表駆動によるＢＵＰの高速化

Table-Driven Bottom Up Parser in Prolog

田村直良, 沼崎浩明, 田中穂積
Naoyoshi Tamura, Hiroaki Numazaki, Hozumi Tanaka
横浜国立大学工学部電子情報工学科, 東京工業大学工学部情報工学科
Yokohama National University, Tokyo Institute of Technology

BUP (Bottom Up Parser in Prolog) については様々な技法の高速化が成され、また左外置文法(Extra position Grammars)や熟語への対応など記述性の面でも色々な改良がみられた。本論文では、これらの資産を損なう事なく高速化を達成する手法について報告する。従来のBUPでは、生成規則の選択については特別な機構は持っていなかったが、本方式では駆動表と先読みの機構から非決定性を制限する。本方式は、原理的には、BUPとLC(1)パーザの動作にかなりの対応があることによっている。LC(1)パーザは、LC(1)なる文脈自由言語のあるクラスに属する文に対して決定的に構文解析できる。本方式は、これを一般の文脈自由文法に拡張したものであるが、拡張により生ずる動作の非決定性は、従来のBUPで行われているバックトラックの範囲にある。その他本論文では、実現と560個程の生成規則からなる実用規模の文法に対する実験結果について報告する。

１．はじめに

BUPは、松本らにより開発されて以来[Matsumoto]、様様の高速化の技法が取り入れられて来た[田中]。また、文法の記述の面でも、XG[今野]、TRIE辞書による効率化、熟語への対応[上脇]などの環境が整備され、自然言語処理環境LangLABとして整備された[徳永]。本論文では表駆動によるBUPの高速化について述べる。

文脈自由言語には、LL(k)、LR(k)、LC(k)などのクラスがあり、それぞれについて決定的なパージング法が提案されている。これらのパージング法に対して、一般の文脈自由言語の文を解析させると、その動作は非決定的になるが、無造作に生成規則の適用を試みるのではなく、ある程度決定的に動作する処理過程もある。今回報告する方法は、LC(1)パーザを基にしたもので、BUPに適用してその動作の非決定性を減少させるものである。一方、LR(k)ベースのパーザでは、還元すべき規則が決定されるまでは適用可能な規則の閉包を状態とするなど効率が良く、スタックの操作を工夫した［富田］は、さらに高速である。今回の対象であるBUPを高速にするにしても、そこまでは到達しないかも知れない。しかし、LRパーザは先読み記号(列)と駆動表、スタックにより完全に制御され、駆動表は与えられた文法規則のみにより決定される。このため、熟語に対しても個々に規則を記述しなければならず、関係代名詞節の記述に有利な左外置形の文法記述も難しい。我々は、上記のような、BUPで実現されているこれまでの蓄積を損なう事なく、わずかの改良で高速化を達成させたい。

解析システムに高速性は常に求められるものであるが、単に速度を求めるのなら、特定の使用を前提としてそれなりの高級言語でシステムを実現することにより高速化が可能であろう。しかし、logic grammarもしくはlogic programmingの範囲内でいかに高速な処理系をあたえるかが、開発環境までも含めた場合には重要である。また、解析システムを利用する側としては、システムの変更による影響は極力避けたいものである。よって一つの方向として、ここで述べるような高速化が意味を持つことになる。

以下第２章では、本方式の原理についてLC(1)パーザとの対応関係を見ながら説明する。第３章では実用的な規模の文法の使用を前提とした実現について説明する。また、第４章では規則数560個程の英語の文法に対して実際に構文解析を行った結果を示し、本方式を検討する。

２．構文解析の原理

ここでは、BUPを表により駆動するための原理について説明する。

２．１　BUPの原理

はじめに、基になるBUPの原理について説明するが、まず left corner を定義する。なお、本論文中、Ｎは非終端期号の集合を、Σは終端期号の集合を、Ｐは生成規則の集合を、Ｓは開始記号を表す。また、Ｖは文法記号の集合（ＮＵΣ）である。

［定義］ Left Corner

文脈自由文法G＝（V，Σ，P，S）において、

$X_1 \in V$を$X_0 \rightarrow X_1 \cdots X_n \in P$の、または単に（ある生成規則の）left cornerとよぶ。

BUPは、原理的には文脈自由文法G＝（V，Σ，P，S）に対して以下の(1)(2)(3)から作られる公理から、(4)を証明する過程を構文解析としたものである。ここで、

LC(s,g,w)　(s,g∈V,w∈Σ*)は、left corner sとそれに続く終端記号列wからgを根とする部分木が造られることを表す。また、ST(s,w)は、終端記号列wからsを根とする部分木が造られることを表す。(2-1)は、各生成規則ごとにつくられる。図1に示すように、gを根とする部分木の left corner s_0とそれ以後の文字列w、部分木$s_2,\ldots,s_n$があるとき、gを根とする部分木のleft corner s_1とそれ以後の文字列 $w_2 \ldots w_n w$があることを示している。(2)はg自身がgを根とする部分木のleft corner であることを示している。また(3)は、gを根とする部分木のleft corner sとそれ以後の文字列wがあって、sから終端記号aが導出されるとき、文字列awからgを根とする部分木が作られることを示している。証明すべき定理は(2-4)であり、入力文字列xを葉とする構文木の根が開始記号Sであることを証明するものである。また、それぞれに対応して、BUPにおける実現を示す。

(1) 全ての生成規則 $s_0 \rightarrow s_1, s_2, \ldots, s_n \in P$に対して、

```
LC(s1,g,w2...wnw) ←                          (2-1)
        ST(s2,w2),....,ST(sn,wn),LC(s0,g,w)

s1(G,D1,D0) :-                               (2-1')
    goal(s2,D1,D2),
    goal(s3,D2,…),
    …,
    goal(sn,…,Dn),
    s0(G,Dn,D0).
```

(2)　LC(g,g,ε) ←　(2-2)

ただしg∈V、またεは空文字列

```
s(s,X,X).                                    (2-2')
```

(3)　ST(g,aw) ← LC(s,g,w)　(2-3)

ただしaw∈Σ*、s → a ∈P

```
goal(G,D1,D0) :-                             (2-3')
    dictionary(C,D1,D10),
    P =.. [C,G,D10,D0],
    call(P).
```

(4)　← ST(S,x)　(2-4)

ただしSを開始記号、xを構文解析すべき文字列とする。

```
:?- goal(s,X,[]).                            (2-4')
```

BUPでは、(2-1')をBUP節と呼ぶ。手続き的に解釈すると、まず（ボトムアップ的に）s1が得られたとき、この節が呼び出される。すると、s2,s3,....,snの生成を（トップダウン的に）呼び出し、これらが成功するとs0が（ボトムアップ的に）完成したとしてs0を呼び出す。また、(2-2')を停止節と呼ぶ。これは、手続き的に解釈すると、sを目標に（ボトムアップ的に）解析を進めている途中left corner sが得られたとき、これが目標に一致したことを表す。(2-3')は、goal節と呼ばれる。ただし、

dictionary(C,D1,D10)

は、差分リストD1の先頭がカテゴリCに属し、その残りの文字列がD10であることを示す。これは、手続き的には、（トップダウン的に）Gを作るとき、まず1文字読んでそれをleft cornerとして解析を進めることを表す。

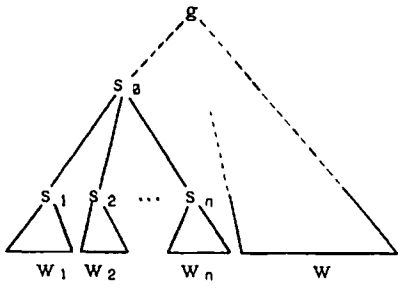

図1　式(2-1)が表す構造

2．2　BUPにおける非決定性

以前のBUPの高速化の際に導入されたwf_goal, fail_goal, linkは、失敗成功を記録することにより解析の無駄を省くものであり、非決定性を根本的に解消するものではない。ここでは、まずBUPにおける非決定性について考察する。BUPには、次のような非決定性がある。

① left cornerが決まった後、それをどの生成規則に属するものとするかについての選択。つまり(2-1)式の形式で表された生成規則について、この式の左辺（left corner s1）と同じ形を持つ式が複数ある場合その中からどれを選ぶかと言うことである。

②目的（goal）の部分木が完成したとするか、goalと同じ名前のleft cornerが完成したとするかについての判断。つまり、(2-1)の形式の規則において s_1=gの場合、(2-1)の形式の規則を選ぶか(2-2)の形式の規則を選ぶかと言うことである。本来この非決定性は、左再帰(left recursion)を持たない文法については存在しない。

③同義語について。一般的なパーザでは、各単語に対して辞書によりそのカテゴリや意味情報を求める。このとき単語に多義性がある場合にはどの意味を選ぶかについての非決定性がある。

④補強項について。生成規則中に埋め込まれた意味処理プログラムに非決定性がある場合が考えられる。

⑤生成規則中の選言について。選言の範囲にleft cor-

nerを含むことはBUPでは禁止されている。それ以外の部分に選言がある場合は、①と等価な非決定性となる。

このうち③、④は、構文解析の方式によらない本質的な非決定性である。また④は、Prologのような言語で補強項を記述する場合の利点であり、この非決定性を許さない場合が多い。⑤は、単に文法の記述上の便宜からくるものであり、本研究においては扱わないことにする。BUPにおける非決定性のうちBUPの解析原理ゆえにもたらされたものは①、②である。以後①による非決定性を適用可能性、②による非決定性を左再帰性と呼ぶことにする。次節で述べる我々の方式は、LC(1)パーザとBUPの類似性を利用して、これらの非決定性をLC(1)パーザの駆動表により減少させることによって高速化をもたらすものである。

2．3　LC(k)の紹介[Aho]

ここで今回の高速化のきっかけとなった LC(k) (Left Corner Grammars with k-lookahead)とそのパーズ法について説明する。まず、left corner導出を定義する。

[定義] (left corner導出)
最左導出(leftmost導出)

$$S \underset{lm}{\overset{\cdot}{=}>} wA\delta \qquad (w\in\Sigma^*,\ \delta\in V^*) \qquad (2\text{-}5)$$

があり、かつ非終端記号Aが(2-5)においてAを生成した規則のleft cornerでないとき、次のように書き、これをleft corner導出と呼ぶ。

$$S \underset{lc}{\overset{\cdot}{=}>} wA\delta \qquad (2\text{-}6)$$

次にLC(k)を定義する。なお、$FIRST_k(\beta\gamma\delta)$は、記号列$\beta\gamma\delta$から生成される終端記号の列について、先頭からk文字以内の部分記号列の集合を表す。

[定義] LC(k)
次の条件が満たされるとき、G＝(V, Σ, P, S)をLC(k)と呼ぶ：

$S \underset{lc}{\overset{\cdot}{=}>} wA\delta$ とし、任意の先読み記号列uに対して、 $A \overset{\cdot}{=}> B\gamma$ であり、かつ(1)(2)を満たすような生成規則 $B\rightarrow\alpha$ が高々1個存在すること。

(1)(a)もし、$\alpha = C\beta$, $C\in N = V-\Sigma$なら、
　　$u\in FIRST_k(\beta\gamma\delta)$、かつ
　(b)さらにC＝Aであるときには、
　　uは$FIRST_k(\delta)$の要素ではない。
(2)もしαが非終端記号から始まらなければ、
　　$u\in FIRST_k(\alpha\gamma\delta)$。

条件(1a)は、wと、Cから導出される文字列、先読み記号列$u\in FIRST_k(\beta\gamma\delta)$が分かれば、規則$B\rightarrow\alpha$が一意に決定されることを保証するものである（図2a）。条件(1b)は、left corner Aが左再帰であるとき、Aのインスタンスが認識されると、先読み記号列uにより、それが規則B→Aγのleft cornerであるのか、wAαのAなのか決定できることを表している（図2b）。条件(2)は、αが非終端記号で始まらないとき$FIRST_k(\alpha\gamma\delta)$が分かると、left corner導出においてwBがあれば、次の規則がB→αであることが一意に決定されることを示している（図2c）。

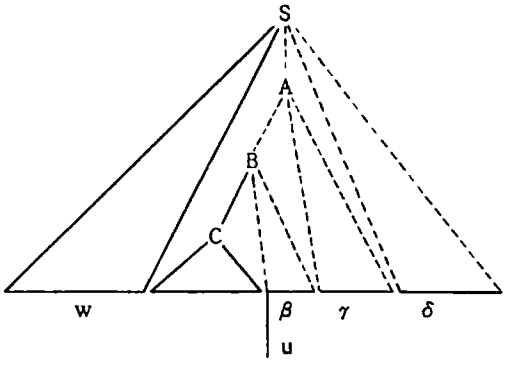

図2 a　条件（1 a）

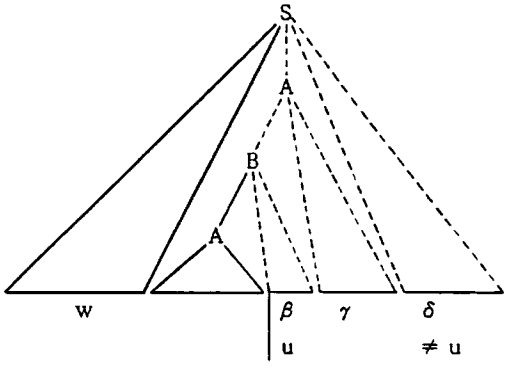

図2 b　条件（1 b）

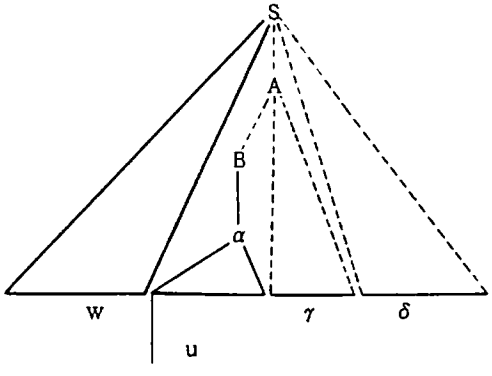

図2 c　条件（2）

LC(1)については、次のようなパーザで決定的な構文解析が可能である。

まず、スタックプッシュ・ダウン・シンボルの集合を$\Gamma = N\cup\Sigma\cup(N\times N)\cup\{\$\}$と定義する。ここで$はスタックの底であり、N×Nは[A,B]なる非終端記号の組（lc pairと呼ぶ）であり、Aは現時点の解析の目標、

Bは現在認識されたleft cornerを表す。パーザは駆動表Tをもとに駆動される。Tは、$\Gamma \times (\Sigma \cup \{\varepsilon\})$ から $(\Gamma^* \times (P \cup \{\varepsilon\})) \cup \{pop, accept, error\}$ への写像であり、現在のパーザの状態が入力に対していかに変化するかを示すものである。

つまり、パーザのconfigurationを $(w, X\alpha, \pi)$ で表すとして（残りの入力 $w \in \Sigma^*$、スタック $X\alpha$、Xがスタックのトップ、部分パーズ π）、

(1) もし $T(X,a)=(\beta, i)$, $X \in N \cup (N \times N)$ なら、
$(aw, X\alpha, \pi) \vdash (aw, \beta\alpha, \pi i)$

(2) もし、$T(a,a)=pop$なら、
$(aw, a\alpha, \pi) \vdash (w, \alpha, \pi)$

xを解析する文字列、Sを開始記号とし、
$(x, S\$, \varepsilon) \vdash^* (\varepsilon, \$, \pi)$
となる遷移が得られたとき、π がxのパーズとなる。

駆動表は、LC(k)の定義に従って次のように生成される。

(1)(a) $T([A,C],a) = (\beta[A,B], i)$
ただし、任意の $A \in N$, $a \in FIRST_1(\beta\gamma\delta)$
$S \underset{lc}{\overset{\cdot}{=}>} wA\delta$ かつ $A \overset{\cdot}{=}> B\gamma$
に対して、p_i: $B \rightarrow C\beta$ $(B, C \in N)$

(b) $T(A,a) = (\alpha[A,B], i)$
ただし、任意の $A \in N$, $a \in FIRST_1(\alpha\gamma\delta)$
$S \underset{lc}{\overset{\cdot}{=}>} wA\delta$ かつ $A \overset{\cdot}{=}> B\gamma$
に対して、p_i: $B \rightarrow \alpha$
（α の先頭は非終端記号ではない）

(2) $T([A,A],a) = (\varepsilon, \varepsilon)$
ただし、任意の $A \in N$, $a \in FIRST_1(\delta)$
$S \underset{lc}{\overset{\cdot}{=}>} wA\delta$

(3) 任意の $a \in \Sigma$ に対して、
$T(a,a) = pop$

(4) $T(\$, \varepsilon) = accept$

(5) それ以外、
$T(X,a) = error$

2．4　高速化の原理

本方式における高速化は、2．2節で述べた非決定性を、前節で述べたLC(1)のパージングを基に減少させることを基本にしている。もともとLC(1)パーザは、LC(1)言語に対して決定的に構文解析するものである。LC(1)パーザをDCG等により記述された一般的な文脈自由言語に応用した場合、その言語に属する文のみを受理するパーザが容易に得られる。しかし、LC(1)言語の性質を満たさない部分の可能性を調べるために、構文解析操作は非決定的になる。ところが、この非決定性は BUPの構文解析における非決定性よりも、先読みの機構によりガイドされて、かなり制限されたものである。この非決定性をバックトラックにより全解探索したとしても高々BUPと同等である。高速化の原理もここにあるわけである。この節では、高速化の原理を述べる。

2．4．1　BUPとLC(1)パーザの比較

BUPとLC(1)パーザの動作は対応するところが多い。BUPの各動作を駆動表の生成法からみたLCパーザの動作と比較してみる。

(1)BUP節：(2-1')の形式のBUP節は、駆動表の生成(1a)に対応する。(1a)は、Aを目標に解析を進め、left corner Cが得られた時点の動作を指示するものである。このとき、Aをleft cornerに持つ生成規則 p_i の残りの部分 β を処理すると、新たにleft corner Bが得られることを示す。

(2) goal節：(2-3')の形式のgoal節は、駆動表の生成(1b)に幾分似ている。goal節では、1文字を読みそのカテゴリをleft cornerとしてBUP節を呼び出す。これに対して(1b)では、終端記号で始まる生成規則 p_i を解析し、その規則の左辺Bを新しくleft cornerとして解析を進めることを指示する。右辺が1個の終端記号から成る場合が goal節での処理に対応する。

(3)停止節：(2-2')の形式の停止節は、left cornerと解析の目標が一致した場合であり、駆動表の生成(2)に対応する。BUPでは、goal節から始まった一連の処理が目標に到達して終了する部分であり、LC(1)パーザでは、スタックをポップ・アップさせる項目である。

2．4．2　「どの規則を選ぶか」

前節での対応を基に、2．2節で述べたBUPにおける非決定性を減少させる手段を考える。

ここでは、まず適用可能性について考える。適用可能性は、ある目標の下でleft cornerが決まった時点で、次にどのBUP節を選ぶかに関する非決定性であった。これは、前節の (1)から BUPの動作とLC(1)パーザが完全に対応していることが分かり、LC(1)パーザの駆動表をそのまま用いることが出来る。すなわち、(2-1')の形式のBUP節に駆動表を検索する部分を加える。

全ての生成規則 $pi=s_0 \rightarrow s_1, s_2, \ldots, s_n \in P$ に対して、

```
s1(LC, DI, DO) :-                              (2-7)
    lh(C, DI),
    t(LC, C, NLC, pi),
    goal(s2, DI, D2),
    goal(s3, D2, …),
    …,
    goal(sn, …, Dn),
    s0(NLC, Dn, DO).
```

ここで、lh(C, DI)は差分リストDIの先頭の文法カテゴリがCであることを表すものであり、t(LC, C, NLC, pi)は、駆動表を検索する部分で、LC(1)の駆動表の項目

T([G, s1], C)=(s2...sn[G, s0], pi)

と、完全に等価である。ただし、LC、NLCは、それぞれ[G,s1]、[G,s0]をコード化したものが値として扱われる。これは、現在のleft corner s1と目標からなるlc pairがLCであり先読み記号がCであるときに、用いるべき生成規則がpiであり、新しいleft cornerと目標からなるlc pairがNLCであることを表している。駆動表にlc pair LCをエントリーとする項目があるかどうかは、BUPで用いられていたlink節による到達可能性に対応するものであり（後述）、ここではさらに先読み記号と関係させることで制約をより精密なものにしている。

なお、文法記号si（$2 \leq i \leq n$）が終端記号の場合、

```
goal(si,Dj,Di)
```

の代わりに、

```
Dj=[si|Di]
```

とするが、これはLC(1)駆動表の生成(3)に対応する。

2．4．3　「部分木は完成したか」

次にBUPの非決定性の内「目的（goal）の部分木が完成したとするか、goalと同じ名前のleft cornerが完成したとするかについての判断」（左再帰性、2．2節②）であるが、これは2．4．1節(3)の対応関係からLC(1)パーザと完全に対応しているのが分かる。よって、(2-2')の形式の停止節に駆動表の検索の部分を加える。

つまり、全ての非終端記号sについて、

```
s(LC,X,X) :-                                    (2-8)
    lh(C,X),
    follow(LC,C).
```

ここでfollow(LC,C)は、LC(1)駆動表の項目

$T([s,s],C)=(\varepsilon,\varepsilon)$

と等価である。ただしLCは、lc pair[s,s]をコード化したものである。これは、sを解析の目標としているときにleft corner sが完成したことに対応している。そのとき、先読み記号Cが許される状況にあるかどうかにより動作を制限するのである。駆動表の生成(2)から分かるように、Cがsに続くことができるかどうかを制約として用いる。

2．4．4　goal節の対応

2．4．1節(2)にあるように、goal節は、生成規則の右辺が1個の終端記号からなる場合のLC(1)パーザの動作と対応している。ただしこれをgoal節で利用するためには次のように修正する必要がある。

駆動表の生成法

(1)(b') $T'(A,a) = ([A,a],\_)$

ただし、任意の $A \in N$ に対して、

$$S \underset{lc}{\Rightarrow} wA\delta \quad \text{かつ} \quad A \Rightarrow a\gamma \qquad (a \in \Sigma)$$

これに対応して、新しいgoal節は次のようになる。

```
goal(G,DI,DO) :-                                (2-9)
    dictionary(C,DI,DIO),
    lc(G,C,LC),
    P =.. [C,LC,DIO,DO],
    call(P).
```

lc(G,C,LC)は、LC(1)駆動表の項目

$T'(G,C)=([G,C],\_)$

と等価である（ただしLCは[G,C]をコード化したもの）。これにより読み込まれた先読み記号が適したものであるかどうかが確認される。適したものである場合、その先読み記号を新しくleft cornerとして目標であるGと組み合わせてlc pair LCを作る。left cornerと目標Gからなるlc pairが存在するかどうかは、BUPにおけるlink節による制約に対応している。

3．実現について

この章では、これまでの議論に基づいて実現された構文解析システムBUP-TDについて、実現上のいくつかの事項について説明する。筆者らは、当初から生成規則の数560個程の実用的な文法での使用を前提にシステムの開発を進めてきた。原理的にはさほど重要でなくても文法の規模故実用上効率化しなければならない部分もある。これらについても多少触れることにする。なおシステムはSun-3/140におけるC-Prolog（インタプリタ）上に実現されている。

3．1　駆動表の実現

駆動表は、実験で用いた文法に対してerror以外の項目が35000個程である。かなり大規模な表であるために、一項目を検索するのに経過時間1秒以上を要してしまう。BUP-TDでは、C-PrologとCプログラムの結合性の良さを利用して、駆動表はCプログラムの静的領域上に実現している。また、表の検索は文法記号またはlc pair（のコード）と先読み記号をキーにハッシングを用いている。

検索の高速化のためにProlog以外の手続き的プログラムの使用を許すとすると、パーザの主要部分、極論すればシステム全体をその言語で書き直せば良いのではないか、という反論が生ずるかも知れない。本研究においては、表駆動による高速化を重点的に調べるために余計なオーバーヘッドは減らすとの立場を取っている。

3．2　先読みの機構

BUP-TDでは、BUP節、goal節、停止節で入力文字列のカテゴリが調べられる。語に多義性がある場合にも、いろいろな箇所から参照される同一な語に対してそのカテゴリは一貫していなければならない。一方、可能な多義性は全て調べなければならない。これらを単純に解決するには、全ての可能な組み合わせのカテゴリの列をあらかじめ作っておくことである。しかし、これでは無駄な計

算を繰り返すことが多い。ここでは、一旦辞書を引くとその情報（カテゴリ名）を制御が移る部分に順次渡していく。そのために、次のように各述語に２つの引数を用意する。

goal(G,....,Fst,Fol)、

left corner nに対して n(G,....,Fst,Fol)

Fstは、goal節の場合目標をGとするときの先頭のカテゴリ名である。left corner nについてはnに続く文字列の先頭のカテゴリである。また、Folは、Gを目標とした入力の部分の解析が成功したときこれに続く部分の最初のカテゴリ名である。

図３(b)に、DCGで表された(a)の規則をこの形式に変換したものを示す。FstとFolは、差分リストと同様に各述語に流される。先読みのための述語lh(Fst,I)は、Fstがバインドされていないときのみ実際に辞書を引くように変更されている。図３(c)は、停止節の例である。このなかで、$35は、lc pair [noun_phrase,noun_phrase]をコード化したものである。

```
noun_phrase(np(Det,adj(Adjs),n(Noun))) -->
    determiner(Det),
    adjectives(Adjs),
    noun(Noun).
```

図３(a) 例えば１４番目の生成規則

```
determiner(G,Det,Info,I,O,Fst,Fol) :-
    lh(Fst,I),
    t(G,Fst,NG,14),
    goal(adjectives,Adjs,I,I1,Fst,F1),
    goal(noun,Noun,I1,I2,F1,F2),
    noun_phrase(NG,np(Det,adj(Adjs),n(Noun)),
                Info,I2,O,F2,Fol).
```

図３(b) (a)の変換

```
noun_phrase($35,I,I,S,S,Fol,Fol) :-
    lh(Fol,S),
    follow($35,Fol).
```

図３(c) 停止節の例

３．３　これまでの諸技法との融合

本方式は、BUPに対してこれまでに行われてきた高速化や記述性の向上のための手法とは本質的に独立の手法であり、併用が可能である。まず、これらをどの様に生かすかについて説明する。なお図４に、従来のBUPとBUP-TDにおいて、実際に用いられているgoal節を示す。

①補強項および意味情報の扱い：

BUP同様、補強項は生成規則中右辺の先頭以外ならどこにでも自由に書くことが出来るし、それが評価される時点も従来のBUPと同様である。

②link：

BUPにおいてlink節は、生成しようとしているleft cornerが、いくつかのleft cornerを経由して目標まで到達するかどうかを表明するものであり、構文解析に先だって全ての文法記号に対してこの到達可能性が計算されている。つまり、link(A,G)は、left corner Aが目標Gに到達することを示すものであり、これはlc pair [G,A]に他ならない。本方式では、駆動表を調べる際にlc pairをエントリとして用いており、表中に目的の項目が存在することは、その前提としてlc pairが有効なものであることを保証している。よって、link節による制約は、本方式には完全に含まれていることになる。

③wf_goal,fail_goal：

wf_goal/fail_goalは、ある目標について、解析に成功/失敗したときの目標と解析を開始したときの文字列を記録するものである。これにより再度解析するときは、同様な計算を繰り返す事なく、記録してあるものを利用するわけである。本方式においても、全く同様な機構を実現できる。ただし、記録する内容は従来のものが、

wf_goal(G,I,A,O)

G:目標、A:意味内容、I:入力文字列、O:残りの文字列

fail_goal(G,I)

G:目標、I:入力文字列

であったのに対して、BUP-TDでは、さらに

wf_goal(G,I,A,O,Fst,Fol)

Fst:文字列の先頭のカテゴリ名、Fol:残りの文字列の先頭のカテゴリ名

fail_goal(G,I,Fst)

Fst:文字列の先頭のカテゴリ名

となっている。この拡張の理由は、単語の多義性に対処するためにある。つまり、記録した時点とこれを利用するときで、文字列の先頭のカテゴリを違って解釈することを防ぐためである。

④形態素処理,TRIE辞書：

入力された語に対する処理と、辞書の構成であるが、本方式のパーザはそれらの処理結果をそのまま用いれば良く、全く変更なく使用できる。熟語処理の部分は、若干の修正は必要だが、本質ではない。

⑤xg：

左外置文法(Extraposition Grammars,XG)への対応は、現在まだ実現されていない。ここでは、対応のための方針について述べる。

BUP-XGの主要部分をまず示す。

```
goal_x(G,GARG,x(G,GARG,X1),X1,D,D).
goal_x(G,A,X,Y,DI,DO) :-
        X=x(T,TARG,X1),
        T¥==G,
        P=..[T,G,TARG,A,X,X1,Y,DI,DO],
        call(P).
```

これらは、goal節の後方に書かれており、正規の構文解析動作が失敗した後に起動される。まず、(a) Xリストx(G,GARG,X1)中に目標があるかどうかが調べられる。あれば、それで目的達成であるし、なければ(b)Xリストの先頭のカテゴリTを先読みにより得られたと同様にleft cornerとみなして解析を進める。

(a)に対しては、駆動表による制限を設けない。(b)に対しては、新しいlc pairが得られたとして解析を進めれば良い。

XGへの対応は原理的には可能であるが、先読みの機構やwf_goal、fail_goalとの組み合わせに関して考慮しなければならない問題もある。

```
goal(G,A,I,O) :-
      ( wf_goal(G,I,_,_) ;
        fail_goal(G,X),!,
        fail ),!,
        wf_goal(G,I,A,O).
goal(G,A,I,O) :-
        dictionary(C,A1,I,IO),
        L=..[C,G],
        call(L),
        P=..[C,G,A1,AA,IO,O],
        call(P),
        A=AA,
        assertz(wf_goal(G,I,A,O)).
goal(G,_,I,_) :-
      ( wf_goal(G,I,_,_) ;
        assertz(fail_goal(G,I)) ),!,
        fail.
```

図4(a) 従来のgoal節

```
goal(G,A,I,O,Fst,Fol) :-
      ( wf_goal(G,I,_,_,Fst,_) ;
        fail_goal(G,X,Fst),!,
        fail ),!,
        wf_goal(G,I,A,O,Fst,Fol).
goal(G,A,I,O,C,Fol) :-
        dictionary(C,A1,I,IO),
        lc(G,C,NG),
        P=..[C,NG,A1,AA,IO,O,_,Fol],
        call(P),
        A=AA,
        assertz(wf_goal(G,I,A,O,C,Fol)).
goal(G,_,I,_,Fst,_) :-
      ( wf_goal(G,I,_,_,_,_) ;
        assertz(fail_goal(G,I,Fst)) ),!,
        fail.
```

図4(b) 実際のgoal節

4．実験と検討

この章では、実現された構文解析システムBUP-TDと従来のBUPとの比較実験、およびその結果について考察する。

4．1　実験結果

生成規則の数555個からなる英語の文法について、付録に示す17個の文を解析した結果を表1に示す。この文法は、DCGで書かれた汎用的な文法であり、非合理的な構造を排除するための補強項を持つ。また、計算時間はCPU時間であり導出木を出力するための時間は含まれていない。なお、使用した計算機はSun-3/140(16MHz,8MB)、Prologは、C-Prolog(インタプリタ)である。

この結果、

①平均して、BUPでは一語あたり1.47秒、BUP-TDでは、1.14秒の処理時間であり、30%程度の高速化がみられる。

②文5、7、10、15については、逆に遅くなっている。この割合は、BUPに対して4%(文7)～30%(文5)である。

本方式は、従来のBUPで行っている制御に加えて駆動表による制約を設けているものである。従って、原理的には従来のBUPより遅くなる要因はない。②における遅くなった原因として、先読みの機構と駆動表の検索の機構に関するオーバーヘッドの増加が考えられる。文5、7、10、15では、駆動表による高速化よりもこれらのオーバーヘッドの増加の方が勝ったものと考えられる。使用した例文が少ないために詳しい検討は難しいが、これらの文に共通する統語的構造もほとんど無く、いずれ検討すべき項目であろう。

一方、[杉村]によると、LangLAB（本論文ではBUPとして扱っている）とSAX[松本]では、インタプリット時では同程度の速度であるが、コンパイル時ではSAXの方が5～10倍速いとしている。この理由として、SAXにバックトラックがないことと、LangLABにおいてアサートされたゴールがインタプリタ・モードで解釈されることをあげている。本方式でもこれらが解決されたわけではないので、コンパイル時ではやはりSAXの方が高速であろう。

4．2　比較と検討

ここで、①BUPと②本方式を扱うときのLC(1)パーザについて比較検討してみる。

2．2節におけるBUPの非決定性についての検討および2．4．1節でのBUPとLC(1)パーザの比較から、高速化は left corner が決まったときに(a)「どの規則を選ぶか」と、(b)「部分木は完成したか」(left recursionの判断)についての非決定性を制限することにある。この両者について検討する。

(a)について：

ここではleft recursionはないものと仮定する。また、次の構文解析の時点を考える。

構文解析の目標が　　$g \in N$、
認識されたleft cornerが　$s_1 \in N$、
先読み記号が　　$c \in \Sigma$。

s_1を右辺の先頭に持つ生成規則の左辺をs_0とする。

さらに、$g \stackrel{*}{\Rightarrow} s_0\alpha$となる確率を

$$p1 = P\{g \stackrel{*}{\Rightarrow} s_0\alpha\}$$

とする。

① この時点でBUPにおける非決定性は、s_1をleft cornerとして持つ生成規則の数を$n1(s_1)$とすると、

$p1 \cdot n1(s_1)$通り（期待値）　　(4-1)

である。

②s_1をleft cornerとして持つ生成規則のうち、右辺の第2項から先読み記号cが導出される規則、および右辺の長さが1項（s_1のみ）で、$g \stackrel{*}{\Rightarrow} s_0\alpha$なる$\alpha$の先頭としてcが導出される規則の数を$n2(s_1, c)$とすると、cが先読み記号として許される確率は、

$n2(s_1, c) / n1(s_1)$　　(4-2)

したがって、考察している時点での本方式における非決定性は、(4-1)と(4-2)より、

$p1 \cdot n2(s_1, c)$通り（期待値）　　(4-3)

である。非決定性の減少から高速化が成されるとすると、速度の比は

$n2(s_1, c) / n1(s_1)$　　(4-2)

となる。

(b)について：

図2bにあるように、ある目標Aに向かって（ボトムアップ的に）解析を進めていて、left corner Aが得られたとき、先読み記号がuである場合である。

① 従来のBUPでは、目的が達成されたとするのと、例えleft recursionがなくても途中のleft cornerとして再度Aに向かって処理を続けるのと2種類の場合を調べる。

② 本方式では、結局のところある目標についてleft recursionがあるかどうかについての判断はできないが、途中のleft cornerが得られたとした場合、上記(a)による制約が働く。left recursionについての対策として、構文解析に先だってleft recursionのある文法記号とない文法記号を区別し、ない記号については停止節のみ起動させる、などの機構がさらに必要である。

5．まとめ

本論文では、BUPの高速化のためにLC(1)パーザを応用した。

まず、BUPの原理について触れ、解析動作の非決定について考察した。解決できる2種類の非決定性があることを述べた。次にLC(1)とLC(1)パーザについて説明した。

BUPとLC(1)パーザの動作にはかなりの対応があることを述べた。さらに、この対応関係を基に実現した表駆動によるBUP、実現されたシステムBUP-TDについて説明した。また、実用規模の英語文法について実験を行い、約30%の高速化が見られた。最後に、従来のBUPと本方式の非決定性について検討を行ったが、left recursionのない文法記号については表駆動以外の機構が必要であることが分かった。

文	木の数	語数	BUP	BUP-TD
1	2	5	8.7	4.4
2	1	7	7.2	4.2
3	1	6	8.3	5.7
4	1	8	9.3	7.9
5	1	5	3.8	4.9
6	1	7	9.2	6.7
7	1	11	15.2	15.7
8	2	8	12.3	8.6
9	1	5	5.2	4.2
10	1	5	7.3	7.8
11	1	3	6.1	3.0
12	1	5	8.0	4.0
13	2	6	8.0	4.9
14	2	8	9.5	7.2
15	2	9	10.8	11.4
16	1	4	6.0	3.0
17	5	21	45.9	36.6

表1　従来のBUPとBUP-TDの速度比較（単位：秒）

今後の課題としては、左外置文法、left recursionへの対応、実現されたシステムをさらに整備することなどがある。また、オーバーヘッドにより従来のBUPよりも解析時間が遅くなった例文も見られることから、プログラミング技法上の検討も必要であろう。

謝辞

最後に本研究について貴重なコメントを頂いた横浜国立大学中川裕志助教授ならびに査読者諸氏に感謝したします。

参考文献

[Aho]：Aho, A.V. and Ullman, J.D., "The Theory of Parsing, Translation, and Compiling, Volume 1 : Parsing", Prentice-Hall, pp. 362-367, 1972.

[Matsumoto]：Matsumoto, Y., et al., "BUP - A Bottom-Up Parser Embedded in Prolog", New Generation Computing, Vol. 1, No. 2, pp. 145-158, 1983.

[Tomita]：Tomita, M., "An Efficient Augmented-Context-Free Parsing Algorithm", Computational Linguistics, Vol. 13, No. 1-2, pp. 31-46, 1987.

[上脇]：上脇正、田中穂積、「辞書のTRIE構造化と熟語処理」、LPC'85, pp. 329-340, 1985.

[今野]:今野聡、田中穂積、「左外置を考慮したボトムアップ構文解析システム」、コンピュータソフトウェア、Vol.3,No.2,pp.19-29,1986.
[杉村]:杉村領一、他、「ロジックプログラミングをベースとした自然言語解析システムの比較」、情処自然言語処理研究会、57-2,1986.
[田中]:田中穂積、上脇正、奥村学、沼崎浩明、「自然言語処理のためのソフトウエアシステムLangLAB」、LPC'86,pp.5-12,1986.
[徳永]:徳永健伸、岩山真、上脇正、田中穂積、「自然言語解析システムLangLAB」、情処論、Vol.29,No.7,pp.703-771,1988.
[松本]:松本裕治、「論理型言語に基づく構文解析システムSAX」、コンピュータソフトウェア、Vol.3,No.4,1986.

付録　実験に用いた例文

1 Tell me when he came.
2 Some of them were on the table.
3 Did he give them to her?
4 Could she have been given many of them?
5 What should I do then?
6 This is hard for me to do.
7 He broke the vase her uncle had given up to him.
8 You could break this vase with that hammer.
9 You could break it up.
10 Who gave her that book?
11 who wrote it?
12 By whom was it written?
13 Where did he try to go?
14 Why did you want John to attack it?
15 How long will it take you to go there?
16 I open the window.
17 The annotations provide important information for other parts of the system that interpret the expression in the context of a dialogue.