解　説

# 自然言語処理における Prolog†

田中穂積†† 松本裕治†††

## 1. はじめに

Prolog の実行が，一階述語論理の定理証明プロセスとして見なしうることは，Kowalski によって明らかにされている．一方，文脈自由文法規則を用いた構文処理プロセスが定理証明プロセスとして見なしうることもすでに明らかにされていた．この二つの事実と，Prolog プログラムの基本であるホーン節の形式と文脈自由文法規則の形式とが良い対応をなすことから，文脈自由文法規則を Prolog プログラムに変換し，それを実行することで構文処理を行う可能性が読み取れる．文脈自由文法規則には非終端記号が現れるが，ホーン節でそれに対応するものは述語名である．述語は任意個の引数がとれる．このことから，ホーン節による文法規則の表現は，文脈自由文法規則の拡張になっていることが分かる．以下で説明するが，それにより，文法規則にさまざまな補強を施すことができるだけでなく，構文処理の精度を上げたり，構文処理と意味処理と文脈処理とを融合させた自然言語処理をも素直に行うことが可能になる．

以上の基本的な考え方を Prolog 上に自然に埋め込み，その有効性を実証したのは，Prolog を着想し実現したフランスのマルセイユ大学の Colmerauer である．彼が示した手法によれば，自然言語処理で必要とされているパーザの機能をそっくり Prolog プログラムの実行機能に委ねることができる[1]．パーザの作成が不必要になるのである．

その後イギリスのエジンバラ大学では，Warren が Dec-10 Prolog を開発した．Pereira と Warren は，Colmerauer の考え方を発展させ，Dec-10 Prolog に DCG とよばれる文法形式を埋め込んだ[2]．その後 DCG⁺に，疑問文や関係代名詞文にみられる左外置変形を扱う機能を持たせた XG を開発した[4]．その概要を2章と3章で説明する．Prolog プログラムの実行戦略はトップダウン&縦型探索法であるから，DCG，XG による自然言語処理もその戦略に従う．しかしトップダウン&縦型探索法によりすべての解を求めようとすると，文法規則に左再帰規則が含まれている場合には無限ループに陥る．ボトムアップ法によればこの問題は解消する．電子技術総合研究所で筆者等は，Prolog によりボトムアップに自然言語を処理する手法を開発した[3]．これは2章の後半で説明する．3章では左外置の扱いを述べる[4]．なお以下のプログラムの説明では，Dec-10 Prolog の記法を用いる．したがって述語の引数に現れる変数は大文字ではじまり，定数は小文字ではじまることに注意されたい．

## 2. DCG (Definite Clause Grammar)

### 2.1 DCG による文法記述形式

マルセイユ大学の Colmerauer の開発した Metamorphosis Grammar[1] の考え方を一層洗練した DCG の形式を説明する．これは Pereira and Warren が，Dec-10 Prolog 上に開発した文法記述の形式である[2]．DCG による小さな日本語文法と辞書項目の記述例を以下に示す．

(A)　$s(S) \dashrightarrow pp(PP), v(V),$
　　$\{\mathrm{interp}(PP, V, S)\}.$

(B)　$pp([N, P]) \dashrightarrow n(N), p(P).$

(C)　$n$([学校, location]) --> [学校].

(D)　$p$(へ) --> [へ].

(E)　$v$([行く, past]) --> [行った].

(A)，(B) は文法規則の記述を，(C)，(D)，(E) は辞書項目をあらわす．DCG の形式と文脈自由文法規則との対応は明らかだろう．以下では(A)から(E)を DCG 節とよぶことがある．(A) の { } で囲まれた述語を除き，DCG の各述語には文脈自由文法規則の非終端記号が対応している．DCG による文法記述の特徴をまとめると次のようになる．

† Natural Language Parsing in Prolog by Hozumi TANAKA (Department of Computer Science, Tokyo Institute of Technology) and Yuji MATSUMOTO (Section of Machine Inference System, Electrotechnical Lab.).

†† 東京工業大学工学部情報工学科
††† 電子技術総合研究所パターン情報部推論システム研究室

（i） 文脈自由文法規則の非終端記号を述語とみなし，これに任意個の引数を持たせることができる．

（ii） 記号 --〉の右側の任意の場所に｛｝で囲まれた(任意個数の)述語が付加できる．これは補強項とよばれる．補強項により構文処理の精度を上げたり，構文処理，意味処理，文脈処理を融合させることも可能になる．

DCG により，どのように自然言語が処理されるかを以下で見てみよう．処理対象文は，「学校へ行った」であるとする．

### 2.2 トップダウン法

トップダウン法による自然言語処理を行うために，Pereira 等の開発した DCG トランスレータは，(A)～(E)を，次の(a)～(e)の Prolog プログラムに変換する．

(a) $s(S, X, Z) :- pp(PP, X, Y), v(V, Y, Z), \mathrm{interp}(PP, V, S).$

(b) $pp([N, P], X, Z) :- n(N, X, Y), p(P, Y, Z).$

(c) $n([\text{学校}, \text{location}], [\text{学校}|X], X).$

(d) $p(\text{へ}, [\text{へ}|X], X).$

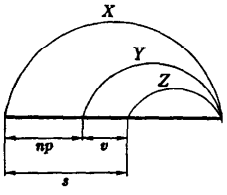
(e) $v([\text{行く}, \text{past}], [\text{行った}|X], X).$

ここで注意すべきことは，｛｝で囲まれた述語 interp 以外の各述語に，新たに2つの引数の対が付加されていることである．たとえば(a)の $X, Z$ と，$X, Y$ と，$Y, Z$ の対は次のように解釈される．

（i） リスト $X$ からリスト $Z$ を差し引いた残りのリストが $s$ である．

（ii） リスト $X$ からリスト $Y$ を差し引いた残りのリストが $np$ である．

（iii） リスト $Y$ からリスト $Z$ を差し引いた残りのリストが $v$ である．



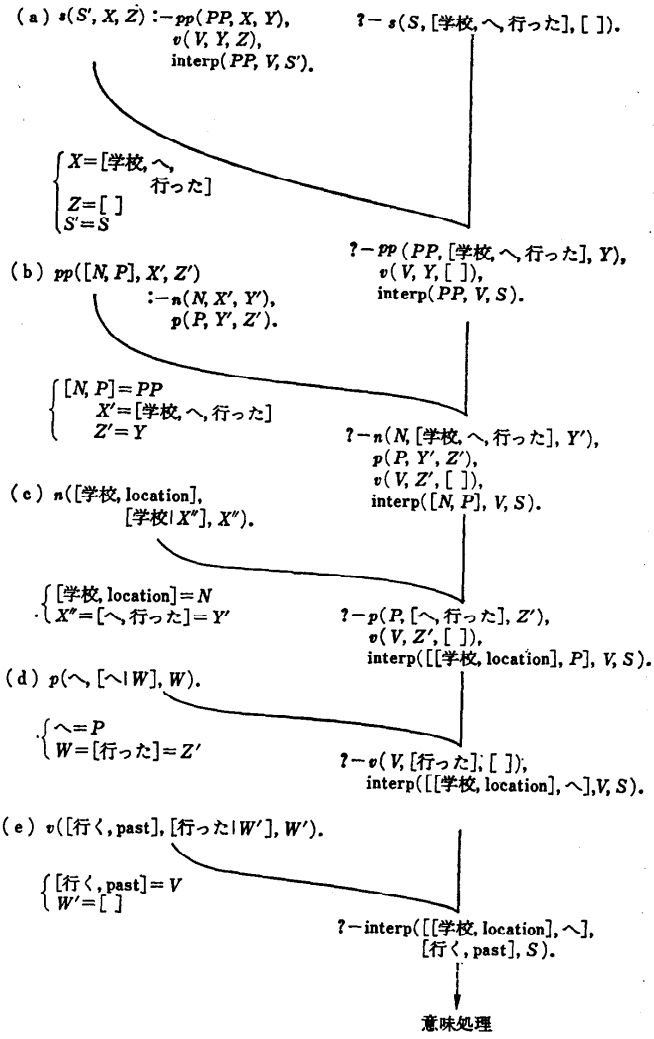




図-1 トップダウンに構文処理が進み意味処理が始まる直前の様子

差分リストの考え方は本特集号「Prolog の言語機能詳説」で説明されている．この差分リストにより，$s$ が過不足なく2つの部分文 $pp$ と $v$ とに分割され，前半の部分文が $pp$ で，後半の部分文が $v$ であることが保証される．｛｝で囲まれた補強項は，処理対象部分文を消費しないので，部分文に対する差分リストが付加されない．

ここでトップダウン法による自然言語処理がどのように進むかみてみよう．処理は次のトップレベルのゴールを実行することからはじまる．

① ?-$s(S, [\text{学校}, \text{へ}, \text{行った}], [\,])$.

これは，[学校，へ，行った] が $s$(文) であるとい

うゴールを達成せよということを意味している．このゴール実行の様子を図-1 に示す．図-1 は(a)のボディにある述語 $v$ まで実行が進んだ様子を示す．この図から，トップダウンに構文処理が進むとともに，辞書から意味処理に必要な情報が持ち込まれて，述語 interp の各引数に束縛される様子が理解できるだろう．これは Prolog のユニフィケーション機能により素直に実現される．

述語 interp として，連用修飾句の意味 ($PP$) と動詞の意味 ($V$) とを用いて意味処理を行うプログラムを用意し，結果を第3引数 $S$ にかえすものとすれば，トップレベルのゴールの実行により，変数 $S$ に意味処理結果を得ることができる．ここで，意味処理で良く使われる格フレームの考え方を利用したければ，(e)の辞書項目記述をそれに合せて変えれば良い．なお述語 interp の実行の直前までで構文処理が成功裏に終了したことになる．

上の例では文法が簡単すぎて，構文処理が決定的に行われる．もう少し文法の規模が大きくなると，文法に含まれる曖昧さのために非決定的な処理が増える．非決定的処理は Prolog の実行機能に委ねられるが，それにより意味処理結果が不具合であれば，バックトラックさせて別の構文処理を行うことができる．{ }で囲まれた部分により，構文処理と意味処理とが素直に融合できるのである．これは，Winograd 等が主張した望ましい自然言語処理の方式であるとともに，人間の自然言語処理の方式に近いと考えられる．

これまでの説明では，{ }で囲まれた部分に意味処理を行うプログラムが仮定されていた．{ }で囲まれた部分は，ボディの任意の場所に挿入できるだけでなく，どのような処理プログラムを書くかはユーザにまかされている．そこで，この部分を利用して，たとえば主語が三人称単数の場合に，一般動詞現在形に語尾変化（s が付く）があるという呼応関係が成立しているかどうかを調べることができる．そして調べた結果を構文処理過程に反映させることができるが，これは文法規則を精密化する道を与える．

この種の文法の精密化の例として，日本語で「きのう」などという時をあらわす名詞は，助詞をともなわなくても連用修飾句 ($pp$; postpositional phrase) になりうることを記述してみる．これは(B)を次の(B)′ に変えて，「きのう」に対する辞書項目記述を(F)とすれば良い．

(B)′ $pp([N,P])\text{--}\rangle n(N),$
$(p(P);\ \{\mathrm{test}(\mathrm{time},N)\}).$

(F) $n([\text{きのう}, \text{time}, \text{past}])\text{--}\rangle[\text{きのう}].$

述語 test を次のように定義しておく．

test $(A,[H|T])$:—member $(A,T)$.
member $(H,[H|T])$:—!.
member $(H,[\_|T])$:—member $(H,T)$.

さらに，もし単文中に「きのう」があらわれると，動詞も過去形(past)でなければならないという呼応関係を記述したければ，(A) を次の (A)′ に変えればよい．

(A)′ $s(S)\text{--}\rangle pp(PP),\ v(V),$
$\{\mathrm{tconcord}(PP,V),$
$\mathrm{interp}(PP,V,S)\}.$

「きのう行った」という文が入力されれば，図-1 から分かるように，$PP$ と $V$ にはそれぞれ，[[きのう, time, past], $P$] と [行く, past] が束縛されるから，述語 tconcord は，変数 $PP$ 中に動詞の時制を強制する past があるかどうか調べ，もしそれがあれば，変数 $V$ 中にもそれがあることを調べるプログラム tconcord を書けばよい（読者自ら試みられたい）．

構文木 (parsing tree) を得たいことがあるかもしれない．この時には，(A)から(E)の各述語に一つ引数を増やして，次のようにすれば良い．

$s([s,A,B],S)\text{--}\rangle pp(A,PP),\ v(B,V),$
$\{\mathrm{interp}(PP,V,S)\}.$
$pp([pp,A,B],[N,P])\text{--}\rangle n(A,N),\ p(B,P).$
$n([n, \text{学校}],[\text{学校}, \text{location}])\text{--}\rangle[\text{学校}].$
.....

以上のようにして次のゴール：

?-$s(Tree,S,[\text{学校}, \text{へ}, \text{行った}],[\ ])$.

を実行すると，このゴール実行後の変数 $Tree$ は

$[s,[pp,[n, \text{学校}],$
$[p, \text{へ}]],$
$[v, \text{行った}]]$

となる．読者は容易にそれを確かめることができるだろう．

本節の Pereira & Warren の方法をまとめてみよう．

(i) DCG 形式の文法 ((A)から(E)) を，それとほぼ相似で1対1に対応する Prolog プログラム ((a)から(e)) に変換する．

(ii) トップレベルのゴールとして $s(S,$〈入力文のリスト〉, [ ]) を実行すると，変換後の Prolog プログラムが動作し，トップダウン＆縦型探索法による処理が進む（これは Prolog プログラムの実行戦略であ

左再帰規則：

np --->np, relative_clause.

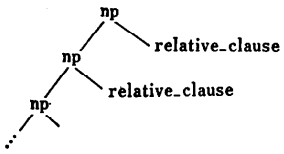



図-2　左再帰規則と無限ループ

る)．したがって構文処理用の特別なプログラム(パーザ)を作る必要がない．

(ⅲ)　(A)から(E)の DCG のボディ (-->の右側) の任意の場所に{ }で囲んだ補強項を書くことができる．補強項として，文法的呼応関係を調べるなど，より精密な文法記述を行ったり，意味処理，文脈処理を行うプログラムを記述することができる．補強項の実行結果は，構文処理の実行制御に反映させうる．これは，意味処理，文脈処理と構文処理を極めて素直に融合させる道を与える．

本節で述べた Pereira & Warren の開発した自然言語処理の手法は，以上の優れた特徴を有しているが，トップダウン& 縦型探索法による実行戦略を採用しているため，一つの欠点がある．それは左再帰規則を適用すると無限ループに陥るという欠点である (図-2 参照)．ただ一つの解を得たければ，文法規則を並べかえ，適用順序を工夫してこの欠点を避けることが可能かもしれない．しかし，一般に自然言語処理では，構文処理結果が多数得られることは避けられない．そのうちどれが良いかを決めるのは，意味処理や文脈処理の役割であり，文法規則の並べかえでは解決できない．

左再帰規則を ε (空ストリング) を含む3つの規則に変換し，トップダウン法における左再帰規則の問題を解消しようとする考え方もある．しかし，これは不必要な非終端記号の導入を許し，必ずしも自然な解決法とはいえない．

次節に述べるボトムアップ法では，こうした問題を避けることができる．

### 2.3　ボトムアップ法[3)]

以下で述べる方法は，電子技術総合研究所で，筆者等により開発された．これは，DCG 形式の文法を変換して，これと一対一対応の Prolog プログラムを作り出すがこれを実行するとボトムアップ&縦型探索法による自然言語処理を行うものである．この変換を行うプログラムは BUP トランスレータとよばれ，Prolog で書かれている[5)]．BUP トランスレータの使用者であっても，文法規則の記述は DCG 形式で行うから，自然言語処理を行う戦略がボトムアップ& 縦型探索法であることを除き，2.2 の後半で説明した優れた特長は，ほとんどそのまま引き継ぐことができる．そのうえ，ボトムアップ法を採用したことによる利点，すなわち左再帰規則を問題なく扱えるようになったことと，

(ⅰ)　トップダウン的な予測情報を用いて，無駄な計算を排除する機能 (link 節)，

(ⅱ)　バックトラックにともなう再計算を排除する機能，

とを持ち，高速な自然言語処理が可能になる．

2.2 の方法は，処理対象文の長さ ($n$) に対して指数オーダの計算時間を要すので効率的ではない．これに対して上記(ⅱ)の機能を組み込んだ本節の方法は，$O(n^3)$ の計算時間に近いと推測されている．実際実験によれば，(ⅱ)の機能を組み込むと，自然言語処理 (特に構文処理) のスピードが一桁近く上がることが報告されている．

以下ではその基本的な考え方を説明する．BUP トランスレータは，DCG 形式で書かれた(A)から(E)の文法規則を，(a 1)から(e 1)の Prolog プログラムへ自動的に変換する．

```
(a 1)  pp(G, PP, A, X, Z):-link(s, G),
                           goal(v, V, X, Y),
                           interp(PP, V, S),
                           s(G, S, A, Y, Z).
(b 1)  n(G, N, A, X, Z):-link(pp, G),
                         goal(p, P, X, Y),
                         pp(G, [N, P], A, Y, Z).
(c 1)  dict(n, [学校, location], [学校|X], X).
(d 1)  dict(p, へ, [へ|X], X).
(e 1)  dict(v, [行く, past], [行った|X], X).
```

BUP トランスレータは，さらに (f 1)から(i 1)の link 節と，(j 1) から (n 1) までの停止節を出力する．

```
(f 1) link(pp, s).      (g 1) link(n, s).
(h 1) link(n, pp).      (i 1) link(X, X).
(j 1) s(s, S, S, X, X). (k 1) pp(pp, PP, PP, X, X).
(l 1) n(n, N, N, X, X). (m1) p(p, P, P, X, X).
(n 1) v(v, V, V, X, X).
```

link 節は，$\alpha(\cdots)$ --> $\beta(\cdots)$, …. なる DCG 節があれば，link($\beta$, $\alpha$) が成立し，link は反射的 ((i 1)) かつ推移的関係にあるから，DCG 節の集合が与えられれば link 節の集合は簡単に計算できる．後述するように述語 link($\beta$, $\alpha$) は，$\beta$ を根とする部分構文木が生

長して，将来$\alpha$を根とする部分構文木ができる可能性（$\beta$から$\alpha$への到達可能性）があることを示している．今$\alpha$をトップダウン的な予測だとすれば，ボトムから生長してくる部分構文木のうち，$\alpha$に到達しえない無効なものを link 節を用いて事前に排除できることになる．

以上の他に，goal 節 (o 1) が組み込まれているものとする．

(o 1)　goal (*G*, *A*, *X*, *Z*):—dict (*C*, *B*, *X*, *Y*),
link (*C*, *G*),
*Pr*=..[*C*, *G*, *B*, *A*, *Y*, *Z*],
call (*Pr*).

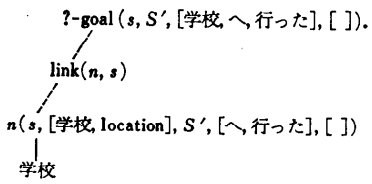
自然言語処理は，以下のトップレベルのゴールを実行することからはじまる．

①　?-goal (*s*, *S'*, [学校, へ, 行った], [ ]).

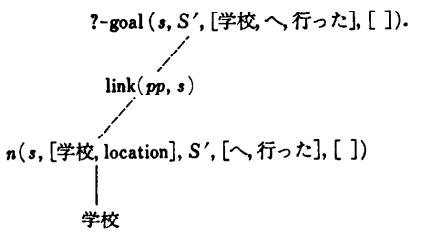
①により (o 1) が選択され，新しいゴール②が作り出される．

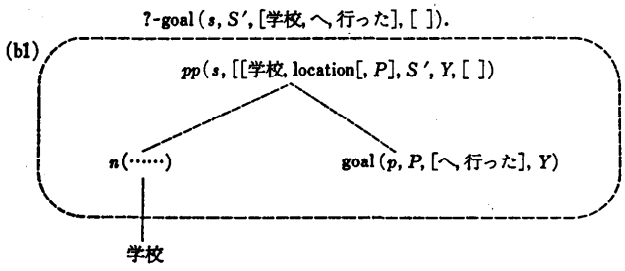
②　?-dict (*C*, *B*, [学校, へ, 行った], *Y*), link (*C*, *s*),
*Pr*=..[*C*, *s*, *B*, *S'*, *Y*, [ ]], call (*Pr*).

②の最初の述語 dict により「学校」に対する辞書 (c 1) が選ばれる．その結果，*C*=*n*, *B*=[学校, location], *Y*=[へ, 行った] となり，*Pr* に渡される．その前に link (*n*, *s*) を実行するが，それは (g 1) により成功する．



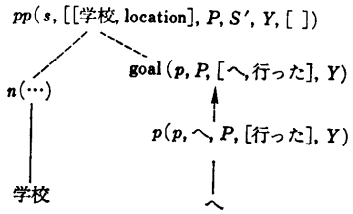

②の最後の call 文では，*Pr* すなわち述語 *n*(*s*, [学校, location], *S'*, [へ, 行った], [ ]) が実行される．これは (b 1) のヘッドとユニファイする．(b 1) のボディの最初の述語 link (*pp*, *s*) は，(f 1) により成功するから (b 1) の選択は有効であったことが分かる．



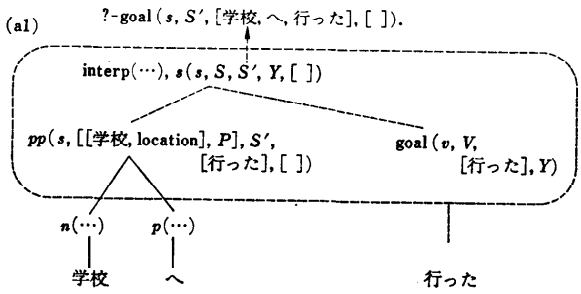

そこで，(b 1) のボディの link 節以降の述語を実行する．その様子は上図のようにあらわすことができる．

ここで，(b 1) のボディのなかでは新しい goal (*p*, *P*, [へ, 行った], *Y*) が実行される．先ほどと同様に (o 1) により辞書びきが行われ，最終的に *Pr*=..[*p*, *p*, へ, [行った], *Y*]，すなわち，*p* (*p*, へ, *P*, [行った], *Y*) が call される．




この述語 *p* は，停止条件節 (m 1) により *P*=へ, *Y*=[行った] として成功するので (o 1) の goal 節の実行は成功したことになる．したがって (b 1) のボディの最後の述語 *pp* を実行する．容易に分かるように，これは (a 1) のヘッドとユニファイする．(a 1) のボディの link (*s*, *s*) は (i 1) により成功する．ここまでの様子を以下に図示する．

goal (*v*, *V*, [行った], *Y*) がどのように実行されるかは，これまでの説明から読者が追跡できるのではないかと思う．最終的に，(a 1) のボディの述語 interp



を実行し，変数 $S$ に意味処理結果を返し $s(s, S, S', Y, [\,])$ を実行するが，これは停止節 (j1) により，$S'=S$ として，意味処理結果が $S'$ に返され，トップレベルのゴール goal($s$, $S'$, [学校, へ, 行った], [ ]) の実行が終了する．

以上で，(a1) から (e1) の Prolog プログラム（これは(A)から(E)の DCG 節に一対一に対応する）が，goal 節 (o1) とともに link 節 ((f1)から(i1)) と停止節 ((j1)から(n1)) を用いて，ボトムアップ & 縦型探索法に基づく自然言語処理が行われることを説明した．そして link 節の利用により，これが単純なボトムアップ法ではなく，トップダウン的な予測をも利用した方法になっていることに注意したい．実際，文法の規模が大きくなるにつれて，処理速度に対する link 節の効用は非常に大きなものになる．これが本節の冒頭の(i)に述べたことである．

われわれは goal 節 (o1) にわずかな修正を施すことで，より一層の高速化をはかることができる．これを (o2), (o3) に示す．

```
(o2)  goal(G, A, X, Y):-
          wf_goal(G, _, X, _), !,
          wf_goal(G, A, X, Y).
(o3)  goal(G, A, X, Z):-
          dict(C, B, X, Y),
          link(C, G),
          Pr=..[C, G, B, AA, Y, Z],
          call(Pr),
          assertz(wf_goal(G, AA, X, Z)),
          A=AA.
```

goal に関して一度成功した計算は wf_goal として assertz しておく．これは (o3) で行う．assertz した結果があれば，これは (o2) により再利用する．この時 (o2) のボディ中のカット記号 ! により，(o3)の実行が阻止されるので，同じ計算を繰り返さないことが保証される．

同様な考え方を失敗結果に適用することも可能である．goal に関して一度失敗した計算を記憶しておき，同じ失敗を二度と繰り返さないようにすることができる．辞書が大量になり，二次記憶に格納せざるを得なくなった時に，一度引いた辞書項目を高速記憶に記憶しておき，以後それを再利用することも同じ技術で可能になる．これらは，goal 節の定義をわずかに修正，拡張することで可能となる．以上が，**2.3** の冒頭(ii)で述べたことである．詳細は文献 3) を参照されたい．

2.3の BUP トランスレータは，$\alpha$-->[ ] ($\varepsilon$ 規則) を Prolog プログラムに変換できない（読者は，その理由を考えてみよ）．新世代コンピュータ技術開発機構 (ICOT) では，$\varepsilon$ レデューサを開発し，この問題を解決している．

2.3 で述べた筆者等の方法をまとめてみよう．

(i) DCG 形式の文法 ((A)から(E)) を，それとほぼ相似で 1 対 1 に対応する Prolog プログラム (a1)から(e1)に変換する．

(ii) トップレベルのゴールとして goal($s$, $S'$, 〈入力文のリスト〉, [ ]) を実行すると，変換後の Prolog プログラムが動作し，link 節によるトップダウンによる予測制御を利用したボトムアップ & 縦型探索法による自然言語処理が進む．組み込まれている goal 節のわずかな拡張により，バックトラックによる再計算を避けた効率の良い自然言語処理が可能になる．

(iii) 2.2 の(iii)と同じ．

2.2 に述べた方法と比べて，2.3 はやや動作の追跡に困難さを感じた読者がいたかもしれない．ここで注意しておきたいことは，2.3 で述べた方法を利用して自然言語処理を行う場合でも，2.2 と同様に，2.1 で述べた DCG の形式を用いるので，DCG の形式さえ分かっていれば良いということである．

2.2 と 2.3 の方法の概略を**図-3** に示す．

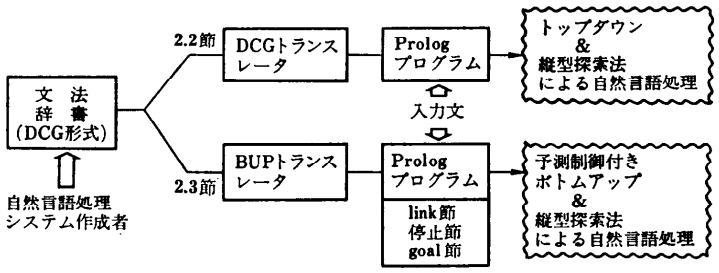



**図-3** Prolog による自然言語処理の方法

## 3. XG (Extraposition Grammar)

英語などの疑問文や関係代名詞文には左外置 (left extraposition) とよばれる変形がある．この左外置は，痕跡 $t$ (trace) を用いて次のように表現される．

The man that$_i$ [$s$ John met $t_i$] is a grammarian.

痕跡は文の表層には現れないから，それを文法を使って同定する必要がある．エジンバラ大学の Pereira は，左外置を扱うために，DCG に拡張を施した XG (Extraposition Grammar) を開発した[4]．XG による単純な英文法記述を以下に示す．

(AA) $s$--〉np, vp.
(BB) np--〉det, $n$, relative.
(CC) np--〉proper_noun.
(DD) np--〉trace.
(EE) vp--〉$v$, np.
(FF) vp--〉$v$.
(GG) relative--〉 (rel_marker, $s$; [ ]).
(HH) rel_marker…trace--〉rel_pronoun.
(II) rel_pronoun--〉[that].
.....

(DD) と (HH) が XG 記述の代表例である．関係節を処理するために，relative (GG) を実行すると，そのボディの rel_marker により，(HH) を実行することになる．(HH) では trace という記号を $x$-リスト (後述) にプッシュする．処理は文の右側に進み，rel_pronoun の that を受理して，rel_marker の実行が成功裏に終了する．そこで (GG) のボディの rel_marker の次の $s$ の処理に進む．

解析対象文が先の The man that John met is a grammarian であるとすれば，この時の $s$ は John met…を処理することになる．(AA) を用いると，np として John (proper_noun) が受理され，vp の処理に進む．vp を処理する規則 (EE) を適用すれば，v は met を受理し次の np の処理に移る．この時文法規則 (BB) もしくは (CC) の対象とする部分文は is a grammarian である．ところが先頭が is で np ではないので，(BB) と (CC) の適用は失敗，バックトラックにより，最後の (DD) が選択され，(DD) のボディの述語 trace が実行される．trace (後述する (jj)) は XG の組込み述語により，先に $x$_list にプッシュした trace をポップし，左外置された np の痕跡を発見する．

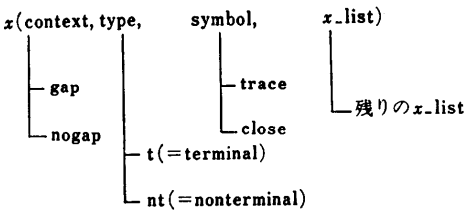



**図-4** $x$_list の構造
(記号 nogap, t の使用法は文献 4) 参照)

以上が XG の動作の概略である．ここで「…」を用いた XG の記法 (HH) によると，rel_marker と trace とで囲まれた John met と，その外置の要素とを結び付けることが禁止される．それにより，Ross の提案した Complex_NP 制約を素直に実現できる[4]．そのためには，(GG) を次の (GG 1), (GG 2) にすればよい．

(GG 1) relative--〉open, rel_marker, $s$, close.
(GG 2) open…close--〉[ ].

XG がどのような機構により動作するかを説明しよう．たとえば XG で書かれた文法規則 (AA) は，(aa) に変換される．

(aa) $s(X, Z, XL0, XL)$:—np$(X, Y, XL0, XL1)$, vp$(Y, Z, XL1, XL)$.

ここで，第1，第2引数の対は，2.2 の (a) と同様，差分リストによる部分文をあらわす．第3，第4引数の対は，差分リストによる $x$_list (外置リスト) の位置をあらわす．$x$_list は 図-4 の構造をしている．すでに述べたように，$x$_list は「…」を含む文法規則と，組込み述語 trace で操作される．

(HH) は (hh) に変換される．

(hh) rel_marker$(X, Y, XL0, x$ (gap, nt, trace, $XL))$
:—rel_pronoun$(X, Y, XL0, XL)$.

(hh) の rel_marker の第4引数では，rel_pronoun の $x$_list $XL$ に，trace 情報をプッシュしていることに注意．

組込み述語 trace の定義は次のようである．

(jj) trace$(X, X, XL0, XL)$:—
virtual (trace, $XL0, XL$).

(kk) virtual$(NT, x(C,$ nt, $NT, XL), XL)$.

(jj) と (kk) は，$x$_list のなかに trace がプッシュされ，それが先頭にあればそれをポップする操作を行うものである．

(GG 2) は (gg 2) と (gg 3) に変換される．

(gg 2) open$(X, X, XL, x$ (gap, nt, close, $XL))$.

(gg 3) close(*X*, *X*, *XL*0, *XL*):—
virtual(close, *XL*0, *XL*).

(gg 2)により，*XL* 中の trace は (close がプッシュされるため) 外から見えなくなる．これにより Complex_NP 制約が実現されるが，詳細は文献4)を参照されたい．

これまで述べた方法は，2.2 のトップダウン法の利点を生かした左外置処理技術であった．XG の考え方を2.3で述べたボトムアップ法で実現することはできないだろうか．ただちに気付くことは，2.3 の方法では，ε 規則の (GG 2) が扱えないことである．またボトムアップであるため，常に入力文に対して辞書びきを行うため，どの時期に trace の存在を予測したら良いかが問題になる．本節で述べた方法はトップダウン法を採用しているため，トップダウン的に np を予測し，それがなければ痕跡の存在を仮定し (dd) を実行した．このようなことは純粋なボトムアップ法ではできない．しかし，2.3 では link 節により，トップダウン的な予測を使いながら，ボトムアップ法による自然言語処理が進むことを述べた．前者の問題に対しても，DCG を拡張し，BUP_XG とよばれる記法を導入して，XG よりも素直な文法記述により 2.3 の方法に，XG の考え方を組み込むことが可能であることを東京工業大学の今野等が示した[6]．それによれば XG でtrace 等のプッシュを行うために導入した不自然な非終端記号 (rel_marker) を用いる必要がないだけでなく goal 節のわずかな修正と拡張により，左外置が 2.3 の方法で簡単に扱える．詳細は文献6) を参照されたい．

## 4. おわりに

Prolog と自然言語処理とは極めて整合性がよいことをさまざまな角度から説明した．そしてそこから新しい自然言語処理技術が生れていることを事例に従って説明した．紙面の都合で触れられなかったが，形態素処理も Prolog で素直に記述できる[7]．Prolog に文字列処理パッケージを付加して，文献7) の方法を更に簡潔に記述し，またそれを高速に実行できることが報告されている[8]．2章と3章で述べたことを総合すると，文字列処理機能と，大量の辞書項目が即座に検索できる機能を持った Prolog マシンが実現すれば，それが自然言語処理マシンであるといってよいだろう．第五世代コンピュータの出現が待たれるゆえんである．最後に，2.2 で述べた方法は，かつて自然言語処理で良く使われた ATNG[9] と等価な能力を持つこと[2]，3章で述べた左外置の扱いは，ATNG での hold, unhold の考え方の拡張になっていること，また 2.3 の方法は，電総研で田中等が開発した拡張 LINGOL[10] の考え方を受け継ぎ，電総研の元吉が発想したこと[11]を指摘して結びとする．

## 参考文献


1) Colmerauer, A.: Metamorphosis Grammar, in Bolc (ed.): Natural Language Communication with Computers, pp. 133-190, Springer-Verlag (1978).
2) Pereira, F. and Warren, D.: Definite Clause Grammar for Language Analysis, Artif. Intell., Vol. 13, pp. 231-278 (1980).
3) Matsumoto, Y. et al.: BUP—A Bottom-up Parser Embedded in Prolog, New Generation Computing, Vol. 1, No. 2, pp. 145-158 (1983).
4) Pereira, F.: Extraposition Grammar, Am. J. Comput. Linguist., Vol. 7, No. 4, pp. 243-256 (1981).
5) 松本，清野，田中：BUP トランスレータ，電子技術総合研究所彙報，Vol. 47, No. 8, pp. 679-697 (1983).
6) 今野，田中，高倉：ボトムアップ構文解析システム BUP での左外置変形の処理，情報処理学会自然言語処理研究会資料，44-1 (1984).
7) 安川，田中：Prolog による形態素処理と熟語処理，情報処理学会自然言語処理研究会資料，32-4 (1982).
8) 梅村他：文字列処理機能を導入した Prolog: Shape Up について，Proc. of Logic Programming Conf. '83 (1983).
9) Woods, W. A.: Experimental Parsing System for Transition Network Grammar, in Rustin (ed.): Natural Language Processing, Algorithmic Press (1971).
10) 田中，佐藤，元吉：自然言語処理のためのプログラミング・システム—拡張 LINGOL について，電子通信学会論文誌，Vol. J 60-D, No. 12, pp. 1061-1068 (1977).
11) 元吉文男：LINGOL コンパイラ，情報処理学会記号処理研究会資料，21-3 (1982).


(昭和59年9月18日受付)