論理型言語Prologの構造と応用上の問題点

杉 本 英 二

要　　約

条件と結論を書くだけでプログラムができあがる論理型言語 Prolog は人工知能の応用として種々のエキスパートシステムに利用されている。しかし、 Prolog による処理には、主に２つの問題点が有る。１つには、非決定的なアルゴリズムの実現のために中間結果を記録するメモリの容量の限界から、長い計算を行えない問題点。２つには、データの基本単位をアトムという小さい単位で行い、すべてのアトムを辞書で一意に管理するので、規模が大きいアプリケーションでは管理のための辞書メモリの限界とデータの検索の非行率さの問題点。これらの問題点のため大規模のアプリケーションの開発が不可能となる。しかし、データだけの規模が大きい応用に対しては、データベース管理システムとの間で適切なインターフェイスを付加すれば、第２の問題点は解決出来る。

０。　はじめに

論理型言語は第５世代コンピュータ開発機構の活発な活動によって着実に応用分野を広げている。特に Prolog はその処理系の普及にともなって多くの応用プログラムが作られるようになり、論理型言語の代表的言語となっている。

最近報告されている Prolog の応用の多くはパーソナルコンピュータ（パソコン）で開発されたものが多く、実用と言うよりは実用化に向けた研究という色彩が濃い。実用化研究の対象は、これまでプログラム化出来ないと見られていた分野に向けられている。例えば、株・金融などの相談、被保険者の健康診断書を基にして保険金額の算定をするシステムなど知識を対象にした応用が研究されている［N］。近い将来に期待されている人工知能の本格的な実用化の準備として、パソコン上の Prolog を使って知識の表現と処理方法の研究が金融機関を中心に始まっている。

ところが、最近の具体的な応用の広がりとともに Prolog に対する期待とは

逆に、限界が見えて来ている。

(1) プログラムとデータが入りきれない。

(2) 長い計算では、スタックオーバーフローとなり実行が不可能になる。

(3) 実行速度が遅い。

Prolog の仕組を調べると、(1) (2)の限界は主記憶容量の限界であり、(3)は計算速度の限界であるか、あるいはアルゴリズムが適切を欠いているかである。一般に Prolog では非決定的アルゴリズムのために、手続き言語よりかなり遅い。

Prolog システムを使って、オモチャ的な試作からより現実的な応用に移る時、このような限界に直面するような例、また Prolog に関する行過ぎた宣伝から過大な期待を抱き失敗する例などがあるようである。パソコンＰＣ９８０１を使った我々の経験でも、Prolog KABA ［K］上でＯＰＳ５のインタープリタ［O］を起動してもほとんど有効な結果すら出さないうちにメモリがパンクする、また同じく KABA 上で Concurrent Prolog のコンパイラ［U］では１００以内の素数の計算でパンクしてしまう。エキスパート構築用ツール Shell-KABA ［B］でも短い質問を数回繰返しただけで、データベースの限界に到達してしまう。

これらの限界は、パソコンからワークステーション、ミニコン、センターマシンへとより大型のコンピュータを使えば片付く問題であると見られる。しかし、大型のコンピュータの利用となると共用化のために様々の制約が加わり不便になるので、実用的な使い方をしたいのであればコンピュータは出来るだけ小さい方が望ましい。規模が大きくなる応用例の対象を考察すると、３つのパターンが考えられる。

(1) データが多い。

(2) プログラムが多い。

(3) 計算が長い。

本来、Prolog ではプログラムとデータの区別をしないが、人間の側の使い方を考えてあえて区別した。データと言う時は、少なくとも本体を持たない節を

指している。上で述べた我々の幾つかの応用例では、プログラムが多くて計算が長いという特徴であった。このような場合は主メモリを大きくするより他に計算を進める方法が無い。しかし、企業体などの応用では「データが多い」だけという例が以外と多いのではなかろうか？

Prolog プログラミングの代表的な３つの例では、このようなデータが多くなる例であった。これを第１章で例示する。第２章では、Prolog システムの仕組について解説し、Prolog システムの限界がどこに有るかを見る。第３章では Prolog システムとデータベース管理システムの結合について考察する。

１。 Prolog プログラミングの例

Prolog がどのように使われるかを Prolog のプログラミングの例を示す。さらにアプリケーションとしての規模が大きくなるとは、どの部分が大きくなるかを指摘する。次の Prolog の典型的なプログラミングの３つの例について考察する：

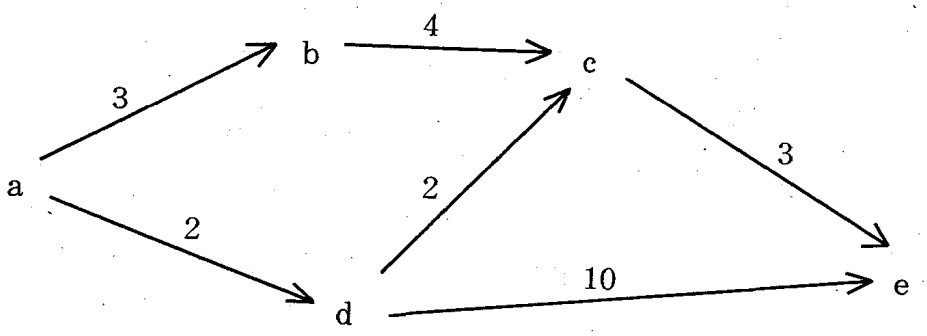
・再帰を使うプログラム
・構造を持つ情報の処理
・データベースとしての Prolog

A．再帰を使うプログラム

プログラムの再帰的定義は、過去２０年以上の長い間人工知能の研究に使われて来た Lisp 言語の歴史とその実績からみて、知的なプログラミングにおいて必須の条件と考えられる。 Prolog はこの Lisp 文化の遺産の多くを論理的プログラミングの中に受継いでいる。その例として、グラフの探索のプログラミングを見ることにする。

[グラフの表現]

[図１] のグラフを考える。ａ，ｂ，ｃ，ｄ，ｅは都市の名前と考え、矢は



[図1] グラフ

それらの都市間の移動可能な方向を示す。矢についている数値は移動のための時間を表わしている。都市aからeまでの最短時間を知りたいというのがここの問題である。

まずグラフ情報を Prolog プログラムに書き表すために、矢に注目し、各々の矢の情報をすべて記録すれば全体としてこのグラフを表わしている考えよう。そこで、

$$a \xrightarrow{3} b$$

を

arrow(a,b,3).

と表わそう。矢は6本だからグラフは、 Prolog 言語で次のように書かれる。

```
arrow(a,b,3).
arrow(b,c,4).
arrow(c,e,3).
arrow(a,d,2).                                    (A1)
arrow(d,c,2).
arrow(d,e,10).
```

また、ある都市Xからある都市Yに、あるH単位の時間で行くことが出来る

ことを、

go(X,Y,H)

と表わすとしよう。

（注：英大文字で書かれたX，Y，Hは変数を示すことになっている。従って、X，Y，Hは代名詞である。これに対してａ，ｂ，ｃ，ｄ，ｅなどは固有名詞である。）

このとき、矢がある２つの都市の間では常にgoの関係にあるから、

```
go(a,b,3).
go(b,c,4).
go(c,e,3).
go(a,d,2).                                    （Ａ２）
go(d,c,2).
go(d,e,10).
```

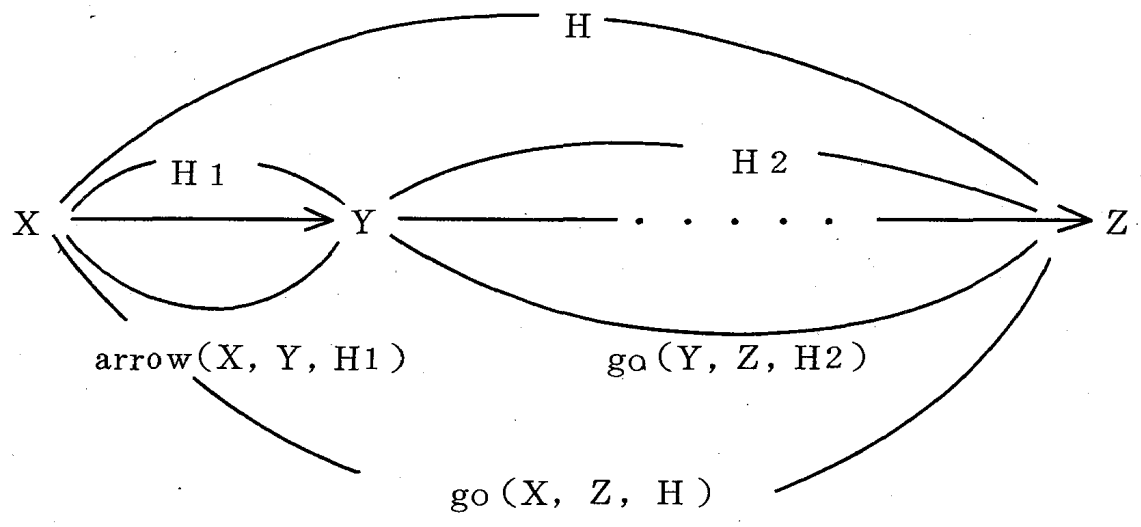
と表わすことが出来る。しかし、この情報は次のプログラムを与えれば、データ（Ａ１）を使って簡単にこの情報を得ることが出来るから、データとしてわざわざ与える必要はない。

```
go(X,Y,H) :- arrow(X,Y,H).                    （Ａ３）
```

つまり、１行の（Ａ３）だけで、６行の（Ａ２）と同じ効果を持つ。

[再帰的定義]

さて go の再帰的定義を考える。ＸからＺまでＨ時間で行くことが出来るとしよう。これが可能ならば、ＸからＺへの経路上にＸの隣の都市Ｙがあるはずだ。それを arrow(X,Y,H1) としよう。そして、このＹから終点の都市Ｚまで行けるはずであるから、それを go(Y,Z,H2) と表わそう。時間の関係式はＨ＝H1+H2が成立している。これらの関係を［図２］に表わす。



[図2] go(X,Z,H)の関係図

この関係を素直にプログラム化すれば、

```
go(X,Z,H) :- arrow(X,Y,H1), go(Y,Z,H2), H is H1+H2.      (A4)
```
(注)

（A4）において、goの定義をするのにgoを使っている所が再帰的定義と言われるところである。再帰的定義では、定義しようとする対象の構造を構成的に作ることが肝要なことで、この例のように go(X,Z,H) の定義をするのにこれより距離が狭くなった go(Y,Z,H2) で定義することで達成している。この再帰は（A3）の適用で終る。プログラム全体はこれに（A1）を加えたものになる。Prolog プログラミングではプログラムの評価が上から下に進められるので（A3）と（A4）の順序が重要となる。再帰の度に目標の都市に行着いたかどうかをチェックするために、（A3）の次に（A4）が来るようにプログラムを配列することに注意を払う必要がある。従って、こうなる。

```
go(X,Y,H) :- arrow(X,Y,H).
go(X,Z,H) :- arrow(X,Y,H1), go(Y,Z,H2), H is H1+H2.      (A5)
```

H is H1＋H2というのは、H1とH2の和を計算しその結果をHに代入することを示している。この is 述語は前もって Prolog システムに用意されているので、利用者が定義しないで使える。この様な述語を組込述語という。

[例１] この例の場合のプログラムの実行を次に示す。

```
?-go(a,e,X).        ａからｅまで何時間で行けるか？
X = 10;             解答の直後に ； を入力すると、別解を答える
X = 7;              これが最小時間を与えている。
X = 12;
no                  別解が無くなった時には no と答える
```

[規模に関する考察]

この例の場合、規模に関係することは２つある。データの大きさと、求める経路の長さである。データは、都市の個数と、都市と都市の間の関係つまり矢の本数である。例えば、全国の輸送ネットワークを考えるなら、全国の都市の数と、都市間の路線の数がデータの個数であり、経路は出発点から到着点までの間に通る都市の数が経路の長さである。経路の長さだけ（Ａ５）の再帰的定義を引用することになる。

問題は、アプリケーションを動かすそうとしている Prolog システムが（Ａ１）で表わされるデータをすべて記録できるであろうか、また求める経路の長さが計算の限界に入っているであろうか、である。

Ｂ．構造を持つ情報の処理

情報の多くはばらばらな情報ではなく、互に関連し合っている。その関連こそ重要な情報であるとみるのが、情報の構造である。従来データベースなどで構造という時、ある記録から関連する記録を取り出すためのリンク（ポインター）を指していた。しかし論理プログラミングでは、情報の関連を数学上の関係概念でとらえ、関数という項でこれを取扱う。以下に、言語処理の例を用いて構造を持つ情報の処理方法を見る。

[関数による構造表現]

関数はその引数に項を持つものである。例えば、変数Ｘは項で、その父親を

father(X) と表わすことにすれば、このXの父の父を、

father(father(X))

と書くことも出来よう。特に関数は、引数部に書かれる幾つかの項をまとめて新たな構造を作るので複合項と呼ばれ、構造を持つ情報の処理に極めて重要な働きをしている。では、構造を持った情報とは何かを説明しよう。

I like Tom.　　（B1）

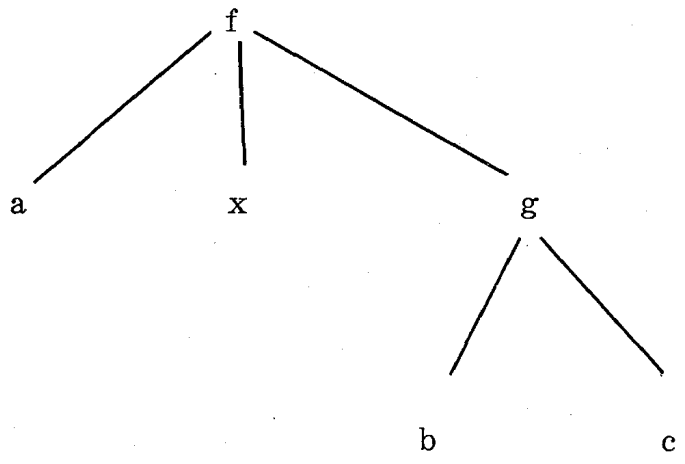
という英文の構文を認識することを考える。この文は3つの単語から出来ている。この単語をさらに文字に分解して考えることも可能ではあるが、文の理解という目的には意味がない。だから単語 like が4つの文字l，i，k，eで出来ていることが事実であっても、単語 like を1つのものと考える。likeをこれ以上分けることが出来ない1つのもの、つまりアトムと考える。大文字で始まっている単語IとTomを Prolog によって変数と解釈されないように'で囲み、単語をアトムとする。次の3つがアトムという項であることが確定した;

'I'　like　'Tom'　　（B2）

（B2）では3つのアトムを紙の上に並べて書いてあるが、コンピュータの中では、このままでは順番が指定されていず、3つの項はバラバラな状態で存

〔項について〕
Prolog の情報の表現と操作はすべて項を単位として行われる。都市の名前a，b，c，d，eは項であるし、都市と都市の間の関係を表わす arrow(a, b, 3)、arrow(b, c, 4)なども項である。さらに変数も項の1つと考えることにより、ある都市Xというのは項である。また項を引数に持つ述語 go(X, Z, H)も項となる。これらの概念を Prolog 文法の言葉で区別すると次のようになる。上記の例で、a，b，c，d，e等の小文字で始まる名前をアトムと呼び、時間を表わす数を整数と呼ぶ。これら2つを定数と呼び、項の1つである。X，Y，Z，H等の大文字で始まる名前を変数と呼び、これも項の1つである。 arrow(a, b, 3)、arrow(b, c, 4) や再帰的定義の go(X, Z, H)等を述語と呼び、これらも項である。この他に関数というカテゴリーも項である。

在している。そこで、アトムや項をコンピュータの中で順番に並べるという構造を導入する。この構造を作るのが関数であり、そのために用いられるのがリストである。リストの説明の前に一般の関数の図式表現を与えよう。例えばｆとｇを関数として f(a, x, g(b,c)) の図式は［図３］で与えられる。



［図３］　関数 f(a,x,g(b,c)) の図式表現

そこで、（Ｂ２）を（Ｂ１）の順番に並べる図式表現を［図４］に与える。２変数関数ｆを使って、３つのアトムを縫い合わせるように組合せている。ここで、アトムの並びの最後を[]というアトムで終了させる。この構造をリストという。普通に使われている関数記号はｆではなく、ドット関数・であるから、実は、

.('I', .(like, .('Tom', [])))　　（Ｂ３）

と書かれている。これでは読みにくいので、＋（ａ，ｂ）と表わさず、関数記号を演算記号＋や－などと同じ様に関数記号を中置きの表現、つまりａ＋ｂのように表現すると、

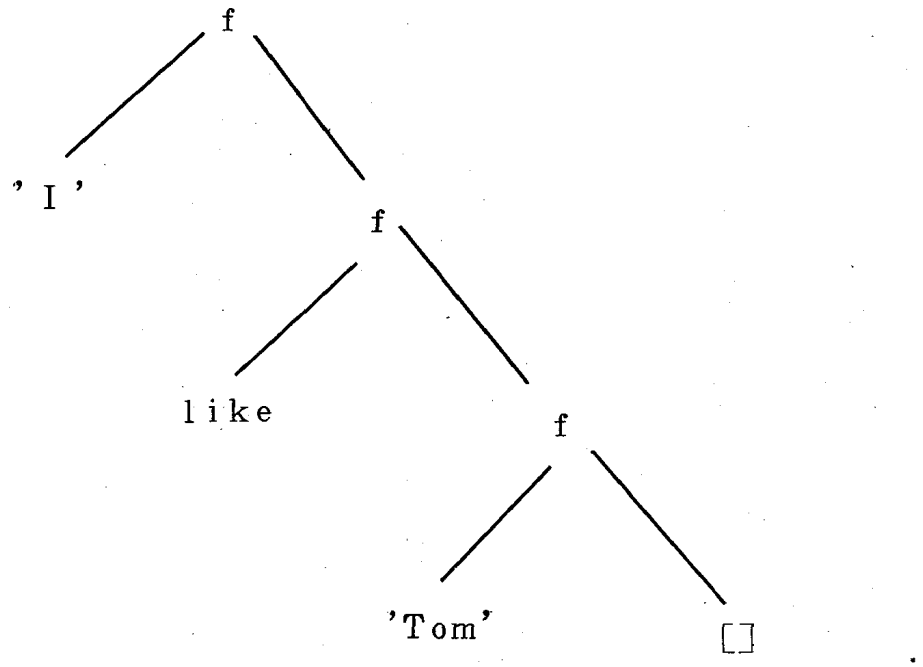
'I' . like . 'Tom' . []　　（Ｂ４）

人間にとって読みやすくなる。これをリストと呼んでいる。まったく同じ内容だが、内容をより端的に分かりやすく表現するために次のように表現すること

がさらに一般的である。

['I', like, 'Tom']　　（B5）

（B1）はこのように3つのアトムからなるリストという複合項として表現できる。



[図4]　f('I', f(like, f('Tom', [])))の図式表現

リスト構造に対する操作は次の2つの操作しかない。

(1)　リストの先頭に1つの項を付加えること。

(2)　リストの先頭から1つの項を取り出すこと。

「項Tをリスト Lの先頭に加えて出来た新しいリストがNである」という述語 cons(T,L,N) の定義を

cons(T, L, T.L).　あるいは、cons(T, L, [T | L]).　　（B6）

と書ける。つまり、Nは今までのリストLの前に項Tを . で繋いで出来るリスト T.L　であることを定義している。

Prolog 言語が他の言語と大きく違っている特徴の１つに「データの入力と出力の方向が決ってはいない」ことがある。この特性を使うと、今まで出力と考えていた３番目の引数の所にリストをデータとして与えると、そのリストの先頭の項を第１引数から取り出せるという、従来のプログラム言語では思いもしなかった様なことが出来る。だから、（Ｂ６）の定義だけで２つのリストの操作が出来る。

[構文解析プログラム]

さて、構造を持つ情報の処理として構文解析プログラムを例にする。
出来るだけ簡単にした文脈自由文法の英文の構文を次のように決めておく。

(1) 文　　－＞　名詞句＋動詞句
(2) 名詞句　－＞　固有名詞
(3) 名詞句　－＞　代名詞
(4) 名詞句　－＞　冠詞＋名詞
(5) 動詞句　－＞　他動詞＋目的語
(6) 目的語　－＞　名詞句

ここで用いることが出来る単語を次に列挙する。

(7) 固有名詞　－＞　'Ｔｏｍ'
(8) 代名詞　　－＞　'Ｉ'
(9) 名詞　　－＞　ｍｏｎｅｙ
(10) 冠詞　　－＞　ａ
(11) 冠詞　　－＞　ｔｈｅ
(12) 他動詞　　－＞　ｌｉｋｅ
(13) 他動詞　　－＞　ｈａｖｅ

ここで示した文法をより詳細にし、単語の量を増やせば実際に近い英文になるであろう。

これを Prolog のプログラムとして書換える。最近の Prolog には漢字を使えるので、プログラムも日本語化したのが［譜１］である。

```
(1)  sentence（InData，文（_名詞句，_動詞句），RestData）  :-
         np（InData，_名詞句，RData），
         vp（RData，_動詞句，RestData）.
(2)  np(InData，名詞句（_固有名詞），RestData) :-
         p_noun(InData，_固有名詞，RestData).
(3)  np(InData，名詞句（_代名詞），RestData) :-
         pro_noun(InData，_代名詞，RestData).
(4)  np(InData，名詞句（_冠詞，_名詞），RestData) :-
         art(InData，_冠詞，RData),
         noun(RData，_名詞，RestData).
(5)  vp(InData，動詞句（_他動詞，_目的語），RestData) :-
         vt(InData，_他動詞，Rest),
         obj(Rest，_目的語，RestData).
(6)  obj(InData，目的語（_目的語），RestData) :-
         np(InData，_目的語，RestData).
(7)  p_noun( ['Tom' | RestData] ，固有名詞('Tom'), RestData).
(8)  pro_noun( ['I' | RestData] ，代名詞('I'), RestData).
(9)  noun( [money | RestData] ，名詞(money), RestData).
(10) art( [a | RestData] ，冠詞(a), RestData).
(11) art( [the | RestData] ，冠詞(the), RestData).
(12) vt( [like | RestData] ，他動詞(like), RestData).
(13) vt( [have | RestData] ，他動詞(have), RestData).
```

[譜1] 構文解析プログラム

[例2]　次にこのプログラムの実行例を示す。

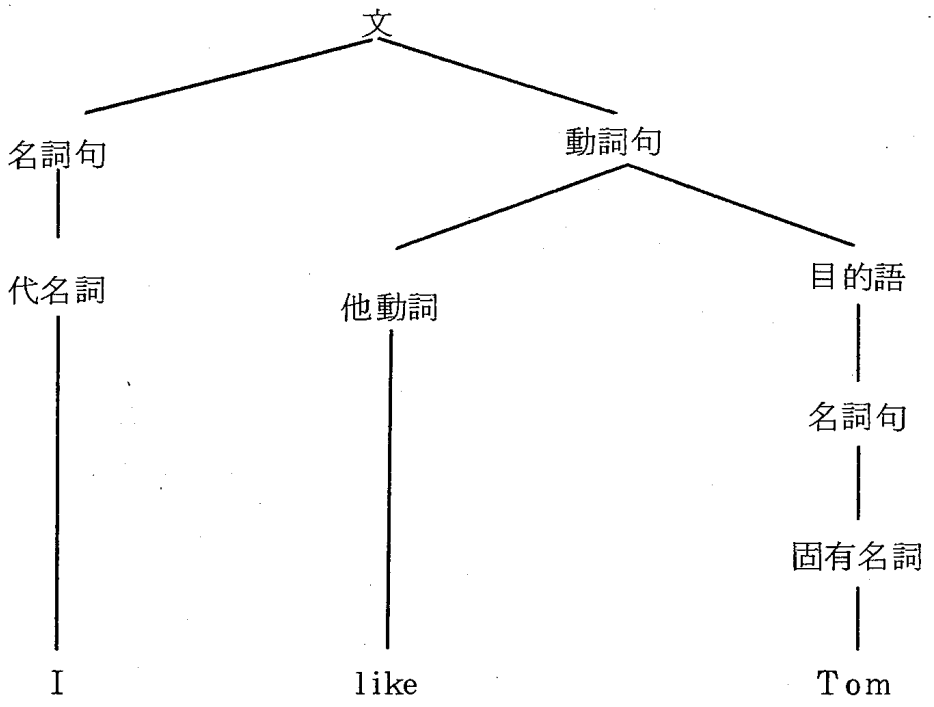
```
?-sentence( ['I', like, 'Tom'] , X, []).
X = 文(名詞句(代名詞('I'), 動詞句(他動詞(like),
```

目的語(名詞句(固有名詞('Tom'))))

yes

この関数表現を図式すると［図５］のようになる。［例２］では、英文のデータをリストという項で与えるため、あるいは得られた結果が複合項であるため、人間にとって Prolog を使うことが使い良いものではないように見える。

しかし、人間と Prolog の間のインターフェースのためにプログラムを書くことができるので、必ずしも人間がむき出しの Prolog を使わなければならないというのではないことを指摘しておく。



［図５］"I like Tom"の構文解析図

[規模に関する考察]

最近の自然言語処理では例外処理の文法を多く含めるようになって来たとはいえ、１つの文法を定めるには、構文概念である非終端記号の個数には限りがあり、数千個もあれば十分であろう。一方では、終端記号の単語の数は固有名詞も含めて数万個あるいはこれ以上にものぼり、その数は桁違いに大きい。実

用的な応用、例えば新聞に書かれている情報の自動的な整理をさせようとする場合、新聞に掲載される単語の中で互に異なる単語の数はいかほどであろうか？

C．データベースとしての Prolog

関係データベースに関する著名なC．J．Date の本［D］の中から、関係データベースの例題を一部引用する。この例では、幾つかの部品の製造業者から部品を購入し、課ごとに部品の在庫管理をしている。また各課の従業員ファイルもある。この様なシステムは企業体ならどこにでも見られるものであろう。これを Prolog で処理するプログラムの例を示す。

データベースのサンプルを次に示す。

業者

sno	sname	status	city
s1	smith	20	london
s2	jones	10	paris
s3	blake	30	paris
s4	clark	20	london

入荷

sno	pno	qty
s1	p1	300
s1	p2	200
s1	p3	400
s1	p4	200
s2	p1	300
s2	p2	400
s3	p2	200
s4	p2	200
s4	p4	300

部品

pno	pname	color	weight	city
p1	nut	red	12	london
p2	bolt	green	17	paris
p3	screw	blue	17	rome
p4	screw	red	14	london

[関係表を Prolog で表現する]

このデータを Prolog で、次のように表わしておく。関係の名前を述語の名前にし、項目は表の順序で引数と対応させる。

```
業者(s1, smith, 20, london).
業者(s2, jones, 10, paris).
業者(s3, blake, 30, paris).
業者(s4, clark, 20, london).
入荷(s1, p1, 300).
入荷(s1, p2, 200).
入荷(s1, p3, 400)
入荷(s1, p4, 200).
入荷(s2, p1, 300).
入荷(s2, p2, 400).
入荷(s3, p2, 200).
入荷(s4, p2, 200).
入荷(s4, p4, 300).
部品(p1, nut, color, 12, london).
部品(p2, bolt, green, 17, paris).
部品(p3, screw, blue, 17, rome).
部品(p4, screw, red, 12, london).
```

[データベースの言語　ＳＱＬ]

最近関係データベースの言語として、ＳＱＬ(Structured Query Language)という言語が注目され、１９８６年にはＡＮＳＩで規格化された。この言語は、ＩＢＭの１０年間の研究で到達した関係データベース用の言語である。SELECT で項目を選択し、FROM で情報を引出すファイルを指定し、WHERE で条件を決めるという非常に簡単な文法であるが組合せが系統的に考えられており、データベース操作のほとんどを定義できる。以下に示す例でこの言語のおよその姿

が理解可能であろうからＳＱＬの詳細は省略する。この言語を使って幾つかのデータベース操作を定義し、これと同じ内容を'Prolog で定義し、実行結果を見る。ＳＱＬの例題は、Ｌｉ［Ｌ］から採った。

［Ｑ１］パリの業者の名前とステータスをみたい。
これは、１つのファイルから条件にあう情報を検索する基本的なデータベース検索である。

```
SELECT sname, status        sname と status を見る。
FROM   業者                 業者のファイルの中の
WHERE  city = paris         city の項目が paris である。
```

これを、Prolog で実行すると次の様になる。

```
?-業者(_, Sname, Status, paris).
Sname = jones
Status = 10;
Sname = blake
status = 30;
no
```

Prolog では、興味が無い項目は、下線_の無名変数で示し、求めたい項目は大文字で始まる変数で指定する。

［Ｑ２］部品 p2 を供給している業者の名前を求めよ。
これは、２つのファイルの突き合わせを行う基本的な操作である。

```
SELECT sname
FROM   業者, 入荷
WHERE 業者.sno = 入荷.sno, 入荷.pno = p2
```

入荷ファイルで p2 を出荷した業者の番号を知ることが出来るが業者の名前を直接求めることが出来ないので、２つのファイルの突き合わせを項目 snoで行う。どちらの sno かを区別するために、ファイル名の接頭辞をつけている。

```
?-入荷(Sno, p2, _), 業者(Sno, Sname, _, _).
Sno = s1
Sname = smith;                    第1の解
Sno = s2
Sname = jones;                    第2の解
Sno = s3
Sname = blake;                    第3の解
Sno = s4
Sname = clark;                    第4の解
no
```

あるいは、求めたいものを一気に求めるには setof(X,Y,Z) を使う。Xは求めたい項、Yは求めるものの条件、Zは1つ1つの解Xをまとめたリストである。つまり、求めたい解として smith, jones, blake, clark があるわけで、この1つ1つを取り出す条件が「入荷(Sno, p2, _), 業者(Sno, Sname, _, _)」で、その解は、条件の中の Sname に表われる。この Sname をXとすれば、自動的にZにXのリストが表われる。

```
?-setof(Sname, q2(Sname), Z), write(Z).
 [blake, clark, jones, smith ]        アルファベット順に並び換えられ
yes                                   ている
```

ただし、q2(Sname): −入荷(Sno,p2,_), 業者(Sno,Sname, _, _).

［Q 3］少なくとも1種類は赤の部品を供給している業者の名前を求めよ。この操作では、1つのファイルを検索した結果を次の検索に利用できるという系統的な方法を繰返して複雑な検索を可能にする。

```
SELECT  sname
```

```
FROM    業者
WHERE   sno IN                          ‘業者の sno が以下の手続き
          (SELECT   sno                 で得られる sno の集合に入
           FROM     入荷                っているという条件。
           WHERE    pno IN
                    (SELECT  pno
                     FROM    部品
                     WHERE   color = red ) )
```

```
?-setof(Sname, q3(Sname), ANS), write(ANS).
 [clark, smith]
yes
```

ただし、q3(Sname):－部品(Pno, _, red, _, _), 入荷(Sno, Pno, _),
業者(Sno, Sname, _, _).

［ＳＱＬと Prolog ］

このようにＳＱＬ言語で書かれた質問は大きく書換えることなく Prolog 言語で書き表すことができる。［Ｑ１］と［Ｑ２］の書換えは直接的であるし、setof 述語を使っても大きな違いはない。［Ｑ３］では多少の Prolog の知識が有れば素直に書換えることが出来る。ここで示したように、逆に Prolog の利用者であればＳＱＬ言語のプログラム相当の Prolog プログラムは自由に書けるであろう。

Liは、A PROLOG DATABASE SYSTEM ［L］の中で、データベースシステムのインターフェイスに Prolog システムを置き、ＳＱＬなどの問合せ言語をPrologを用いて論理式に翻訳し、これを検索効率の点で最適化して Prolog 言語に変換し、この Prolog に変換された質問の実行でデータベースにアクセスするというシステムを提案している。この方法を使えば、Prolog 言語でＳＱＬを Prolog に翻訳できることを示している。つまり、インターフェイスはProlog

であっても利用者からはＳＱＬ専用に見えるデータベースを作れる。

[規模に関する考察]

ここでの規模の問題は、アトムとしての業者や部品の大きさと、トランザクションとしての入荷件数が重要である。関係ファイルの組は１つの述語だけを持つ節で表現される。検索が Prolog の単一化という非常にコストを伴う検索方法で行われる場合、組の個数が大きければ大変な検索時間がかかるであろう。

2。Prolog システムの内部構造

1。で示したプログラムは、Prolog 言語の処理系で実行される。従って、この処理系での実行の限界が結果の限界になる。 Prolog プログラムが実行されるシステム内部構造を見ながら、この処理系の限界を考察する。

[Prolog の実行]

1。のグラフ探索の例を用いて、Prolog の実行を説明する。プログラムは（Ａ１）と（Ａ５）をそのまま使う。説明に関係する部分だけを次に示す。特には Prolog の実行戦略や単一化（ユニフィケーション）・アルゴリズムを述べることはしない。これらに関しては、［S］［G］［F］などがある。

```
arrow(a,b,3).                                          ( a )
arrow(b,c,4).                                          ( b )
go(X,Y,H) :- arrow(X,Y,H).                             ( c )
go(X,Z,H) :- arrow(X,Y,H1), go(Y,Z,H2), H is H1+H2.    ( d )
```

質問が都市 a から都市 c までの時間を求めるものであったとして、質問から解答までの Prolog の実行の道筋を示そう。

```
?-go(a,c,ANS).          最初の質問                     ( e )
```

引数を3つ持つ go は2つ有り、まず（c）の節の可能性を試す。（c）とそれぞれ対応する引数との間で単一化が行われ、X←a, Y←c, H←ANS という単一化によって単一化が成功するので、（c）は次の節に書換えられる。

go(a,c,ANS) :- arrow(a,c,ANS).　　（c'）

この成功によって、最初の質問は ?-arrow(a,c,ANS). に変換される。従って、この新しい質問に対する解を探すため、引数を3つ持つ arrow の節を上から順番に質問と単一化が出来るか試すが、第1引数が a で、第2引数が b である arrow 節は存在しないので、新しい質問は失敗する。この失敗によって（c'）の採用は否定され、次の節（d）との可能性を試すことになる。このように一旦採用していた節を放棄して新しい節を探すという手続きを再試行という。この再試行によって、

go(X,Z,H) :- arrow(X,Y,H1), go(Y,Z,H2), H is H1+H2.

が呼出され、単一化：X←a, Z←c, H←ANSの成功によって、この式は

go(a,c,ANS) :- arrow(a,Y,H1), go(Y,c,H2), ANS is H1+H2.　　（f）

となり、（f）の3つの質問が順番に実行される。2番目の質問は1番目の質問の解答を得るまで待たされる。3番目の質問も同様に2番目の質問の解答があるまで待たされる。

?-arrow(a,Y,H1).	1番目の質問
?-go(Y,c,H2).	2番目の質問
?-ANS is H1 H2.	3番目の質問

まず、1番目の質問から実行される。

?-arrow(a,Y,H1).　　（g）

この質問の解答として、（a）が選ばれ、単一化はY←b,H1←3 が確定する。

この単一化によって、2番目の質問は、

?-go(b,c,H2) （h）

となる。この質問に対して（c） go(X,Y,H3) :- arrow(X,Y,H3). が選ばれる。ここで、名前の混乱を避ける為に H が H3 と改名されている。（c）は単一化：H3←H2 が行われた節

go(b,c,H2) :- arrow(b,c,H2)

によって、新しい質問 ?-arrow(b,c,H2) に変換され直ちに実行される。新しい質問の実行は、節（b）との単一化で H2←4 が確定し、第2の質問に対する解答が終了した。これで、H1、H2 の値が確定し ANS is H1 H2 の計算を行うことが出来る。つまり、3+4 の計算の値 7 を ANS に代入する。これによって、すべての計算が終了し ANS の解答 7 を得ることが出来た。

[計算のプロセスを記憶するスタック]

上の実行例の最初の質問に対して解の可能性は（c）と（d）の2通りあった。まず（c）を試して、（c）の採用があとの計算に都合が良くないと分かったら（c）を諦め（d）の可能性を試した。このように Prolog の処理系は質問に対する解答のために計算を実施するが、その計算は必ずしも一直線ではないことが多い。そのため、 Prolog システムは複数の解答の存在を認めた上で計算を進めるという非決定的アルゴリズムを採用し、再試行のために計算の中間結果をすべて記憶していることが他の言語には無い特徴である。この特徴を最大限利用するのが Prolog プログラミングであり、未知の解を探索するために用いられる。この様な特徴をもたらす中間結果の記録は、各単一化の成功の度に行われ、そのとき記録すべき基本的なものは次の3項目である。

(1) 呼出しの句（質問）
(2) 呼出しに応じて単一化に成功した節
(3) 単一化情報

これらの情報は非決定アルゴリズムを実現するために、計算の再開のために(1)どんな質問で、(2)どこまで進んでいたか、(3)どんな計算をしたかという情報を記録するものである。

1つの質問の呼出しによって作られるこれらの情報が有効な期間は、呼出しの成功によってこの情報が作られてから、この呼出しに対する再試行までの間である。上に説明したように呼出しは順次行われ、成功か失敗かの評価は常に最新の呼出しについて行われるので、否定が起きる可能性は常に最新の呼出しについてである。否定された情報は将来解答に採用される可能性がないので、その時点でメモリの有効活用のため消去され、次の呼出しのために使われる。このような使い方に最も適したデータ構造はスタック(stack)である。

通常(1)と(2)が記録されたスタックを制御スタックと呼び、(3)の単一化情報が記録されたスタックを変数スタックと呼ぶ。 Prolog システムの計算限界はこの制御スタックと変数スタックという2種のスタックにどれだけのメモリが用意されているかによって決まると言って良い。

[データベース]

一般にデータベースという時には、一定の形式にそってディスク装置に記録されたデータを指すが、Prolog でデータベースという時は、Prolog のための情報（節）を記録するための Prolog システムの中に組込まれたメモリの領域、あるいはここに記録された内容を指している。このデータベースは Prolog システムにおいて最も基本的であるので、その構造を調べる。

Prolog 言語の中のアトムつまり、文字、記号、数字、アルファベットで始まる名前などを一意に管理するために、それぞれのアトムに整数の番号をつけて管理している。アトムの名前からこの番号を探すメカニズムがハッシュ関数(hash)である。ハッシュ関数は、文字列の集合から適当な範囲の整数の集合への写像として定義される。ハッシュにおいて異なる文字列が重複して同じ値を取る場合、これらの文字列を同義語といい、同義を避けるために特別なアルゴリズムが追加される。このアルゴリズムの追加によって、すべてのアトムには

一意の整数が付けられ、ハッシュ関数によってその値を高速に計算することが出来る。関数はアトムである関数名（関数子）と引数の個数（引数価）で管理され、アトムと同じアルゴリズムで整数が付けられる。今後はアトムと言う時は関数も含めていることにする。

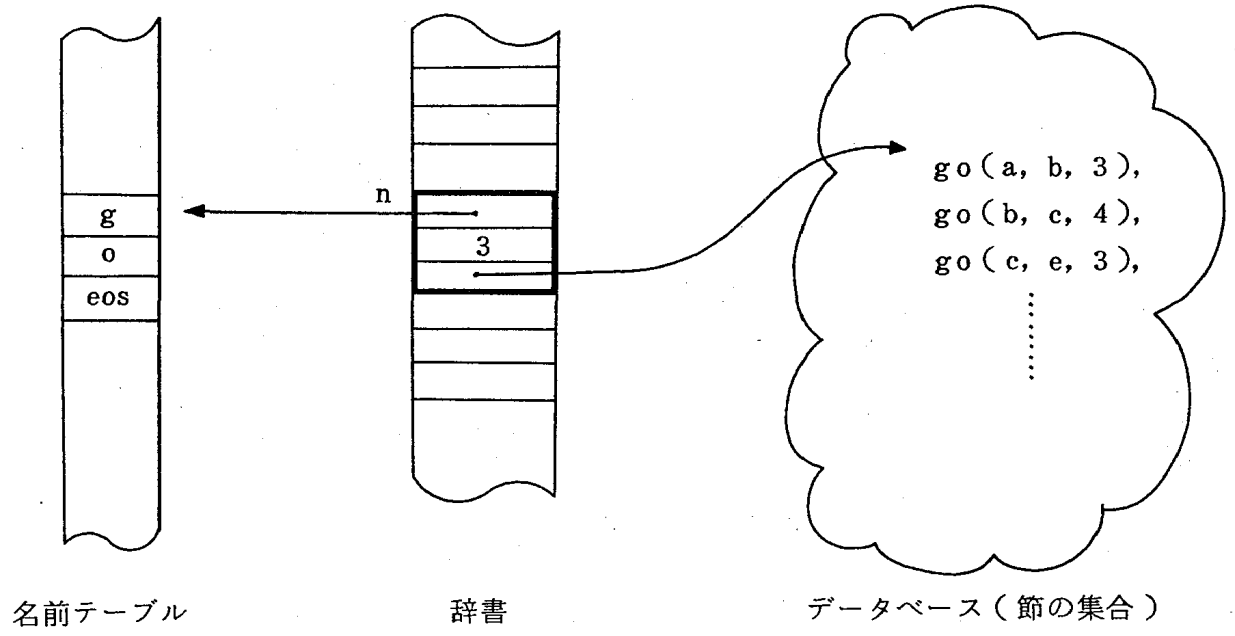
アトムの情報を一意に管理するために用意された表がある。先に付けられた整数はこの表の中の位置（アドレス）を示している。この表をハッシュ・テーブルと呼ぶ。つまり、アトムに関する次の情報がアトムに付けられた整数の番号に対応する表の位置に記録される。

(1) アトムの名前

(2) アトムの引数価

(3) このアトムを節頭に持つ節の集合

多くの処理系では、(1) (3)の情報はポインターで記録され、それぞれの情報はこれとは別の表のポインターで示される位置に記録されている。［図 6］に 3 引数の go 述語の定義を示す。［図 6］に示したように、データベースは(3)の節が記録さているメモリの領域あるいは節そのものを指す。これらの情報を記録することによって、アトムを一意に管理でき、さらに質問として与えられ



［図 6］辞書とデータベース

たアトムに対応する節の集合を直ちに検索できる。従ってこの表を辞書と呼ぶこともある。

次にデータベースの中での節あるいは、項の表現を見る。関数や述語を含めてアトムには、識別のために一意のアドレスが付けられている。節や項はこれらのアドレスを用いて組み立てられている。この組み立てのために、システムの実現の都合からデータ構造として幾つかの選択が考えられている。その主な基本データ構造として次の3つがある［Na］。

(1) 固定長レコード形式

(2) 可変長レコード形式

(3) 2進リスト形式

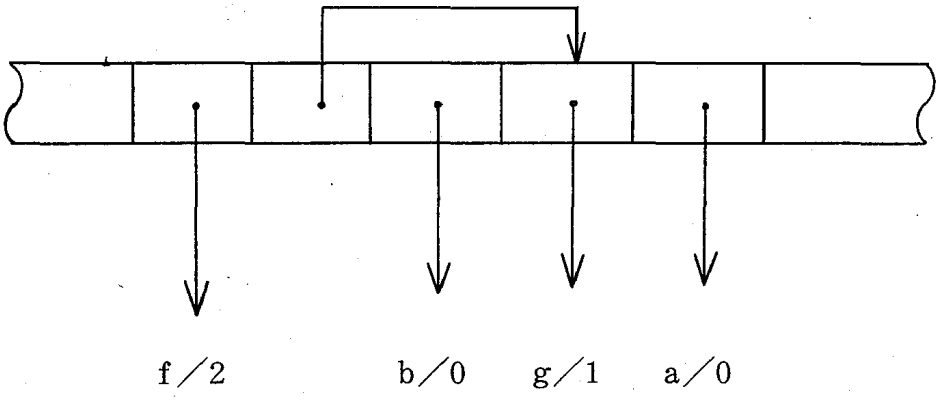
その構造図を［図7］に示している。これらのデータ構造のうちどれを用いるかは、システムの特性と制作者の思想的好みによって選択されている。例えば最も早く作られたマルセイユ Prolog では可変長レコード形式を使い〔S2〕、LISPで実現されている prolog/KR ［Nk］などは2進リスト形式である。

Prolog のデータベースは、プログラムあるいはデータとして入力された節を記録している。その記録は上に示したように様々な方法があるが、いずれの方法でも辞書で管理されたアトムのアドレスを参照している。このように、Prolog ではすべての情報は基本になるアトムを組合せて表現されている。

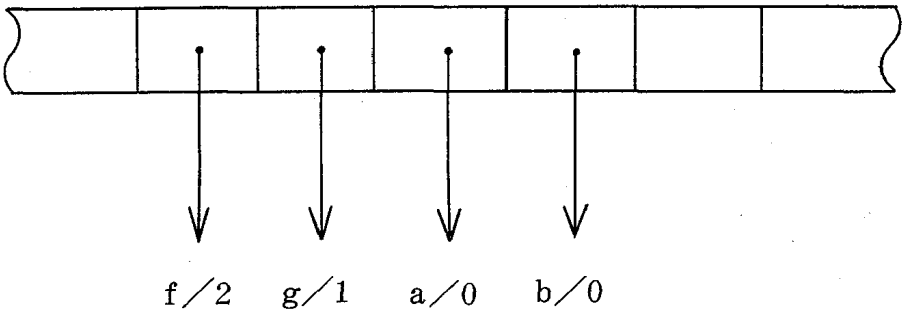
このような方法で Prolog がどれだけの情報を取入れることが出来るかは、アトムのテーブルである辞書のサイズと、節や項を記録するデータベースの領域のサイズの大きさによって決る。従って、このメモリの領域が大きければ、より多くのプログラムとデータを Prolog の取込むことが出来る。

一方、 Prolog の実行時にデータベース・オーバーフローという表示が出て、実行が異常終了することがある。実行の途中で新たなプログラムを取入れた訳でもないのにこの様なことが起こる場合がある。 Prolog の計算途中で生成される項や数値などはデータベースに新たに記録されので、余分のスペースがデータベースに無い場合には、上記の異常終了になる。このあたりの事情は処理系の作り方に依存しているのでそれぞれ特殊な場合が多いようである。

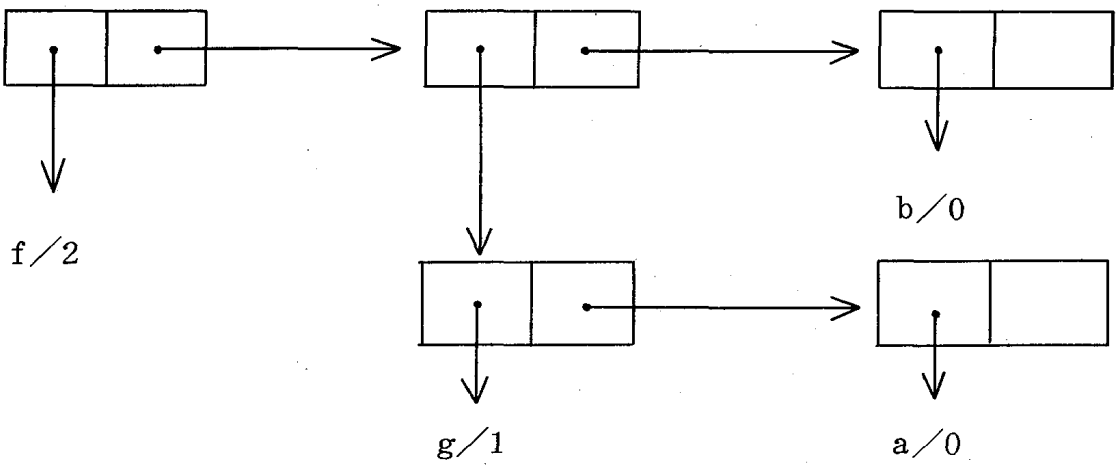
(1) 固定長レコード形式



(2) 可変長レコード形式



(3) 2進リスト形式



［図7］項 f(g(a),b) の表現形式

---

f/2, a/0, b/0, g/1は、それぞれ2引数の関数 f, 引数なしの関数 a, b 引数1の関数 g のアドレスを示している。

3。 Prolog システムとデータベース管理システム

本章では、小さな Prolog システムを用いても規模が大きなデータを持つ応用を実現するために、 Prolog とデータベース管理システムとの結合を提案する。

[Prolog データの 3 つの特徴]

1 章で示したように企業体での Prolog の応用を考えると、多くの応用例でデータのみの規模が大きいことがわかる。また、これらデータへのアクセスには 3 つの特徴がある。

第 1 の特徴は、検索が単純である。例えば、 2 章の質問（ c ’）

?-arrow(a, c, ANS).

では、「 arrow の関係の中で、第 1 項目が a で、第 2 項目が c である組があるか？ もし有れば、第 3 項目の値を ANS に代入しなさい」というものである。

第 2 の特徴は、データ構造が単純であるので関係で表わせる。第 1 章の arrow データ（ A 1 ）は、

arrow			
	a	b	3
	b	c	4
	c	e	3
	a	d	2
	d	c	2
	d	e	10

と表わせるし、第 1 章の「データベースとしての Prolog」では、データそのものが既に関係であった。

しかし「構造を持つ情報の処理」では、［譜１］の（７）から（13）のデータ構造は複雑である。例えば、（７）の「Tom が固有名詞である」というデータを

```
p_noun( ['Tom' | RestData] , 固有名詞('Tom'), RestData).
```

と表わしてある。第１、第２引数とも関数表現になっており、記録要素の値が文字列と数値に限定される関係データベースの組の要素となれない。もともとこの表現は、すべてを Prolog で実行することを前提にして書かれているので関数表現になっているが、次の様にデータとして用いられた節をプログラムにし、データ部分だけを単純化することが出来る。

```
p_noun( ['Tom' | RestData] , 固有名詞('Tom'), RestData) :-
        p_noun_data('Tom', 固有名詞).
```

この p_noun_data('Tom', 固有名詞）の部分が新たなデータである。これなら、関係データベースとして表現できる。(8)から(13)までの節も同様に書換えができる。この書換えはここだけの特殊な取扱いではなく、企業体の多くのデータの場合には、 Prolog プログラミングの技術的な方法でデータの単純化は可能であろう。

第３の特徴は、検索されるデータ数はデータベースと比較して少ない。つまり、第１の特徴で見たように、検索には単純ながら条件がつけられる。この条件のために、データベースに記録されたデータの多くが条件を満たさず、全体の大きさと比較すれば極めて小さい部分のデータのみが検索される。例えば、第２章では ?-arrow(a, c, ANS) の質問に該当する組は存在しなかった。

[外部データベースの導入]

このように単純な検索のみであれば、第２章で説明したような Prolog 内に組込まれているデータベースに記録するより、 Prolog とは異なるデータベース（外部データベース）にデータを書込んでおき、必要な時にこのデータベー

スにアクセスするという方法が考えられる。その概念図を［図８］に示す。

この方法には、幾つかのメリットがある。 Prolog システムに関しては、データが外部に記録される分だけ主メモリが少なくても良いので、小さな Prologシステムであっても、データの規模が大きな応用にも実用化できる。一方、外部のデータベース管理システムを使うので、このためのハードとソフトを用意しなくてはならないが、次のようなメリットがある。

・大量のデータの管理が容易になる。

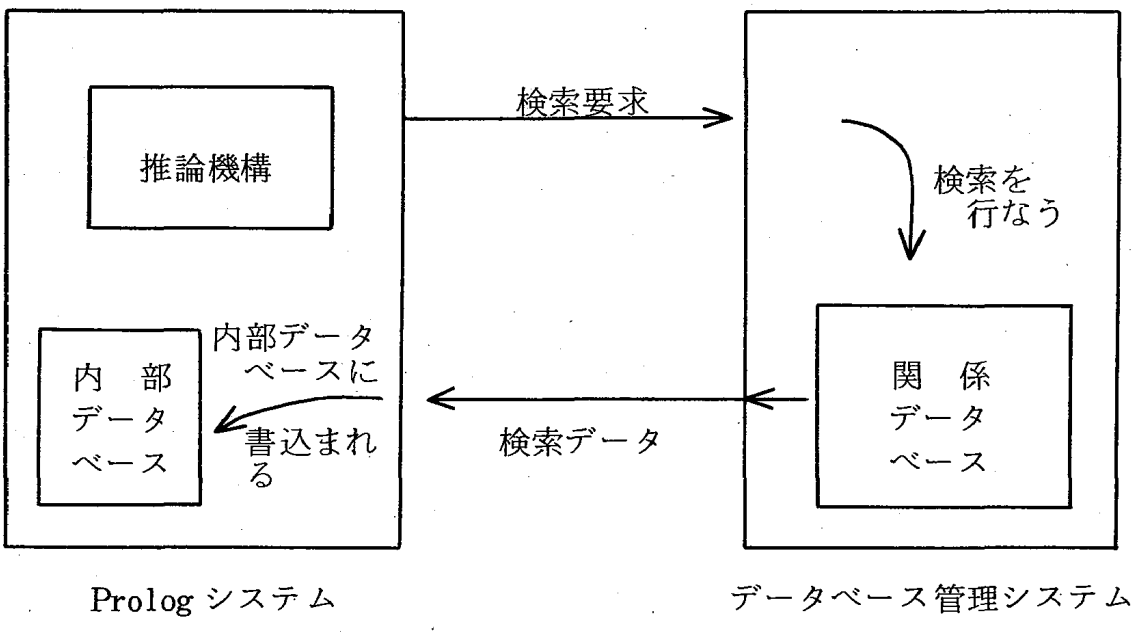
・データ検索が早くなる。

しかし現在のところ、 Prolog システムとデータベース管理システム（ＤＢＭＳ）とを結合するには、２つの問題がある。

(1) Prolog システムからＤＢＭＳへ検索命令を渡す標準的な手続きがない。

(2) ＤＢＭＳから Prolog システムへ検索データを渡す標準的な手続きがない。

現在のパソコンの多くは、まだマルチタスクになっていないので Prolog システムとデータベース管理システムの２つのシステムを同時には実行できない。従って、２台のパソコンにそれぞれのシステムを起動しＲＳ２３２Ｃなどの通



［図８］ Prolog システムとデータベース管理システムの結合

信インターフェイスを用いて結合するのが当面の方法であろう。しかも上で指摘した第3の特徴は、この通信容量が大きくなくても構わないことを示しているので、通信コストを考える上でも都合が良い。

さらに、通信ということを延長して考えると、ネットワーク結合になるだろう。今日は、かつての大型コンピュータを中心にしたシステムの構成法から、ネットワークを中心とした分散処理へとコンピュータの応用法が変って来ている。このような展望に立つと、 Prolog システムが通信のための標準的な手続きを必要としているのは、上記の2つの問題を含めて現在的な課題である。

4。 参考文献

[B] バイナス・ユニー 「Shell KABA」

[D] C.J.Date "An Introduction to DataBase Systems",Addison-Wesley

[F] 古川康一 「Prolog 入門」、オーム社

[G] 後藤滋樹 「PROLOG 入門」、サイエンス社

[K] 荻野、桜川、柴山、Prolog KABA Reference Manual、岩崎技研工業

[L] Ｌｉ， "A PROLOG DATABASE SYSTEM", John Wiley & Sons Ltd
訳「PROLOG データベース・システム」近代科学社

[Ｎａ] 中村克彦 「Prolog と論理プログラミング」、オーム社

[Ｎｋ] 中島秀之 " Prolog/KR User's Manual"

[N] "エキスパート・システムの実用化に入った大手金融機関"、
日経コンピュータ 1986.10.27

[O] 小沢、小林「 OPS5 on Prolog 」, Computer Today 1986/5, No.13

[S] 杉本英二 「述語論理型言語 Prolog の紹介」、小樽商科大学、
商学討究 第33巻第1号 1982年

[Ｓ2] 杉本英二 「マルセイユ Prolog システムの拡張」、
小樽商科大学, 商学討究 第35巻第4特別号 1985年

[U] 上田和紀 「並列論理型言語の実用処理系」日本ソフトウエア科学会第

1 回大会論文集 3D-5