ボトムアップ構文解析システムＢＵＰの高速化

今野　聡，奥村　学，田中穂積（東京工業大学　工学部）

１．はじめに

Prologを用いた構文解析システムとして有名なＤＣＧ(1) はトップダウン・パーザであることから左再帰の文法規則が扱えないという欠点を有している。この欠点を改良するためにボトムアップ・パーザＢＵＰ(2,3) がＥＴＬ，ＩＣＯＴによって開発された。このＢＵＰにはＥＴＬによって高速化がすでに図られている(4) 。ところがこの高速化ＢＵＰの効率は、その高速化にPrologの組込み述語assertを用いているため、入力文が長くなるほど低下してしまう。本稿では簡単な改良によりこの効率の低下を防止できるようになったのでそれについて報告する。

２．ＢＵＰの基本アルゴリズム

本章ではoriginalＢＵＰ(2,3) の基本アルゴリズムを例を用いて説明する。originalＢＵＰにおけるgoal節（ＢＵＰの動作をコントロールする述語）および辞書引き用述語'dictionary'を以下に示す。

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

例　辞書および文法規則

```
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.
```

step 1　入力文の解析は述語'goal'の呼び出しによって開始される。すなわち、

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

で開始される。

step 2　'goal'を呼び出すと、定義から述語'dict'（'dictionary'）が呼び出され、解析中の文のリストの先頭単語の辞書引きを行なう。上例では、

```
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　step 2が成功すると、'goal'では定義より得られた文法カテゴリと同じ名前の述語を呼び出す（P =.. [C,G,Y,Z],call(P) ）。上例の場合、

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

となる。これは文法カテゴリnpが発見されたことを意味している。

step 4　step 3で呼び出された述語に対して適用可能な文法規則を探し、適用する。上の例では適用した結果、

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

が呼び出される。

step 5　step 4での文法規則の適用が成功するということは、すなわち文法規則を使って構文解析木を下から上へ一段成長させることになり、完成した文法カテゴリに対して再び文法規則を適用する。例ではｓ（文）が完成し、解析を終了する。

３．高速化ＢＵＰ

この章では高速化のためoriginalＢＵＰに加えられた改良点(4) について説明する。

３．１．辞書引き結果の登録

originalＢＵＰにおいては、どこかで解析に失敗するとバックトラックして同じ単語に対して何回も述語'dictionary'をcallするようなことがあり、効率が非常に悪い。そこで述語'dictionary'は次のように変更された。

```
d1) dictionary(C,X,Y) :-
      wf_dict(_,X,_), ! ,
      wf_dict(C,X,Y).
d2) dictionary(C,X,Y) :-
    ( dict(C,X,Y) ; morph(C,X,Y) ),
      assertz( wf_dict(C,X,Y) ),fail.
d3) dictionary(C,X,Y) :-
      wf_dict(C,X,Y).
```

d2) で辞書引きした結果を'wf_dict' という述語名で登録し、d1) ではその登録結果を用いて今辞書引きしようとしている文字列がすでに辞書引きされたものかどうかチェックし、すでに登録されているものに関しては辞書引きを行なわないようにしている。

3．2．成功ゴール，失敗ゴールの登録

ＢＵＰはProlog上にimplement されているからその解析はdepth-first に進んでいく。そのためどこかで解析が失敗するとバックトラックが起きる。この際それまでの処理結果をすべて忘れてしまうので、同じ計算を何回も繰り返す可能性がある。すなわちＢＵＰでは、同じ文字列に対し同じゴールを予測して、何回も同じ解析を繰り返す可能性がある。そこでgoal節は次のように変更された。

```
g1)  goal(G,X,Z) :-
       ( wf_goal(G,X,_), ! ,
         wf_goal(G,X,Z) ;
         fail_goal(G,X), ! ,fail ).
g2)  goal(G,X,Z) :-
         dictionary(C,X,Y),
         link(C,G),
         P = ..(C,G,Y,Z),call(P),
         assertz( wf_goal(G,X,Z) ).
g3)  goal(G,X,Z) :-
       ( wf_goal(G,X,_) ;
         assertz( fail_goal(G,X) ) ), ! ,
         fail.
```

g2) で成功したゴールを登録し、またg3) で失敗したゴールを登録し、g1) ではそれらの登録結果を用いて、現在の文字列，予測ゴールにおける解析がすでに成功ゴール，失敗ゴールとして登録されているかどうか調べ、登録されているなら再計算を行なわないようにしている。

4．ＢＵＰの改良

本章では、高速化ＢＵＰの効率低下防止のため今回加えた改良について説明する。高速化ＢＵＰの効率が入力文の長さが長くなるほど低下するのは、

成功ゴール，失敗ゴール，辞書引き結果を登録する際'assert'する述語の引数として、現在解析している文字列をリストの形で持っていなければならない

ことが最大の原因だと考えられる。なぜならそのために

1） 処理結果をassertする時間
2） unification の時間

が非常に増大し、その結果、時間効率が大幅に低下すると考えられるからである。このように非効率さが、現在解析している文字列をリストの形で保存していることにかなり起因していることから、リストの形で保存することをやめ、インデックスを用いることにした。すなわち文"I bought a book yesterday."が入力された時には、図１のような対応表を解析前に作り、この対応表の数字（例えば、(a,book,yesterday)に対しては３）を現在解析している文字列に対するインデックスとして、文字列のリストの代わりに述語の引数として持とうということである。図２に、リストの形，インデックスの形それぞれを用いて上の入力文に対して解析を行なった時の、登録された成功ゴールのリストを示す。直感的に見てassertに必要な記憶領域が相当違うというのは明らかであろう。

```
text(5,(i,bought,a,book,yesterday)).
text(4,(bought,a,book,yesterday)).
text(3,(a,book,yesterday)).
text(2,(book,yesterday)).
text(1,(yesterday)).
text(0,()).
```

図１

5．検討

４章で述べた改良の効果を検討するため、いくつかの例文に対する処理時間および使用した記憶領域を求めた（表１，２）。表１，２における入力文の”リストの長さ”とは、英語の場合には前例のように単語数となるが、我々が作成している日本語のシステム(5) の場合には自動分かち書き処理を行なうため１文字１文字のキャラクタ数となる。表１，２からわかるとおり、今回の改良による処理時間の短縮，使用した記憶領域の減少はともに、入力文が長いほど顕著である。例文３では、処理速度は約７割改善され、また記憶領域も半分以下ですむ。

このように今回の改良によって効率はかなり改善され、また高速化ＢＵＰにおいて今後の課題となっていた”使用する記憶領域の減少”が実現できた。また今回の実験からこの改良は、一般に”リストの長さ”（＝単語数）がそう多くはならない英語などより、日本語など”リストの長さ”（＝キャラクタ数）が相当数にまで達するこう着言語に対して有効であることがわかった。

```
wf_goal(vp,(bought,a,book,yesterday),(a,book,yesterday)).
wf_goal(nomhd,(book,yesterday),(yesterday)).
wf_goal(obj,(a,book,yesterday),(yesterday)).
wf_goal(vp,(bought,a,book,yesterday),(yesterday)).
wf_goal(advp,(yesterday),()).
wf_goal(obj1,(a,book,yesterday),(yesterday)).
wf_goal(np,(a,book,yesterday),(yesterday)).
wf_goal(vp,(bought,a,book,yesterday),()).
wf_goal(s,(i,bought,a,book,yesterday),()).
wf_goal(ncomp,(bought,a,book,yesterday),(a,book,yesterday)).
wf_goal(ncomp,(bought,a,book,yesterday),(yesterday)).
```

図２（ａ）　リストの形

```
wf_goal(vp,4,3).
wf_goal(nomhd,2,1).
wf_goal(obj,3,1).
wf_goal(vp,4,1).
wf_goal(advp,1,0).
wf_goal(obj1,3,1).
wf_goal(np,3,1).
wf_goal(vp,4,0).
wf_goal(s,5,0).
wf_goal(ncomp,4,3).
wf_goal(ncomp,4,1).
```

図２（ｂ）　インデックスの形

６．おわりに

本稿では、ＢＵＰの時間的な効率改善ということで、高速化ＢＵＰに新たに加えた改良点について述べた。高速化ＢＵＰにおいては、解析中の文字列をリストの形のまま引数とした述語をassertしていたため、

１）　assertする時間

２）　unifIcation の時間

が非常に増大する。また

３）　記憶領域が相当量必要

という問題点があった。今回、リストの形で引数としていた文字列をインデックス（数字）の形に変更したことで、これらの問題点をすべて解決することができた。今後は、現在ＢＵＰ上で開発中の英語および日本語の文法規則を充実させ、意味解析まで含めた自然言語システムを目指す予定である。

謝辞

御討論いただいたＢＵＰ検討会参加の皆様，また東京工業大学情報工学科田中研究室の諸氏に感謝します。

参考文献

(1)：F.Pereira & D.Warren,
"Definite Clause Grammar for Language Analysis
--A Survey of the Formalism and a Comparison with Augmented Transition Networks",
Artificial Intelligence,13,pp231-278,
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資料34-6，
1982

(3)：松本裕治，他，
"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, 1983

(4)：松本裕治，清野正樹，田中穂積，
"ＢＵＰの高速化"，
情報処理学会自然言語処理研究会資料39-7，
1983

(5)：田中穂積，小山晴生，奥村　学，
"ボトムアップ構文解析システムＢＵＰの拡張と日本語文法の試作"，
Proc. of THE LOGIC PROGRAMMING CONFERENCE '84, 1984

例文

1．　私は花子を見た。

2．　その状態は固体から液体、気体へと変わっていく。

3．　物質はすべて粒子からできていて、液体や気体では粒子は絶えず動きまわっている。

例文	リストの長さ	リストの形	インデックスの形
1	２０	663(1326)	588(1124)
2	４８	1793(4523)	1386(3422)
3	８０	4135(8004)	2359(4947)

表１　処理時間の比較（msec）

（）内の値はトータル・タイム

例文	リストの形	インデックスの形
1	6656	4608
2	27136	12800
3	57856	27136

表２　使用した記憶領域の比較