解　説

ロジック・プログラミングと計算言語学†

田中穂積†† 新田義彦†††

1. はじめに

計算言語学には次の二つの異なる研究があると思う.

(1) 計算機を利用した言語学の研究.

これはさらに二つに分けられる.

(1.1) 計算機によって大量の言語データを処理し言語の統計的な性質を明らかにする.

(1.2) 計算機上に言語理論のモデルを作り動作させ, その結果に基づき言語理論の精密化を図る.

(2) 自然言語処理技術に関する研究, またはその応用システムに関する研究. これは計算機のもつ機能の高度化に関係している.

ロジック・プログラミングは(1.2)と(2)に対する計算言語学の研究に有力な研究手段を提供する. 言語学ではさまざまな規則を用いて文の言語学的な構造を明らかにしたり, 得られた構造を別の構造に変換することがよく行われる. 計算機で同様な処理を行う技術が自然言語処理技術であるが, ロジック・プログラミングは自然言語処理に都合の良い枠組みをもっている[3),4),8),10),12)]. これは, 計算言語学の(2)の研究に直接関係するとともに, (1.2)についても言語理論のモデルを計算機上に作り動作させる作業をきわめて容易にする. 言語理論についていえば, ロジック・プログラミング言語で重要な役割を果たすユニフィケーションをベースにした文法理論が最近新しい理論として注目されている[14)].

2章では, ロジック・プログラミング言語の典型である Prolog の基本計算機構を簡単に復習しておく. 3章と4章では, Prolog の基本計算機構をそのまま利用して, 構文解析, 意味解析などの自然言語処理を行うことが可能なことを実例により示す.

ここで次のことを注意しておきたい. 2章では Prolog の基本計算機構をやや形式的に説明するが, それは Prolog を用いて自然言語処理を行う場合に, 2章で述べたことの理解が前提になる, ということではない. 2章で述べたことを理解すれば, なぜ Prolog と自然言語処理との整合性が良いかの理論的背景が比較的はっきり理解できるという程度の意味でしかない.

2. 逐次計算機上での Prolog の基本計算機構

以下の説明では, 最も標準的な DEC-10 Prolog の記法を用いることにする[2)]. Prolog プログラム節 (以下略してプログラム節とよぶ) の形式には以下の(1)と(2)がある.

(1) 含意節; $A :- B_1, \cdots, B_n$.

ここで $A, B_1, \cdots, B_n$ はそれぞれ一つの述語を表す. A はヘッド, $B_1, \cdots, B_n$ はボディと呼ばれる. いずれも否定記号をもたない述語である. ボディに現れる記号「, 」は, 連言記号を表す.

(2) 単位節; A.

これは含意節のボディを空にしたものに対応している.

プログラム節の有限集合を Prolog プログラムと呼ぶ. Prolog プログラムの実行 (Prolog プログラムを用いた計算) の意味を説明するために, さらに次の二つの節を定義しておく.

(3) ゴール節; ?- $B_1, \cdots, B_n$.

これは含意節のヘッドを空にしたものに対応する. ボディに現れる各述語 B_i をサブゴールと呼ぶ.

(4) 空節; ?-.

これは含意節のヘッドとボディを空にしたものに対応する.

逐次計算機上の Prolog プログラムの実行は, ゴール節のボディの述語 (サブゴール) を左から右に順に実行していくことから始まる.

† Logic Programming and Computational Linguistics by Hozumi TANAKA (Tokyo Institute of Technology) and Yoshihiko NITTA (Advance Research Laboratory, Hitachi, Ltd.).

†† 東京工業大学工学部
††† (株)日立製作所基礎研究所

(a) ゴール節 G_i が

G_i : ?- $A_1, \cdots, A_k$.

であり，プログラム節 C_i が

C_i : A :- $B_1, \cdots, B_q$.

であるとする．このとき G_i のボディの最左端にある A_1 と A のおのおのの引数に含まれる変数に“適当な代入 θ_{i+1}”を施し，両者を同一化 ($A_1\theta_{i+1}=A\theta_{i+1}$) することができれば，$C_i$ の選択により“A_1 は実行された”といい，新しいゴール節 G_{i+1} を得る．

G_{i+1} : ?- $(B_1, \cdots, B_q, A_2, \cdots, A_k)\theta_{i+1}$.

A_1 に対して同一化により C_i が選択できなければ A_1 の実行は失敗 (failure) し，G_i が失敗したとして後述する後戻りを行う ((c)参照)．

ここで，G_i で「A_1 が実行された」ということは，「A_1 の実行が G_{i+1} での $(B_1, \cdots, B_q)\theta_{i+1}$ の実行に置き換えられた」という意味であり，後述する「A_1 の実行が成功した」ということを必ずしも意味しないことに注意．また「両者を同一化するための適当な代入」とは「最も一般的なユニファイア (mgu ; most general unifier) による代入」のことである．mgu の正確な定義は文献 7) を参照のこと．A_1 と A とが mgu により同一化可能なら，両者はユニファイ可能であるという．同一化操作のことをユニフィケーションという．

(b) G が与えられ，この G に対して(a)を繰り返すと，ゴール節の系列 $G, G_1, \cdots, G_n$ と，代入の系列 $\theta_1, \cdots, \theta_n$ とを得る．もし G_n が空節になれば，G の実行 (Prolog プログラムの実行) は成功 (success) して終了する．このとき G に含まれる各変数に対して代入の合成 $\theta_1 \cdots \theta_n$ を施すと，答えを得ることができる．

上の説明ではさほど明確ではなかったと思われるが，G_{i+1} の形から，$(B_1, \cdots, B_q)\theta_{i+1}$ の実行がすべて成功してから A_2 が実行されることがわかる．A_1 は，$(B_1, \cdots, B_q)\theta_{i+1}$ の実行がすべて成功したときに成功し，次の A_2 の実行に移る．以下同様に A_1 から A_k の実行がすべて成功すれば，G_i は成功したことになる．このとき G_i から始まるゴール節の系列の最後の G_n は空節になっているということを(b)は述べている．

いうまでもなく，C_i としてユニット節が選択されると，G_{i+1} のボディのサブゴール数は G_i より1減る．このようにして G が最終的に，空節 G_n に到達可能な場合のあることがわかる．

(c) なんらかの理由により，G_{i+1} のボディの先頭にある $B_1\theta_{i+1}$ の実行に失敗すると，ゴール G_{i+1} の実行は失敗したとして，G_i に戻り A_1 を再実行する．このことを後戻りという．A_1 の再実行とは，A_1 とユニファイ可能なヘッドをもつ別のプログラム節 C_i' を選択し，(a)と同様な操作を繰り返すことをいう．後戻りは，A_1 とユニファイ可能なヘッドをもつプログラム節がなくなるまで続くが，なくなると A_1 の実行は失敗し，後戻りはさらに G_i から G_{i-1} に伝播していく．

したがって A_1 の実行に一度成功したとしても，その後の A_2 などの実行結果により後戻りが発生し，A^1 が再実行され，最終的に再実行に失敗することもある．

以上，逐次計算機上での Prolog の基本計算機構をまとめると以下のようになる．

(P1) プログラム節の選択 (呼び出し) は，プログラム節のヘッドとのユニフィケーションにより行われる．

(P2) 後戻りが発生すると，プログラム節の再選択が自動的に行われる．

逐次計算機上の Prolog プログラムの実行であるからさらに，

(P3) プログラム節の選択は，プログラム作成者が記述した順に一つずつ行う．

(P4) ゴール節のボディにあるサブゴールは，左から右に一つずつ実行する．

3. ロジック・プログラミングと構文解析

構文解析は，文法を用いて文の構文構造を認識するものである．説明を簡単にするために，最もよく使われる文脈自由文法規則を用いた構文解析を説明し，Prolog との関連を説明する．

3.1 構文解析の方法

文脈自由文法規則は，次の形をしている．

(A) $x \longrightarrow y\ z$

構文解析の立場からすれば，この文法規則には次の2とおりの読み方がある．矢印 (⟶) の向きに沿う読み(1)とその逆向きの読み(2)である．

(1) 解析すべき文の一部が非終端記号 x にまとめられるためには，文の一部が，非終端記号 y にまとめられる前半部分と，非終端記号 z にまとめられる後半部分とに (過不足なく) 分割されるはずである．

(2) 解析すべき文の一部の前半部分が非終端記号 y に，後半部分が非終端記号 z にまとめられることがわかると，前半と後半を合わせた部分を非終端記号 x

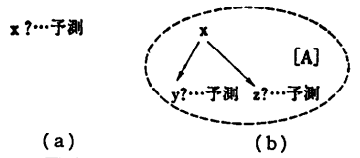


図-1 トップダウン構文解析

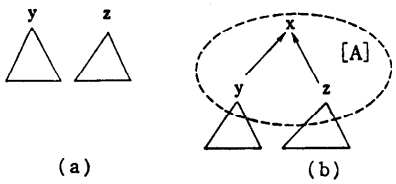


図-2 ボトムアップ構文解析

にまとめることができる.

(1) の読みはトップダウンに, (2) の読みはボトムアップに構文解析する場合に使われる. これを図-1, 図-2 に示す.

3.2 トップダウンの構文解析

ここで前節(1)の読みは, さらに次のように解釈される. 解析すべき文の一部を x にまとめようとしてゴール (予測) x を立てると, 文法規則のなかで, 矢印の左に x をもつ (A) を選択して, サブゴール (予測) y と z を立てて, さらにそれらが成り立つかどうかを調べる(図-1).

この説明は, 前章の (a) で述べた Prolog プログラムの実行に関する説明と全く同じであることに注意したい. 各文法規則に対して, 文法規則の左辺をヘッドに, そして右辺をボディにもつ Prolog プログラム節を対応させることができる. トップダウンによる構文解析は, ゴール x を実行することから始まる. すなわちゴール x の実行により, x とユニファイ可能なヘッドをもつ (文法規則 (A) に対応する) Prolog プログラム節を選択し, 新しいサブゴール y と z を作る. 以下同様にしてサブゴール y と z の実行を続ければよい.

以上のような考え方をすると, トップダウンに構文解析するためのプログラム (パーザ) は, 前章の (P1), (P2), (P3) で代用可能なことがわかる. これは, Colmeraure が Metamorphosis Grammar で初めて具体的に示した[3]. その後 Pereira らは Metamorphosis Grammar を洗練した DCG (Definite Clause Grammar) と呼ばれる文法記述形式を開発し, DCG の形式で書かれた文法規則を Prolog プログラムに変換するシステムを作成した[12]. 以下では, Pereira らの方法を説明する.

DCG による文法記述の例を(a)に, Prolog プログラムへの変換結果を(b)に示す. (c)は構文解析を行うために最初に実行するゴールを示す.

```
(a) 1) s(S)-->np(NP), vp(VP),
            {interp(NP, VP, S)}.
    2) np(NP)-->(pronoun(NP) ; propernoun
            (NP)).
    3) np(NP)-->det(DET), noun(N),
            {interp(DET, N, NP)}.
    4) pronoun(N)-->[you].
    .....
(b) 1) s(S, U, V) :-np(NP, U, W),
            vp(VP, W, V), interp(NP, VP, S).
    2) np(NP, U, V) :-(pronoun(NP, U, W) ;
            propernoun(NP, U, W)), V=W.
    3) np(NP, U, V) :-det(DET, U, W),
            noun(N, W, V), interp(DET, N, NP).
    4) pronoun(N, [you|X], X).
    .....
(c) ?-s(S, [you, run], []).
```

(a)と比べて(b)では引数が二つ増えている. この引数の対には, 解析すべき文の部分が差分リストとして与えられている. たとえば述語 pronoun の第2, 第3引数の [you|X] と X は, リスト [you|X] の後から X を差し引いた残りの [you] が pronoun であることを示している. DCG のボディには, 選言記号「;」を書くことができる. (a)-2) は, np は pronoun か propernoun のいずれかであることが記述されている. DCG では非終端記号は述語名として現れる. したがって, 意味解析を行うための述語をボディに {,} でくくって挿入しておくと (これを補強項とよぶ), 非終端記号名の述語とが全く区別なく実行され, 構文解析と意味解析との融合が自然に行われる. これは望ましい自然言語処理の在り方であるとして, かつて心理学者が注目した方法である[13]. すべての解析結果を得たければ, 解析結果を得てから強制的に後戻りすればよい.

たとえば (a)-1) に現れる補強項 interp(NP, VP, S) は, np と vp の解析結果である NP と VP とを用いて意味解析を行い, その結果を S に返す述語である. このように Prolog プログラム節のボディには, 意味解析を行う部分 (interp(...)) と構文解析を行う部分 (np(...), vp(...)) とが混在し区別なく実行されるので, 構文解析と意味解析とを融合させることができるのである. なお補強項がボディに記述できるの

で，DCG 文法規則により文脈依存の処理をも行うことができる．

ここで(c)のゴールがどのように実行されるかを追跡した結果を以下に示す．ここで(b)-1) などは，文法規則の適用を示す．

```
?-s(S, [you, run], []).
        ↓(b)-1)
?-np(NP, [you, run], W'),
        ↓     vp(VP, W', []), interp(NP, VP, S).
        ↓(b)-2)
?-(pronoun(NP, [you, run], W");
        ↓     propernoun(NP, [you, run], W")),
        ↓     W'=W",
        ↓     vp(VP, W', []), interp(NP, VP, S).
        ↓(b)-4)
?-vp(VP, [run], []), interp(NP, VP, S).
        ↓
    以下省略
```

以上をまとめると次のようになる．文法記述者が DCG とよばれる記法で文法を記述すると，それと1対1に対応する Prolog プログラムに変換することができる．変換後の Prolog プログラムに，解析すべき文を引数に与え，(開始記号 s に対応する) 述語 s を予測として実行すると ((c)参照)，トップダウンに構文解析することができる．

しかしトップダウンの構文解析には一つの問題がある．それは，"np-->np, srel." のように，左再帰規則がある場合に，すべての構文解析結果を得ようとして無限ループに陥ることである．ボトムアップの構文解析ではこの問題を避けることができる．

3.3 ボトムアップの構文解析

前節では，DCG 形式で文法規則を記述しておきさえすれば，Prolog の基本計算機構をそのまま利用してトップダウンの構文解析が可能なことを示した．ボトムアップの構文解析についても，同様な方法が開発されている[8]．この方法は BUP とよばれているが，正確にいうと，3.1(2) の純粋なボトムアップ構文解析法とは異なり，ボトムアップの構文解析をベースにし，しかもトップダウン的な予測をも利用する方法である．したがって，能率良く構文解析を行うことができる．これは次の(3)に示す考え方に基づいている．

(3) いま文の一部がボトムアップ解析により非終端記号 y にまとめられることがわかったとしよう．そうすると，文脈自由文法規則の矢印の直後に非終端記号 y をもつ規則を取り出す．今の場合それが 3.1 の(A)であったとすると，非終端記号 y にまとめられた部分に直接後続している残りの部分が，非終端記号 z にまとめられるかどうかを調べるため，次に予測 z を立てる (これがトップダウン的な予測に対応している)．もし予測 z が成功すると，非終端記号 y と z でまとめられる部分は，一つの非終端記号 x にまとめられることがわかったとして，今度は非終端記号 x に対して(3)を繰り返し，構文解析をボトムアップに進める (図-3)．

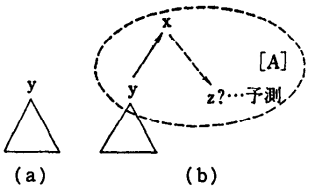


図-3 トップダウン予測付きボトムアップ構文解析

(3)の説明から明らかなように，構文解析を行うための文法規則の選択は，前節(a)の DCG 文法規則の矢印の直後の非終端記号をキーにして行われる．(トップダウンの場合には，矢印の左の非終端記号がキーになることに注意) Matsumoto らの方法[8]のポイントは，Prolog の基本計算機構を利用してこのような文法規則の選択法を実現するために，前節(a)の DCG 文法規則を，矢印の直後の非終端記号 np をヘッドとする Prolog プログラム節に変換することである．

Matsumoto らの方法では，DCG 形式で記述された文法規則を，後述する Prolog プログラムに変換する．変換後の Prolog プログラムに，解析すべき文を与え，開始記号 s を引数にした述語 goal (後述) を実行すると，トップダウン予測付きのボトムアップ構文解析を行うことができる．

たとえば 3.2(a)-1) と 4) の DCG 文法規則は，BUP とよばれるトランスレータにより (d)-1) と 4) の Prolog プログラム節に変換される．

```
(d) 1) np(G, NP, T, U, V) :-link(s, G),
                            goal(vp, VP, U, W),
                            interp(NP, VP, S),
                            s(G, S, T, W, V).
    4) dict(pronoun, [hearer], [you|X], X).
```

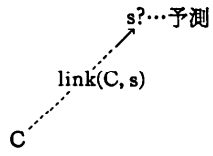
次のゴールを実行すると構文解析が始まる．

```
(e) ?-goal(s, S, [you, run], []).
```

説明を簡単にするために，最も簡略化された述語 goal の定義を与える．

```
(f)  goal(G, A, U, V) :-dict(C, B, U, W),
                        link(C, G),
                        Nt=..[C, G, B, A, W, V],
                        call(Nt).
```

予測Gと解析すべき文Uとが与えられて，(e)の述語 goal が実行されると，(f)のボディでUに対して辞書引きを行い (dict(C, B, U, W))，辞書引き結果から品詞Cを得る．link(C, s) は，得られた品詞Cをもとに構文解析がボトムアップに進んだとき，将来予測sを満たし得るかどうかを調べる述語である．



幸いにして link に関する情報は，DCG 文法規則が与えられればあらかじめ計算できる．

ボトムアップの構文解析が進み，今 np としてまとめうる部分がわかれば，(d)-1) が選択され，そのボディの実行に移る．まず初めに s から，与えられた予測Gに到達可能かどうかが調べられる (link(s, G))．ついで vp を新しい予測とした述語 goal が実行される (goal(vp, …))．もし goal (vp, …) が成功すると次は np と vp とをまとめて s にすることが可能なことがわかるので，s(G, …) を実行する．詳細な動作については文献 15) を参照して欲しい．

文献 8) には，後戻りによる再計算を避ける方法が述べられている．それにより約1けた構文解析の高速化が図れることが報告されている．最近田中らは，link 節をわずかに変えることでさらに 6.5 倍の高速化が図れることを報告している[17]．それによれば，本節で述べたトップダウン予測付きのボトムアップ構文解析の速度は，実用的にも十分な速度に達したと考えられる．

ロジック・プログラミングの枠組みで，ボトムアップ構文解析を並列に行う方法も開発されている[9]．逐次計算機上であっても，コンパイルすると BUP 以上のスピードが得られることが報告されている．並列マシンの研究の進展とともに今後が期待される．

3.4 DCG を拡張した文法記述

DCG の枠組みで，Prolog の基本計算機構をそのまま使った構文解析の方法を説明してきた．その他にも DCG を拡張した文法記述方式により，Prolog 上で関係代名詞などを含む文に見られる左外置変形を扱うことができる．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，文法規則の数を大幅に減らすことが可能となり，文法の見通しが良くなる．この痕跡発見機構を 3.2 の方法に組み込んだものとして Pereira の XG (Extraposition Grammar) が[13]，また 3.3 の方法に組み込んだものとして今野らの XGS (Extraposition Grammar with Slash Category) がある[5]．

4. ロジック・プログラミングと意味解析

4.1 意味解析の基本的な考え方

意味解析の基本となるのは，辞書項目の記述である．辞書項目の記述としてよく用いられるものとしてフレーム表現がある．フレームはスロットの集合からできている．最も典型的な辞書項目記述例を図-4 に示す．

スロットは，スロット名 (subj)，制約条件 (human)，アクション (=agent) の3組からなる．ここで制約条件は，このスロットを満たすことのできるフレーム (これをフィラという) の意味的性質を記述する．たとえばフィラがこのスロットを満たす (埋める) ことができるためには，フィラが文中で，主格 (subj) の位置を占め human でなければならない，ということが記述されている．フィラがこのスロットを満たすとアクション (=agent) を起動し，フィラの深層格を agent であるとする．フィラがこのスロットを満たすことができなければ，つぎのスロット (iobj) を選択し同様なことを行う．フィラの満たしうるスロットが発見できなければ，self スロットを介して上位のフレーム act のもつスロットにアクセスする．

田中らによると[16] DCKR[6] とよばれる形式で辞書項目を記述しておけば，以上に述べた意味解析の手順のほとんどすべてを Prolog の基本計算機構に任せることができる．これについて簡単に説明する．

DCKR では，フレームを構成する各スロットを，

```
[sell, unit,
    [subj, human, =agent],
    [iobj,  human, =goal],
    [obj, thing, =object],
    [self, act]]
```

図-4 辞書項目記述例

```
        :- op(100, yfx, -), op(100 yfx, :).
(a)  sem(sell, subj: F-In-Out) :-
                sem(F, isa: human),
                extractsem((agent: F), In, Out).
(a)  sem(sell, iobj: F-In-Out) :-
                sem(F, isa: human),
                extractsem((goal: F), In, Out).
(c)  sem(sell, obj: F-In-Out) :-
                sem(F, isa: thing),
                extractsem((object: F), In, Out).
(d)  sem(sell, P) :-
                isa(act, P).
(e)  isa(X, P) :- P=isa: Upper; sem(Upper, P).
```

図-5　DCKR による図-4 の辞書項目記述例

sem 述語をヘッドとする一つのホーン節で表現する．ただし，この sem 述語の第1引数にはフレーム名がくるものとする．図-4 の記述を簡略化した DCKR で表現したものを述語 isa の定義とともに図-5 に示す．

図-5 の(a)から(d)が sell に関するフレーム記述である．各ホーン節が図-4 の各スロットに対応している．スロット名は，ホーン節のヘッドの第2引数の先頭に書く．変数Fがフィラである．スロットに対応するホーン節のボディには，意味的制約条件とアクションの対が記述されている．たとえば図-5(a) のボディの

sem(F, isa : human),
extractsem((agent : F), In, Out)

という記述は，フィラFが人間 (human) なら，フィラFの深層格を動作主 (agent) とするアクション extractsem ((agent : F), In, Out) を起動し，深層格構造を抽出するものである．ここで，フィラFが人間であるかどうかを調べる sem (F, isa : human) はフィラFに対する意味的制約条件を表している．

extractsem は，抽出した深層格構造を In に付加した結果を Out に返す．

図-5 の記述から，スロットの選択は

sem(sell, iobj :mary-I-O)

を実行すれば，Prolog の基本計算機構 (2章(P1), (P2), (P4)) により自動的になされることがわかる．(スロットの選択を行うタイミングは，たとえば 3.2 節の DCG 記述中の補強項 {interp(VP, subj : NP, S)} の実行時である)．スロットが選択されると，2章(P3)によりスロットのボディが実行され，意味的制約条件の検査とアクションが実行される．意味的制約条件として，sem 述語の連言と選言とを任意に組み合わせたものが記述できるが，その解釈は Prolog インタプリタが行ってくれる．

もし図-5(a)から(d)の実行に失敗すれば，図-5(e)が選択され，sell の上位の act フレームのスロットが自動的にアクセスされる．これは知識継承が Prolog のユニフィケーション機構により自動的になされる例にもなるが，興味ある読者は文献 6) を参照するとともに，図-5 のプログラムの実行を少し追ってみるとよい．

このように，辞書項目記述形式を DCKR にすれば，意味解析の中核をなすプログラムを Prolog に組み込みの基本計算機構ですべて代用可能なことがわかる．

さらに図-5 の記述はコンパイルすることができるので，高速化を図ることができる．これまでの辞書項目の記述形式では，辞書項目を一つの大きなデータ構造として表現していたため，それをコンパイルできないのとよい対照をなす．

意味解析結果をも DCKR 形式にしておけば，質問文から得られた DCKR 形式の結果をそのまま実行して答えを得ることができる．この場合 Prolog インタプリタがそのまま推論エンジンとして働く．

5. ロジック・プログラミングと自然言語理解システム

これまで述べてきたことから，自然言語理解システムの一般的な構成図として図-6 が考えられる．

図-6 の左側には言語解析を行う部分があり，中心

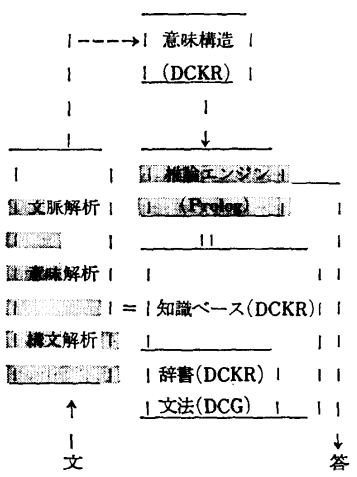


図-6　自然言語理解システム
■■■ は，Prologの基本計算機構で代用可能な部分を示す．

に知識ベースがある．文が入力されると，知識ベースを用いて構文解析，意味解析，文脈解析が行われる．これまでの説明から，ロジック・プログラミングの枠組みを用いれば，図-6 の網目をかけた部分は Prolog の基本計算機構に任せることが可能なことが理解できよう．したがって，Prolog マシン＋alpha が自然言語解析マシンであると結論してもよいように思われる．ここで alpha がなにかと問われれば，それは知識ベースマシンであるということになるだろう．いずれにせよこのような考え方は，我が国の第五世代コンピュータ計画の目指す方向と一致している．

6. おわりに

文脈解析の問題について触れておきたい．文脈を理解するためには，文を解析した結果としての知識の形式をどのようにしたらよいかを深く考えなければならない．文脈理解の過程でさまざまな推論が行われることはよく知られている．したがって文脈解析についても，推論機構の裏付けのある知識を文から抽出しておくことは重要であろう．最近では状況を考慮した意味論が文脈理解の研究に重要であると認識されている[1]．その方向の研究として文献 11) が注目される．

従来自然言語処理は，道具作りに労力を要すことから困難な研究であるとされていた．特に言語学の立場から計算言語学の分野にアプローチする場合に大きな障害になっていたように思う．ロジック・プログラミングの枠組みで計算言語学の研究を行う場合には，そうした障害は減少する．本稿によりロジック・プログラミングの枠組みでの自然言語の研究の意義が理解されれば幸いである．なお本稿で述べた方向の研究の先駆けとして文献 4) がある．現在でもその内容の新鮮さに驚かされる．一読を勧めたい．

謝 辞 筆者の一人にロジック・プログラミングと自然言語処理の関係を示唆してくれたのは，当時の上司であった淵 一博 ICOT 所長である．筆者らのよき指導者であり助言者である同博士に感謝する．

参 考 文 献

1) Barwise, J. and Perry, J.: Situations and Attitudes, The MIT Press (1983).
2) Clocksin, W.F. et al.: Programming in Prolog, Springer-Verlag (1981).
3) Colmeraure, A.: Metamorphosis Grammar, in Bolc (ed): Natural Language Communication with Computers, Spring-Verlag, 133-190 (1978).
4) 淵 一博：述語論理的プログラミング——EPILOG の提案，情報処理学会第 1 回記号処理研究会資料 (1977.7).——情報処理，Vol. 26, No. 11, pp. 1298-1306 (1985) に再録．
5) 今野 聡他：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編，コンピュータソフトウエア，Vol. 3, No. 2, pp. 115-125 (1986).
6) 小山晴生他：Definite Clause Knowledge Representation, Proc. of LPC' 85, ICOT, pp. 95-106 (1985).
7) Lloyd, J.W.: Foundations of Logic Programming, Springer-Verlag (1984).
8) Matsumoto, Y. et al.: BUP—A Bottom-up Parser Embedded in Prolog, New Generation Computing, Vol. 1, No. 2, pp. 145-158 (1983).
9) 松本裕治：並列構文解析，情報処理学会自然言語処理研究会資料 (1986).
10) Mellish, C.S.: Computer Interpretation of Natural Language Descriptions, Ellis Horwood Limited (1985).
11) Mukai, K.: Unification over Complex Indeterminates in Prolog, Proc. of LPC '85, ICOT pp. 271-278 (1985).
12) Pereira, F. et al.: Definite Clause Grammar for Language Analysis—A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, No. 13, pp. 231-278 (1980).
13) Pereira, F.: Extraposition Grammar, American Journal of Computational Linguistics, Vol. 7, No. 4, pp. 243-256 (1981).
14) Shieber, S.M.: An Introduction to Unification-Based Approaches to Grammar, The Center for the Study of Language and Information, Stanford University (1986).
15) 田中穂積他：自然言語処理における Prolog，情報処理 Vol. 25, No. 12, pp. 1396-1403 (1984).
16) 田中穂積他：知識表現形式 DCKR とその応用，日本ソフトウェア科学会編コンピュータソフトウェア，(掲載予定)．
17) 田中穂積他：自然言語処理のためのソフトウエアシステム LangLAB, Proc. of LPC '86, ICOT, pp. 5-12 (1986).
18) Winograd, T.: Understanding Natural Language, Academic Press (1972).

(昭和 61 年 6 月 30 日受付)