仮想機械による述語論理型構文解析プログラムの効率改善について†

上原邦昭†† 落谷亮†††
角所収†† 豊田順一††

本報告は自然言語処理プログラム PAMPS と，その仮想機械 PAMPS マシンについて述べたものである．PAMPS による文法の記述は述語論理に基づいているために，文法を宣言的に解釈することができる．そのため PAMPS を用いて言語理解システムを作成・修正することは容易である．しかしながら当初開発されていた PAMPS の実行メカニズムは，文法規則の逐一解釈・実行というインタプリタ形式であったため，高速な処理速度が得られなかった．今回試作した仮想機械 PAMPS マシンは，PAMPS 文法を PAMPS マシン命令にコンパイルし，その命令列を直接実行するものである．PAMPS マシン命令は，パターンマッチング用命令とパージングアルゴリズム用命令の2種類に分けて考えることができる．それぞれは PAMPS の実行メカニズムの特性を十分に考慮した高レベルの命令体系になっている．また PAMPS マシンのデータ構造は，DEC-10 Prolog での構造共有化技法に基づいているが，逐次処理，並列処理のいずれにも依存しない汎用性のあるものとなっている点に特色がある．この結果インプリメントした言語に違いはあるが，PAMPS マシンはインタプリタに比べて20倍から76倍もの処理速度改善となっている．

1. まえがき

人間と計算機の会話媒体に自然言語を用いることは理想である．しかしながら言語理解システムを実用化しようとする場合，考慮すべき問題点が少なくとも二つある．一つは自然言語に関する知識がまだ十分でなく，新しい知見が発見されるたびに絶えず文法を修正・変更する必要があることである．他の一つは人間と計算機の会話を円滑に行うために，十分に高速な応答速度を達成する必要があることである．とくに後者の問題はシステムが大規模化するにつれて，非常に重要な問題となるであろう．

先に筆者らは，自然言語処理用プログラムPAMPS[1]について報告した．PAMPS による文法の記述は，述語論理に基づいているために言語の宣言的記述が可能であり，言語理解システムの作成や修正が容易になるという利点がある．この点で PAMPS は第一の課題を達成していると考えられる．しかし文献1)で報告した PAMPS の実行メカニズムは，文法規則の解釈—実行というサイクルで動作するインタプリタ (PAMPS インタプリタと呼ぶ) として実現されている．したがって言語理解システムの開発段階ではトレース等の対話的な診断情報を容易に得ることができ有用であるが，高速な処理速度が得られないという欠点があった．

本報告では，この効率の問題を解決する手段として仮想機械 PAMPS マシンとそのコンパイラを提案する．PAMPS マシンの特徴としては，

1) PAMPS の実行メカニズムの特性を十分に反映するような高レベルの PAMPS マシン命令を設定していること，

2) PAMPS のパージングアルゴリズムである上昇並行型 (bottom-up, parallel) の特徴を考慮して，PAMPS マシン用のデータ構造を開発していること，

である．この結果，PAMPS マシンは PAMPS インタプリタに比べて20倍から76倍もの処理速度改善となっている．また筆者らの提案するデータ構造は，Warren らの DEC-10 Prolog の構造共有化技法[2]に基づいているが，バックトラックを含む逐次型処理あるいは並行型処理のいずれにも依存しないという点に特徴がある．Warren らのデータ構造が Prolog の下降縦型探索のみに限定されていたことを考えると，筆者らのデータ構造は最近研究が盛んな述語論理型言語の逐次処理マシンによる擬似 OR—並列処理を実現する上でも有用であると考えられる．

† Improvement on the Performance of Predicate Logic Oriented Parsing Program on a Virtual Machine by KUNIAKI UEHARA (The Institute of Scientific and Industrial Research, Osaka University), RYO OCHITANI (Department of Information and Computer Sciences, Faculty of Engineering Science, Osaka University), OSAMU KAKUSHO and JUNICHI TOYODA (The Institute of Scientific and Industrial Research, Osaka University).

†† 大阪大学産業科学研究所
††† 大阪大学基礎工学部情報工学科

2. PAMPS の概要

本章では PAMPS について以下の議論で必要な概念のみを説明する．PAMPS については文献1)で詳しく述べられている．

PAMPS 文法は，文法規則，辞書規則，局所 Horn 節からなる．文法規則は各非終端記号にいくつかの引数を与えた拡張文脈自由型規則で，

$A(t_{01}, \cdots, t_{0l}) \rightarrow B_1(t_{11}, \cdots, t_{1m}), \cdots, B_n(t_{n1}, \cdots, t_{np})$.

の形で表現される．辞書規則は，

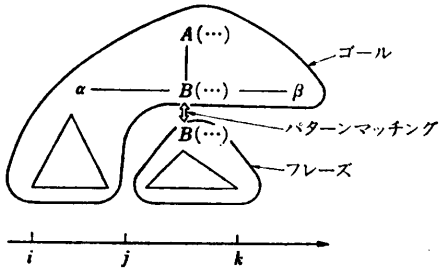
$A(t_{01}, \cdots, t_{0l}) \rightarrow [w]$.

の形で表現される．$A, B_i\ (1 \leqslant i \leqslant n)$ は非終端記号，wは終端記号である．非終端記号の引数は項である．項は変数，数，アトム，複合項からなる．複合項 $f(t_1, \cdots, t_n)$ は項名 f と引数 $t_i\ (1 \leqslant i \leqslant n)$ からなる．引数はまた項である．

構文解析は上昇並行型で行われ，文法に曖昧性のある場合は同時にいくつかの部分構文木（フレーズと言い換えることもある）が生成される．また将来成長する可能性のない無駄な部分構文木の生成を抑えるために，入力語の先読み機能と，次にどんな非終端記号が期待できるかという予測（ゴールと呼ぶ）が導入されている．構文解析が進行する各段階で，ゴールとフレーズの非終端記号間でパターンマッチングが行われ，引数を通じて文脈情報が受け渡される．**図1**にゴールとフレーズの例とその表記法を示す．

拡張文脈自由型規則のみで記述が困難な規則の適用条件は，局所 Horn 節（述語論理での Horn 節と同等）で定義される“手続き”によって記述可能である．局所 Horn 節は，

$P(x_1, \cdots, x_s) \leftarrow Q_1(y_1, \cdots, y_m), \cdots, Q_n(z_1, \cdots, z_p)$.



ゴールは$\langle B(\cdots)\beta, i, j, A(\cdots)\rangle$,
フレーズは$\langle\lambda, j, k, B(\cdots)\rangle$
で表す．

図1 ゴールとフレーズの例
Fig. 1 Pictorial representation of a goal and phrase.

と

$P(x_1, \cdots, x_s)$.

からなる．ただし，P, Q_i を述語名，述語名と引数をあわせて述語と呼ぶ．また引数は項である．

局所 Horn 節は，

$A(t_{01}, \cdots, t_{0l}) \rightarrow B_1(t_{11}, \cdots, t_{1m}), \cdots, \{P(x_1, \cdots, x_s)\}, \cdots, B_n(t_{n1}, \cdots, t_{np})$.

のように文法規則右辺の中カッコで囲まれた述語列が局所 Horn 節の左辺とパターンマッチングすることによって起動され（手続きの呼出し），Prolog と同様に下降縦型探索で実行される．

3. PAMPS マシンのデータ構造

PAMPS で記述した文法は，PAMPS コンパイラによって仮想機械 PAMPS マシンの命令列にコンパイルされる．この命令列を直接実行するものが PAMPS マシンである．本章ではこの PAMPS マシンのデータ構造について述べる．

3.1 項の表現

項は PAMPS マシンの内部で**図2**に示すようなスケルトンとして表現される．たとえば複合項

$S(np(\text{john}), vp(v(\text{likes}), *x))$

は**図3**で示すスケルトンとして表現される．

パターンマッチングした結果新しく組み立てられる項は，項の構造を示すスケルトンを指すポインタ S と，束縛された変数の値を格納したデータ領域を指すポインタ F の組 $\langle S, F\rangle$ で表される．この変数のデータ領域を変数フレームと呼び，文法規則が呼び出される度に1個作られる．変数フレームはいくつかの変数セルからなり，個々の変数セルには束縛された変数の

項	スケルトン
数 i	int (i)
アトム a	atom（aのアトム番号）
変数 $*v$	var（$*v$ の変数番号）
複合項 $f(t_1, \cdots, t_n)$	label l ⋮ l: fn（fの項名番号） $\tilde{t}_1$ ⋮ $\tilde{t}_n$ fn end

アトム名，変数名，項名はすべて固有の番号に変換する．変数番号は，各文法規則，辞書規則，局所 Horn 節内でそれぞれ固有に与えられる．l はラベル，$\tilde{t}_i$ は項 t_i のスケルトンを表す．

図2 項とそのスケルトンの対応関係
Fig. 2 Correspondence between terms and their skeletons.

値が書き込まれる．任意の変数セルは，変数フレームの先頭アドレスと変数番号をインデックスとして一意に決定できる．このようなデータ表現法を構造共有化技法[2]と呼ぶ．例として複合項 $np(*x, *y, \text{sing}, *z, *x)$ がパターンマッチングの後で

np(john, $*y$, sing,
　n(john human), john)

となったとする．このとき新しく組み立てられた複合項はスケルトンと変数フレームを用いて図4のように表現される．

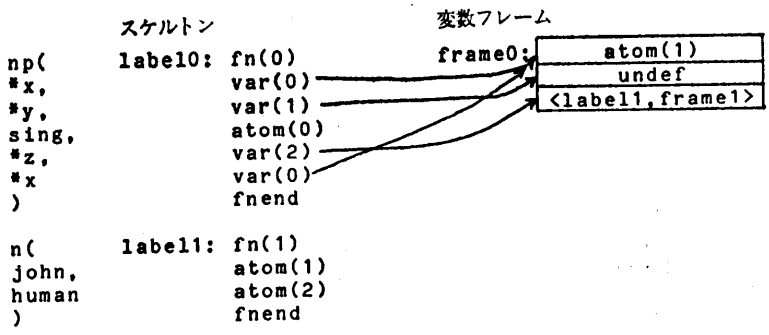
```
11: fn(0)          s(
    label(12)      np(john),
    label(13)      vp(v(likes),*x)
    fnend          )
12: fn(1)          np(
    atom(0)        john
    fnend          )
13: fn(2)          vp(
    label(14)      v(likes),
    var(0)         *x
    fnend          )
14: fn(3)          v(
    atom(1)        likes
    fnend          )
```

項名	項名番号
s	0
np	1
vp	2
v	3

アトム名	アトム番号
john	0
likes	1

変数名	変数番号
*x	0

図3　複合項 $s(np(\text{john}), vp(v(\text{likes}), *x))$ のスケルトン
Fig. 3　Skeleton of the compound term $s(np(\text{john}), vp(v(\text{likes}), *x))$.



frame 1 は複合項 n(john, human) を含む文法規則の変数フレーム先頭アドレスを指す．

図4　スケルトンと変数フレームによって表現された複合項の例
Fig. 4　A compound term represented by a skeleton and variable frame.

3.2 PAMPS マシンでの構造共有化技法

自然言語のための文法は通常曖昧性を伴うので，PAMPS による並行型の構文解析では，一つのフレーズと複数のゴール，あるいは逆に一つのゴールと複数のフレーズが同時にパターンマッチングするといった競合状態が生じる場合がある．競合状態の例を図5に示す．この例は一つのゴールと二つのフレーズがパターンマッチングするために生じる競合状態を表したものである．

文法規則が呼び出されたとき，その規則に含まれる変数はすべて束縛されておらず，値 undef（変数の値は未定であることを示す）をもつものとする．変数のこのような状態を explicit-undef と呼ぶ．状態①では辞書規則とのパターンマッチングの結果，変数 $*y$（スケルトンは var(2)，以後カッコ内はスケルトンを表す）には値 const 4 (atom (4)) が書き込まれる．変数 $*x0$(var(0)), $*x1$(var(1)) の変数セルは値 undef のままで変化はない．図5で示した変数フレームはすべてこの呼び出された文法規則の状態である．各辞書規則に対して作られる変数フレームは，議論を簡単にするために省略する．

状態②では変数 $*x0$(var(0)) の変数セルに const 0(atom(0)) と，パターンマッチング番号2が書き込まれる．このパターンマッチング番号は，パターンマッチングごとに与えられる固有の数であり，後に競合が起こった時点で，変数セルに書き込まれた値がいずれのパターンマッチングで束縛されたものかを決定するために用いる．状態②では 変数 $*x1$(var(1)) はまだ束縛されていない．

状態③ではゴール①が別の辞書規則とパターンマッチングし，競合が生じる．変数 $*x0$(var(0)) の変数セルにはすでに値 const 0(atom(0)) が書き込まれているが，この値をそのまま変数 $*x0$(var(0)) に束縛された値とみなすことはできない．このような競合を防ぐために，子孫―祖先関係表という概念を導入する．

子孫―祖先関係表は，パターンマッチングの履歴を示したもので，行列として表される．関係表の縦横はそれぞれパターンマッチング番号に対応する．たとえば i 行 j 列に1が立っていれば，パターンマッチング番号 i をもつゴール（フレーズ）は，パターンマッチング番号 j のゴール（フレーズ）の子孫である（j は

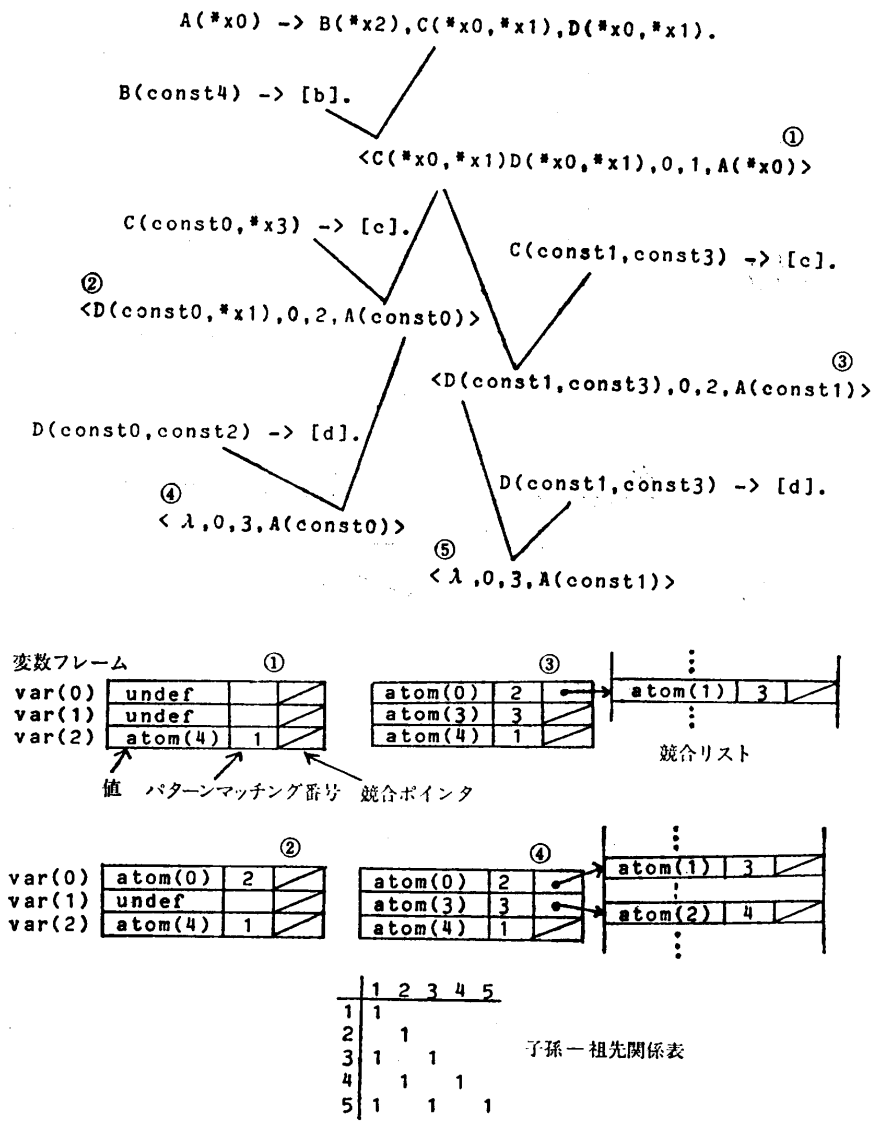


図 5 一つのゴールと二つのフレーズによる競合状態
Fig. 5 Pictorial representation of the conflict between a goal and two phrases.

i の祖先である）という*. PAMPS は上昇並行型で動作するので，一般に複数個の部分構文木が同時に作られる．もしパターンマッチング番号 i と j の間に子孫ー祖先関係が成り立たなければ，それらは別の部分構文木の成長で実行されたパターンマッチングである．変数セルに書き込まれた値が祖先のパターンマッチングで束縛されたものであれば，その値をそのまま変数の値とする．もしそうでなければ，その値は別の部分構文木で束縛されたものであり，現在のパターンマッチングでは変数はまだ束縛されていないということになる．

さて3行2列には1が立っていないので，変数 $*x0$ (var(0)) に書き込まれている値は祖先で束縛されたものではない．状態③で新しく変数 $*x0$(var(0)) に束縛される値 const 1(atom(1)) を表現するために，競合リストと呼ぶフリーセル領域に変数 $*x0$ の変数セルを作り，この変数セルに束縛される値とパターンマッチング番号3を書き込む．同時に変数フレーム上の競合ポインタは競合リスト上の変数セルを指す．変数 $*x1$(var(1)) の変数セルには値 const 3(atom(3)) とパターンマッチング番号3を書き込む．競合ポインタは何も指さない．

状態④では変数 $*x0$(var(0)) の変数セルに値 const 0(atom(0)) が書き込まれているが，ここでも競合の可能性がある．値 const 0(atom(0)) はパターンマッチング番号2で束縛されたものである．子孫ー祖先関係表の4行2列には1が立っているので，変数セルの値は祖先のパターンマッチングで束縛されたものであることがわかる．変数 $*x1$(var(1)) の変数セルに書き込まれている値 const 3(atom(3)) は，パターンマッチング番号3で束縛されたものであるが，関係表4行3列には1が立っておらず，祖先のパターンマッチングで束縛されたものではない．したがって現時点で変数 $*x1$(var(1)) はまだ束縛されていない．このように変数フレーム，競合リストのいずれにも値が書き込まれていない変数の状態を，さきほどの explicit-undef に対して implicit-undef と呼ぶ．

以上のデータ構造は Prolog の実行のような下降縦型探索においても有効である．この意味で筆者らの

* 子孫ー祖先関係表はグラフ理論での有向グラフの可到達性行列 (reachability matrix) と同一のものである．

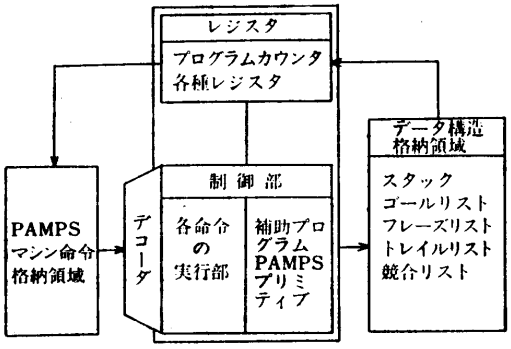


図 6 仮想 PAMPS マシンの概念構成図
Fig. 6 Conceptual structure of the virtual PAMPS machine.

データ構造は探索方式に依存しない汎用性のある述語論理型言語用のデータ構造となっている.

4. 仮想機械 PAMPS マシン

4.1 PAMPS マシンの制御構造

図6は後述の PAMPS マシン命令を実行する仮想機械 PAMPS マシンの概念構成図である. PAMPS プリミティブは PAMPS にシステム組込みとして与えられている述語を実行するサブルーチン群からなる. 補助プログラムは PAMPS マシンの初期設定, および解析結果の表示用プログラム群などからなる.

データ構造格納領域にあるスタックには変数フレームが作られる*. 前章の説明では述べなかったが, パターンマッチングが失敗すれば, すでに書き込まれた変数セルの値をすべて元にもどす操作 (値を undef にする) が必要である. この操作のために, パターンマッチングによって変数セルに値が書き込まれると, その変数セルのアドレスはトレイルリストと呼ぶデータ領域に記録される. パターンマッチングの失敗時にはトレイルリストを参照しながら変数セルを初期化する. ゴールリスト, フレーズリストにはそれぞれ構文解析過程で生成されるゴール, フレーズに関する情報が書き込まれる.

4.2 PAMPS 文法のコンパイル

本節では PAMPS 文法のコンパイルの原理について述べる. PAMPS のパージングアルゴリズムに従って文法規則と辞書規則に手続き的解釈を与える. 文法規則

$$A(t_{01}, \cdots, t_{0l}) \rightarrow B_1(t_{11}, \cdots, t_{1m}), B_2(t_{21}, \cdots, t_{2p}).$$

の右辺非終端記号 B_1, B_2 はそれぞれ手続きへの入口, B_1, B_2 の引数はパターンマッチング操作を規定した手続きの本体と考える. 規則左辺の非終端記号 A は手続きの呼び出し, A の引数は手続き A を呼び出すための値受渡し用引数と考える. 辞書規則

$$A(t_{01}, \cdots, t_{0l}) \rightarrow [a].$$

は入力語 a によって起動する手続きの呼出し, A の引数は手続き呼出し用引数と解釈する. 上記の文法規則は手続き呼び出し B_1 により (すなわちフレーズ B_1 の生成により) 起動する. B_1 の手続き本体でパターンマッチングを行うと, B_2 の前で処理を中断し, ゴールを設定する. フレーズ B_2 が生成されると, B_2 の前で中断されていた手続きは再開し, さらに B_2 の手続き本体でパターンマッチングを行う. パターンマッチングが成功すると, 続いてフレーズ A を生成して処理を終了する. 生成されたフレーズは待ち行列 (フレーズリスト) に入れられ, 行列の先頭から次々と手続き呼出しが起こり, 上述の動作が繰り返される. 待ち行列が空になるとさらに次の入力語を読み込み解析を続ける.

局所 Horn 節は縦型逐次探索で実行するために, 文法規則とは逆に右辺を手続き呼出しの並び, 左辺を手続き入口と考える.

PAMPS 文法に以上の手続き的解釈を与えた場合, PAMPS マシン命令にコンパイルするためには,

1) パターンマッチングは, ほとんどの場合変数への値の代入, あるいは同値判定といった単純な操作に分解できる. したがって各非終端記号の引数を出現の仕方に応じていくつかの場合に分類し, それぞれの場合に対応した高レベルの命令を設定すること.

2) 構文解析の途中で無駄な規則検索の手間を省くために, あらかじめ解析の可能な道筋に関する情報を抽出しておき, 実行時にはその情報に従って解析を進めること.

などが必要である. 次節では PAMPS マシン命令をパターンマッチング用命令とパージングアルゴリズム用命令の二つに分けて説明する.

4.3 PAMPS マシン命令

文法規則 $A(\cdots) \rightarrow B_1(\cdots), B_2(\cdots), B_3(\cdots).$ は**図7**のような命令列にコンパイルされる. pattern-matching (B_i) は, 非終端記号 B_i の引数にほぼ1対1となるパターンマッチング用命令列を表す. パターンマッチング用命令の詳細は 4.3.1 項で述べる. init 命令は B_1 のパターンマッチング用命令列の実行が終わった後で, この文法規則に出現する残りの変数セルを確保

* 実際には局所 Horn 節の変数は大域変数と局所変数という2種類に分類される. それぞれの変数用にスタックがあるが, 本報告では省略する.

し初期化する命令である．neck 命令はスタックに変数フレームを1個作る命令である．gset 命令，pset 命令は，それぞれゴール，フレーズの生成情報をゴールリスト，フレーズリストに記録し，さらに手続きを中断する命令である．手続き再開時にはこれら gset 命令，pset 命令の次の命令から実行できるように，ゴールリスト，フレーズリストに手続き中断時のアドレスが記録される．call(A) 命令は手続きAに制御を移す命令である．

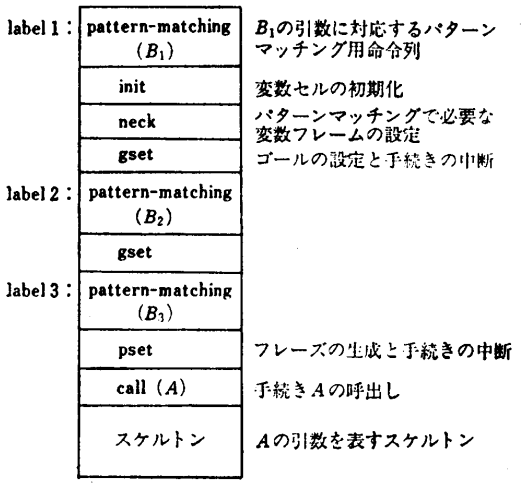
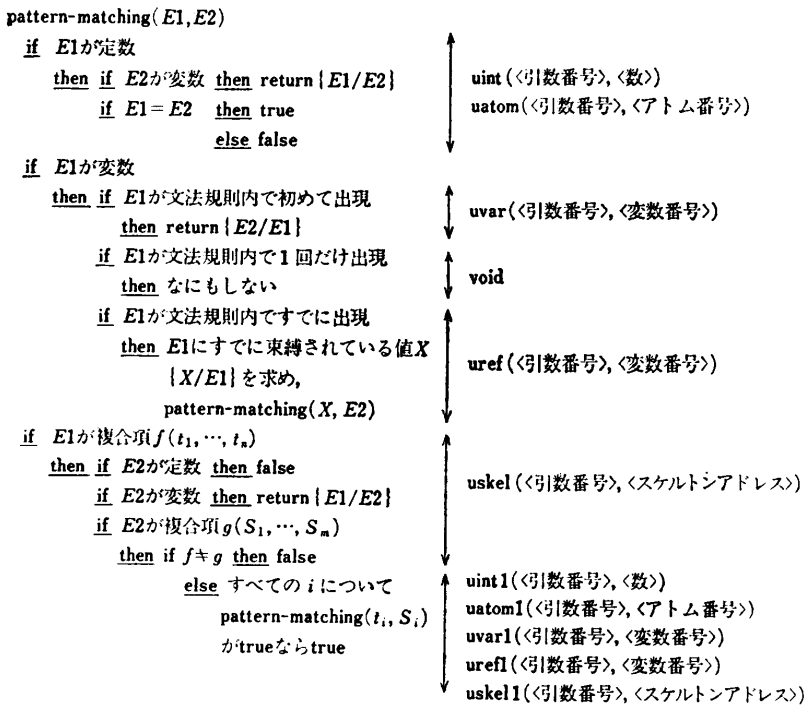


図 7 文法規則のコンパイル模式図
Fig. 7 Schematic representation of a compiled grammatical rule.

4.3.1 パターンマッチング用命令[2)]

パターンマッチング用命令を説明するために，**図8**にパターンマッチングのアルゴリズムを示す．ただし $E1$ を文法規則右辺（ゴール側），$E2$ を文法規則左辺（フレーズ側）の非終端記号に含まれる引数とする．4.2 節でも述べたように，$E1$ はパターンマッチング用命令，$E2$は手続き呼出しの値受渡し用引数となる．図8はパターンマッチングのアルゴリズムとパターンマッチング用命令の対応を示している．このうち uvar 命令は代入操作，uint 命令，uatom 命令，uskel 命令は $E2$ の出現状況により同値判定か代入操作のいずれかを行う．また uref 命令は $E1$ にすでに束縛されている値を求め，同値判定か代入操作を行う．void では何もする必要はなく命令は設定されない*．このようにコンパイル時に引数の出現状況を調べ，それぞれに対して高レベルの命令を設定することにより，パターンマッチングの処理時間を高速化することができる．



第1オペランドの引数番号は，非終端記号での引数の出現位置を示す．ただし，命令uint1などの引数番号は，その複合項での引数の出現位置を示す．

図 8 パターンマッチングのアルゴリズム
Fig. 8 Procedural representation of pattern matching algorithm.

4.3.2 パージングアルゴリズム用命令

まず文献1)で示したパージングアルゴリズムのうち，主要部分を**図9**に示す．文法規則に 4.2 節で示した手続き的解釈を与えると，パージングアルゴリズムは手続きを起動するための方法を明示したものと考えることができる．図8で示したパージングアルゴリズムを八つの部分に分け，それぞれを PAMPS マシン命令として設定する．このうち try-C 命令，try-R 命令は中断されている手続きを再開する命令である．たとえば try-R 命令は call 命令で

* 語尾に1のついている命令は，複合項の引数に対して設定される命令である．またこの引数よりも深い位置に出現する引数については命令を設定しない．

呼ばれた手続きを再開するための条件として関係 follow を調べ*，もし関係が成り立てば，中断されていたパターンマッチング用命令列に制御を移す．同様に try-L 命令，try-R 命令は新たな手続きを開始する命令である．

文法規則 $A(\cdots) \rightarrow B_1(\cdots), \cdots, B_n(\cdots)$. $(n \geqslant 2)$ の場合，try-L 命令は B_1，try-C 命令は B_i $(2 \leqslant i \leqslant n-1)$，try-R 命令は B_n へ制御を移す命令である．また文法規則 $A(\cdots) \rightarrow B(\cdots)$. の場合，try-D 命令は B へ制御を移す命令である．コンパイル時にはこれら文法規則右辺の非終端記号を try-R 命令等の四つの部分に対応づけ，同一非終端記号同士でまとめている．**図 10** はパージングアルゴリズム用命令の実行の流れをフローチャートとして図示したものである．

それぞれのフレーズ⟨λ, ?i, j, ?B⟩について
(それぞれのゴール⟨B ?C ?α, ?k, i, ?A⟩について
D∈first(C)ならば
pattern-matching; ゴール⟨Cα, k, j, A⟩の設定　　enter
それぞれのゴール⟨B, ?k, i, A⟩について　　entry
D∈follow(A)ならば　　try-C("C", ⟨飛び先アドレス⟩)
pattern-matching; フレーズ⟨λ, k, j, A⟩の生成　　entry
それぞれの文法規則 ?A→B ?B′ ?αについて　　try-R("A", ⟨飛び先アドレス⟩)
A∈first(C)となるゴール⟨?C ?γ, ?k, i, ?F⟩
が存在し，かつ D∈first(B′)ならば，　　try-L("A", "B", ⟨飛び先アドレス⟩)
pattern-matching; ゴール⟨B′α, i, j, A⟩の設定
それぞれの文法規則 ?A→Bについて
A∈first(C)となるゴール⟨?C ?γ, ?k, i, ?F⟩
が存在し，かつ D∈follow(A)ならば，　　try-D("A", ⟨飛び先アドレス⟩)
pattern-matching; フレーズ⟨λ, i, j, A⟩の生成)

○pattern-matching はゴールとフレーズの非終端記号間でのパターンマッチング操作を表す．
○D は先読み語を表す．
○⟨飛び先アドレス⟩はパターンマッチング用命令列への飛び先のアドレスを示す．

図 9 PAMPS のパージングアルゴリズム
Fig. 9 Procedural representation of PAMPS parsing algorithm.

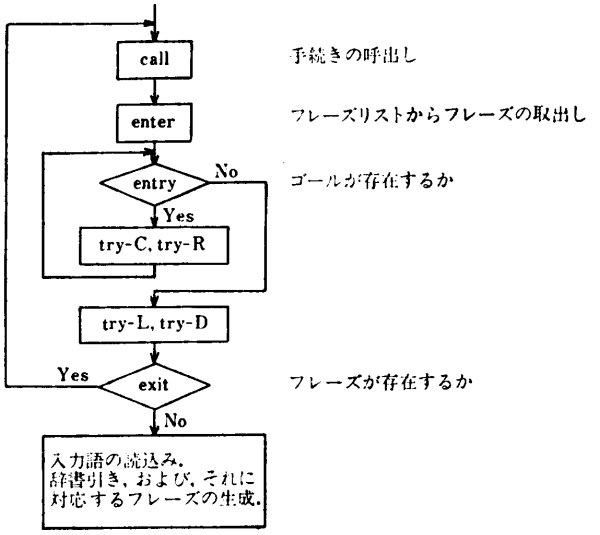


図 10 パージングアルゴリズム用命令の実行流れ図
Fig. 10 Execution flow of PAMPS machine instructions for parsing.

4.3.3 辞書規則と局所 Horn 節のコンパイル

実用的な言語理解システムでは辞書のサイズも大規模なものとなる．PAMPS マシンでは辞書規則を高速に検索するために，ハッシング技法を用いて，入力語を検索の鍵として検索できるようにしている．

また局所 Horn 節は DEC-10 Prolog の Prolog コンパイラとまったく同様の形式で PAMPS マシン命令にコンパイルされる．命令形式，データ構造ともにほとんど等しいので本報告では省略する．

付録に簡単な文法とそのコンパイル例を示す．

5. PAMPS マシンの性能評価

本章では PAMPS マシンと PAMPS インタプリタとの性能比較を行い，PAMPS マシンの優位性を立証する．

5.1 測定に用いる文法

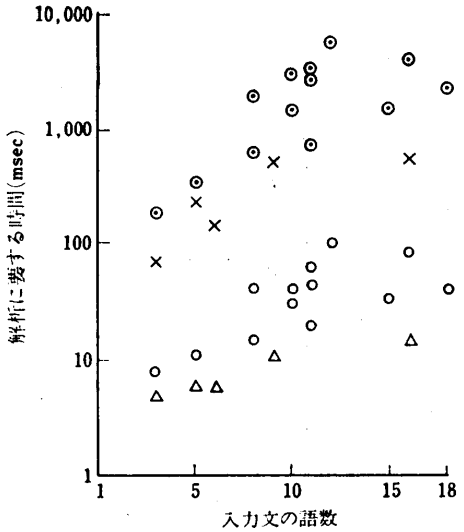
性能比較の測定対象として選んだ文法は，Simmons と Chester の物語を解析する Logic Grammar[3](LG と，Pereira と Warren の Definite Clause Grammar[4](DCG) である．LG は Lisp と Prolog を融合した言語 HCPRVR で記述されている．DCG は Prolog で記述された文法である．両者はいずれも下降逐次型で構文解析するように動作する．これら二つの文法を PAMPS 文法に変換し測定対象とした．LG に与える文は V-2 ロケットに関する 13 文からなる物語[3]であり，DCG に与える文は文献 4) の付録 2 に示されている 5 文である．

5.2 動的諸特性の測定結果

5.2.1 インタプリタと仮想機械の実行速度比較

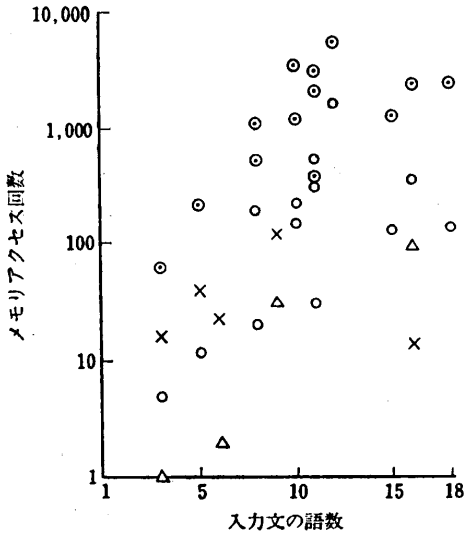
PAMPS インタプリタは ACOS-1000 上の Lisp 2.1 でインプリメントされ，さらに Lisp コンパイラ

* 関係 follow, first についての説明は，紙面の都合上省略する．詳しくは文献 1) を参照のこと．



⊙はLGを，×はDCGをPAMPSインタプリタで実行した場合，
○はLGを，△はDCGをPAMPSマシンで実行した場合である．

図 11 PAMPS マシンとインタプリタの速度比較
Fig. 11 Comparison of execution time between PAMPS machine and interpreter.



⊙はLGを，×はDCGをPAMPSインタプリタで実行した場合，
○はLGを，△はDCGをPAMPSマシンで実行した場合である．

図 12 PAMPS マシンとインタプリタのオーバヘッド比較
Fig. 12 Comparison of overhead between PAMPS machine and interpreter.

によってコンパイルされている．PAMPS マシンは FORTRAN によってインプリメントされている．実現された言語が異なるので，単純な速度比較はできないが，**図 11** で示したように，入力語数が長くなるにつれて両者とも指数関数的に実行時間を要していることがわかる．また PAMPS マシンは PAMPS インタプリタに比べて 20 倍から 76 倍もの速度向上になっていることがわかる*．とくに入力語数が長くなるにつれて速度向上比は大きくなっていることがわかる．この原因として，PAMPS 文法が PAMPS マシン命令に直接コンパイルされていることもあるが，両者のデータ構造に違いがあることも大きく影響していると考えられる．次項ではデータ構造の観点から両者を比較する．

5.2.2 データ構造の比較

まず簡単に PAMPS インタプリタのデータ構造を説明する．PAMPS インタプリタのデータ構造は，PAMPS マシンのそれとは異なり，文献 6) で提案された構造共有化技法に基づいている．PAMPS マシンのデータ構造との大きな相違点は，束縛された変数の値を格納する方法である．PAMPS インタプリタでは変数の値を変数 Var と値 Term の結合対 (Var . Term) として表現する．各パターンマッチングごとに結合対からなるリスト (mgu リスト) が作られる．後のパターンマッチングで変数 Var が出現するときはいつでも，Var がある値 Term に束縛されているかどうか，その祖先となるゴールやフレーズをさかのぼり mgu リストを調べる．もし Var と Term が結合対として mgu リストに出現するならば，Var は Term に束縛されているとして処理を進める．これを deep-binding 法と呼ぶ．これに対して 3 章で説明した PAMPS マシンのデータ構造を shallow-binding 法と呼ぶ．

shallow-binding 法と deep-binding 法の定量的比較にあたって，それぞれの方式のオーバヘッドと考えられるものをメモリアクセス回数で換算する．deep-binding 法では，束縛された値を検索するために，変数が出現する度に mgu テストの結合対を調べる操作が必要となる．この束縛された値を発見するまでの結合対へのアクセス回数を deep-binding 法でのオーバヘッドとする．

次に shallow-binding 法でのオーバヘッドを考える．shallow-binding 法では変数同士でパターンマッチングが行われた場合，互いにその変数の値が implicit-undef あるいは explicit-undef ならば，二つの変数セルはポインタで結合される．このような変数を共有変数と呼ぶ．後にいずれかの変数に値が束縛さ

* 文献 1) では 5 倍から 12 倍となっているが，これは形式言語理論の教科書[9] にでているような数個の文法規則からなるような例の場合である．

れると，両方の変数は同一の値をもつことになる．このように共有された変数の値は変数間のポインタをたどることによって求められる．このたどりの回数を A_s とする．またある変数が競合状態にある場合，変数の値は競合リスト上にある．この値を検索するために競合ポインタをたどる回数を A_c とする．したがって shallow-binding 法のオーバヘッドは A_s+A_c で表される．

図12は両者のオーバヘッドを各測定文法に対して求め，その結果を図示したものである．この図からはばらつきが大きく一概にはいえないが，入力文が長くなるにつれて，したがって競合状態が多くなるにつれて，両者ともオーバヘッドが大きくなることがわかる．また筆者らの提案した shallow-binding 法は，deep-binding 法に比べて2倍から23倍ものオーバヘッドの改善となり，筆者らのデータ構造はメモリアクセスの観点からも妥当であることがわかる．

6. むすび

本報告では述語論理に基づく自然言語処理用プログラム PAMPS と，それを効率よく実行する仮想機械 PAMPS マシンについて報告した．PAMPS マシンに導入した shallow-binding 法と呼ぶ構造共有化技法と PAMPS 文法のコンパイルにより，構文解析に要する時間は大幅な短縮となった．

今後はインタプリタと仮想機械のインプリメントする言語，データ構造を統一すること，コンパイラの最適化処理を考えてさらに効率のよい PAMPS マシン命令を設定することが必要である．しかしながらまず Lisp でインタプリタを実現し，その考察に基づいて仮想機械を設計したのは，開発工程から考えると大変望ましいものであったと思われる．

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井上哲也氏，三上理氏に感謝する．また博士課程在学の河合和久氏には PAMPS マシンのデータ構造を検討していただき，とくに感謝する．

参考文献

1) 上原，豊田：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，情報処理学会論文誌，Vol. 24, No. 4, pp. 496-504 (1983).
2) Warren, D. H. D.: Implementing Prolog—Compiling Predicate Logic Programs, Technical Report 39 & 40, Department of Artificial Intelligence, University of Edinburgh (1977).
3) Simmons, R. F. and Chester, D.: Relating Sentences and Semantic Networks with Procedural Logic, *Comm. ACM*, Vol. 25, No. 8, pp. 527-547 (1982).
4) Pereira, F. C. N. and Warren, D. H. D.: Definite Clause Grammars for Natural Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13, No. 3, pp. 231-278 (1980).
5) Aho, A. V. and Ullman, J. D.: *The Theory of Parsing, Translation, and Compiling*, Vol. 1, *Parsing*, p. 541, Prentice-Hall, Englewood Cliffs (1972).
6) Boyer, R. S. and Moore, J. S.: The Sharing of Structure in Theorem Proving Programs, in Meltzer, B. and Michie, D. (eds.), *Machine Intelligence*, Vol. 7, pp. 101-116, Edinburgh University Press, Edinburgh (1972).

付録

付図1に簡単な PAMPS 文法と，文法規則4)のコンパイル例を示す．ただしラベル $l16$ と $l17$ は省略した部分を指している．

(昭和58年7月20日受付)
(昭和59年1月17日採録)

```
1) SENTENCE() -> S(*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(*num)),*num) ->
          DET(*y,*num),ADJ(*z),NOUN(*x,*num).
5) NP(np(propn(*y)),*num) -> PROPN(*y).
6) PRED(pred(verb(*x),obj(*y)),*num1) -> VP(*x,*num1),NP(*y,*num2).
7) NP(vp(*x),*num) -> V(*x,*num).
8) DET(the,sing) -> [the].
9) ADJ(tall) -> [tall].
10) NOUN(man,sing) -> [man].
11) V(reach,sing) -> [reaches].
12) PROPN(new-york) -> [new-york].
```

```
                .
                .
                .
11:  uvar(1,0)      DET(*y,
     uvar(2,1)      *num),
     init(2,4)
     neck(4)
     gset           ADJ(
12:  uvar(1,2)      *z),
     gset(7)        NOUN(
13:  uvar(1,3)      *x,
     uref(2,1)      *num).
     pset(4,0)      NP(
     call(19)
     label(14)      np(  ),
     var(1)         *num) ->
14:  fn(2)          np(
     label(15)      noun(*x),
     label(16)      art(*y),
     label(17)      mod(*z),
     label(18)      nbr(*num)
     fnend          )
15:  fn(3)          noun(
     var(3)         *x
     fnend          )
16:  fn(4)          art(
     var(0)         *y
     fnend          )
```

```
17:  fn(5)          mod(
     var(2)         *z
     fnend          )
18:  fn(6)          nbr(
     var(1)         *num
     fnend          )
                      .
                      .
                      .
19:  enter
     entry(110)
     try-R("PRED",116)
110: try-D("SUBJ,117)
     exit
111: enter
     try-L("NP","ADJ",11)
     exit
112: enter
     entry(113)
     try-C("NOUN",12)
113: exit
114: enter
     entry(115)
     try-R("NP",13)
115: exit
                      .
                      .
                      .
```

付図 1 簡単な PAMPS 文法とそのコンパイル例
Fig. A.1 Simple PAMPS grammar and its compiled codes