自然言語処理のための ソフトウエアシステムＬａｎｇＬＡＢ

田中穂積、上脇正、奥村学、沼崎浩明
〈東京工業大学工学部〉

1・ はじめに

筆者らはこれまで、ロジックプログラミングの枠組みで自然言語処理の研究を進めてきた。ロジックプログラミングの枠組みと自然言語処理との整合性が良いと考えたからである［田中 84］。両者の整合性の良さは、フランスのマルセイユ大学のColmeraureの開発した Metamorphosis Grammar[Colmeraure 78] において初めて明確に示された。

Colmeraure が Metamorphosis Grammar で示した基本的な考え方は次の様なものである。まず、拡張文脈自由文法の一形式である Metamorphosis Grammar で書かれた文法規則を、それと一対一に対応する Prolog プログラム (Horn節) に変換する。変換後の Prolog プログラムに自然言語の文を与えて実行すると、トップダウンで縦型探索 (depth first)による 構文処理を行うことができる。この方法によれば、従来、自然言語を構文処理する場合に作成しなければならなかったパーザの作成が不要になる。Prolog のインタープリタに組み込みの機能をそのまま用いて、パーザを代用することが可能になるからである。

Metamorphosis Grammar のもう一つの重要な特徴は、構文処理と意味処理とを融合させた自然言語処理が可能になる、ということである。Metamorphosis Grammar の形式に自然な拡張を施すことで、それが可能になるのであるが、これは認知心理学の観点からも好ましいことであると考えられる。

その後イギリスのエジンバラ大学の Pereira 等は、Metamorphosis Grammar の考え方をさらに洗練した Definite Clause Grammar (DCG) を開発している[Pereira 80]。Pereira等の DCG も Metamorphosis Grammar と同様、（構文処理と意味処理とを融合した）トップダウンで縦型探索の自然言語処理を行う Prolog プログラムに変換されるが、この方式には一つの大きな問題がある。それは、左再帰的な文法規則があると、構文処理の過程で無限ループに陥るという問題である。この問題はボトムアップな構文処理を行うことにより避けることができる。

電子技術総合研究所の松本等の開発した方法によると、DCG の形式で書かれた文法規則を、ほぼそれと一対一に対応する Prolog プログラムに変換し、この Prolog プログラムに自然言語の文を与えると（構文処理と意味処理とを融合した）ボトムアップで縦型探索の自然言語処理を行うことができる[松本 83]。これは、BUPシステムと呼ばれている。

その後東京工業大学の今野等は、BUPシステムに左外置変形が扱えるような拡張を施した BUP_XGと呼ばれるシステムを開発している[今野 86]。本稿で説明する自然言語処理システム LangLAB は、この BUP_XG をベースにした自然言語処理のためのソフトウエアシステムである。２章では、DCG による文法記述形式と BUPシステム の基本的な考え方を簡単に説明し、LangLAB システム で採用されている DCG_XG と呼ばれる文法記述形式を説明する。

LangLAB システム には BUP_XG と比較しておよそ 6.5倍ほど高速な構文処理アルゴリズムが組み込まれている。３章ではこの高速化の方法を実験結果とともに示す。［松本 83］では、再計算を避けるアルゴリズムを導入することによって、構文処理速度がおよそ１桁改善可能なことが報告されているから、再計算を避けるアルゴリズム導入以前の、最も初期のBUPシステムに比べてLangLAB システムは、構文処理に関しておよそ６０倍以上の高速化がはかられていることになる。筆者は LangLABシステムによる構文処理については、十分満足できる速度が得られたと考えている。４章では、LangLAB システムのもつ文法開発支援環境について述べる。

機能の面から見ると BUPシステムのスーパーセットが BUP_XGシステムであり、BUP_XGシステムのスーパーセットが LangLAB システムになっている。

2・ ＤＣＧ，ＢＵＰ，ＤＣＧ_ＸＧ，ＢＵＰ_ＸＧ

LangLAB システム では文法規則の記述は、BUP_XG システムで用いた形式、すなわち DCG に左外置が扱えるよう拡張を施した DCG_XG 形式を用いる。その具体的な説明に入る前に、DCG とBUP システムの関係を説明しておく。これは、３章の理解の基礎にもなる。

2・1 ＤＣＧとＢＵＰシステム

BUPシステム では、文法記述者が書いた DCG 形式の文法規則は、BUPトランスレータによって、BUP節に変換される。このBUP節は Prolog インタープリタにより最終的にProlog プログラムに変換される。変換後の Prologプログラムには、処理すべき自然言語の文が与えられて、ボトムアップで縦型探索の自然言語処理が行われる（図１）。以下では説明を簡単にするために、構文処理の流れにだけ焦点を当てるので、DCG には意味処理を行うプログラムが補強されていないものを用いる。したがって、純粋に構文処理だけを行う例を示す。まず図２－１のDCG形式の文法規則と辞書は、BUPトランスレータによって図２－２に示すBUP節に変換される。この時、後述する link節も計算され BUP節に付加される。図３－１のBUP節は更にPrologインタープリタによって図２－３のPrologプログラムに変換されて実行される。この Prolog プログラムに現われる各非終端記号に対応した述語と述語 goal には、二つの引数が付加されている。後述するがこれらは、処理すべき文の部分の差分リストによる表現になっている。

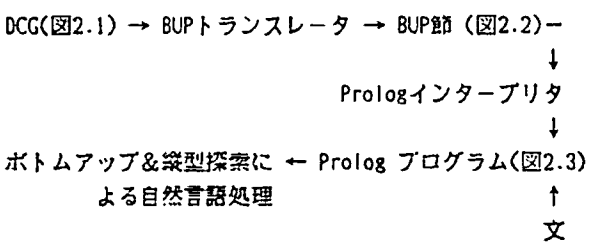

図1　BUP システムによる自然言語処理の様子

```
(a1)   s --> np, vp.
(b1)   np --> pron.
(c1)   pron --> [you].
(d1)   vp --> [walk].
```

図2－1　DCGによる文法規則例

```
(a2)   np(G) --> {link(np,G)},
                 goal(vp),
                 s(G).
(b2)   pron(G) --> np(G).
(c2)   dict(pron) --> [you].
(d2)   dict(vp) --> [walk].
```

図2－2　図2－1を変換後のBUP節

```
(a3)   np(G,X,Z) :- link(np,G),
                    goal(vp,X,Y),
                    s(G,Y,Z).
(b3)   pron(G,X,Y) :- np(G,X,Y).
(c3)   dict(pron,[you|X],X).
(d3)   dict(vp,[walk|X],X).
```

図2－3　最終的なPrologプログラム

BUPシステムにはボトムアップな構文処理を行うための述語goalが用意されている。その定義を図3に示す。

```
(A1)   goal(G,X,Y) :- ( wf_goal(G,X,_) ;
                        fail_goal(G,X),!,fail ),
                      !,wf_goal(G,X,Y).
(A2)   goal(G,X,Z) :- dict(C,X,Y),
                      link(C,G),
                      P =.. [C,G,Y,Z],
                      call(P),
                      assertz(wf_goal(P)).
(A3)   goal(G,X,Y) :- ( wf_goal(G,X,_) ;
                        assertz(fail_goal(G,X) ),
                      !,fail.
```

図3　述語 goal の定義

図2－3の文法規則を用いて "you walk." という文を構文処理する手順を説明する。

（1）　"you walk." という文の処理は、つぎの述語goalの実行により開始する。

```
?- goal(s, [you,walk],[]).
```

これは、[you,walk]の後方から[]を差し引いた残りの部分（これを差分リストという）、即ち[you, walk]という文が将来 s というゴールを満しうる（s という根をもつ構文木ができる）かどうかを調べることを意味している。

（2）　述語 goal の実行により、(A1) が呼出されるが (A1) では再計算を避けるために成功結果と失敗結果とが、それぞれ wf_goalとfail_goalにあるかどうか調べる。

（3）　上の例の場合には (A1) の実行は失敗するので、つぎの (A2) が選ばれる。

（4）　(A2) のボディでは、述語 dict(C,[you,walk], Y) を実行し、入力文の単語のリストX の先頭から辞書引きが始る。これは、図2－3の (c3)とマッチするので、文法カテゴリCに pronが、Y に[walk]が返される。

（5）　(A2)のボディでは次の述語link(pron,s)が実行されるが、これは構文処理が先に進み、構文木が下から上向きに (bottom-up) に成長したとき、pronからゴール s に到達可能かどうかを調べる述語である。link述語はあらかじめBUPトランスレータによって（与えられた文法規則の集合から）計算されている。今の場合それによりlink(pron, s)が成功するものとしよう。そうすると次は文法カテゴリpronと同じ名前の述語を呼出す：

```
P =.. [pron,s,[walk],[]),call(P)。
```

ここで call(P) は、call(pron(s,[walk],[])) であることに注意しよう。

（6）　pron(s,[walk],[])の実行により、今度は(b3)が呼出され、そのボディの np(s,[walk],[])が実行される。これは、npを根とする構文木が完成したことを意味している。np(s,[walk],[])の実行により、次は(a3)が呼出される。(a3)のボディでは、link(s,s)が調べられるが、これは必ず成功する。linkの検査の後では次の goal(vp,[walk],[]) が実行される。これは、np までの処理が終り、次に処理すべき単語の系列が walkで始まり、その満たすべきゴールが vp であることを予測していることになる。

以下同様にしてトップダウンの予測を利用したボットムアップの構文処理が進むが、停止条件節がないと goal 節の実行が成功裏に終了しない。停止条件節については [Matsumoto 83]を参照されたい。さて(A2)のボディの最後では、 今実行中のゴールが成功したので成功結果を wf_goal としてアサートしておく。同じ計算がバックトラックにより繰り返される時には、(A1)により直ちに成功結果を取り出すことができる。(A3)は失敗結果をアサートしておくための節である。

最後に構文処理の過程で意味処理を行いたければ、例えば以下に示すようにDCGのボディ中に、 意味処理用の Prolog プログラムを中括弧で括って挟み込めば良い。意味処理結果を控えておくために、各非終端記号に対応する述語に引数をもたせておく：

```
s(Ssem) --> np(NPsem),
            vp(VPsem),
            {seminterp(NPsem,VPsem,Ssem)}.
```

これは最終的に、

```
s(Ssem,X,Z) :- np(NPsem,X,Y),
               vp(VPsem,Y,Z),
               seminterp(NPsem,VPsem,Ssem).
```

のように変換され実行される。seminterpでは、NPsemとVPsemとを用いて意味処理した結果をSsemに返す。

2・2　DCG_XGとBUP_XGシステム

英語の関係代名詞節の埋め込み文は、平叙文中の名詞句が一つ欠落した構造をしているが、これは先行詞が関係代名詞節の左方に移動してできたと考えられる。この様な語句の移動を左外置と呼んでいる。この場合、移動した跡には痕跡を残すと考え、この痕跡をシステムに発見させることを前提にして文法規則を記述すると、そうでない場合に比べて、文法規則の数や文法カテゴリの数を減らすことが可能になり、文法の見通しが良くなる。

トップダウンの構文処理システムの ATNや[Woods 71] XGには[Pereira 81]、痕跡発見機構が組み込まれている[Winograd 83]。トップダウンの構文処理システムの場合には、次に処理すべき文の部分がどの非終端記号に縫められるべきかが予測できる。したがって、予測された文法カテゴリが特定のもの、例えば np の時のみ痕跡の存在を仮定することができるので、効率良く痕跡を発見することができる。

ところが純粋にボトムアップの構文処理システムの場合には、その様な予測が行えないから、処理すべき文の単語と単語の間のほとんど全てに痕跡があるとして処理を進めなければならないので、効率が悪い。幸いなことに、前節で述べた様に BUPシステムの場合には、ボトムアップを基本としているものの、トップダウンの予測をも利用している。[今野 86]はこの点に着目して BUPシステムに痕跡発見機構を組み入れたBUP_XGシステムを開発している。LangLABシステムは、このBUP_XGシステムをベースにした自然言語処理システムである。

BUP_XGシステムでは痕跡をサーチし発見するために、[Pereira 81]の XG で導入された Xリスト(extraposition list) を用いる。それに伴い goal節も変更されるが、その詳細は [今野 86] を参照して頂くこととし、LangLAB システム の使用者は、左外置が扱えるよう DCG を拡張した DCG_XG ([今野 86]ではこれを XGSと呼んでいる) に

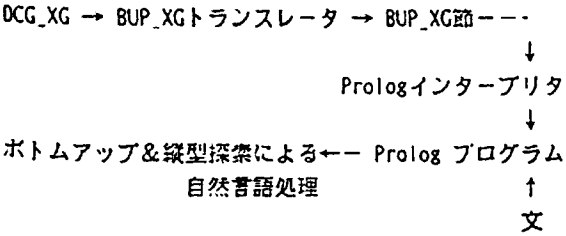

図4　BUP_XG システムによる自然言語処理の様子

よる文法記述形式を理解するだけで十分だろう。

さて DCG_XG による文法記述では、次の記法が許されている：

```
np --> det, noun, srel../np.
```

記号 "../" (スラッシュとよぶ) に後続するカテゴリをスラッシュ・カテゴリとよぶ。この記法は GPSG を参考にしたものである [Gazdar 82]. "srel../np" は、srel の中に痕跡となるスラッシュ・カテゴリ np が一つ存在することを表している。

スラッシュ・カテゴリと痕跡との対応例を図5に示す。文法記述者は、 DCG_XG の形式で文法を書きさえすれば、LangLAB システムに組み込まれた BUP_XGシステムによって、自動的にスラッシュ・カテゴリと痕跡との対応がとられる。

DCG_XG には、以下に示すような "<" と ">" で囲んだ記法がある。

```
np --> det, noun, <srel../np>.
```

この場合には、srel 中の (スラッシュ・カテゴリ np に支配される) 痕跡は、det と noun とからなる名詞句以外とは対応付けができない。したがって英語の場合、複合名詞句制約に違反した非文法文の構文処理に成功することはなくなる。ここでく,>で囲んだ記法は、 [Pereira 81]の

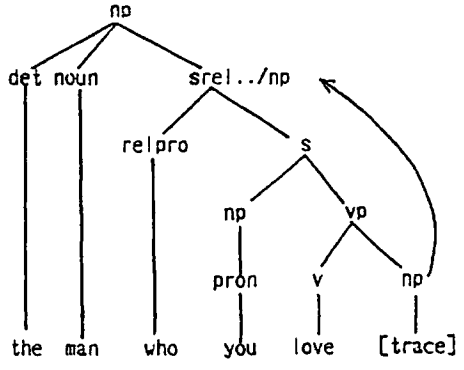

図5　スラッシュ・カテゴリと痕跡との対応

open, close カテゴリで括ることに対応しているが、LangLAB システムで採用した今野らのDCG_XGの記法の方が簡潔であると思うがどうだろうか。

DCG_XG の形式で書かれた上記した文法規則が BUP_XG トランスレータによりどの様な BUP_XG節に変換されるか、またBUP_XGシステムには coordinate structure を扱う機能もあるがそれらがどの様に実現されているか、については[今野 86]を参照されたい。

3・　LangLABトランスレータによるBUP_XG節の最適化

LangLAB システムには前節で説明した BUP_XG節をおよそ6・5倍高速化し、使用記憶容量を節約した構文処理アルゴリズムが組み込まれている。これは以下で説明する LangLAB トランスレータによって DCG_XG から最適化したBUP_XG節を生成することで実現される。図6に LangLAB システムの概要を示す。3章では、LangLAB トランスレータについて説明する。 LangLAB トレーサについては、4章で説明する。

３・１　Ｌｉｎｋ節の除去による高速化

使用文法規則の数が多くなるにつれて、 BUP_XGトランスレータにより生成されるlink節の数は著しく増大する。ちなみに筆者等が使用している英語の文法規則の数はおよそ400であるが、BUP_XG トランスレータにより生成されるlink節の数は約600に達している。

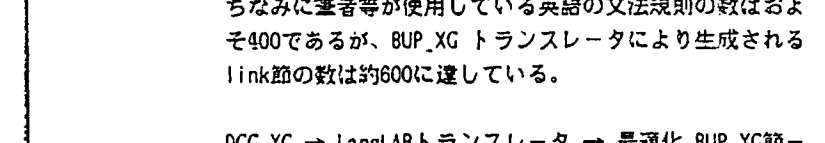

図６　LangLAB システムによる自然言語処理の様子

したがって、link節の検索時間の短縮が望まれる。

BUP_XG節のボディに現れる述語 linkを調べてみると、必ずその第一引数はアトムであり、第二引数は変数になっている。link(a,G) の形をしているのである。LangLAB トランスレータはDCG_XGを変換する時、BUP_XG節のボディに現れる link(a,G)を a(G) に変えたものを出力する。これを以下ではリンク対とよぶ。

たとえば２・２の次の文法規則：

```
np --> det, noun, srel../np.
```

は、大略次のBUP_XG節に変換される。 ここで、np(G) がリンク対であることに注意。

```
det(G,...) --> {np(G)},
               goal(noun,...),
               goal(srel,...),
               np(G,...).
```

このような最適化によって、非終端記号 np からゴール G への到達可能性を調べるための時間が大幅に短縮される。述語名 np をキーにしたハッシュ検索がなされるためである。

一方 goal節のボディに現われる辞書引き後の link(C,G) は（図３（Ａ２））、

```
Link =.. [C,G],
call(Link)
```

に変更する。

さてリンク対の実際の呼び出し時の様子を観察してみると、第一引数 G は常にアトムになっているから決定的な動作をする。したがって、BUP_XGトランスレータが計算し出力するlink節のボディには、カット記号を入れることができる。そこで、LangLAB トランスレータは、BUP_XGトランスレータが出力する link(f,g) を次のものに変えて出力する。

```
link(f,g). ===>  f(g) :- !.
```

以上の様にして link節を別の形にして除去することにより、構文処理速度をおよそ６倍向上させることができる。筆者等はこの改善により、LangLAB システムの構文処理速度は十分満足できるレベルに達したと考えている（３・４の実験結果参照）。

３・２差分リストのインデックス化表現による記憶の節約

２・１では、文の処理対象部分が差分リストとして表現されることを説明した。使用文法規則の数が少なく小規模であるうちは、この差分リストによる表現も大きな問題とはならない。構文処理過程で生じる中間的な成功結果と失敗結果とは、それぞれwf_goalとfail_goalとしてアサートされることを２・１で述べた。この時、述語wf_goalの最後の２引数は、処理対象部分文の差分リストになっている。 文法の規模が大きくなり、しかも処理すべき文の長さが長くなるにつれて、アサートされるwf_goal節とfail_goal節の数が著しく増大し、そのために大きな記憶容量を必要とする。この記憶容量は、差分リストをインデックス化することで節約できる。また計算結果を再利用する場合にも、それを検索するためのユニフィケーション時間が短縮されるため、 構文処理速度が幾分改善される[今野 84]（３・４の実験結果参照）。

インデックス化のために、たとえば "you walk." という文が入力された段階で、

```
text(0,[]).
text(1,[walk]).
text(2,[you,walk]).
```

をアサートしておく。そして辞書引きのとき与えられるインデックスの対から、述語 text を呼出して差分リストを復元し、実際の辞書引きを行う。したがって、goal 節のボディに現れる （辞書引きのための） 述語 dict は、LangLABシステム では dictionaryという述語に変えられている。

述語 dictionary は、インデックスの対から（処理対象部分文の）差分リストを復元して辞書引きするだけでなく、辞書引きに失敗した場合には形態素処理を行う。また、辞書引きか形態素処理のいずれかに成功した場合には、その成功結果を wf_dict としてアサートしておき、再計算に備える。

３・３　直接辞書引きによる高速化

```
np --> det, noun, srel../np.
```

という文法規則は、概略次の BUP_XG節に変換されることを３・１で述べた。

```
det(G,...) --> {np(G)},
               goal(noun,...),
               goal(srel,...),
               np(G,...).
```

ここで noun は単語の品詞であるとしよう。単語の品詞は非終端記号に属すが、他の非終端記号と異なり、書き換え規則により別の非終端記号に展開されることはない。言い換えると、単語の品詞からは書き換え規則により単語（終端記号）にしか展開されない。したがって BUP_XG 節の右辺から、noun の様な非終端記号に対する述語 goal を呼出す場合には、述語 goal の代りに述語 dictionary（３・２参照）を用いて、直接辞書引きしても良い。上記した BUP_XG 節を次のように変えてもよいのである。

```
det(G,...) --> {np(G)},
               dictionary(noun,...),
               goal(srel,...),
               np(G,...).
```

LangLAB トランスレータは、与えられた文法規則をみて、直接辞書引き可能な非終端記号を抽出し、上に示すBUP_XG節を出力する。直接辞書引きを行う BUP_XG 節を用いることにより、アサートされる wf_goal,fail_goal の数が減り、構文処理速度が1割弱改善される（3・4の実験結果参照）。

3・4　BUP_XG節の最適化に関する実験結果

LangLABトランスレータの出力する最適化 BUP_XG 節により、構文処理に関し、およそ6・5倍程の高速化が得られることをすでに述べた。それを確認するために次の（C）に示す25の例文に対して実験を行った。（D）に示す実験結果はC－Prologインタープリタによる実験結果である。コンパイルすると、構文処理時間は更に（D）の少なくとも1／5程度に短縮できよう。なお使用文法規則数は、DCG_XG の形式でおよそ400ほどである。

（A）使用計算機：SUN3/160ワークステーション

（B）使用Prolog：C－Prologインタープリタ

（C）使用例文：

1 There are three on the table now.
2 There are some men here from the city.
3 Those are the books she read.
4 Were they given to her?
5 Could she have been given many of them?
6 I saw many more books than she did.
7 I saw that they were there.
8 The books that were on the table are difficult to read.
9 My uncle gave the girl several books.
10 She was given some books by her uncle.
11 The books were given her by her uncle.
12 He gave up the books to her.
13 You could break this vase with that hammar.
14 That hammar could break this vase easily.
15 Did not those people want him to try to do it?
16 Are there any more left?
17 Is this any harder for him to do than that was?
18 This is a film developed in the research.
19 The system for interpreting dialogues is developed.
20 She was given more difficult books by her uncle.
21 Do not do that!
22 Be very careful not to do it too quickly.
23 Read these books, if you have time.
24 If you have time, read these books.
25 This paper presents an explanatory overview of a large and complex grammar that is used in a computer system for interpreting English dialogues.

（D）実験結果と評価

実験は上記した各例文について、全ての構文処理結果が得られるまでの時間（秒）を以下の4つの場合に分けて測定した。

（a）最適化する前のBUP_XG 節を使用、
（b）Link節の削除（3・1）、
（c）差分リストのインデックス化（3・2）、
（d）直接辞書引き（3・3）。

測定時間には、形態素処理の時間も含まれている。また、例文番号に括弧付きで添えられた数字は、得られた構文処理結果の数を表す。

例文番号	(a)	(b)	(c)	(d)
1(2)	13.11	1.85	1.83	1.73
2(1)	9.78	1.51	1.61	1.53
3(1)	17.15	3.13	3.08	3.07
4(2)	27.60	4.45	4.40	4.35
5(1)	32.43	5.61	5.43	5.10
6(2)	46.5	8.05	7.80	6.92
7(1)	13.2	2.23	2.13	1.92
8(1)	52.56	8.55	8.31	8.10
9(1)	24.00	3.93	3.85	3.63
10(2)	43.21	7.11	6.88	6.48
11(1)	34.01	5.46	5.31	4.87
12(1)	15.61	2.45	2.43	2.20
13(2)	24.35	3.86	3.76	3.35
14(2)	15.38	2.81	2.73	2.43
15(1)	72.01	10.8	10.46	9.60
16(1)	17.78	3.03	3.10	3.03
17(1)	25.11	4.63	4.48	4.15
18(1)	30.10	4.93	4.83	4.60
19(1)	25.73	4.41	4.46	4.42
20(4)	82.75	14.53	14.00	12.68
21(1)	18.61	2.63	2.61	2.43
22(3)	18.50	3.26	3.16	3.10
23(1)	26.68	4.56	4.40	4.08
24(1)	57.48	9.23	9.06	8.76
25(8)	186.36	47.48	46.55	39.93

以上の結果を比較すると、次のことが分かる。

（1）　（a）／（b）＝5．99
（2）　（a）／（c）＝6．10
（3）　（a）／（d）＝6．52

3・1節、3・2節、3・3節に述べた最適化により、構文処理速度が次第に向上し、最終的には6・52倍の高速化が実現されていることが分かる。特に link 節の除去による高速化が顕著である。Prologインタープリタではなくコンパイラが使用できれば、構文処理時間は更に1／5に短縮されようから、LangLAB システムは十分な構文処理速度に達した考えている。

4・　LangLABの文法開発支援環境

本章では、LangLAB システムの文法開発支援環境について説明する。前章の図6に示した LangLAB トレーサについては、実例とともに説明する。

4・1　文法規則の変更

文法が大規模になるにつれて、文法開発支援環境が重

要になる。文法を開発する段階では、DCG_XG の形式で書かれた文法規則を修正したり削除したり、新しい規則を付加することが頻繁に行われる。従来のトランスレータでは、この様な変更が生じるたびに、すべての文法規則を最初から変換し直さなければならなかった。文法規則の数が少ないうちは、これはさほど大きな問題ではないが、文法規則の数が２００を越えると変換に時間がかかるので問題である。ＬａｎｇＬＡＢ トランスレータでは、一度変換した DCG_XG 形式の文法規則の一部を変更して再変換する場合に、変更されなかった文法規則については、すでに変換済みの BUP_XG 節をそのまま用いる機能があるため、変換速度が大幅に改善されている。

このような機能の実現はさして困難ではないように見える。しかし、 LangLAB トランスレータは（BUP_XG の）link節に相当するリンク対を計算するために全ての文法規則を見る必要がある。このリンク対の計算を再変換に際して高速に行うためには、様々な情報を管理しておく必要がある。これら再変換に必要となる情報の管理は LangLAB システム が全て行ってくれる。それにより文法規則の変更が全体の１割であれば、再変換に要する時間も、全部を変換し直す時間の１割程度に抑えることができる。

４・２　構文処理過程のトレース機能

文法規則の開発中には様々なエラーが発生する。LangLAB システム は DCG_XG 形式で書かれた文法規則を、最適化された BUP_XG節に変換し、それを Prolog プログラムとして実行し構文処理を行うシステムである。 もし、DCG_XG 形式の文法規則に {、} で囲まれた意味処理用の Prolog プログラムを補強すれば、構文処理過程で意味処理をも行ってしまうことも可能となる。この様な方式で自然言語処理を行っているため、文法規則に含まれるエラーは、Prolog プログラム実行時のエラーとして顕在化する。この時文法規則開発者は、エラーの原因を探るため構文処理過程をトレースすることが必要になるが、これまでのトレーサは、Prolog に組み込みの機能で代用されていたため、不必要なまで詳細なトレースを行うことを余儀なくされていた。

LangLAB システム に組み込まれているトレーサは、以下の機能を持っている：

（１）文法規則適用過程のみのトレース機能、
（２）それまでに作られた（中間的な）構文木を図示する機能、
（３）トレースを途中で中断し、文法規則修正後に、修正された文法規則を用いたトレースの再開機能。

以上の機能を持つトレーサを実現する方法として次の２通りが考えられる。

（ｉ）DCG_XGを変換した最適化BUP_XG節のインタープリタを別途作成し、その中にトレーサを埋め込む、
（ｉｉ）DCG_XGを解釈しBUP_XGシステムの動きをシミュレートするインタープリタを作成し、その中にトレーサを埋め込む。

（ｉ）は [ICOT 83] の方法であり、（ｉｉ）は我々の LangLAB システムが採用した方法である。（ｉ）の方法は、上記した（３）の機能を持たせようとした時に問題となる。なぜならシステム使用者が修正する文法規則は、DCG_XG 形式の文法であり、その変換結果である最適化BUP_XG節ではないからである。言い換えると、我々から見える文法規則は DCG_XG 形式の文法規則である。したがって、もし（ｉ）の方法を採用するなら、DCG_XG 形式の文法規則を修正後に、それを最適化 BUP_XG 節に変換し直してからトレーサを働かせなければならない。文法規則の開発中途では、文法規則の修正は頻繁に起きる。 その都度LangLAB トランスレータを動作させてDCG_XG形式の文法規則を変換し直さなければならないということは、文法規則開発者にとって煩わしく、しかも変換に時間がかかるため好ましいことではない。そのためか、[ICOT 83] には（３）の機能は含まれていない。しかし、文法開発者の立場からすると（３）は最も重要な機能である。

それに対して（ｉｉ）の方法では、システム使用者がDCG_XGを修正後ただちにトレーサを働かせることができる。実際、（ｉｉ）の方法を採用した我々のLangLAB システムには（３）の機能が実現されている。

LangLAB システム の文法開発支援環境の使用例を付録に示す。ただしこの例では小規模な実験文法を用いる。そして、辞書項目 "with" の品詞が誤って pron になっている。トレーサを動作させた構文処理過程でそれが発見されたので、構文処理を一旦中断し、品詞を prep に修正して構文処理を続行する様子が記録されている。ただし、emacs エディタを動作させて辞書を修正する部分は記録から省略されている。

５・　おわりに

LangLAB システムの構文処理速度に関して、筆者らは十分満足できるレベルに達したと考えている。LangLAB システムは、今後自然言語処理システムを開発するための有力なツールとなると考えている。ただ、現在の LangLAB システムには、not only ... but also ...などといった複雑な熟語を柔軟に扱う機能が組み込まれていない。しかし[上脇 85]は、 BUP_XG システム上で熟語を柔軟に扱う方法を開発している。これは、辞書の構造をTrie構造化[Aho 83]して実現したものであるが、現在これを LangLAB システム 上にインプリメントする作業を進めている。

今後は構文処理のレベルを越えた自然言語処理（たとえば意味処理）をロジックプログラミングの枠組みの中で行いたいと考えている。筆者らはすでにDCKRとよばれる意味表現形式を提案している[小山 85],[田中 86], [奥村86]
それによれば、意味処理に要するプログラム作成労力が著しく軽減される。また、単語の多様な意味記述が可能になるので、従来困難だと思われていた、複雑で高度な意味処理を行うことが可能になると考えている。いささかオーバーな言い方をすれば、LangLAB システムや DCKR とよばれる意味表現形式を用いることにより、構文処理や意味処理に必要とされていたプログラム作成労力の一部が軽減されて、誰でも自然言語処理における最も困難な問題に、直接取り組むことが可能になってきているといえよう。以上

の考え方にしたがって、ロジックプログラミングの枠組みの中で今後各種の自然言語処理応用システムを作成したいと考えている。

謝辞

本研究は ICOT と ETL と東京工業大学の田中研究室との間で行われた BUP 検討会での議論がベースになっている。BUP 検討会を主宰されたICOTの第一研究室の松本裕治氏ならびに参加者の皆様に感謝する。

参考文献

[Aho 83] Aho,A.V., Hopcroft,J.E. and Ullman,J.D.: Data Structures and Algorithms, Addison-Wesley (1983).

[奥村 86] 奥村学、高倉伸、田中穂積：意味記述用言語 SRL/0 の設計と DCKR、情報処理学会自然言語処理研究会資料 54-5,(1986).

[Colmeraure 78] Colmeraure,A.: Metamorphosis Grammar, in Bolc.(ed.): Natural Language Communication with Computers, Springer-Verlag, 133-190(1978).

[Gazdar 82] Gazdar,G.,Pullum,A.F.:Generalized Phrase Structure Grammar: A Theoretical Synopsis, Indiana University Linguistics Club,(1982).

[ICOT 83] 昭和57年度通商産業省委託業務電子計算機基礎技術開発成果報告書－－基礎ソフトウエア・システム編、第二分冊、ICOT, 251-265(1983).

[上脇 85] 上脇正、田中穂積： 辞書のＴＲＩＥ構造化と熟語処理、Proc.of Logic Programming Conference '85,ICOT,329-340(1985).

[今野 84] 今野聡、奥村学、田中穂積：ボトムアップ構文解析システム BUP の高速化、 日本ソフトウエア科学会第１回大会論文集、3A-2(1984).

[今野 86] 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、日本ソフトウエア科学会誌、コンピュータソフトウエア、3,2,115-125 (1986) .

[小山 85] 小山晴生、田中穂積：Definite Clause Knowledge Representation, Proc.of Logic Programming Conf.'85, ICOT, 95-106(1985).

[Matsumoto 83] Matsumoto, Y. et.al.: BUP--A Bottom-up Parser Embedded in Prolog, New Generation Computing, 1, 2, 145-158(1983).

[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, Journal of Artificial Intelligence,13, 231-278(1980).

[Pereira 81] Pereira, F.: Extraposition Grammar, American Journal of Computational Linguistics, 7, 4, 243-256(1981).

[田中 84] 田中穂積、松本裕治 ： 自然言語処理における Prolog、情報処理、25,12, 1396-1403(1984).

[田中 86] 田中穂積 ： 知識表現形式 DCKR とその応用、ICOT-TR (印刷中) .

[Winograd 83] Winograd,T.: Language as a Cognitive Process, Vol.1:Syntax,Addison-Wesley(1983).

[Woods 71] Woods,W.A.: Experimental Parsing System for Transition Network Grammar, in Rustin(ed.): Natural Language Processing, Algorithmic Press (1971).

付録 LangLAB トレーサの使用例

```
| ?- start.

 Do you use your morph & util ? n

morph consulted 19776 bytes 6.1 sec.
util consulted 8363 bytes 2.86667 sec.

 c)onsult  or  r)e-consult : c.

 input DCG file ? : gram.

 consult another (y/n) ? : n

input sentence : i open the door with a key.

i open the door with a key.

 set say (y/n) : n

call goal : sentence :                          トレース開始
call dict : i
set  cat  : pron                                辞書引き
pick rule : np(_1113,_1114,np(pron(i))) -->
                pron(a,_1036,pron(i)).  :       規則抽出
pick rule : sentence(_1338,_1339,sentence(np(pron(i)),_1341)) -->
                np(_1113,_1114,np(pron(i))),
                vp(_1344,_1345,_1341).  :       規則抽出
call goal : vp :                                サブゴールの実行
call dict : open                                辞書引き
set  cat  : v
```

```
pick rule : vp(_1678._1679.vp(v(open)._1681)) -->
                v(_1600._1601.v(open)),
                np(_1684._1685._1681).  :        規則抽出
call goal : np :                                 サブゴールの実行
call dict : the
get  cat  : det                                  辞書引き
pick rule : np(_2016._2017.np(1._2018._2019)) -->
                det(_1933._1939.det(the)),
                n(_2023._2024._2018),
                adjp(_2025._2026._2019).  :      規則抽出
call goal : n :                                  サブゴールの実行
call dict : door                                 辞書引き
get  cat  : n
exit goal : n
call goal : adjp :                               サブゴールの実行
call dict : with                                 辞書引き
get  cat  : pron                                 誤りの発見
fail goal : adjp : q                             失敗. q は quit

[ execution aborted ] ／
^Z
Stopped
$ emacs gram
          文法規則の修正を行う.
$ fg      デバッガを再開する.
prolog bupdb.save
| ?- start.
 c)onsult  or  r)e-consult : r.          文法の再登録

 input DCG file ? : gram.

data.ctrl consulted 160 bytes 0.0333394 sec.

 reconsult another (y/n) ? : n

input sentence : i open the door with a key.

i open the door with a key.

 set spy (y/n) : y

 input goal category : adjp.             スパイポイントの設定

 spy another (y/n) : n

call goal : sentence : s                 スキップ
call goal : adjp :                       スパイポイント
call dict : with                         辞書引き
get  cat  : prep                         訂正された部分
pick rule : adjp(_1940._1941.adjp(prep(with)._1943)) -->
              prep(_1862._1863.prep(with)),
              np(_1946._1947._1943).  : t     部分木の表示

 |-sentence
    |-np
    |  |-pron -- i
    |-vp
       |-v -- open
       |-np
          |-det -- the
          |-n -- door
          |-adjp
             |-prep -- with

call goal : np : n                       no trace
exit goal : n
exit goal : np
exit goal : adjp
exit goal : sentence                     構文解析成功
```