buffer(w1);
       j:=0;
loop:  j:=j+1;
try-W: buffer1:=buffer2;
       buffer2:=set-buffer(wj+1);
       for each buffer1 element E1(...)
         if state<?B(...)?γ,?k,j-1,?A(...)> and E1∈first(B)
           then for each buffer2 element E2(...)
             if E2∈follow(E1)
               then generate state<λ,j-1,j,E1(...)>;
       for each state<λ,?i,j,?B1(...)>
try-C:   (if state<?B2(...)?C(...)?α,?k,i,?A(...)> such that B1=B2
           then for each buffer2 element E2(...)
             if E2∈first(C)
               then (pattern-matching(B1(...),B2(...),θ);
                     generate state<C(...)αoθ,k,j,A(...)oθ>)
try-R:     if state<?B2(...),?k,i,?A(...)> such that B1=B2
           then for each buffer2 element E2(...)
             if E2∈follow(A)
               then (pattern-matching(B1(...),B2(...),θ);
                     generate state<λ,k,j,A(...)oθ>)
try-L:     for each rule ?A(...) -> ?B2(...)?B'(...)?α such that B1=B2
           if state<?C(...)?γ,?k,i,?F(...)> and A∈first(C)
             then for each buffer2 element E2(...)
               if E2∈first(B')
                 then (pattern-matching(B1(...),B2(...),θ);
                       generate state<B'(...)αoθ,i,j,A(...)oθ>)
try-D:     for each rule ?A(...) -> ?B2(...) such that B1=B2
           if state<?C(...)?γ,?k,i,?F(...)> and A∈first(C)
             then for each buffer2 element E2(...)
               if E2∈follow(A)
                 then (pattern-matching(B1(...),B2(...),θ);
                       generate state<λ,i,j,A(...)oθ>)
       )
       go to loop;
termination: if state<λ,0,m+1,S'(...)> is generated
               then return acceptance;

set-buffer(w) = {D(...)|D(...) -> [w]}

pattern-matching(A(...),B(...),θ): 非終端記号A(…), B(…)をパターンマッチングし，
                                   置換θを求める手続き
A(…)○θはパターンマッチングの結果であることを示す．
?Xは‘…であるすべてのXについて’と読む．
```

図7 PAMPS のアルゴリズム
Fig. 7 Parsing algorithm of PAMPS.

* 2章でのゴールとは意味が異なることに注意．

内で局所的に行われる．

3.5 曖昧性の処理

PAMPSのような並行型解析アルゴリズムの特徴として，文法に曖昧性がある場合，解析中に頂点が同一の複数の部分構文木が生じることがあげられる．もしユーザがただ1個の部分構文木を望むならばPAMPSの書換え規則に解析の優先順位を与え，その得点計算により，複数の部分構文木のうちいずれかを選ぶことができる．この得点は部分構文木の各節点がもつ得点と，その部分構文木を成長させるために適用した書換え規則の優先順位との和により決定され，最高点をもつ部分構文木が最も優先される．

4. PAMPS の構文解析例

本章では図5で示した文法を用いて，以下の単文

The tall man reaches New York.

を解析する過程を**図8**を用いて説明する．

(1) アルゴリズムの初期設定として，ゴール①を生成する．

(2) try-W：入力文の先頭語 the，先読み語 tall を読み，$buffer_1$, $buffer_2$ にそれぞれ，DET (the), ADJ (tall) を入れる．DET は first (SENTENCE) に，ADJ は follow (DET) に含まれるので，辞書規則 (8) を適用し，フレーズ②を生成する．

(3) try-L：文法規則 (4) の左辺 NP は first (SENTENCE) に，$buffer_2$ の ADJ は first (ADJ) に含まれる．したがってフレーズ②の非終端記号 DET (the) と文法規則 (4) の右辺最左非終端記号 DET ($*y$) をパターンマッチングする．ここで変数 $*y$ は the に置換される．続く局所 Horn 構成内に含まれる述語 F-GET はあらかじめシステムに組み込まれている述語で，3個の引数はそれぞれ，単語，特徴素タイプ，特徴素値を指定する．F-GET は辞書規則の特徴素列から特徴素値を求め，その値を第3引数とパターンマッチングする．第3引数が変数のときは特徴素抽出述語として，すでに値をもっているときは特徴素チェック述語として働く．ここでは辞書規則(8)より特徴素タイプ NUM の値 •(sing, •(plural, •(mass, nil))) を求め，変数 *num1 をこの値で置換する．この手続きを終えると，ゴール③を設定する．

(4) try-W：バッファを1語左へシフトし，$buffer_1$ には ADJ (tall) を，$buffer_2$ には先読み語 man に対応する非終端記号 NOUN (man) を入れる．(2)と同様の過程を経て，辞書規則(9)を適用し，フレーズ④を生成する．

(5) try-C：$buffer_2$ の NOUN は first (NOUN) に含まれるので，フレーズ④の非終端記号とゴール③の予測記号をパターンマッチングし，ゴール⑤を設定する．

(6) try-W：(4)と同様の操作後，辞書規則(10)を適用し，フレーズ⑥を生成する．

(7) try-R：(6)の操作で入れら

```
$ the tall man reaches new-york.← 入力文
① <SENTENCE()T(),0,0,S'()> ⟵ 初期設定

   ** 1. **
② <λ,0,1,DET(the)>                                    冠詞 the の数情報
   : F-GET(the,num,*num1)                                    ↓
③ <ADJ(*z) NOUN(*x) F-GET(*x,num,*num2) MEMBER(*num2,.(sing,
    .(plural,.(mass,nil)))),0,1,NP(np(noun(*x),art(the),mod(*z),
    nbr(*num2)),*num2)>

   ** 2. **
④ <λ,1,2,ADJ(tall)>
⑤ <NOUN(*x) F-GET(*x,num,*num2) MEMBER(*num2,.(sing,.(plural,
    .(mass,nil)))),0,2,NP(np(noun(*x),art(the),mod(tall),
    nbr(*num2)),*num2)>

   ** 3. **
⑥ <λ,2,3,NOUN(man)>
   : F-GET(man,num,*num2) MEMBER(*num2,.(sing,.(plural,.(mass,nil))))
   : MEMBER(sing,.(sing,.(plural,.(mass,nil))))←冠詞と名詞の数の一致を調べる
              ↖名詞 man の数情報          ↙主語の数情報
⑦ <λ,0,3,NP(np(noun(man),art(the),mod(tall),nbr(sing)),sing)>
⑧ <λ,0,3,SUBJ(subj(np(noun(man),art(the),mod(tall),nbr(sing))),sing)>
⑨ <PRED(*y,sing),0,3,S(s(subj(np(noun(man),art(the),mod(tall),
                  ↖主語から渡された数情報
    nbr(sing))),*y))>

   ** 4. **
⑩ <λ,3,4,V(reach,reaches)>
   : F-GET(reaches,num,*num)                              述語の数情報
⑪ <λ,3,4,VP(vp(reach),sing)>                                  ↓
⑫ <NP(*y,*num2),3,4,PRED(pred(verb(vp(reach)),obj(*y)),sing)>

   ** 5. **
⑬ <λ,4,5,PROPN(new-york)>
⑭ <λ,4,5,NP(np(propn(new-york)),*num)>
⑮ <λ,3,5,PRED(pred(verb(vp(reach)),obj(np(propn(new-york)))),sing)>
⑯ <λ,0,5,S(s(subj(np(noun(man),art(the),mod(tall),nbr(sing))),
    pred(verb(vp(reach)),obj(np(propn(new-york))))))>
  主語と述語の数が一致するのでフレーズ⑯が生成される(フレーズ⑮とゴール⑨から)
   : OUT(s(subj(np(noun(man),art(the),mod(tall),nbr(sing))),
    pred(verb(vp(reach)),obj(np(propn(new-york))))))

    s(subj(np(noun(man),art(the),mod(tall),nbr(sing))),
     pred(verb(vp(reach)),obj(np(propn(new-york)))))
   ↖システム組込み述語 OUT からの出力
⑰ <λ,0,5,SENTENCE()>
⑱ <T(),0,5,S'()>

   ** 6. **
⑲ <λ,5,6,T()>
⑳ <λ,0,6,S'()>
```

印：のついたものは，局所 Horn 構成内のゴールを示す．

図 8 ゴール，フレーズの生成例
Fig. 8 Illustration of generated goals and phrases.

れた $buffer_2$ のVは，follow(NP) に含まれる．したがってフレーズ⑥の非終端記号と，ゴール⑤の予測記号をパターンマッチングする．続いて局所Horn構成内の手続きを実行する．まず F-GET によって変数 ＊num 2 は sing に置換され，次のサブゴール MEMBER (sing, • (sing, • (plural, • (mass, nil)))) に移る．このサブゴールは局所 Horn 節 (13) とパターンマッチングが成功し，局所Horn構成内の実行を終える．その結果フレーズ⑦を生成する．

同様の過程を経て，最後に入力文の統語的構造を出力し，解析を終了する．この出力は文法規則(1)に含まれるシステム組込み述語 OUT によって行われる．

5. DCG との比較

現在実働している述語論理型構文解析プログラムとして，Definite Clause Grammar (DCG)[12]，Simmonsらのプログラム[16]，Metamorphosis Grammar[4] などがある．このうち DCG は解析方式を除くと，PAMPS と類似したものであり，すでにいくつかの自然言語処理システムの一部として応用されている[11),15]．本章ではこの DCG と PAMPS との比較により，PAMPS の特徴をさらに明確にする．

1) 文法の形式

DCG では文法規則の右辺に非終端記号だけでなく，終端記号列を置くことができる．また CFG の λ-規則に対応する右辺のない文法規則が許される．PAMPS は辞書規則内に特徴素列が書け，規則に優先順位を与えることができる．また DCG はアルゴリズムの特性上，左回帰規則に対してはうまく動作しないので，規則の記述に注意する必要がある．

2) 解析方式

DCG は下降逐次型解析なので一度の解析過程で考慮している構文木はただ1個である．これに対して PAMPS は上昇並行型をとるので，文法に曖昧性のある場合には，可能な構文木がすべて同時に生成される．

3) 優先順位

DCG では構文解析結果の優先順位付けは，規則の並びにのみ依存する．逆に PAMPS では各規則に優先順位を与え，その順位計算により解析結果の優先順位を変更できる．

4) 手続きの埋め込み

DCG も PAMPS と同じく文法規則内に手続きを埋め込むことができる．この手続きは構文解析アルゴリズムと同様に下降逐次型探索で実行される．PAMPS の構文解析は上昇並行型で，局所 Horn 節集合で定義された手続きは下降逐次型探索で実行される．したがって PAMPS は構文解析プログラムとプログラム言語 Prolog を融合したものと考えることができる．

6. 実験および考察

本報告では，述語論理に基づく構文解析プログラム PAMPS について述べた．PAMPS は阪大大型計算機センターの ACOS-1000 上の Lisp 2.1 で実現されている．構文解析の基本アルゴリズムは約300行，局所 Horn 構成を含む PAMPS 全体では約2,000行である．このなかには文法の入出力，トレースなどのユーティリティも含まれている．試作した文法規則は約110ほどあるが，関係代名詞，接続詞句における省略，イデオムなどに関する規則はまだ十分に組み込まれていない．この文法を用いての解析は1語当り 40 msec 程度で実行されている．

今後の問題点としては，

1) 解析速度を向上させること．
2) 文法作成を容易にするために，システム組込み述語を充実させること．
3) 文法の拡張，修正を容易にするために，解析の進行状況を視覚的に表示するための機能，解析の実行途中でユーザの指定により解析の流れを一時的に制御する機能などをもつこと．

などがあげられる．1) に対しては，PAMPS の文法規則を Fortran で実現した仮想機械 PAMPS マシンのオブジェクトコード列にコンパイルし，それを直接実行する PAMPS マシンを開発済みである．簡単な比較実験によると，PAMPS マシンでの実行は5倍から12倍もの速度向上になることが確かめられている[18]．また，2)，3) については現在考察中である．

PAMPS, DCG などの述語論理に基づく構文解析プログラムは，Montague 文法などの言語理論[12] や，関係データベースとも関連が深く[7),11]，将来自然言語処理に新たな光を投げ掛けることが期待される．

謝辞 本研究を進めるにあたり，有益な助言討論をしていただいた修士課程在学の落谷 亮氏，河合和久氏，野口要治氏に深く感謝します．

参考文献

1) Aho, A. V. and Ullman, J. D.: *The Theory of Parsing, Translation, and Compiling,*

Volume 1, Parsing, p. 541, Prentice-Hall, Englewood Cliffs (1972).
2) Bouckaert, M., Pirotte, A. and Snelling, M.: Efficient Parsing Algorithms for General Context-Free Parsers, *Inf. Sci.*, Vol. 8, No. 1, pp. 1-26 (1975).
3) Clocksin, W. F. and Mellish, C. S.: *Programming in Prolog*, p. 279, Springer-Verlag, New York (1981).
4) Colmerauer, A.: *Metamorphosis Grammars, in Natural Language Communication with Computers*, in Bolc, L (ed.): *Lecture Notes in Computer Science*, No. 63, pp. 133-189, Springer-Verlag, New York (1978).
5) Earley, J.: An Efficient Context-Free Parsing Algorithm, *Comm. ACM*, Vol. 13, No. 2, pp. 94-102 (1970).
6) 渕 一博: 述語論理的プログラミング —EPILOG の提案—, 情報処理記号処理研究会資料 1-2 (1977).
7) Gallaire, H. and Minker, J. (eds.): *Logic and Data Bases*, p. 455, Plenum Press, New York (1978).
8) Kowalski, R.: *Predicate Logic as Programming Language*, Proc. IFIP 74, pp. 569-574, North-Holland, Amsterdam (1974).
9) Kowalski, R.: A Proof Procedure Using Connection Graphs, *J. ACM*, Vol. 22, No. 4, pp. 572-595 (1975).
10) Loveland, D. W.: *Automated Theorem Proving: A Logical Basis*, p. 405, North-Holland, Amsterdam (1978).
11) McCord, M. C.: Using Slots and Modifiers in Logic Grammars for Natural Language, *Artif. Intell.*, Vol. 18, No. 3, pp. 327-367 (1982).
12) Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Language Analysis —A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, No. 3, pp. 231-278 (1980).
13) Pereira, F. C. N.: Extraposition Grammars, *Am. J. Comput. Linguist.*, Vol. 7, No. 4, pp. 243-256 (1981).
14) Pratt, V. R.: LINGOL—A Progress Report, Proc. 4th IJCAI, pp. 422-428 (1975).
15) Silva, G. and Dwiggins, D.: Toward a Prolog Text Grammar, *SIGART Newsletter*, No. 73, pp. 20-25 (1980).
16) Simmons, R. F. and Chester, D.: Relating Sentences and Semantic Networks with Procedural Logic, *Comm. ACM*, Vol. 25, No. 8, pp. 527-547 (1982).
17) 田中, 佐藤, 元吉: 自然言語処理のためのプログラミングシステム—拡張LINGOLについて—, 信学論, Vol. J 60-D, No. 12, pp. 1061-1068 (1977).
18) 上原, 落谷, 河合, 豊田: 新しいパーザ PAMPS とそのコンパイラについて, 情報処理自然言語処理研究会資料 30-4 (1982).
19) Uehara, K. and Toyoda, J.: A Pattern Matching Directed Parser: PAMPS, Linguistic Society of America Meeting Handbook: 1981, pp. 116, ACL/LSA Sessions on Computer Modeling of Linguistic Theory (1981).

(昭和57年10月12日受付)
(昭和58年1月17日採録)