自然言語処理 30－4
（1982. 3. 25）

新しいパーザPAMPSと そのコンパイラについて

上原邦昭、落谷亮、河合和久、豊田順一
大阪大学基礎工学部

1. はじめに

自然言語（特に英文）を解析するパーザとして、我々はPAMPS（PAttern Matching Parsing System）を開発してきた。文献(1)、(2) このパーザは解析制御のためにpattern matchingを用いているところに特徴がある。pattern matching (unification)は、LISPでのdata selector (car, cdr)、data constructor (cons)等を単一の機能として実現できるため、記述形式に一貫性があり、更にはside effectがないため、明晰性が保てるという利点がある。またパージングの制御に必要な機能の大部分は統語／意味情報の転送と判定であるが、これらもpattern matching 機能に統一できるという利点がある。

またパージングを効率よく行なうためには、Marcusも指摘しているように、入力データの利用（bottom-up）、予測の利用（top-down）、先読みデータの利用（looK-ahead）が必要である。文献(3) PAMPSでは、Earleyのアルゴリズムを上述の目的に合うように変形したものを用いている。Earleyのアルゴリズムでの Predictor に相当する部分は、LINGOLでのoracleと同様に文法規則を前処理して、ビットテーブルの形で保存している。bottom-up 過程で成長した部分構文木は、このテーブルによりfilterにかけられ、将来成長する可能性のあるもののみが残される。Earleyのアルゴリズムで用いられている先読み機能は、不用な部分構文木の生成を抑制することしかできない。我々のアルゴリズムでは更に"不用な予測"をも先読み機能により削除することにより、パージングの探索空間／時間を縮少している。ここで"不用な予測"とは、予め文法規則で予測されていた非終端記号のうちで、先読み語を最左端とする終端記号列として導出できないものをいう。言い換えると、予測非終端記号を設定するとき、その予測非終端記号が次の先読み語の親となりうるかどうか調べ、もし親となりえないならば、その予測非終端記号は構文木を組み立てるうえで不用になるということである。この先読み機能についても予め先読み語を計算しておくことができる。結果はoracleと同様に、ビットテーブルの形で保存している。

2. PAMPSのアルゴリズム

まず、Earleyのアルゴリズムでの項（item）に対応してゴールとフレーズを定義する。項〔A← α・Bβ, i〕j をゴール（Bβ, i, A, j）と呼ぶ。項〔A←α・, i〕j をフレーズ（A, i, j）と呼ぶ。$A \overset{*}{\Rightarrow} w\cdots$ という導出が存在するとき、即ちAから導出される任意の文形式のうちwが最左端に現われるとき、関係$R^*(A, w)$が成り立つ。$S \overset{*}{\Rightarrow} \cdots Aw\cdots$ という導出が存在するとき、即ちwがAの直接右隣りに並ぶとき、関係Follow (A, w)が成り立つという。

パージングは左から右へ1語ずつ進行し、部分構文木はbottom－upに作られる。フレーズ(B, i, j)が生成された段階で、以下の4種類の動作が実行される。

1) try C: ゴール(BCα, i, A, k)(α=λ でもよい)がある場合、先読み語 w_{j+1} について $R^*(C, w_{j+1})$ ならば、ゴール(Cα, j, A, k)を設定する。
2) try R: ゴール(B, i, A, k)がある場合、先読み語 w_{j+1} について Follow (A, w_{j+1}) ならば、フレーズ(A, k, j)を生成する。
3) try D: ゴール(Cγ, i, E, k)かつ $R^*(C, A)$ となる規則 A←Bがある場合、先読み語 w_{j+1} について Follow (A, w_{j+1}) ならば、フレーズ(A, i, j)を生成する。
4) try L: ゴール(Cγ, i, E, k)かつ $R^*(C, A)$ となる規則 A←BDα がある場合、先読み語 w_{j+1} について $R^*(D, w_{j+1})$ ならば、ゴール(Dα, j, A, i)を設定する。

以上の4種類の動作を繰り返し、もはや実行すべきことがなくなったならば、更に1語読み込み(w_{j+1})動作を続ける。新たに先読み語は w_{j+2} となる。

説明の都合上、実際のアルゴリズムと若干異なるので、詳しくは文献(2)を参照のこと。以上のアルゴリズムをEarleyの文献[4]で示された例題に適用した場合以下のような効果があることがわかる。

表1. 先読みの効果

	文法	AE	UBDA	BK
	入力文	a+a×a	X^4	X^4
Earley 先読みなし	フレーズ生成数	14	18	24
	ゴール設定数	8	10	4
Earley(1語先読み)	フレーズ生成数	11	15	21
	ゴール設定数	7	10	4
PAMPS(1語先読み)	フレーズ生成数	11	15	21
	ゴール設定数	4	6	3

3. 文法と辞書の記述

文法規則は、各非終端記号に引数を与えた拡張CFG rule である。引数として定数、変数、更に変数と定数からなる複合項(データ構造)が許される。

```
SENTENCE(ques) <-
  auxil(*atype,**,**,*nums),np(*nums),vp(*atype,**,**).
```

辞書項目は、終端記号、非終端記号、特徴素リストからなる。特徴素リストは特徴素タイプ、特徴素のペアからなる。

```
(they n(npos,ns(x,o,x),them)
     ((ntype pro) (pers third)))
```

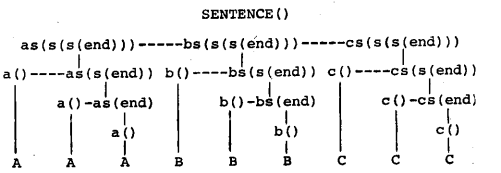
特徴素リストは、システム組み込み関数 F-get (終端記号、非終端記号、特徴素タイプ、特徴素が代入される変数)によりアクセスされる。

4. 解析制御

自然言語を解析する際、CFGでは能力が弱いので、何らかの補強が必要となる。このため、大別して1) 部分構文木の上位レベルノードに、あるいは同レベルの

```
SENTENCE() <- as(*x),bs(*x),cs(*x).
as(s(*x)) <- a(),as(*x).     as(end) <- a().     a() <- A.
bs(s(*x)) <- b(),bs(*x).     bs(end) <- b().     b() <- B.
cs(s(*x)) <- c(),cs(*x).     cs(end) <- c().     c() <- C.
```

図1. $a^n b^n c^n (n \geq 1)$ の導出木



他ノードに統語/意味情報を送ること、2) 送られた情報を判定してその規則の適否を決めることが必要である。これらの機能はpattern matching として単一化することが可能である。図1に言語 $L(G)=\{a^n b^n c^n \mid n \geq 1\}$ を導出するための規則と、その導出例を示す。各非終端記号に付け加えられた引数により、情報の転送、判定がスマートに表現できる。実際の言語解析でpattern matching をどのように用いるかについては、文献(2)でいくつか例が示されている。

5. Extra Condition[5]

文法記述においては、文法規則数、カテゴリー数の増加を防ぐため、規則内にその適用条件を埋め込み、条件の判定により規則の適否を決定することが必要である。PAMPSでは、この規則適用条件記述をprocedure call として規則内に含むことを許している。

```
head(*ntype,*nums,third,*word) <-
    n(pos,*nums,*word),F-get(*word,ntype,*ntype), in(*ntype,[comm,prop] .
in(*x,[*x *1]).
in(*x,[** *1]) -> in(*x,*1).
```

上の文法規則では、中カッコでくくられている部分が規則適用条件となっている。中カッコの中はホーン節集合である。この記法を文献(6)の定義に従い、local ホーン構成と呼ぶ。中カッコの中がlocalに証明されれば、適用条件を満足したとして、更にパージングは進行する。尚中カッコの中では top-down、depth-first で証明が行なわれる。

6. PAMPSコンパイラ

PAMPSのような述語論理的文法記述の場合、宣言的意味論と手続き的意味論の両方を持つことが従来より指適されている。文献(5) 文法規則が宣言的意味論を持つと見なした場合、文法規則と共にそれを解釈実行するためのメカニズムが必要となる。前章までの考察は主としてこのメカニズムをどのように構成するかについての議論であった。本章以降では、文法規則を手続き的意味論に基づき、文法規則自身を低レベル機械語に翻訳し、それを直接実行するようなメカニズム、即ちPAMPSコンパイラと仮想機械PAMPS マシンについて言及する。また、2章のアルゴリズムによって文法規則を解釈するメカニズムをPAMPS インタプリタと呼ぶ。

6.1. PAMPS コンパイラの概念 文法規則をインタプリタにより解釈実行すれば、(1) 適用可能な文法規則を規則集合の中から逐一検索しなければならず、(2) pattern matchingも統一的なアルゴリズムで実行しなければならない。

まず(1)の問題について考える。次の文法は、入力記号列によって駆動されるオートマトンとしてとらえることができる。即ち、ある状態からスタートして、入力記号に応じた遷移によって別の状態に移る様子をラベル付き有向グラフとして表現できる。ここで、各ノードは状態に、各アーク間のラベルは状態遷移の際に必要なアクション(先読みのチェック、トップダウン過程の予測チェック、pattern

matchingの処理）を示している。

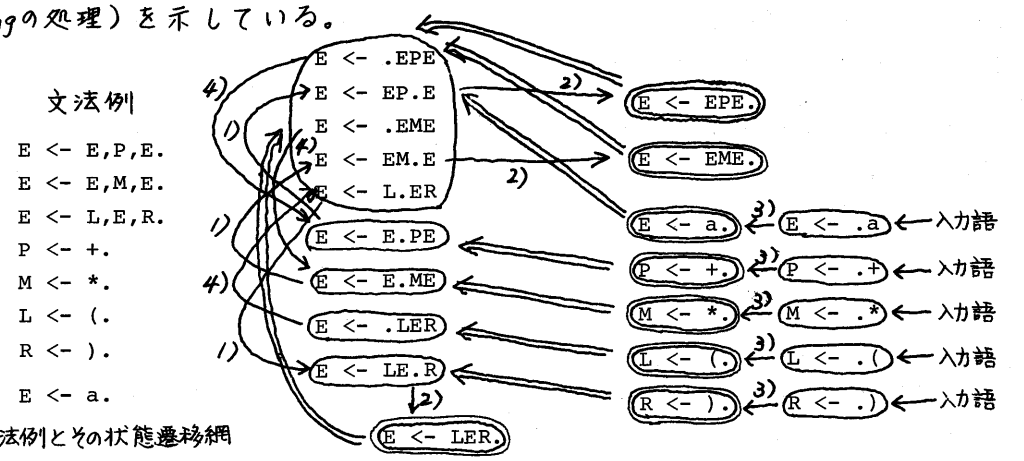


図2. 文法例とその状態遷移網

ラベルtryC 等は、２章の４分類のアクションに対応している。状態２重丸ではフレーズが生成され、そのフレーズはqueueに入れられる。queueの先頭要素から順次、状態遷移（手続き呼び出し）が行なわれる。queueの要素が無くなれば、入力を１語読み込む。

上述のように文法入力時に、あらかじめ道筋に関する情報を取り出しておき、実行時には、この道筋に従ってパージングすることで、無駄な規則検索の手間を省くことができる。

パージング制御に使用されるpattern matchingは単純な代入と同値判定に還元できる。

```
Sentence() <- Noun-phrase(*x),Vreb-phrase(*x).
Noun-phrase(Single)
```

この場合、ゴールNoun-phraseとフレーズNoun-phraseとのpattern matchingは、*x：＝Singleという代入に帰着する。

```
Sentence() <- Noun-phrase(Single),Verb-phrase(Single).
Verb-phrase(Plaural)
```

この場合に引数どおしでの必要な操作は、Single≒pluralを判定することである。このように非終端記号の各引数を、それぞれの出現状況に応じていくつかの場合に分け、それぞれに必要な操作を低レベル語に翻訳し、処理を高速化する。また、PAMPSはアルゴリズム上bottom upに動作するので、手続き処理用コード（pattern matching用コード）にコンパイルする部分は、文法規則の右辺である。左辺は、手続き呼び出し用コードにコンパイルする。以上がPAMPSコンパイルの原理である。PAMPSマシンはFortranにより実現された仮想機械である。

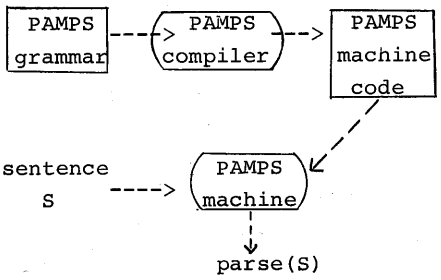


図3. PAMPSマシン概念図

6.2. PAMPSマシンコード　文法規則A← E、P、T.は図４のように翻訳される。pattern matching用コードは以下の４グループに分けられる。

1) 変数が、その文法規則内で初めて出現する場合は、相手引数の値を代入する。
2) 変数が、文法規則内で一度しか出現しない場合は、いかなる処理も必要としない。
3) 定数あるいは複合項の場合は、相手引数を調べ、同値の場合のみ成功、もし相手変数が具体化していない場合は、相手に代入する。
4) 変数が既に文法規則内で出現している場合は、自分自身に代入された値を調べ、相手側の引数とpattern matchingする。

更に、パージングコードとして、前述のtry C 等の他、手続き呼び出し用コード、手続き呼び出し制御用コードなどがある。PAMPSマシン用コードは、これら約50種類のコードから構成されている。

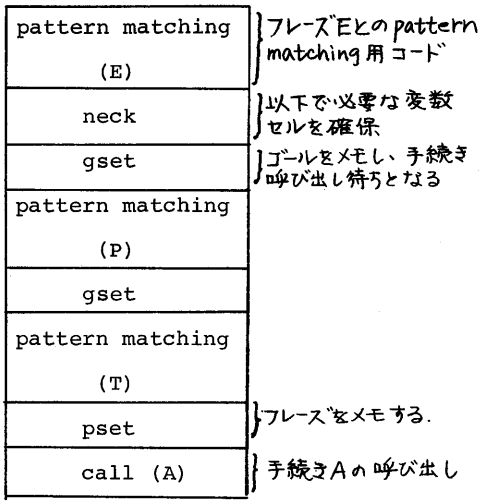


図4. 文法規則 A←EPTの模式コード列

7. PAMPS マシンのデータ構造

PAMPS マシンのデータ構造は、Boyer-Moore の sharing structure 技法に基づいている。しかし、PAMPS は本質的に bottom-up、parallel で動作するため、Warren 等の Prolog で用いたデータ構造のように、1つの変数に1つの変数セルを与えるだけではうまく処理できない。たとえば、1つのフレーズが複数個のゴールと、あるいは1つのゴールが複数個のゴールと同時に pattern matching する可能性がある。この状態を競合(conflict)と呼ぶ。我々は、各時点で行なわれた pattern matching に個有の番号を与え、更に各 pattern matching 間の祖先—子孫関係をテーブルとして表現することにより、競合問題を解決した。

図5について考える。フレーズ B(1)と規則 A(X)←B(Y)、C(X)、D(X). が pattern matching した場合、ゴール C(X)が設定される。この場合、変数Xのセルには何も書かれていない。--- ①

ゴール C(X)とフレーズ C(2)の pattern matching が行なわれると、variable stack 上の変数Xのセルに、値 int(2)とこの時点での pattern matching 番号2が書き込まれる。競合はまだ起っていないので、conflict pointer は nil である。--- ②

フレーズ C(3)が生成されると、ゴール C(X)と pattern matching する。変数Xのセルには値 int(2)が書かれているが、並列性のため、この値をそのまま変数Xの値と

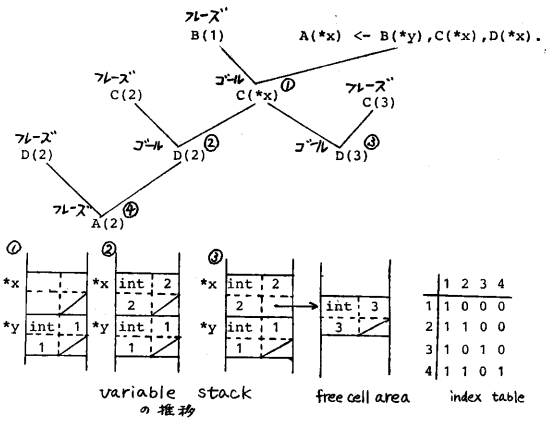


図5. データ構造例

みなすことはできない。そこで、このpattern matchingでの変数文のセルとして、新しいセルがFree cell areaから1つ取られ、新しいセルに値int(3)とpattern matching番号3が書かれる。同時にvariable stack上にある変数文のセルのconflict pointerは、Free cell areaの新しいセルを指す。… ③

次にフレーズD(2)が生成され、ゴールD(2)とpattern matchingする。まず、ゴール側の引数文のdereferenceが行なわれる。変数文のセルには値int(2)が書かれているが、これをそのまま変数文の値と判断することはできない。(競合の可能性) その値が現在のpattern matchingでの真の値かどうかの判定のために、index tableを用いる。index tableはビットマトリックスとして構成され、i行j列のビットは、pattern matching番号jの節点が、pattern matching番号iの祖先であるかどうかを示している。変数文のセル上の値int(2)のpattern matching番号は2で、現在のゴールのpattern matching番号は2であるからindex table2行2列を調べる。2行2列にはビットが立っているので、このセルは現在のpattern matchingのゴールの祖先であることがわかる。これより文の値はint(2)となり、pattern matchingは成功する。… ④

ボトムアップ、パラレル型パーサでは途中結果がすべて残されるが、自然言語入力文は高々数十語であるので、我々のデータ構造で十分対処可能である。

9. 計算機実現

PAMPSコンパイラとPAMPSマシンの有効性を確認するために、計算機実験を行なった。PAMPSインタプリタをLISP1.9でインプリメントし、それをコンパイルしたものと、PAMPSマシンをFortranでインプリメントしたものを、同一文法と同一入力文を用いて速度比較した。その結果PAMPSマシンの方が約1.6倍スピードアップできることがわかった。PAMPSマシンはプログラムの最適化をしていないので、まだいくらか高速化できると思われる。現在は、PAMPSコンパイラとPAMPSマシンの最適化とPAMPSマシン用の自然言語処理用文法を平行して作製している。文法規則は総数で約150個ほどある。

10. 参考文献

1. K. Uehara et al. "A Pattern Matching Directed Parser: PAMPS" ACL/LSA sessions on Computer Modeling of Linguistic Theory, NY (1981).
2. 河合他. "パターンマッチングによるパーザPAMPSについて" 信学技報 AL81-94 (1982).
3. M.P. Marcus "A Theory of Syntactic Recognition for Natural Language" M.I.T. Press (1980).
4. J. Earley "An Efficient Context-Free Parsing Algorithm" CACM Vol.13, No.2 (1970).
5. D.H.D. Warren "Impremented Prolog" D.A.I. Research Report No.39, 40 (1977).
6. 渕 "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 1-2 (1977).