自然言語処理 39－7
（1983. 9. 16）

BUPの高速化

松本裕治（電総研） 清野正樹（松下電器）
田中穂積（電総研）

1. はじめに

自然言語処理システムの作成においては，記述能力の高いプログラミング言語が必要である。Prologは第一階述語論理を基礎としたプログラミング言語で，バックトラック機能をもっているため，パターン・マッチングや探索を含むような処理に向いている。Prologを用いた構文解析システムとしては，DCG (Definite Clause Grammar) [1] が有名である。DCGでは各文脈自由文法規則をPrologのプログラム (definite clause)，各文法カテゴリをPrologの述語で表現するため，文法規則の中にプログラムを埋め込んだり，文法カテゴリに引数をもたせたりすることができる。しかし，DCGは完全なトップダウン・パーザなので，左再帰的な文法規則を扱うことができない。また，辞書と文法規則の扱いを分離できない。筆者らはDCGのこうした欠点を補うためにボトムアップ・パーザBUP [2,3] を開発した。BUPでは，DCGと同様，各文脈自由文法規則をPrologのプログラム，各文法カテゴリをPrologの述語で表現する。また，文法規則をボトムアップ的に適用するので，左再帰規則を扱うことができ，辞書と文法規則の扱いも分離できる。BUPの基本アルゴリズムは，Prolog上への実装の容易さを重視したため，時間的効率があまり良くなかった。本稿では，BUPの基本アルゴリズムに簡単な改良を加えるだけでCocke-Younger-Kasami法[4]やEarley法[4]に匹敵するアルゴリズムを実現することができたので，これについて報告する。なお，筆者らはDECsystem-10 Prolog [5] を用いているため，プログラムの表現方法はこれの記法に従っている。

2. BUPの原理

本章では，ボトムアップ・パーザBUPの基本的な動作原理について説明する。

2.1 BUPの基本形

BUPが扱うことのできるのは，ε規則を含まないサイクル・フリーな文脈自由文法規則である。文脈自由文法規則は，一般に，次の二つの形式のいずれかで表わすことができる。

```
1) C -> C1,C2, ... ,Cn   (n>=1)
2) C -> a
```

大文字と小文字はそれぞれ非終端記号と終端記号を表わしている。BUPではこれらの規則を次のようなPrologの節に変換する。

```
1') c1(G,X1,X) :- goal(c2,X1,X2), ... ,goal(cn,Xn-1,Xn),c(G,Xn,X).
2') dict(c,[a|X],X).
```

DECsystem-10 Prologでは，大文字と小文字はそれぞれ変数と定数を表わしているため，文法規則中の非終端記号も小文字に変換してある。これらの節は，さらにDCGの記法に従って次のように記述できる。

```
1") c1(G) --> goal(c2), ... ,goal(cn),c(G).
2") dict(c) --> [a].
```

上の述語'goal'は次のように定義されている。

```
3) goal(G,X,Z) :-
        dict(C,X,Y),P=..[C,G,Y,Z],call(P).
```

'=..' はDECsystem-10 Prologの組込み述語で，Cを述語，G，Y，Zをその引数とするリテラル表現を作り出し，Pと束縛する。したがって，call(P)によって，C(G,Y,Z)が呼び出される。さらに，文法規則中に現われる各非終端記号に対して，次のような停止条件節を与える。

```
4) c(c,X,X).
```

BUPのアルゴリズムは，上に示したPrologの節を用いて，トップダウン的なゴール・カテゴリの設定とボトムアップ的な文法規則の適用から成り立っている。以下では，BUPの基本アルゴリズムを例を用いて説明する。

辞書と文法規則

```
dict(np,[john|X],X).
dict(vp,[walks|X],X).
np(G,X1,X) :- goal(vp,X1,X2),s(G,X2,X).
```

入力文

```
John walks.
```

基本アルゴリズム

step 1. ゴール・カテゴリの設定

入力文の解析は述語'goal'の呼び出しによって開始される。この時，述語'goal'の第1引数はゴール・カテゴリS（文），第2引数は入力文のリスト，第3引数は空リストである。例に対しては，

```
?- goal(s,[john,walks],[]).
```

によって開始される。

step 2. 先頭単語の辞書引き

述語'goal'が呼び出されると，定義に従って述語'dict'を呼び出す。この時，第2引数に解析すべき単語のリストを渡す。述語'dict'は単語のリストの先頭の単語の辞書引きを行ない，その単語の文法カテゴリを第1引数に，残りの単語のリストを第3引数に返す。例では，

```
dict(C,[john,walks],Y)
```

によって辞書引きを行ない，Cにnp，Yに[walks]が返される。

step 3. 文法規則の呼び出し

各文脈自由文法規則は右辺の先頭の非終端記号に対応する述語を左辺にもつPrologの節に変換されている。述語'goal'では，step 2の辞書引きを成功すると，得られた文法カテゴリと同じ名前の述語を呼び出す。この時，第1引数はゴール・カテゴリ，第2引数は辞書引きした残りの単語のリストである。例では，

```
np(s,[walks],[])
```

となる。これは，入力文とゴール・カテゴリSを対応づける過程で，すでに文法カテゴリnpが発見されたことを意味している。

step 4. 停止条件節のチェック

停止条件節は文法規則よりも前に置かれ，文法規則が呼び出されると，呼び出した文法カテゴリとゴール・カテゴリが一致しているかどうかのチェッ

クに用いられる。例では，呼び出した文法カテゴリnpとゴール・カテゴリsが異なるので，チェックは失敗する。もし，一致していれば，文法規則の適用は行わない。

step 5. 文法規則の適用

適用可能な文法規則がみつかると，元の文脈自由文法規則では右辺の2番目以後の非終端記号に対応する文法カテゴリをゴール・カテゴリとして，述語'goal'を呼び出す。例では，

```
goal(vp,[walks],X2)
```

である。

step 6. 文法規則の呼び出し

step 5で述語'goal'の呼び出しがすべて成功すれば，その文法規則の適用が成功したことになる。すなわち，文法規則を使って構文解析木を下から上に一段成長させたことになり，完成した文法カテゴリに対して再び文法規則を呼び出す。例では，

```
s(s,[],[])
```

となる。

2.2 文脈依存性

前節で示したBUPの基本アルゴリズムは，文脈自由文法を処理するためのものである。これに自然言語の文脈依存性を取り入れる方法として，

1) 文法カテゴリに付加的な引数をもたせる。
2) 文法規則の中にPrologのプログラム（制約条件項）を埋め込む。

という方法を用いる。

Prologの各述語は任意個の引数をもつことができるため，DCGでは各非終端記号に対応する述語が任意個の引数をもつことを許している。しかし，BUPでは述語'goal'がパージングをコントロールしているため，各非終端記号に対応する述語のもつ引数の数を制限したい。そこで，筆者らはDCGの記法で書かれた文法規則をBUP用の文法規則に変換するトランスレータを作成し，その変換の過程で複数個の引数をリストにして1つの引数に置き換えるという方法をとった。前節で示したBUPの文法規則と辞書1)，2)は，次のような形式になる。

```
1') cl(G,Al,A,Xl,X) :-
        goal(c2,A2,Xl,X2), ... ,goal(cn,An,Xn-l,Xn),
        c(G,An+l,A,Xn,X).

2') dict(c,A,[a|X],X).
```

これらのBUP節の中で各A_iは対応するDGGの非終端記号のもつ引数のリストである。同様に，前節の3)，4)は次のような形式になる。

```
3) goal(G,A,X,Z) :-
        dict(C,Al,X,Y),
        P=..[C,G,Al,A,Y,Z],call(P).

4) c(c,A,A,X,X).
```

4)の停止条件節の役割は，ゴール・カテゴリが完成したかどうかのチェックと，完成した場合に引数の情報をそのまま返すことである。

ここで，文法カテゴリに付加的な引数をもたせて，文脈依存的な処理を行なう

例として次のようなDCG記法の文法規則を考える。

```
sentence(s(NP,VP)) --> np(NP,N),vp(VP,N).
```

これをBUP形式に変換すると次のようになる。

```
np(G,[NP,N],A) --> goal(vp,[VP,N]),sentence(G,[s(NP,VP)],A).
```

'sentence'をゴール・カテゴリとしてこの節を呼び出すと，停止条件節

```
sentence(sentence,A,A,X,X).
```

によって，変数Aに'[s(NP,VP)]'が返される。この時，変数Nを用いて文脈依存的な処理を行うことができる。この節が呼び出される時には，すでにnp（名詞句）が完成しており，変数Nには数と人称に関する情報が束縛されている。一方，'goal(vp,[VP,N])'の呼び出しの際にNの値を渡すため，動詞の語尾形に関する情報とマッチしなければ，この呼び出しは失敗する。このように，文法カテゴリに引数をもたせることによって，文脈依存的な処理ができる。

BUPに文脈依存性を取り入れるもう一つの方法が，文法規則の中にPrologのプログラムを埋め込む方法である。例えば，上の文法規則において変数Nの代りに一致を調べるプログラム'check_np_vp'を埋め込むことによって文脈依存性を扱うこともできる。

```
sentence(s(NP,VP)) --> np(NP),vp(VP),{check_np_vp(NP,VP)}.
np(G,[NP],A) --> goal(vp,[VP]),
                 {check_np_vp(NP,VP)},
                 sentence(G,[s(NP,VP)],A).
```

2.3 リンク関係

BUPの基本動作は，2.1節で述べたトップダウン的なゴール・カテゴリの設定とボトムアップ的な文法規則の適用であるが，適用可能な文法規則をすべて適用してしまうので，無駄が多い。これを防ぐために，カテゴリ間の到達可能性を表わすリンク関係を導入した。

BUPにおける各文法カテゴリの呼び出しは，右辺の先頭要素にその文法カテゴリをもつような文脈自由文法規則の起動を意味している。このような文法規則の起動が成功すると，その規則の左辺の文法カテゴリが発見されたことになる。もし，この文法カテゴリがゴール・カテゴリの構成要素になれなければ，この文法規則の適用自体が無駄になる。また，述語'goal'の定義の中で，先頭単語の辞書引きの結果から得られる文法カテゴリがゴール・カテゴリの構成要素になれなければ，それ以上処理を進めても無駄になる。

こうした効率の悪さを改善するために，BUPでは文法規則が与えられた段階でリンク関係を計算し，実行時にリンク関係を用いて探索空間を小さくしている。リンク関係の計算は，まず各文法規則の右辺の先頭要素の文法カテゴリから左辺の文法カテゴリへリンク関係があるとし，さらに推移律を適用してすべてのリンク関係を求めるという手順で行われる。リンク関係を用いた適用可能性のチェックをBUPの各文法規則及び述語'goal'の中に埋め込むと，前節の1'），3）は次のようになる。

```
1') c1(G,Al,A,Xl,X) :-
        link(c,G),
        goal(c2,A2,Xl,X2), ... ,goal(cn,An,Xn-1,Xn),
        c(G,An+1,A,Xn,X).
```

```
3) goal(G,A,X,Z) :-
        dict(C,Al,X,Y),
        link(C,G),
        P=..[C,G,Al,A,Y,Z],call(P).
```

1')をDCGの記法によって書き換えると次のように簡略化される。

```
1") cl(G,Al,A) -->
        {link(c,G)},
        goal(c2,A2), ... ,goal(cn,An),
        c(G,An+1,A).
```

リンク関係を用いたチェック機構は，Prattのアルゴリズム[5]の中で'oracle'（天の声）と呼ばれているものと同じ働きをする。

3. BUPの改良

本章ではBUPの基本アルゴリズムに簡単な改良を加えることによって高速化する方法について説明する。

3.1 辞書項目の登録

BUPのアルゴリズムでは，述語'goal'の中で辞書引きを行なう。これまでの説明の中では述語'dict'による単純な辞書引きだけを示したが，実際に語尾変化のおこる言語を扱うためには，形態素処理も含めた辞書引きを考えなければならない。そこで，述語'goal'の定義を次のように変更した。

```
goal(G,A,X,Z) :-
      dictionary(C,Al,X,Y),
      link(C,G),
      P=..[C,G,Al,A,Y,Z],call(P).

dictionary(C,A,X,Y) :-
      dict(C,A,X,Y) ; morph(C,A,X,Y).
```

述語'dictionary'はまず従来の辞書引きを行ない，これに失敗すると述語'morph'によって形態素処理を行なう。述語'morph'の定義はここでは示さないが，単語の分解とパターン・マッチング，辞書引きなどを含んでいるため，述語'morph'の実行には多くの時間が必要となる。ところが，BUPの基本アルゴリズムでは，部分的に解析を失敗すると，同じ単語に対して何回も述語'dictionary'を呼び出すことになり効率が悪い。そこで，述語'dictionary'の定義をさらに変更した。

```
dictionary(C,A,X,Y) :-
      wf_dict(_,_,X,_), ! ,wf_dict(C,A,X,Y).

dictionary(C,A,X,Y) :-
    ( dict(C,A,X,Y) ; morph(C,A,X,Y) ),
      create_dlist(X,Y,U,V),
      assertz( wf_dict(C,A,U,V) ),fail.

dictionary(C,A,X,Y) :-
      wf_dict(C,A,X,Y).
```

2番目の定義は，従来の定義に，辞書引きあるいは形態素処理の結果を'wf_dict'という述語名で登録する部分を付け加えたものである。述語'create_dlist'は，X，Yと同じ句（単語のリスト）を表わすd-list，U，Vを作り出し，Vを常に変数にしておく。このU，Vを用いて辞書項目を登録すると，同じ単語が同一文中に二度現われても，無駄な形態素処理を防ぐことができる。

1番目の定義は，現在注目している単語の辞書項目がすでに登録されているかを調べ，登録されていればその内容をそのまま返し，2番目の定義を実行しないためのものである。

3.2 成功ゴールの登録

BUP のアルゴリズムはボトムアップ的にdepth-firstで文法規則を適用していくため，ゴール・カテゴリの解析が失敗するとバックトラックがおこる。このバックトラックはPrologのバックトラック機構をそのまま利用して暗に記述されている。Prologはバックトラックがおこるとそれまでの計算結果を忘れてしまうので，同じ計算を再びくり返す可能性がある。すなわち，BUP は入力文の同じ位置から同じゴールを予測して何回も同じ解析をくり返す可能性がある。BUP でバックトラックがおこるのは述語'goal'の中なので，その定義を変更し，同じ計算をくり返さないようにした。新しい定義は次のようになる。

```
goal(G,A,X,Z) :-
      wf_goal(G,_,X,_), ! ,wf_goal(G,A,X,Z).

goal(G,A,X,Z) :-
      dictionary(C,Al,X,Y),
      link(C,G),
      P=..[C,G,Al,A,Y,Z],call(P),
      assertz( wf_goal(G,A,X,Z) ).
```

2番目の定義は，従来の定義とほとんど同じであるが，解析が成功した場合に'wf_goal'という述語名で結果を登録する部分を追加してある。

1番目の定義は，再び入力文の同じ位置から同じゴール・カテゴリが予測されたときに，すでに処理結果が登録されているかを調べ，登録されていればその内容をそのまま返し，2番目の定義を実行しないためのものである。

3.3 失敗ゴールの登録

前節では成功するゴール・カテゴリの解析をくり返さない方法を示した．この考え方を更に拡張して，失敗したゴールも登録することによって，失敗することがすでにわかっている計算をくり返さないようにすることができる。述語'goal'の新しい定義は次のようになる。

```
goal(G,A,X,Z) :-
    ( wf_goal(G,_,X,_), ! ,wf_goal(G,A,X,Z) ;
      fail_goal(G,X), ! ,fail ).

goal(G,A,X,Z) :-
      dictionary(C,Al,X,Y),
      link(C,G),
      P=..[C,G,Al,A,Y,Z],call(P),
      assertz( wf_goal(G,A,X,Z) ).

goal(G,A,X,Z) :-
    ( wf_goal(G,_,X,_) ;
      assertz( fail_goal(G,X) ) ), ! ,fail.
```

2番目の定義は，前節の2番目の定義と全く同じである。

3番目の定義は，新しく追加した定義で，もし2番目の定義の実行中にゴール・カテゴリが全く完成しなければ，'fail_goal'という述語名で失敗したゴール・カテゴリを登録するためのものである。

1番目の定義は，入力文のある位置からあるゴール・カテゴリが予測された場合に，すでに解析が実行されていれば，同じ計算をくり返さないためのものである。もし，'wf_goal' が登録されていれば，その引数をそのまま返す。もし，'fail_goal' が登録されていれば，述語'goal'の呼び出しを失敗させる。もし，どちらも登録されていなければ，2番目の定義を実行する。

4. 検討

これまで，BUPのアルゴリズムの改良について，段階を追って説明してきた。この効果を調べるために，2つの例文に対する各段階のBUPの処理時間を求めた（表1）。BUPは入力文に対するすべての構文解析木を逐次求めるため，例文2では3通りの解析結果が出力されている。

表1をみると，入力文の長さの長い例文2の方が改良の効果が顕著である。したがって，今回の改良によって，オーダーのクラス自体が上がったと考えられる。また，トータル・タイムの中には，すべての可能性を調べ終わるまでの時間が含まれており，この処理に対する効果が著しい。このことは，例文2のような曖昧性を含む文の解析に有効である。

1. I have a book.
2. BUP analyzes all of the basic kinds of phrases and sentences.

	1.	2.
a. original BUP	208 (969)*	6,391 511 383 (42,234)
b. a. + wf_dict	152 (488)	2,085 169 118 (12,433)
c. b. + wf_goal	183 (381)	1,801 282 278 (3,479)
d. c. + fail_goal	210 (362)	1,628 213 204 (2,771)

* ( )内の値はトータル・タイム

表1．処理時間の比較（msec.）

さて，第2章で示したBUPの基本アルゴリズムは，ボトムアップ＋バックトラッキングなので，解析時間がc^nのオーダーになる。文脈自由文法を扱うパージング・アルゴリズムで，より時間的効率の良いものには，Cocke-Younger-Kasami法やEarley法があり，どちらもn^3のオーダーである。以下では，これらのアルゴリズムをインプリメントした構文解析システムLINGOL[5]と高速化したBUPとを比較してみたい。

LINGOLもBUPもトップダウン機構とボトムアップ機構を組み合わせたアルゴリズムを用いている。ボトムアップ機構は，入力文に対して文法規則を適用するもので，Cocke-Younger-Kasami法が基本になっている。C-Y-Kアルゴリズムでは，入力文を左から右にスキャンして，パーズ・テーブルを作成する。パーズ・テーブルとは，入力文のi番目の単語から始まるj個の単語から成る句に対応する文法カテゴリの集合t_{ij}から構成されている。LINGOLとBUPでは，このパーズ・テーブルの作成方法が異なる。LINGOLでは入力文中の現在注目している位置に適用可能な文法規則をbreadth-firstにすべて適用してパーズ・テーブルを成長させる。BUPでは適用可能な文法規則をdepth-firstに一つずつ適用してパーズ・テーブルを成長させる。BUPの場合，DECsystem-10 Prologには配列やビット・テーブルの概念がないので，述語'wf_dict'，'wf_goal'によってパーズ・テーブルを実現している。また，BUPにはパーズ・テーブル以外に，失敗したゴールを登録するテーブル（述語'fail_goal'によって実現）が存在し，無意味な解析を防いでいる。

トップダウン機構は，ゴール・カテゴリの設定とゴール・カテゴリへの到達可能性のチェックから成り，Pratt法が基本になっている。LINGOLの場合もBUPの

場合も処理はほとんど同じであるが，文法カテゴリ間の到達可能性に関する情報の記憶方法がLINGOLではビット・テーブル，BUPではリンク関係を表わすProlog節と異なっている。

5. おわりに

本稿ではPrologに埋め込まれた構文解析システムBUPの時間的な効率改善について述べた。Prologに埋め込まれた構文解析システムとしてはDCGがあるが，DCGは単純なトップダウン・パーザなので次のような欠点をもっていた。

1) 左再帰規則が扱えない
2) 文法規則と辞書の扱いが分離できない

BUPはDCGの利点を残したままこれらの欠点を補うために開発したパーザである。文法規則と辞書の扱いが明確に分離されているので，それぞれについて高速化をはかることができた。本稿で述べたBUPの高速化の手法は，解析の途中結果を記憶して，同じ計算をくり返さないというものであったが，BUPの動作をコントロールしている述語'goal'を数行修正するだけで実現できた。途中結果の記憶には，Prologの'assert'機能を使ったため，時間的な効率は改善できたが，多くの記憶領域が必要になった。DECsystem-10 Prologは記憶領域の制限が強く，また配列やビット・テーブルを扱えないといった欠点をもっているため，こうした欠点を補ったPrologシステムの開発が望まれる。

筆者らは，現在，BUPを用いた英語の構文解析システムを作成中である。英語の文法規則の開発には，DCGの記法で記述した文法規則と辞書をBUP用の文法規則と辞書に変換するトランスレータ[6]を用いている。文法規則の数が約150，辞書項目の数が約200で，今後さらに充実させてゆく予定である。

謝辞

本研究の機会を与えていただいた電子技術総合研究所中島隆之パターン情報部長に感謝します。日頃，御討論いただいたICOT第2研究室自然言語グループの皆様，また電子技術総合研究所推論システム研究室の諸氏に感謝します。

参考文献

1) Pereira,F. and Warren,D., Definite Clause Grammar for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, 13, pp. 231-278, May, 1980.

2) 松本裕治，田中穂積，Prologに埋め込まれたbottom-up parser：BUP，情報処理学会 自然言語処理研究会，34-6，1982.

3) 松本裕治，田中穂積，平川秀樹，三吉秀夫，安川秀樹，向井国昭，横井俊夫，Prologに埋め込まれたボトムアップパーサ：BUP，Proc. of THE LOGIC PROGRAMMING CONFERENCE '83, 3.1, 1983.

4) Aho,A.V. and Ullman,J.D., The Theory of Parsing, Translation, and Compiling, vol.1 Parsing, Prentice-Hall, 1972.

5) Pratt,V.R., LINGOL -- A Progress Report, Proc. of 4th IJCAI, pp. 372-381, Aug.,1975.

6) 松本裕治，清野正樹，田中穂積，BUPトランスレータ，電子技術総合研究所彙報（予定）.