Prologに埋め込まれたbottom-up parser：BUP

松本 裕治・田中 穂積
（電子技術総合研究所・パタン情報部）

1. はじめに

Prologは論理学を基礎とし、その記述能力の高さから、人工知能向けの新しい言語として注目を集めている。著者らはPrologを自然言語処理への応用に用いてきた[松本82]。著者らが使用しているDec System-10 Prolog[Pereira 78]にはDCG（Definite Clause Grammar）[Pereira 80]というパーサーが、Prologの基本形であるdefinite clauseを利用して実装されている。DCGでは、文法カテゴリがPrologの述語として表現され、文法規則が、いわば、Prologのプログラムにコンパイルされた形になっているため、文法規則の間に自由にPrologのプログラムが記述できるなどの特徴を持つ反面、完全なトップダウンパーサーであるため、左再帰的な文法規則が扱えない、辞書と文法規則が分離されていない、などの欠点を持つ。DCGのもつ特色をなるべく損わず、しかもその欠点を補う目的で、Prolog上にボトムアップ機能をもつパーサーの実装を試みた。本稿で紹介するパーサーBUP（Bottom-Up Parser in Prolog）は、DCGと同様に文法カテゴリがPrologの述語としてコンパイルされた形を取り、文法規則内に自由にPrologのプログラムが記述できる上、左再帰的な文法をも処理することが可能で、また、文法規則と辞書も分離した形で扱うことができる。

次章では、BUPの基本となるパージングアルゴリズムとProlog上への実装法について述べる。アルゴリズム自体はCocke-Younger-Kasami法やEarley法に比べて効率的には劣るが、Prologへの実装の簡潔さを重視して設計されている。DCGは文脈自由文法の文法規則がそのままの形で、Prologのプログラムとして動くが、BUPの場合は、1つの文法規則が1つのPrologの文に対応するものの、補助的なプログラムがいくつか必要である。

3章では、簡単な例をもとに、DCGとBUPの比較を試みる。実行時間の比較として、著者らの日本語文解析のプログラムにおける処理時間の一部を紹介する。

2. BUP：Bottom-Up Parser in Prolog

2・1 BUPのパージングアルゴリズム

本節ではBUPが用いているパージングアルゴリズムを簡単に説明する。アルゴリズムの記述はプログラムの性格上、再帰的な形をとる。文法規則は文脈自由文法の形にのみ制限する。ただし、後で示すように、各文法カテゴリはPrologの述語として表現され、任意数の引数を持つことができるし、また、文法規則中にPrologのプログラム（述語）を自由に挿入できるので、文脈依存的な処理をも記述可能である。大規模な辞書や文法規則をもつ自然言語処理システムへの応用を考えているので、文法規則の記述と、辞書記述は分離して考えている。ただし、一部の制限はあるが、DCGと同様に、文法規則中に終端記号を記述することは可能である。アルゴリズムの実行中は、常にその時点での目標（goal）となる文法カテゴリが記憶されているとする。最初のgoalは「文」である。

以下のアルゴリズムでは、与えられた文脈自由文法には、循環する規則列がな

く（cycle-free），また、ε-規則（ε-rule）も含まれていないと仮定する。文法規則は適当な順に並べられているとする。

次にBUPで用いられている基本的なパージングアルゴリズムの概要を示す。このアルゴリズムは2つの部分，goal部およびrule部，よりなり，互いに呼び合う形で構文解析を行う。本アルゴリズムでは、完全なbottom-up動作だけではなく，常に各時点でのsubgoalを意識し、ある意味ではtop-down機能をも有した形になっている。

[goal部]

目標（goal）と解析すべき記号列をわたされ，その記号列の先頭から目標をみたす部分列を切り取る働きをする。　具体的には、先頭の一単語を辞書で引いてその文法カテゴリを得て，そのカテゴリ名，目標，先頭の一単語を取り除いた記号列を持ってrule部を呼び出す。

[rule部]

カテゴリ名、目標、および解析すべき記号列がわたされる。意味的には、目標を達成する上で，与えられたカテゴリが現在の所発見されたことを示す。よって

1) 目標と与えられたカテゴリが等しければ，この呼出しは終了する。
2) そうでない場合は、与えられたカテゴリ名を文法規則の右辺の先頭にもつ規則を1つ選択し、その文法規則の右辺の残りのカテゴリ名を目標にして，次々にgoal部を呼び出す。
3) 2)のgoal部呼び出しが全て完了すれば，その文法規則の左辺のカテゴリ名，はじめに与えられた目標，および，残された記号列を持ってrule部を再帰的に呼び出す。

Prologへの実装を考えたアルゴリズムであるので、呼び出し失敗時には、自動的に後戻りすることを想定している。[goal部]における辞書引き，および，[rule部]における文法規則の選択は一意的ではないが、辞書項目や文法規則は適当な順に並べられ、失敗による後戻り時に、次々にその順に従って選択される。

なお、top-downの予測制御を行うために、与えられた文法規則の集合から，どの文法カテゴリがどの文法カテゴリに上位につながり得るかを調べておけば、[rule部] 2)において、はじめから無駄とわかっている文法規則の選択を防ぐことができる。Prolog中への実装では、このことも考慮に入れている。後に示すlinkという述語がこれにあたる。

cycle-freeかつε-規則のない文脈自由文法が，任意の入力文について，このアルゴリズムで解析可能なことは、任意の時点で、部分解析木の深さが有限になることを示すことによって証明できる。また、本アルゴリズムが、可能な解析木を後戻りによって全て生成可能なことも、ここではその証明は省略する。

本アルゴリズムは、Earley-Prattのアルゴリズムをちょうどserialに実行するのに近い形をとる。したがってWFST（Well Formed Substring Table）などを用いて部分解析木を保存することは行っていないので，効率的には、指数関数オーダーのアルゴリズムである。これはPrologへの実装における記述の簡潔さを重視

したためである。 Prologの並列化の話題は最近さかんであり、Prologのor-並列を実現する処理系ができれば、このアルゴリズムはEarley-Prattアルゴリズムと同等の効率で動く。

2.2 BUPのPrologへの実装

本節では、BUPのPrologへの実装、すなわち、文脈自由文法の文法規則がどのようなPrologプログラムに変換されるかについて説明する。

文脈自由文法の文法規則は一般に、

1) $S \rightarrow N, V$

2) $S \rightarrow V$

また、終端記号を表わす規則として

3) $N \rightarrow a$

のような形のものがある。ここでは、文法と辞書を分離するため、終端記号は一般に3)のような形でのみ生成されると仮定するが、実際には、DCGと同様に、文法規則中に終端記号を記入することも可能である。

上の3つの規則は、Prolog中ではそれぞれ次の形に変換される。ただし、Dec System-10 Prologの記法を採用するため、文法カテゴリは小文字で表わされる。

1') n(G,X0,X) :- goal(v,X0,X1), s(G,X1,X).

2') v(G,X0,X) :- s(G,X0,X).

3') dict(n,[a|X],X).

これらのうち、1'), 2')が前節でのrule部の働きをする。Gは与えられた目標である。X0, Xなどは入力文の部分列を2つのリストの差で表現するための変数である。DCGの記法を用いると、これらは次のようにも記述できる。

1'') n(G) --> goal(v), s(G).

2'') v(G) --> s(G).

3'') dict(n) --> [a].

上のプログラム中で、goalという述語はgoal部にあたる。この述語は次のように定義される。

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

「=..」はリストの第1要素を述語名にし、他の要素を引数とする述語表現を構成する演算子である。

効率を考えて、link情報を利用する場合は、1'') 2'')はさらに次のように、それぞれ変換される。

1''') n(G) --> {link(s,G)}, goal(v), s(G).

2''') v(G) --> {link(s,G)}, s(G).

すなわち、1)2)の規則は最終的にはカテゴリsの部分列を記述する規則であるから、もしsが、現在の目標であるGにつながらないなら、この規則を使っても意味がない。linkはそれを防ぐ役割を果たす。{}はDCGの記法で、この中の述語は文法カテゴリではなく、Prologのプログラムと解釈される。

rule部 1)の停止条件は、全ゆる文法カテゴリについて、例えば、

5) s(s,X,X).

のように目標と文法カテゴリが等しい場合には、無条件でその処理を終らせるようにする。

〔例1〕

図1の1)～3)の文法を考える。これらはそれぞれBUPとして1')～3')のように変換される。4)～7)は上で述べた補助プログラムである。入力文が'John walks'だとすると、BUPの起動は次の呼出しによって始まる。

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

この質問文の実行経過を示したのが図2である。図中で、「⇒」はunificationの結果得られた新しい本体であることを示す。「=」は等価であること、「↔」は単語とのunificationの結果、その述語が消滅することを示す。

```
1)  S -> NP , VP

2)  NP -> John
3)  VP -> walks


1') np(G,X,Z) :- goal(vp,X,Y),s(G,Y,Z).

2') dict(np,[john|X],X).
3') dict(vp,[walks|X],X).


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

5) np(np,[],[]).
6) vp(vp,[],[]).
7) s(s,[],[]).
```

図1. 文法の例とBUPへの変換

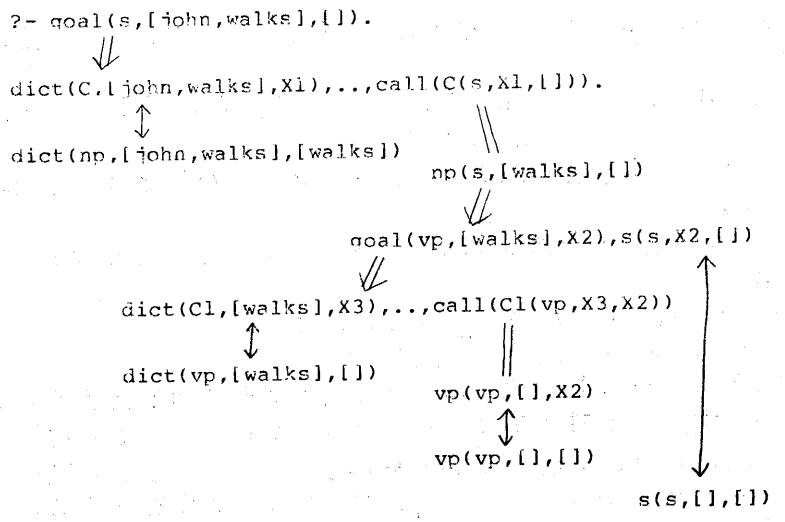

図2. 例1の実行の流れ

3. 評価（DCGとの比較）

今まで述べたように、著者らがBUPを考えたのは、DCGの欠点を補うためである。著者らがまに克服しようとしたDCGの欠点とは、

1) 左再帰的な規則が記述できないこと、

2) 文法と辞書が分離されていないこと　　などである。

BUPはこれらの欠点はなくなっているが、文法がcycle-freeかつε-rule-freeであることは要求する。　またBUPにはDCGにはない補助的なプログラム（goal, link, 停止条件節など）が必要であり、その辺りのオーバーヘッドが小さいプログラムや比較的決定的に解析が進む入力文に対して悪影響する。

文法カテゴリを表わす述語の引数も、常に目標を持ち歩くため多くなる。またこれは前章では述べなかったが、DCGの文法カテゴリを表わす述語が持つ引数の倍の引数をBUPでは持つ必要がある。現在の所、著者らは、DCGの文法カテゴリがもつ引数をひとまとめのリストにし、余分に1つ変数（これはパージングに伴って得られる情報を運ぶ）を付け加え、入力文を表わす2変数を含めると、合計5引数の述語に統一してしまっている。図3に具体的なシステムからの例を示す。BUPの文法規則の中で、変数Iが最終的な情報を運ぶ変数である。これに伴い、停止条件を表わす節も例えば、np(np, I, I, X, X). のように表わされることになる。図3の例は、日本語モンテギュ文法システム[松本82]からの例である。このシステムはDCG, BUPの両方で記述されている。いくつかの日本語文に対する両者の処理時間を表1に示す。それぞれの入力文は、

1. 太郎は目が美しい女を愛している。
2. 大きい箱が上にある机の上に小さい箱がある。
3. 太郎を愛している女を愛している男を花子は愛している。
4. 太郎が愛している女は目が美しい。

例をみると、日本語の埋め込み文などはbottom-upの構文解析の方が向いており、しかも、2, 3のように複数の解釈が存在する場合にも、日本語の解釈として自然なものが一義に決まる。

	DCG		BUP	
	I	C	I	C
1.	859	239	806	158
2.	1536	498	3676	829
	2332	820	8110	2167
3.	3124	1366	418	109
	830	317	1343	287
4.	1303	330	648	122

I : interpreter, C : compiled

表1. 処理時間の比較（msec）

DCG

```
np([A,NS],[np,DET,NOUN]) -->
        det([DH,DS],DET),noun([NH,NS],NOUN),
        {[DH,NH]=A}.

noun([[lambda,X,[box,X]],[[self,box]]],[noun,hako])
                                --> [hako].
```

BUP

```
det(G,I,[[DH,DS],DET]) -->
        {link(np,G)},
        goal(noun,[[NH,NS],NOUN]),
        {[DH,NH]=A},
        np(G,I,[[A,NS],[np,DET,NOUN]]).

dict(noun,[[[lambda,X,[box,X]],[[self,box]]],[noun,hako]])
                                --> [hako].
```

図3. DCGとBUPの比較

cycleを含む文法規則はDCG同様BUPでも扱うことができない。しかし、ε-ruleについては、文法を分割したり、Prologのor表現を使ったりして、BUPに埋め込むことは可能である。

4. おわりに

Prologに埋め込まれたbottom-up parserとしてBUPを提案した。DCGと同様Prologの中に文法規則がコンパイルされた形で埋め込まれる。しかも、文法規則の制限が少なく、辞書が文法規則と分離されているため、大規模な辞書を扱う場合などにもモジュラ性がよい。しかし、文法規則などの開発中には、DCGに比較すると、BUPは途中経過が非常に追い難い。BUP向きのトレーサーの構築が、BUPを自然言語処理の道具として使ってゆく上で是非必要である。

DCGで記述された文法規則をBUPに変換するプログラムは現在作成中で、著者らは、DCGを用いて、簡易版を作っている。現在ICOTでより高速の変換プログラムを作成中である[横井82]。

謝辞

本研究の機会を与えて下さった本所石井治パターン情報部長に感謝します。長年に渡り御指導いただいた淵一博ICOT研究所長に感謝します。討論いただいた横井俊夫ICOT第三研究室長はじめICOT第三研究室の皆様、また本所推論機構研究室の諸氏に感謝します。

（参考文献）

[Pereira 78] Pereira, L., Pereira, F. and Warren, D., User's Guide to DEC system-10 Prolog, Dept. of AI. Univ. of Edinburgh, Sept. 1978.

[Pereira 80] 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. pp231-278. May. 1980.

[Aho 72] Aho, A.U. and Ullman, J.D., The Theory of Parsing, Translation, and Compiling, vol. 1 Parsing, Prentice-Hall, 1972

[Earley 68] Earley, J., An Efficient Context-free Parsing Algorithm, Ph.D. Thesis, Carnegie-Mellon Univ. 1968.

[Pratt 73] Pratt, V.R., "A Linguistic Oriented Programming Language," 3rd IJCAI, pp. 372-381. Aug. 1973.

[Pratt 75] Pratt, V.R., "LINGOL — A Progress Report," 4th IJCAI, pp.422-428, 1975.

[横井 82] 横井俊夫他, "DCG/Bトランスレータ ユーザーズマニュアル," ICOT 所内資料, Oct. 1982.

[松本 82] 松本裕治, 田中穂積, "日本語モンテギュ文法の実働化と質問応答への応用," 情報処理学会 自然言語処理研究会 31-7, May, 1982.

[元吉 82] 元吉文男, "LINGOLコンパイラ," 情報処理学会 記号処理研究会, Dec. 1982.