ICOT Technical Report: TR-070

TR-070

WG4 ワークショップ '83
"知 識 表 現"

編 集
溝口 文雄 (東理大)
古川 康一 (ICOT)

August, 1984

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03) 456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

目　　次

エキスパートシステム設計用の知識表現
LOOKS & POOLS

東京理科大学工学部経営工学科　　溝口　文雄

▲ はじめに ▲

本研究は、論理型プログラム言語を用いてのエキスパートシステム設計用の知識表現言語を開発することを目的にしたものである。

現在のエキスパートシステムはLISPを用いて（特に、INTER-LISP）作成されてきたが、新しい展開を考えると、論理型プログラム言語の持つ、さまざま可能性を検討することは重要である。本研究の動機も、その可能性を出発点にしている。そして、そのひとつの可能性は、すでに、論理型プログラム言語で、プロダクションシステムを作成したときの経験（HIZOGUCHI,F.,1983)で示したように、プログラムの生産性として示すことができる。すなわち、多機能の効率の、効率のよいプロダクションシステムが論理型プログラム言語のDEC-10 PROLOG で実現できた。

本研究は、エキスパートシステムで利用される知識の記述を、さらに柔軟にし、かつ豊かにするための知識表現言語を検討することである。具体的には、LOGIC ORIENTED AND ORGANIZED KNOWLEDGE SYSTEM(略してLOOKS)およびPROLOG OBJECT ORIETED LANGUAGE SYSTEM(略してPOOLS)の２つの言語の開発を進め、これらの言語のイメージに触れることである。

なお、LOOKS & POOLS の２つを合せてLPS と呼ぶ（Logic Programming System）こともある。

▼ 情報処理モデルのイメージ ▼

従来のプロダクションは、人間の情報処理モデルを反映する形で構築された(Nowell,A.,1972)。特に、外界の情報をどのようにコード化し、それを記憶にストアするかに焦点があてられていた。その結果、短期記憶→長期記憶というデータの解釈と転送を中心とする現在のプロダクションシステムの枠組みが開発された。そして、多くのエキスパートシステムが、この枠組みを採用している。外部データを処理する知識は、「IF~THEN」型のルール集合となっている。

確かに、ルール表現の知識は、読みやすさ(Readability)、まとまりやすさ(Modularity)、変更がしやすい(Independence)等の利点を持ち、実際の応用システムの開発には向いている。そのことは、LISPを基礎にするエキスパートシステム、例えば、MYCIN,EMYCIN,OPS等で実証ずみである。

ところで、最近の認知科学で提案されている人間の情報処理のモデルは、図１に示すような構造となっている。すなわち、従来のモデルとは異なって、並列処理および、記憶域の活性状態から、短期と長期の処理の違いを説明している。そして、外部データは記憶内の表現図式（スキーマ：Schema）によって解釈、および処理されるというものである。

もし、我々の情報処理モデルを、エキスパートシステムのイメージとして採用するとすると、このイメージに対応した形の知識構造や、表現の形式を考えていく必要がある。本研究の立場は、明らかに、並列処理の情報処理モデルにおいている。

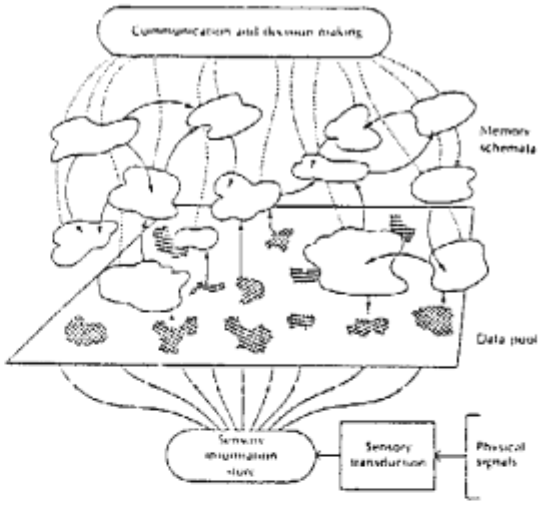


図１．　Bobrow & Norman　並列処理モデル

▼ 対象世界の知識表現言語　...LOOKS... ▼

対象世界（エキスパートシステムの）を知識表現としていくためには、その対象を構造的枠組みとして再構築していく必要がある。LOOKS は、対象世界を、自然な形式で、可読性の良いものにするための知識の組織化を図

った表現言語で、フレーム構造に基礎をおく、フレームの代りに、スキーマと言ってもよい。とにかく、何んらかの知識構造を表現する単位（ユニットないしはプロトタイプ）を使って、知識の組織化を行なうことを目標にしたものである。ただし、Logic Orientedであるために、知識単位を第１階述語論理の形式に変換する必要がある。すなわち、LISPでの知識表現が、最終的には属性リストで扱われるのと同様に、LOOKS では、節の型式で扱っていかなければならない。

したがって、スロットとフィラーから構成されるフレームは、次のような形のassertion として扱われることになる。スロットの関係をR1,...Rnとし、その関係を持つフレーム概念をCとすると、次のように書くことができる。

```
x(C(x)≡  y1,...yn
    R1(x,y1) &  ... & Rn(x, Yn))
```

この表現の特徴は、フレーム概念Cをひとつの構造体で表わしていたものをいくつかのAssertion で置き換えることに相当する。

したがって、次のフレーム形を考えると、Assert形にすることは容易である。

```
(Frame name (slot1 (facet1 (value1 )
                   (facet2 (value1 )

            (slot2 (facet1 (value1 ) ..)..)..)
```

つまり論理的枠組で、フレーム構造を表現することができる。

LOGIC でこうした構造を表現することは、スロット－フィラー間の推移をいわば埋め込んだ形にすることになる。

```
assert(frame name
        ([[slot1, facet1, value1 ],
          [slot2, facet1, value1 ],
                                ..])).

[frame name, [internal predicate]]
```

このことは、関係データベースにおける問い合せが容易に行なえるのに似ている。

PROLOGによる知識表現言語は基本的には、LISPと同様の記述力を持つ。さらに、論理型プログラムの利点を生かせば、別の展開も可能である。そのことは、POOLS で触れる。

▼ LOOKS の特徴

LOOKS は知識の表現単位としてフレームを持つ。したがって、フレームの持ついくつかの特徴を有している。例えば、i) 階層構造　ii) 遺伝性　iii) 相互干渉である。

エキスパートシステムでは、外部データを処理するための手続き付加が重要である。LOOKS では、この手続き付加は、DEMON によって実現されている。

具体的には、図２のような手続きとメッセージ交換によるDEMON 駆動によって、情報処理が進む。この方式は図１の処理モデルを具体化したものである。

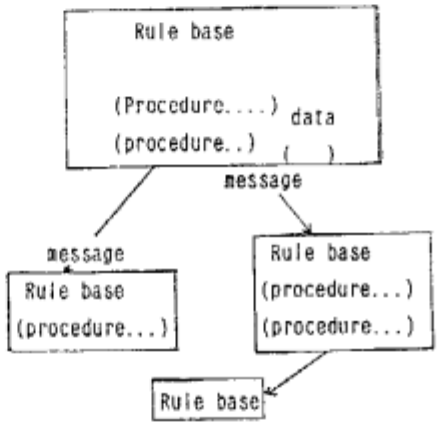
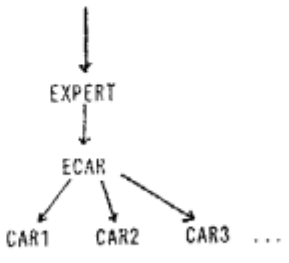


図２．　メッセージにより手続き部の起動

図２におけるRule base はAKO リンクより成る階層構造のルールフレームである。つまり、上位のルール、中位のルール、下位のルールが外部のデータ処理に対して、各々の役割に応じて働くような階層的構造となっている。その役割における構造を示したのが図３である。この図３は故障の診断のルール階層を示したものである。

(a)



(b)

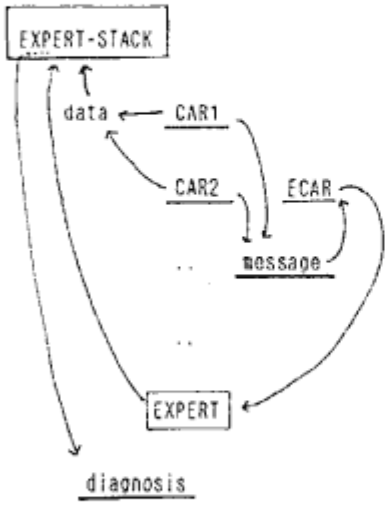


図3　ルールの階層のメッセージ交換による情報処理

(a) ルールの階層
(b) ルール内部の処理

図3(b) に示したように、診断結果を得るまでは、下位のルール群CAR1, CAR2... データ収集を行う。次に、ECARがデータ収集の状態をチェックした段階で、上位のルールであるEXPERTを呼び出す。EXPERTはデータ収集の状態をみて、診断可能ならば、その結果を出力する。この例題の場合には、フレーム内部に書く性質については、充分には利用していない。むしろ、ここでは、ルールの上位－下位と、DEMON の利用で診断がどのように進むかを示したのである。

LOOKS の特徴

1．遺伝性の制御
・AKO リンクの利用による知識の遺伝性とデフォルト計算
・遺伝性の制御はUNIT方式のSame, Restricted, Opti Optional Unique を用いる。

2．　フレーム構造への手続き付加
・メッセージによるDEMON 駆動
DEMON はPROLOGのプログラムとして記述する。

3．　構造的ルールベース
・ルールを、名前つきフレーム構造と考えて、ルール間の階層性を導入し、ルールの構造化を可能にしている。

4．　会話的知識の組織化

5．　システムの挙動、内部構造、結合関係のフレーム表現からHRS(Genesereth, M)と同様の診断が可能である。

ただし、フレーム表現における多重視点(Multiple View) および多重遺伝(Multiple Inheritance)の機能は現在検討中である。

実際の対象領域は、回路の故障診断について、システムに関する知識から推論する方式を、現在、解析中である（片山、満口、1983）。

以上が、対象世界の知識表現の立場から、アプローチしたエキスパートシステム用の言語の開発である。

LOGIC の枠組で、知識表現をどのように考えるかは、今後、検討する部分が数多く残されている。

▼　オブジェクト指向型からの知識表現言語
... POOLS...

オブジェクト指向型言語におけるオブジェクト（Object）は、対象世界の知識表現における構造フレームと微妙な違いを持つ。ただし、知識表現における意味的なクラスの概念や、遺伝性（性質の）等の考え方は、同一の考え方である。

POOLS はXerox PARCのLOOPS マニュアル（Bobrow,D.G.,1983) を基礎にして、PROLOGで作成中の試験システム（本田、1983）である。

POOLS のオブジェクトは次のようにして定義される。

```
Class （クラス名）
  { Super([ クラス名 ,...]);
    metaclass([ クラス名]);
    classvar([クラス変数名 ,...]);
  instvar([ インスタンス変数名 ....]);
       method([関数名 ....])).
```

ただし、methodの関数は

```
関数名(([Selector(Arg)],Self),
                Return_Value) :- !, ... .
```

という型をしていなければならない。

また、Super とmetaclass は省略できない。

```
class(counter) <=
        ( super([compute_counter1,
                 compute_counter2 ;
          metaciass([class]) ;
          instvar([state]) ;
          method([counter,counter1]) ).
```

Objectとしてよく例に使われるCounter の定義例である。

Counter の例をオブジェクト関係で表現したのが図４である。

図４　POOLS におけるオブジェクトの関係　（本田）

Pools はConcurrent　Prologを使わないで、LOOPS を参考にして素直に、オブジェクト指向型言語のインタープリタを作成したものである。使用言語Ｃ－Prologである。

このPOOLS とLOOKS とをどのように融合していくかは、現在検討中である。融合を考えるというよりかは、POOLS で、フレーム表現を考えるというアプローチをとる予定である。課題は数多い。

並列モニターの問題
多重の意味表現
シミュレーションとの融合

```
counter(([clear],Self),Self) :- !,
        get_classvalue((Self,reset),X),
        put_value((Self,state),X).
counter(([up],Self),Self) :- !,
        get_value((Self,state),X),
        Y is X+1, put_value((Self,state),Y).
counter(([down],Self),Self) :- !,
        get_value((Self,state),X),
        Y is X-1, put_value((Self,state),Y).
counter1(([show],Self),X) :- !,
        get_value((Self,state),X).
counter1(([up(N)],Self),Self) :- !,
        get_value((Self,state),X),
        super(Self,[plus(X,N)],Z),
        put_value((Self,state),Z).
counter1(([down(N)],Self),Self) :- !,
        get_value((Self,state),X),
        (Self <- ([minus(X,N)],Z)),
        put_value((Self,state),Z).

class(compute_counter1) <=
        ( super([object]) ;
          metaciass([class]) ;
          classvar([(reset,0)]) ;
          method([plus,minus])  ).
plus(([plus(N1,N2)],Self),Ret) :- !,
        Ret is N1+N2.
minus(([minus(N1,N2)],Self),Ret) :- !,
        Ret is N1-N2.

class(compute_counter2) <=
        ( super([object]) ;
          metaclass([class]) ;
          instvar([x,y,z]) ;
          method([times]) ).
times(([mult(X*Y)],Self),Z) :- !,
        put_value((Self,x),X),
        put_value((Self,y),Y),
        Z is X*Y,
        put_value((Self,z),[X,Y,Z]).
```

∞

▼　むすび　▼

知識表現言語の開発に対して、２つのアプローチを試みた。ひとつは、対象世界の知識表現で、フレームを出発点にしており、ＡＩの問題として解決しやすい。もうひとつは、オブジェクト指向からのアプローチで、実際の知識表現における意味をどのようにオブジェクトに反映させるかなど、今後検討していかなければならない問題を有している。

▼ 文献 ▼

1. MIZOGUCHI,F.,: PROLOG Based Expert System, New Generation Computing, 1(1983) 99-104

2. Newell, A.,: A Theoretical exploration of Mechanisms for Coding the Stimulus, in Helton & Martin (Eds.), CODING Process in Human Memory, John Wiley & Sons (1972)

3. Norman, D.A., & Bobrow, D. G., On the Role of Active Memory Processes in Perception and Cognition, In C. N. Cofer (Ed.), The Structure of Human Memory, San Francisco : Freeman, 1976

4. 片山、溝口、 PROLOGによる知識表現の検討、情報処理学会第２６回全国大会、pp1065-1067, 1983

5. 片山、溝口、 PROLOGによる知識表現：LOOKS, 情報処理学会全国大会 （準備中）

6. Bobrow, D, G., Stefik, M., The Loops Manual, KBVLSI memo, 1983

7. 本田、溝口研打ち合せ資料 1983

8. その他

   溝口、 人工知能における知識表現、数理科学、6. 1983

9. Pereira, F., C-Prolog User's Manual, Version 1.2a, EdCAAD, 1983

▼ PROLOGによるエキスパートシステムの展開について：

・ 推論システム

Rutgers: EXPERT system
- 対象
  - 眼科： 緑内障
  - 整形外科： ムチウチ症
  - 故障診断：原子炉

ADIPS: INTER-LISP
- 故障診断

File translator

APLICOT テスト中

Forward/Backward Reasoning system

・ 知識表現システム

KRS (Knowledge Representation system)
- INTER-LISP
- Frame-base
- Program ドキュメント

東大大型センター
- UTI-LISP

オブジェクト指向型言語

Looks & Pools 進行中

・ 自然言語システム

LINGOL
- HAC-LISP
- INTER-LISP
- UTI-LISP 進行中

(DCG + Slot Grammar +...)

エクスパートシステムのための複合型知識表現とグラフィックス機能の実現

桑原　和宏　　石塚　満
（東京大学生産技術研究所）

1．複合型知識表現ＦＬＯＲＡ

1はじめに

我々は，１９８０年以来図１に示されるように，建築物被害査定のエクスパートシステムＳＰＥＲＩＬをはじめとしていくつかの知識ベースシステムを開発してきた［１－４］．ＳＰＥＲＩＬには被害査定問題に含まれる不確実性，あいまい性を伴う知識の有効な利用を図るためDEMPSTER & SHAFER理論をファジィ集合への拡張した合理的な推論機構が使用されている［５，６］．

さらに，汎用プロダクションシステムを用いて，２次元物体の輪郭形状の認識システム［７］および日本民謡の旋律構造の解析システム［８］が作成された．これらのシステムにも不確実性の取り扱いが導入されている．

汎用プロダクションシステムは，知識がＩＦ－ＴＨＥＮのルール形式でかかれ，それぞれのルールが独立しているという特長がある反面，ルールの数が増えるにしたがい実行速度の低下を招き，また階層的な知識構造を支援していないなどの欠点が明らかになった．

そこで動的にフレームの生成・消滅を繰り返すことによりボトムアップおよびトップダウン探索を組み合わせた効率のよい探索をおこなうＦＢＳＳ（FRAME-BASED SEARCH SYSTEM）を開発した．ＦＢＳＳを用いて３次元物体の認識システムが作成され成果を収めている［９］．さらにＦＢＳＳに不確実性も取り入れ，エクスパートシステム用に汎用化したフレームシステムＦＩＳ（FRAME-BASED INFERENCE SYSTEM）を作成した［１０］．ＦＩＳには効率のよい推論機構が内在しているけれども知識がプロトタイプフレームという形式のみで記述されるため一連の手続き型知識を表現しにくい欠点がある．エクスパートシステムの対象分野においてある程度手続き的な手法が確立されている場合，そのような手法もシステム内に取り込めるようにした方が望ましい．

そこで種々の知識表現形式を可能とするような知識表現システムＦＬＯＲＡを開発している［１１］．ＦＬＯＲＡにおいて知識は基本的にはＦＩＳにおけるようなプロトタイプフレームの形で表現されるが，オブジェクト指向の考え方を取り入れることにより手続き型の知識も容易に表現できるようにしている．ＦＬＯＲＡを用いて既存鉄筋コンクリート造建

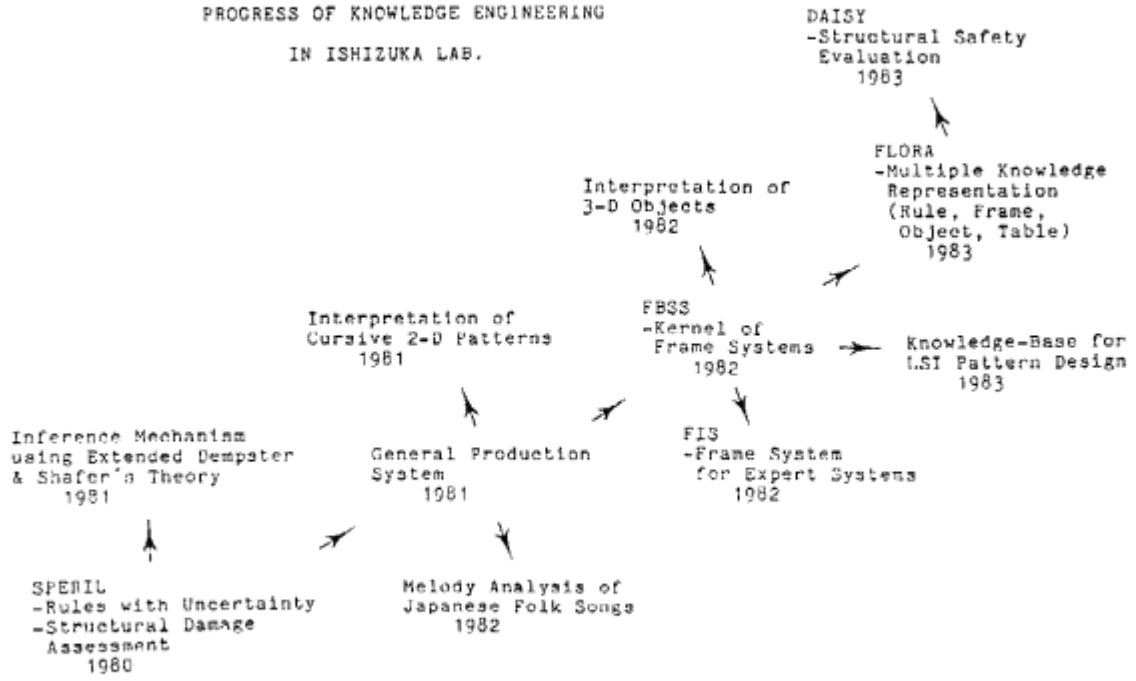


図１　石塚研究室で開発された知識ベースシステム

築物の耐震診断システムＤＡＩＳＹを構築している．

またフレーム型知識を用いてＬＳＩのパターン設計システムを検討中である．

本論文では，ＦＬＯＲＡを中心に述べ，さらに遠隔端末からもアクセス可能な，グラフィックを取り入れたエクスパートシステムについて述べる．

2.ＦＬＯＲＡにおけるオブジェクト

エクスパートシステムでは，問題を階層的にいくつかのサブゴールに分解して表現する場合が多い（図2）．我々が開発をおこなっている耐震診断システムでは，まず耐震指標を求めることが必要となる．耐震指標を求める場合も問題をいくつかのサブゴールに分解して求めていくことができる（図3）．この場合各ゴールの目標は，そのゴールの値を求めることになる．

各ゴールの目標がある値を求めるという点では，同じである．しかし，そのための具体的な方法となると各ゴールそれぞれで異なってくる．たとえば，計算式にしたがって求める，表を参照して求める，ユーザに質問を発して求める，などいろいろな方法が考えられる．

そこで，ＦＬＯＲＡには，オブジェクト指向の考え方を取り入れることにした．まず，各サブゴールをひとつひとつのオブジェクトとして考える．それに対してGETVALUEのメッセージを送ることにより，その各ゴールが値を返すというようにする．このような考え方をとることにより各サブゴールは，その外側に対しては，GETVALUEというメッセージを送ると値を返してくるというオブジェクトにしかみえず，各サブゴールそれぞれの中身は，外側とは独立しているという，利点が生まれる．

3.メソッドの継承

オブジェクト指向の考え方では，各オブジェクトについてメッセージに対するメソッドを規定する必要がある．各オブジェクトごとにひとつひとつメソッドを記述するのでは冗長である．そこで同じメソッドをもつオブジェクトを集め，そこからクラスを構成する．クラスについてもさらに上位のクラスを考えることもできる．オブジェクトがメッセージを受け取ると，それに対するメソッドをまず自分の中からさがし，もしなければ自分の属する上位のクラスの中からさがし，それでもなかったらさらに上位のクラスの中をさがすというように受け取ったメッセージに対するメソッドを求める．

4.ＦＬＯＲＡにおけるメッセージの受け渡し

ＦＬＯＲＡは，PROLOG/KRを用いてインプリメントされている．メッセージの受け渡しは，述語MBOX（図4）がおこなう．メッセージは，RECEIVER，SELECTOR，ARGUMENTからなる．たとえばHEIGHTというオブジェクトにGETVALUEという，メッセージを送る時は，

```
(MBOX HEIGHT GETVALUE ? *ANSWER)
```

を実行させると変数*ANSWERに答えがかえってくる．ここではARGUMENTとしてどんなものともユニファイする

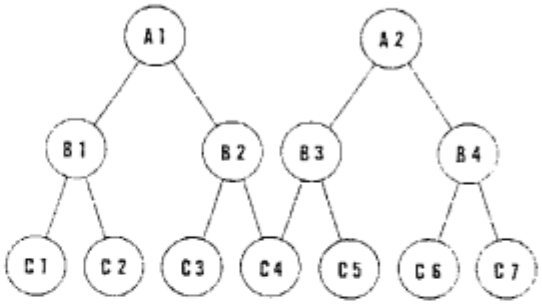


図２　問題の階層的分割

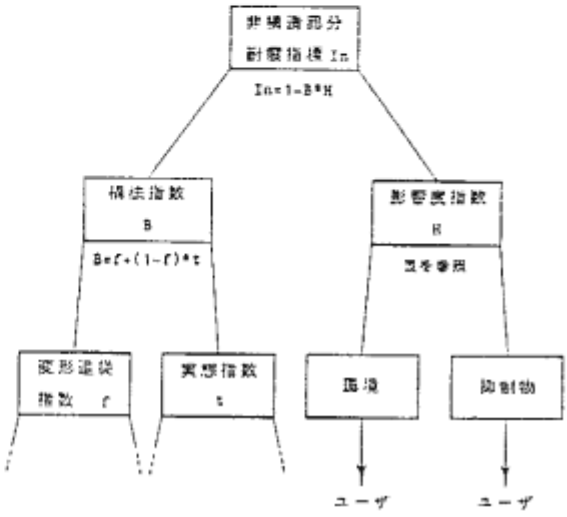


図３　耐震指標算出における問題の分割例

```
; mail box --- manupilate messages...
(as (mbox *receiver *selector *argument *answer)
    (*rec *selector *argument *answer)
    (cut))
(as (mbox *receiver *sel *arg *ans)
    (*receiver type *super)
    (cut)
    (mbox2 (*super) *receiver *sel *arg *ans))
(as (mbox *receiver *sel *arg *ans)
    (error *receiver))
;
(as (mbox2 (*scar . *scdr) *rec *sel *arg *ans)
    (*scar *sel *rec *scdr *arg *ans)
    (cut))
(as (mbox2 (*scar . *scdr) *rec *sel *arg *ans)
    (*scar type *super)
    (cut)
    (cons *super (*scar . *scdr) *supers)
    (mbox2 *supers *rec *sel *arg *ans))
(as (mbox2 (*scar . *scdr) *rec *sel *arg ?)
    (error *scar))
;
```

図４　述語ＭＢＯＸ

変数である？を使っている．MBOXは，まず

（*RECEIVER *SELECTOR *ARGUMENT *ANSWER）

を実行する．もしこれが失敗すれば，次に

（*RECEIVER TYPE *SUPER）

を実行し，そのオブジェクトの属しているクラス*SUPERを求め，述語MBOX2を使ってクラス*SUPERの中に同じようにメソッドを求めていく．

5.ＦＬＯＲＡにおける知識表現

ＦＬＯＲＡにおいては，主として問題をサブゴールに分解して考える．そのサブゴールは，PROLOG/KRにおける述語 に相当している．ＦＬＯＲＡが各サブゴールに要求していることは，次のことである．

「各サブゴールはメッセージに対応したメソッドをもつ
(①) か，または，自分の属しているクラスを定義する
(②) こと」

①については，各サブゴールが

（<SUBGOAL> <SELECTOR> <ARGUMENT> <ANSWER>）

を頭部にもつ節を持つことを意味する．また②は，

（<SUBGOAL> TYPE <SUPER CLASS>）

という節を持つことを意味する．

その他の点については，その上位クラスがその記述形式を決めることになる．

Ⅱ．ＦＬＯＲＡの耐震診断システムへの応用

1.耐震診断システムにおけるクラス

現在のところ耐震指標に関する知識およびそれに必要な上位クラスのインプリメントは，ほぼ終わっている．

ここでは，上位クラスとして次のようなものを用意している．

CONST： 定数

FORMURA： 計算式により求める．

ASK： ユーザに質問を発する．

TABLE： 表形式の知識

CASE： 条件分岐．

VECTOR-ASK： VECTOR型の'ASK'．

VECTOR-FORMULA： VECTOR型の'FORMULA'．

各サブゴールは，この中のどれかに属させる．各サブゴールはFRAME-LIKEな形で表現される．（図5）

2.VECTOR型の導入

耐震指標を求める際に各階ごとに指標を求める必要がある．

```
;
(as (height type ask))
(as (height mes "nankaidate desuka?"))
(as (height constraint (integer positive)))
(as (height type-constraint scalar))
;
(as (area type vector-ask))
(as (area size height))
(as (area mes $index " kai no yukamenseki wo"
                     " oshietekudasai "))
;
(as (eo type case))
(as (eo case (pillar)
             ((1) eo1)
             ((2) eo1)
             ((3) eo2)))
;
(as (eo1 type formula))
(as (eo1 formula times))
(as (eo1 arg (floor owc fw)))
;
(as (floor type formula))
(as (floor formula (*no *io) *x)
    (makefloat *no *n)
    (makefloat *io *i)
    (quotient (! add1 *n *) (! plus *n *i *) *x)))
(as (floor arg (height $index)))
;
```

図５ ＦＬＯＲＡにおける知識表現の例

これに対処するためにVECTOR型の導入をおこなった．すなわち，各階の指標ごとにサブゴールを設定するのではなく，一つの階について指標算出の知識をいれ，それを各階で共通に用いるわけである．VECTOR型は１次元配列であり，INDEXを用いて各階の指標を区別する．ここでは，INDEX$なる述語が現在のVECTOR型におけるINDEXの値をスタック形式に保持している．

3.DATA BASE

現在，インプリメントしてある上位クラスは，値を求めることが目的となっているので，各サブゴールの値が求まるとそれをDATA BASEに記憶させるようにした．したがって，各サブゴールにGETVALUEのメッセージが送られるとまずそのサブゴールの値をDATA BASEの中にさがし，その値があればそれを返し，もしなければ各サブゴールの知識に基づいて値を求めることになる．

DATA BASEは次の４種類の述語に分けて構成する．

①DB-ANS$： SCALAR型のデータのうちユーザからの答えを記憶する．

②DB$： その他のSCALAR型のデータの記憶．

③DB-VEC-ANS$： VECTOR型のデータのうちユーザからの答えを記憶する．

④DB-VEC$： その他のVECTOR型のデータの記憶．

DATA BASEのフォーマットは，次のとおり，いずれも頭部のみの節の形でアサートされる．

（DB-ANS$ <OBJECT-ID> <SUBGOAL> <VALUE>）

（DB$ <OBJECT-ID> <SUBGOAL> <VALUE>）

(DB-VEC-ANS$ <OBJECT-ID> <SUBGOAL> <INDEX> <VALUE>)

(DB-VEC$ <OBJECT-ID> <SUBGOAL> <INDEX> <VALUE>)

ここでOBJECT-IDとは，ARGUMENTとして渡される値のことで，これを指定することにより同じサブゴールについて複数のものを生成することができる．

4.トップレベル

現在の耐震診断システムのトップレベルを図6に示す．ここでは，各オブジェクトにメッセージを送り値を求めている．各指標については，桁行方向とはり間方向の2方向について求める必要があるためARGUMENTの指定をおこなっている．最終的には，IS（構造耐震指標）が0．9以上であれば，安全，それ以下であれば，さらに詳しい調査が必要であるとのメッセージを出して終わる．

Ⅲ．FLORAの拡張

1.探索機構の導入

現時点におけるFLORAには，試行錯誤的に解を求めるようなクラスは，定義されていない．対象分野によっては，探索の過程をプログラマが明らかに指定せずにシステム側に任せた方がよい場合もある．そこでここでは，FIS（FRAME-BASED INFERENCE SYSTEM）の機能を取り入れることを考えている．FISは，問題を階層的にいくつかのサブゴールにわけ，仮説の生成，仮説の検証という過程を通して問題の答えを求めるものである（図7）．これをFLORA上で実現するには，たとえばFISというTYPEのオブジェクトを考え，問題のサブゴールをオブジェクトに対応させる．仮説の生成をおこなうときはその仮説に対応するオブジェクトに対してPRODUCEのメッセージを送るようにし，また，仮説の検証をおこなうときは，その仮説に対応するオブジェクトにVERIFYのメッセージを送るようにすればよい．FIS型のクラスには，それぞれPRODUCE，VERIFYのメッセージに対するメソッドを書いておく．

このようにFLORAには，新たにクラスを定義することにより種々の表現形式を混在させることができる．これが可能なのは，問題をいくつかのサブゴールにわけ，それぞれの間のやりとりはすべてメッセージによることにしたためである．このようにオブジェクト指向の考え方を取り入れたことにより各サブゴール間の独立性が保たれ，また種々の表現形式が可能となっている．

2.問題点

FLORAは東大大型計算機センタM280H上のPRO

```
(as (daisy)
    (init)
    (mbox name getvalue ? *name)
    (mbox totalarea getvalue ? *1)
    (print "hari houkou ni tsuite ukagaimasu")
    (mbox eos getvalue hari *hari)
    (print "keta houkou ni tsuite ukagaimasu")
    (mbox eos getvalue keta *keta)
    (mbox g getvalue ? *g)
    (mbox sd getvalue ? *sd)
    (mbox t getvalue ? *t)
    (mbox is getvalue hari *harians)
    (mbox is getvalue keta *ketaans)
    (append *harians *ketaans *anslist)
    (apply min *anslist *ans)
    (terpri)
    (princ " DAISY assesed that ")
    (princ *name)
    (if (greaterp *ans 0.9)
        (princ " is safe. ")
        (princ " needs further investigation. "))
    (terpri)
    (top))
```

図6　DAISYのトップレベル

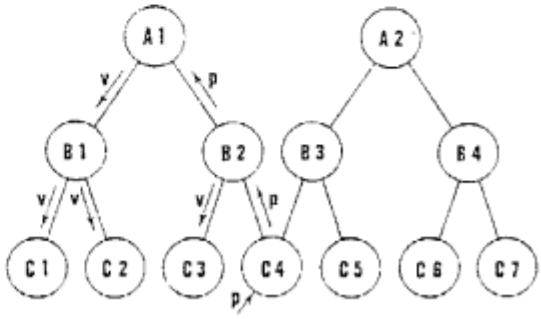


図7　FISの推論過程

LOG/KR上にインプリメントされている．これに耐震指標算出の知識として200行程度入れて走らせると，TSSの標準値である1MBYTEでは，メモリが不足して動かない．組み入れた知識の量は本当に賢いというには程遠いという段階であり，従来型の計算機では，このようにメモリを浪費して真に実用になるものができるかいささか疑問を感じる．新機構の第5世代計算機では解決されるであろうことを期待している．

3.今後の展望

耐震指標算出だけでは，探索的な動作が少なく，わざわざFLORAを用いる必要がないともいえる．今後は，より専門的な知識，経験的な知識を組み入れていく予定である．

Ⅳ．グラフィックの導入

1.図形の取り扱い

エクスパートシステムは，比較的大きな知識ベースを必要とし，またそれらの管理も常時必要となることより，エクスパートシステムは，中央の大型計算機上に実装し，それをTSS環境下においてユーザが共用することが望ましい．最近，携帯用の小型端末が比較的容易に入手することができるので，

電話線を用いてどこからでも目的とするエクスパートシステムを利用することができるようにすることは意義が大きいと思われる．

筆者らが，開発をおこなっている既存建築物の耐震診断のような分野でのユーザとの対話を考えると，通常の文字による対話にくわえて，図形情報も必要となる場合が多い．たとえば，ユーザに対する質問文で，図8に示すような図形をも同時に表示できれば，言葉で長々と説明するよりわかりやすい．しかし，前述のような低速の回線を使ったＴＳＳでは大型計算機で直接図形情報を扱うのでは回線の速度の点で実用的でない．そこで，端末をインテリジェント化し，できるだけ図形処理は端末側でおこなうような分散処理構成をとることにした［１４，１５］（図９）．

2.図形の出力

低速の回線を用いたＴＳＳでは伝送速度が遅いことから，図形を出力するために大型計算機側から基本的なグラフィックコマンドをひとつひとつ送っていては実用にはならない．そこで端末側のグラフィックパッケージにマクロ定義機能を持たせ，主としてマクロを実行させるようにした．

2.1.マクロ定義

マクロ定義の方法として，次の２通りの方法を用意した．

(1)マクロとして実行するコマンド群を定義したマクロ定義ブロック（図１０）を用意し，グラフィックディスプレイへ転送する方法．

(2)マクロ学習機能すなわち，マクロ学習開始コマンドを送り，さらにマクロとして定義するコマンド群を通常の方法で送り，最後にマクロ学習終了コマンドを送るという手順でおこなう方法．グラフィックパッケージ側では，マクロ学習開始コマンドとマクロ学習終了コマンドの間に送られてくるコマンド群からマクロ定義ブロックを作成する．

2.2.マクロの管理

グラフィックパッケージには，ファイルＩＤ，バージョンナンバ，マクロ定義ブロック群などからなるマクロ定義ファイルを持たせ，マクロを番号で管理する．

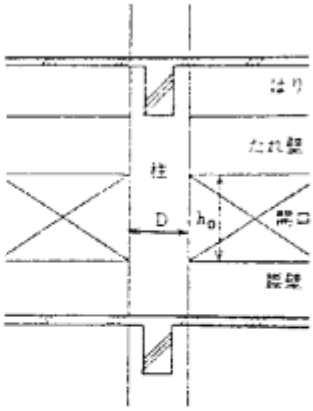


図８　出力図形の例［１６］

2.3.マクロの更新

エクスパートシステムの知識の更新に伴い，端末側のマクロ定義も変更する必要がでてくる．そこでエクスパートシステムの利用開始時に端末側にマクロ定義ファイルのバージョンナンバを問い合わせ，もしそれが最新のものでなければ変更分のマクロ定義のみ大型計算機から送り，端末側でマクロ定義ファイルの更新をおこなうようにする．

3.図形の入力

建築物の耐震診断などの応用を考えると簡単な図形，たとえば建物の平面図などを扱えるようにした方が望ましい．そこでタブレットなどで入力した図形より必要な情報を取り出す（例えば，建物の平面図から床面積を計算するなどの）プログラムを端末側におき，その結果のみを大型計算機の方へ送信するような構成をとる．

Ⅴ．むすび

知識表現としてルール，フレームを用いた知識ベースシステム開発の経験に基づき，現在開発を進めているオブジェクト的な考え方を取り入れた複合型知識表現システムＦＬＯＲＡとそのエクスパートシステムへの具体的な応用である耐震

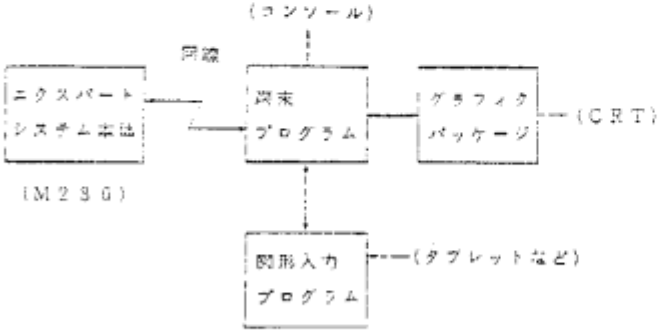


図９　システムの全体構成

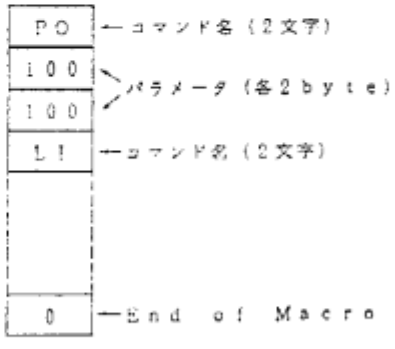


図１０　マクロ定義ブロックの例

診断システムＤＡＩＳＹの概要を紹介した．同時に今後のエクスパートシステムにおいて重要な役割を果たす図形を遠隔端末においても効率的に扱える方式についても記した．

本文中に述べたようにPROLOGによるシステムの実装には従来型の計算機では，たとえ大型機であっても不十分な点があり，第５世代計算機に期待するところが大きい．図形による対話を実現する通信網に関しては，構内については，ＬＡＮにより解決されるであろうが，広域の高速データ通信については，ＩＮＳ網に期待するところが大きい．

REFERENCES

[1] 石塚 ： エクスパートシステム，システムと制御，Vol. 26, No. 10, pp. 624-631, 1982. 10

[2] M.Ishizuka, K.S.Fu, J.T.P.Yao : An Expert System for Damage Assessment of Existing Structures, ICPR, Munich, 1982

[3] ---,---,--- : Rule-Based Damage Assessment System for Existing Structures, Solid Mechanics Archives, 8, pp.99-118, 1983

[4] 石塚 ： 建築物被害査定システムのエクスパートシステム，情報処理学会論文誌，Vol. 24, No. 3, pp. 357-363, 1983. 5

[5] M.Ishizuka, K.S.Fu, J.T.P.Yao : Inference Methods under Uncertainty for the Problem-Reduction Method, to appear in Information Sciences, also Purdue Univ., Civil Eng., CE-STR-81-24 and Elec. Eng., TR-EE 81-33, 1981.08

[6] ---,---,--- : A Rule-Based Inference with Fuzzy Set for Structural Damage Assessment, M.M.Gupta et al. (eds) Approximate Reasoning in Decision Analysis, North-Holland Pub. Co., 1982

[7] M.Ishizuka, M.Numao, Y.Yasuda : A Rule-Based Interpretation of Contour Patterns with Curves, Inter Graphis '83, Tokyo, 1983

[8] 岸井，沼尾，石塚 ： ルールに基づく日本民謡の旋律構造解析，信学会パターン認識と学習研，PRL82-61，同オートマトンと言語研，AL82-73，1982. 12

[9] 沼尾，石塚 ： フレーム型模索システム（FBSS）による３次元図像の解釈，同上PRL82-53，AL82-65，1982. 12

[10] 沼尾，岸井，石塚 ： 動的フレームに基づく推論システムーFIS，第26回情処全大，7C-4，1983. 3

[11] 桑原，石塚 ： 複合型知識表現によるコンサルテーションシステム，第27回情処全大発表予定，1983. 10

[12] Special Issue on Smalltalk, Byte, Vol. 6, No. 8, 1981

[13] D.G.Bobrow, M.Stific : The LOOPS Manual, Xerox PARC, Memo KB-LSI-81-13, 1982

[14] 桑原，石塚 ： エクスパートシステムへの応用を考えたインテリジェント・グラフィック端末，東大生研電気放送会報告，Vol. 33, No. 18, 1983. 6

[15] 桑原，石塚 ： エクスパートシステムの遠隔アクセス用グラフィック端末，信学会情報システム部門全国大会発表予定，1983. 8

[16] 財日本建築防災協会 ： 既存鉄筋コンクリート造建築物の耐震診断基準，1981

知識表現システムＦＬＯＲＡとその応用

桑原　和宏　　石塚　満

（東京大学生産技術研究所）

1.はじめに

コンサルテーションシステムを開発するにあたっては，対象分野の専門家の知識を効率よくかつ容易に表現できるような知識表現システムが望まれる．現在，知識表現システムとしてたとえばプロダクションシステム，フレームシステムなどが開発されている．プロダクションシステムでは，知識は，ＩＦ－ＴＨＥＮルールの形で記述され，知識のモジュラリティが高い反面，構造化された知識が表現しにくい，また推論の制御などの知識が記述しにくいといった欠点がある．一方，フレームシステムでは，構造化された知識を容易に表わすことができ，また宣言的知識，手続き的知識が混在できるなどの特徴を持ち，知識表現システムを考える上で，有効な土台を提供してくれると考えられる．そこで，ここではフレームシステムを基本に考える．

比較的複雑な問題を扱う場合，それをいくつかの部分問題に分割し，部分問題それぞれに応じた知識を表現するのが好ましい．ここでは，細分化された部分問題に関する知識をフレームとして表現し，それらのフレーム間でメッセージのやりとりをおこなうことによって推論を進めていくような構成をとることにした．フレームはメッセージに対して何らかの反応を示すオブジェクトとして考えることができる．フレームにメッセージを送る時，フレームの内部について考える必要がなくなり，その内部構成は他のフレームから独立している．したがって，知識のモジュラリティが高まると同時に種々の形式の知識に対して柔軟に対処できると考えられる．

ここでは，このような考え方に基づいて作成した知識表現システム（これをＦＬＯＲＡと呼んでいる）について述べ，さらにその応用例として既存建築物の耐震診断システムとそれにおける自然言語インタフェースについてふれる．

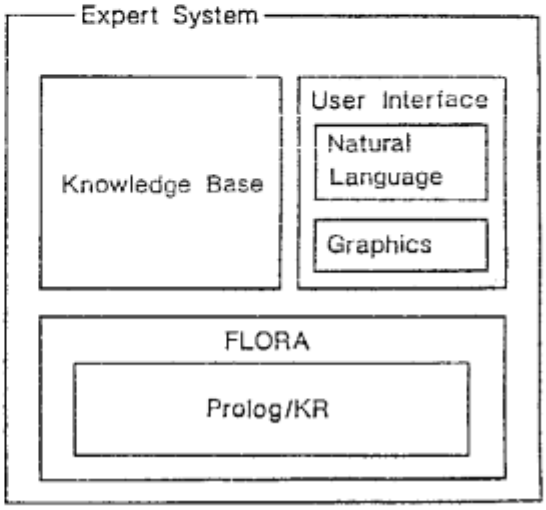


図１　知識表現システムＦＬＯＲＡ

なお，ＦＬＯＲＡはＰＲＯＬＯＧ／ＫＲ［１］を用いてインプリメントした（図１）．

2.フレキシブル知識表現システムＦＬＯＲＡ

2.1 ＦＬＯＲＡにおけるフレーム

ここでは，フレームとして（フレーム，スロット，バリュー）の３つ組を基本としたデータ構造を考えている．これをＰＲＯＬＯＧで表現するため，頭部のみの節で表わす．すなわち，（ＰＲＯＬＯＧ／ＫＲの記法を用いて）

```
(AS (FRAME SLOT1 VALUE1))
(AS (FRAME SLOT2 VALUE2))
      :
      :
```

と表わす．またフレームにはメッセージに対するメソッドを記述する必要がある．これも（フレーム，スロット，バリュー）という3つ組を用いて表現することもできるけれども，スロットの記述とメソッドの記述を明確にわけたいということもあり，（フレームというワク組みからはみだすかもしれないが，）次のような形式で記述することにした．

```
(AS (FRAME
        (MESSAGE-PATTERN)
        *SELF . BODY))
```

ここで，BODYとは，メソッドとして，実行すべき述語である．メソッドの実行に際しては，変数*SELFにメッセージの受け手が渡される．

またMESSAGE-PATTERNはセレクタと0個以上のアーギュメントからなる．PROLOGの論理変数は，入力，出力，どちらにでも使えるので，メッセージに対する返事が必要な時はアーギュメントを利用して返すことにし，特に返事用の変数をシステムとしては用意しないことにした．

2.2 フレームの種類

FLORAにおけるフレームは概念的には，次のように分けることができる．（ここではSMALLTALK［2］における用語を流用している．）

(1)クラス――いくつかのフレームに共通した性質を記述したフレーム．

(2)インスタンス――個々の具体的知識の表現．これには，

①クラスのインスタンスとして動的に生成されるもの．

②あらかじめ与えられた個々の事実．

がある．

FLORAではこのようにフレームは分類されるが，システム内部では，どれも同じように扱っている．

2.3 フレーム間の関係

フレーム間の関係にはTYPEとSUPERという2つの基本的なものがある（図2），TYPEというのは，たとえばインスタンスとクラス間の関係である．同じような性質のインスタンスを集め，1つのクラスを生成しており，インスタンスに対して送られたメッセージに対するメソッドはその属するクラスに記述されている．またクラスについても，同じような性質のものを集め，1つのフレームをつくることができる．各フレームはある一つのクラスに属しており，TYPEスロットにその属しているクラス名を保持する．

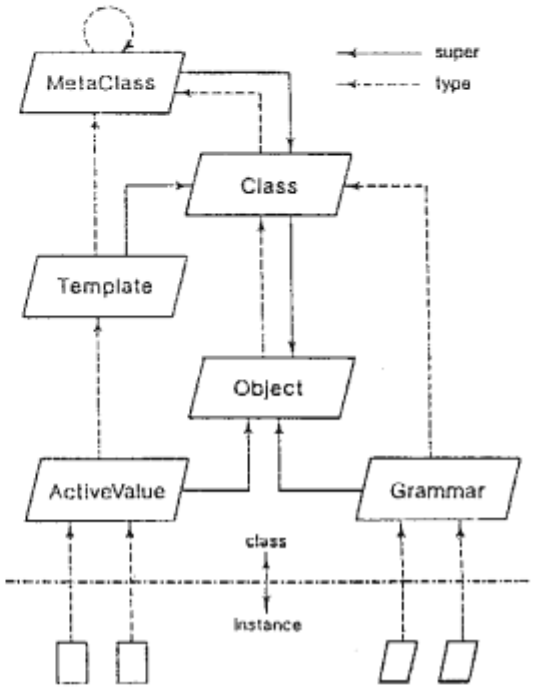


図2　フレーム間の関係

またSUPERという関係は，フレーム間の性質の継承を規定している．そのフレームのSUPERスロットにかかれたフレーム（これをそのフレームのスーパクラスと呼ぶ）の性質ーメソッド，インスタン

ス変数の宣言，クラス変数ーをフレームは継承する．この時，性質を継承するフレームを継承されるフレームのサブクラスと呼ぶ．ただし，同じ名前の変数，同じメッセージパタンをもつメソッドについては継承はおこなわない．また，この性質の継承は単に自分のSUPERスロットにかかれたクラスのみならず，そのクラスのスーパクラスというようにスーパクラスのチェインをたどって継承がおこなわれる．また多重継承を許しており，複数のフレームからの性質を継承できる．

2.4.インスタンス変数，クラス変数

SMALLTALKなどでは，オブジェクトの状態を保持するものとしてインスタンス変数，クラス変数を設けている．インスタンス変数はそのインスタンスに固有の情報を保持するものであり，クラス変数はそのクラスに属するすべてのインスタンスで共有する情報を保持するものである．

FLORAではフレームが基本になっているという立場からインスタンス変数はそのフレームのスロットに対応させ，またクラス変数はその属するクラスのフレームのスロットに対応させることにした．

2.5.メッセージのやりとり

メッセージのやりとりは，基本的には次の述語を用いておこなう．

(¥SEND *RECEIVER
. *MESSAGE-PATTERN)

変数*RECEIVERには，メッセージのあて先を入れる．

また，メソッドのサーチは次のような方針をとることにした．まずメッセージの受け手のTYPEスロットにかかれたフレームのメソッドをサーチし，そのメッセージパタンとマッチするものがあればそれを実行し，もしなければそのフレームのSUPERスロットにかかれたフレームについてメソッドのサーチをおこなう．また，一度メソッドの実行が開始されるとその実行が失敗しても他のフレームにさらにメソッドをサーチすることはしないようにした．ただし，同じフレーム内ではマッチする他のメソッド記述があればその実行はおこなう．このようにしたのは次のような理由からである．

(1)スーパクラスからメソッドを継承するにあたって自分のクラスに同じメッセージパタンのメソッドがあればそれについてはスーパクラスから継承しないという方針をとっており，それを完全にしたい．

(2)PROLOGでは，バックトラックが制御の一つになっておりメソッドの実行が失敗する度に一番大もとのクラスまでメソッドをサーチしていては効率が悪い．

さらにこの¥SEND述語の他に¥SEND-TO，¥SEND-SUPERなる述語を用意している．¥SEND-TOはメソッドのサーチの際にそのスーパクラス群については，サーチをおこなわないもので，たとえばメソッドの継承を無効にしたい時，あるいは，継承の必要がない時に用いる．また¥SEND-SUPERはそのサーチを始めるクラスを指定するもので指定されたフレームのスーパクラスからサーチを始める．

また，直接メソッドの書いてあるクラスを指定してメソッドの実行をおこなう述語としてDO-METHOD（メソッドの継承をおこなう），DO-METHOD-AT（メソッドの継承をおこなわない）などの述語を用意している．

2.6.データ指向プログラミングの実現

データ指向プログラミングはオブジェクトの変数（ここではフレームのスロットに対応）のアクセスにともなって，あるプログラム（述語）が実行されるという考え方を基本としている．このデータ指向プログラミングによりあるプロセスのモニタが容易になるなどの利点が生まれる．また一種のデモンとも考えられる．LOOPS［3］ではACTIVEVA

LUEというものを導入してこれを実現している．ここでは，基本的な考え方をできるだけ統一するために，オブジェクト指向のワク組みで同じような機能を実現することにした．

FLORAではACTIVEVALUEという一つのクラスを設け，変数のアクセスにともないあるプログラムの実行が必要な変数（スロット）については，単なる値ではなく，クラスACTIVEVALUEのインスタンスを代入する．変数に対して値のアクセスのメッセージ－たとえばGETVALUE，PUTVALUE－が来た場合，その変数がクラスACTIVEVALUEのインスタンスであれば，それへメッセージを転送する．そのインスタンスには，アクセスにともなって実行されるべきプログラムについてかいてあり，そのプログラムが起動されることになる．

また，この考え方を使ってデフォルト値，また付加手続きなども実現できる．たとえばデフォルト値を扱うには，GETVALUEに対して実行されるプログラム（GETFN）として，「すでにその変数の値が決まっている（束縛されている）ならばその値を返し，そうでなければ，デフォルト値を返す．」というものにすればよい．またデフォルト値を指定するためにいちいちGETFNを前に述べたように指定するのがわずらわしいときは，新たにたとえばUSEDEFAULTというクラスをクラスACTIVEVALUEのサブクラスとして設け，その変数をUSEDEFAULTのインスタンスとして実現すればよい．USEDEFAULTではインスタンス生成の際，デフォルト値を指定する．そしてGETVALUEというメッセージに対してはそのデフォルト値を参照するようなメソッドをもち，そのほかのメソッド－たとえばPUTVALUEなど－についてはACTIVEVALUEのものを継承する．

このようにすればデータ指向プログラミングもオブジェクト指向のワク組の中で実現できる．またその方がアクセスにともない起動されるプログラムの定義についてもクラス間にインヘリタンスを用いることができ，より容易な記述が可能である．

2.7 スロットの値の書きかえ

PROLOGの制御機構の重要なものとしてバックトラックがある．バックトラックで問題となるのは，ASSERT，RETRACTなどのような副作用をもつ述語についてはバックトラックが起こってもその副作用はもとに戻らないことである．ここではスロットは頭部のみの節で保持しており，スロットの値を書きかえるのに基本的にはRETRACT，ASSERTを使うことになり，そのままでは一度スロットの値を書きかえるとバックトラックが起こっても，もとの値に戻らないことになる．応用によっては，一度値を書きかえたら，バックトラックが起きても，もとに戻さなくてもいいという場合があるかもしれないが，ここでは基本的にはバックトラック時には書きかえたスロットの値はもとに戻すという方針をとった．実際のインプリメンテーションでは効率を上げるためスロットの値の書きかえ用の述語（DEFINE-SLOT）をLISPで定義している．

2.8 フレーム定義の例

ごく簡単なフレームの例としてクラスBOXの定義を図3に示す．BOXはグラフィック画面上に出力されるBOXを表わしている．BOXはクラスSHAPE（図4）のサブクラスとして定義されている．SHAPEはBOXなどのようなグラフィック画面上に出力されるオブジェクトに共通なもの－インスタンス変数として座標X，Y，傾きTILT，色COLOR，またメソッドとして指定の座標へ動かすMOVETO，指定の傾きにするTURNTOなどのメッセージに対するメソッドを持っている．BOXではSHAPEのインスタンス変数，メソッドを継承するとともにさらにインスタンス変数WIDTH，HEIGHTを持ち，メッセージDRAW，ERASEなどに対す

```
(as (box type template))
(as (box super (shape)))
(as (box instvar
         ((width 10)
          (height 10))))
;
(as (box (draw) *self
     (¥gr *self
          ("PC" @color)
          ("TT" @tilt)
          ("PO" @x @y)
          ("BX" @width @height))))
(as (box (erase) *self
     (¥gr *self
          ("PC" @@background)
          ("TT" @tilt)
          ("PO" @x @y)
          ("BX" @width @height))))
```

図3　BOXの定義（一部）

```
(as (shape type template))
(as (shape super (object)))
(as (shape instvar
           ((x 0)
            (y 0)
            (tilt 0)
            (color 10))))
(as (shape background 0))
;
(as (shape (turnto *tilt) *self
     (¥send *self erase)
     (¥send *self
            putvalue tilt *tilt)
     (¥send *self draw)))
(as (shape (moveto *x *y) *self
     (¥send *self erase)
     (¥send *self putvalue x *x)
     (¥send *self putvalue y *y)
     (¥send *self draw)))
```

図4　SHAPEの定義（一部）

るメソッドを持つ．

BOXの新しいインスタンスBOX1を生成するには，

(¥SEND BOX NEW BOX1)

を実行する．またBOX1を表示するには，

(¥SEND BOX1 DRAW)

を実行すればよい．

なお，クラス定義における¥GR述語は，FLORA組み込みのグラフィック出力用の述語である．

3 FLORAの応用例(1)ー耐震診断ー

3.1 耐震診断

既存建築物の耐震診断については，まず耐震指標を算出し，それをもとに総合的に建築物の耐震性の判定をおこなうという方法が提案されている．耐震指標の算出については具体的な計算法が示されており［4］，耐震指標算出だけならば，わざわざ知識表現システムを持ち出さずともFORTRANなどで十分記述できる．しかし，自然言語インタフェースを用いたユーザとの対話的なコンサルテーション，また最終的な判定における専門家の定式化されていない知識の利用を考えると，耐震指標についてもその算出法を知識表現システム上で記述できるようにした方がよいと考えられる．そこで，ここでは耐震指標算出への応用について述べる．

3.2 インプリメンテーション

まず，TATEMONOというクラスを一つ作り，一つの建物に対してそのクラスのインスタンスを作る．耐震指標算出に必要なデータ，たとえば床面積などはそのインスタンス変数として保持する．算出法については，メソッドとして記述しておく．

3.3 クラスASKUSER

自然言語インタフェースを用い，主導権混在型のコンサルテーションシステムの実現を目的としており，そのためユーザが値を入れるような変数についてはユーザとの対話を考えた機能を持たせる必要が

ある．ここでは次のようなものを考えている．

(1)GETVALUEのメッセージに対して：もし，すでに値が決まっていれば，その値を返す．そうでなければユーザに質問を発し，値を求め，その値を返すとともに記憶しておく．

(2)PUTVALUEのメッセージに対して：もし，いまだ値が決まっていなければ値を書きこむ．そうでなければ値の書きかえをおこなう前にユーザに確認を求める．

このような機能を実現するためクラスACTIVE VALUEのサブクラスとしてASKUSERというクラスを設けた．ユーザが値を入れるような変数は，このASKUSERというクラスのインスタンスとして実現される．

3.4 算術式の表現

UTILISP [5] にはINFANT (INFIX NOTATION FACILITY) と呼ばれるユーティリティがあり，INFIX NOTATIONで書かれた算術式を評価できる．

FLORAでは算術式の記述を容易にするためこの機能を導入している．さらに@をつけてインスタンス変数，また@@をつけてクラス変数を表わせるようにした．この機能は¥¥述語を用いて呼び出す．例を次に示す．

```
(¥¥  *SELF
     (*RETURN-VALUE  :=
      0.1  ¥*  @VAR1
           //  @@VAR2))
```

これは，インスタンス変数VAR1の値に0.1を乗じ，それをクラス変数VAR2の値で割った結果を変数*RETURN-VALUEに束縛することを表わしている．この述語の呼び出しに当たっては，変数*SELFに現在のメッセージの受け手を入れておく．

4. FLORAの応用例(2)－自然言語の構文解析－

4.1 構文解析

FLORAの応用例として自然言語の構文解析について述べる．これはDCG (DEFINITE CLAUSE GRAMMAR) [6] をもとにしている．DCGは一階述語論理の節形式で文法を表わし，文脈自由文法の自然な拡張となっている．またPROLOGとの相性がよい．DCGがトップーダウンの解析をおこなうのに対して，ここで述べるものはボトムーアップとトップーダウンを組み合わせた探索をおこなう．

4.2 FLORAにおける実現

FLORAでまずGRAMMARというクラスを設け，各非終端記号をそれぞれGRAMMARのインスタンスとする．クラスGRAMMARの各インスタンスは次のようなスロットを持つ．

(1)TYPE－－GRAMMARという値がはいる．(クラスGRAMMARのインスタンスであることを示す．)

(2)EO－－そのフレーム（非終端記号）が導出規則の右辺にあるような導出規則の左辺にくるフレームのリスト．もしそのフレームが最終的なゴール（ゴールフレームと呼ぶ）であればEOスロットはない．

(3)COND－－導出規則．（複数可）

(4)PARSE－－探索の戦略の指定．トップーダウンまたは，ボトムーアップ．デフォルトはトップーダウン．

インスタンスの例と対応するDCG記述を図5に示す．FLORAでは便宜上クラスGRAMMARのインスタンス名は%で始まる．

クラスGRAMMARには次のようなメッセージに対するメソッドが規定されている．

(1)PARSE-DOWN－－メッセージの受け手を根とする構文解析木を生成する．

```
(as (%subj type grammar))
(as (%subj eo (%s)))
(as (%subj cond
          ((*x)
           (%np *x)
           (%p subj))))
(as (%subj cond
          ((pron)
           (%pron)(%p subj))))
```

(インスタンスの定義)

```
subj(X) --> np(X),p(subj).
subj(pron) --> pron,p(subj).
```

(DCGによる記述)

図5　文法の記述例

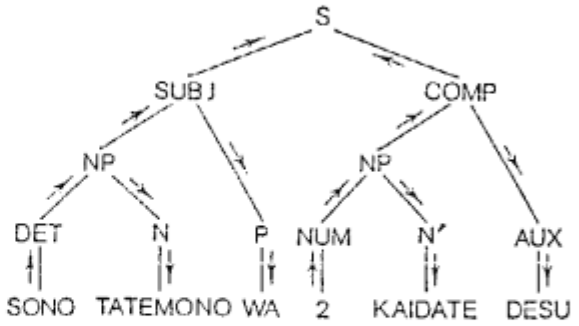


図6　探索の例

(2)PARSE-UPTO――与えられた非終端記号を根として，与えられた非終端記号　から出発する構文解析木を生成する．

(3)PARSE-START――与えられた非終端記号を根とする構文解析木を生成する．単語レベルからの探索に用いる．

(4)PARSE-UPTO?――与えられた非終端記号から出発し，任意のゴールフレームにいたる構文解析木を作成する．

(5)PARSE-START?――単語レベルから始めて任意のゴールフレームにいたる構文解析木を生成する．

簡単な探索の例を図6に示す．探索はまず最初の単語のカテゴリを辞書引きにより求め，そのカテゴリに対応するフレームにPARSE-STARTまたはPARSE-START?のメッセージを送ることにより始まる．この考え方を使って，分かち書きローマ字日本文を対象とした，副腹診断のコンサルテーションシステムの自然言語インタフェースを作成している．

5.おわりに

フレームにメッセージパッシングの考え方を導入したフレキシブル知識表現システムFLORAとその応用例について述べた．FLORAはPROLOG/KRを用いてかかれ，できるだけバックトラック，パタンマッチなどのPROLOGの特徴をいかす方向でシステムの作成を試みた．またPROLOGでシステムを構成し，その上で知識を表現するというよりむしろPROLOGにいくつかの有用な述語を提供したという形をとり，通常のPROLOGプログラムとのインタフェースが容易かつ自然におこなうことができる．

参考文献
1)Nakashima, H., "Prolog/KR User's Manual", University of Tokyo, 1983.
2)Goldberg, A., Robson, D., "SMALLTALK-80 : The Language and its Implementation", Addison Wesley, 1983.
3)Bobrow, D.G., Stefik, M., "The LOOPS Manual", Memo KB-VLSI-81-13, Xerox PARC, 1983.
4)日本建築防災協会, "既存鉄筋コンクリート造建築物の耐震診断基準", 1981.
5)Chikayama, T., "Utilisp Manual", University of Tokyo, 1981.
6)Pereira, F., Warren, D., "Definite Clause Grammar for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks", Artificial Intelligence, 13, pp.231-278, 1980.

知識ベースに基づく Language-oriented Editorの設計

東京理科大学理工学部情報科学科　　武田　正之

1．はじめに

ソフトウェア開発段階における仕様記述からプログラム作成への過程では、種々の知識 — 例えば、ソフトウェア作成方法論、プログラム言語の構文・意味規則、プログラミング技法、最適化手法等 — が必要とされる。これらの知識が計算機で処理できる程度に形式化されるならば、自然言語による表現のあいまい性が除去できると共に、プログラミング支援システムの実現も可能となり、ソフトウェアの生産性を高めることが期待される。

本報告では、ソフトウェア仕様記述からプログラムを合成する段階における、プログラム編集、静的意味解析、エラー診断・回復支援等の機能を持つ会話型エディタを考察対象とし、知識ベースを導入することで異なる言語への適用性、ユーザモデルの考慮によるシステム柔軟性の実現可能性を検討する。第２章では、プログラムの言語の構文や意味を考慮した編集機能を持つ会話型エディタの要求仕様を検討する。第３章ではその実現に必要な機能を抽出し、Prologによるインプリメント可能性を示す。最後に、このようにして実現されるエディタシステムの特徴を考察する。

2．エディタに対する要求仕様

Language-oriented Editorでは対象言語特有の意味やエラー診断手法を知識として備えておく必要がある。またそれらの知識を駆使して柔軟性のあるシステムを実現するためにいくつかの制御機構が要求される。以下ではこの知識と制御機構を中心にしてエディタシステムの構成を検討してみる。

2．1　知　識

エディタシステムには次のような知識が要求される。

(1) 対象言語の構文・意味知識
　FKB (Forward Knowledge Base)
(2) エラー条件の否定的知識
　NKB (Negative KB)
(3) エラー診断・回復処理知識
　BKB (Backtrace KB)
(4) エラー状態の知識
　EKB (Error KB)
(5) ユーザモデルの知識

(1) 対象言語の構文・意味知識　FKB

FKB は編集対象言語の構文や静的意味規則（即ち、コンパイラでチェック可能な範囲）から構成され、対象言語が与えられれば明確に表現される知識である。知識の定義方法はプログラム言語の形式的定義手法[3,4,6]に依存するが、ここではKnuth の属性文法[2]をその基礎とする。その理由は以下の点にある。(a) 属性文法は言語の構文(BNF) に意味規則を付加して拡張した文脈自由文法であり、知識が言語の生成規則(Production Rule) 毎に独立に表現されるため、これら知識のモジュラリティ、拡張性が高く保たれる。(b) 属性文法記述からPrologの一形態であるDCG(Definite Clause Grammar)[5] 記述へは機械的に変換可能であり、意味定義に用いられる属性値の評価をunification 機構により実現でき、実行効率の向上が期待できる。(c) 属性はその値の受け渡し方向に応じて相続及び合成属性の２種類に分類される。この評価方向の情報をエラー診断・修正候補の推論に利用できる。属性文法による言語記述の例と、それをPrologへ変換した例を図１に示す。

(2) エラー条件の否定的知識　NKB

NKB はエラー発生条件を表現している。FKB に従った構文・意味解析の実行中にNKB で指定された禁止条件が成り立つとエラーが発生する。NKB はエラーチェックレベルに対応して階層を形成しており、ユーザの指定に応じて取り出す階層の範囲を変化させることで条件チェックの強さの制御を可能にしている。図１(a) 中error-condition 部がエラー発生条件を表現している。

(3) エラー診断・回復処理知識　BKB

NKB の条件チェックによりエラーが発生した場合、その原因を調査し回復処理を施す必要がある。ユーザによるエラー診断・回復処理を支援するためには、エラー発生時点の環境の表示、エラー原因の候補とその理由の提示等の機能が必要となる。BKB にはこのような機能を実現するための知識が含まれており、

```
<block>          ::= <var-decl>↑env1
                     <statement>↑env2;
        env2 := env1.

<var-decl>↑env   ::= <id>↑name  ':'  <typ>↑type;
        env := cons(name,type).

<statement>↑env ::= <id>↑name  '='  <num>↑type;
  error-condition
    if gettype(env,name)<>type
    then type-conflict-error.
```

(a) 属性文法記述

```
block(*s0,*s) :- var-decl(*env,*s0,*s1),
                 statement(*env,*s1,*s).

var-decl([*name,*type],*s0,*s) :-
                 id(*name,*s0,*s1),
                 terminal(':',*s1,*s2),
                 typ(*type,*s2,*s).

statement(*env,*s0,*s) :-
                 id(*name,*s0,*s1),
                 terminal('=',*s1,*s2),
                 num(*type2,*s2,*s),
                 gettype(*env,*name,*type1),
                 equal(*type1,*type2).

terminal(*token,[*token,*s],*s).
```

(b) Prolog記述

図1．属性文法による構文・意味規則
の記述例とPrologへの変換

エラーチェックレベルに応じていくつかの階層を形成することもある。

(4) エラー状態の知識 EKB

EKB にはエラー状態(warning, fatal 等)、エラー番号に対応したメッセージ等の知識が含まれており、ユーザに応じた適切なエラーメッセージの表示や処理を実現できるよう考慮している。

(5) ユーザモデルの知識

ユーザの能力（novice, expert等）に応じた支援や、誤り傾向の統計処理によるエラー診断方法の改善等の機能を実現するためには、ユーザモデルを導入する必要がある。心理学的検討がこの知識の質を決定するであろう。

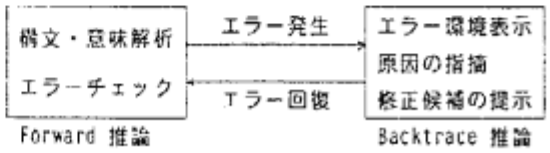
2．2 制御機構

上述の知識群を活用し柔軟性のあるシステムを実現するためには次のような制御機構が要求される。

- (1) Forward/Backtrace 推論機構
- (2) 知識の動的階層化機構
- (3) 協調型問題解決機構

(1) Forward/Backtrace 推論機構

エディタシステムは通常、ソーステキストの構文・意味解析を行いながら前向きにその処理を進めるが、エラーが発生するとユーザの応答に従って後もどりしながらエラー回復処理を行う。エラーの回復が終了するか又はエラーチェックレベルが低く設定されると処理は再び前向きに進行する。このようにシステムはユーザの編集作業に応じて前向き／後向きの処理を繰り返していく。エラー原因の指摘機能は、属性評価方向の情報を積極的に利用し、エラー発生の直接的原因となった属性から依存関係を逆にたどることにより実現される。またその過程においては、エラー時の環境を表示したり、診断理由の説明を行うようなユーザ指向のシステムが望まれる。この機能はいわゆるデバッグコンサルテーションと考えられる。



(2) 知識の動的階層化機構

エラーチェックレベルに応じた条件のチェックやヘルプレベルに応じた説明機能の実現には、知識の動的階層化と階層的知識における知識遺伝機構が必要である。この機構を利用すると、ある知識の意味を一時的に変更することができ、エディタの標準的な機能を変えることも可能となる。例えば、エラーメッセージのコンソール表示をワークファイルへの蓄積に変更したり、キーボードからのテキスト入力をファイルから行うように切換えること等が考えられる。

(3) 協調型問題解決機構

ユーザとの入出力（キーボードからの入力、コンソールへの表示）やファイル処理ルーチン等とエディタシステムの本体とはコルーチンの関係にあることがわかる。これらのルーチンは抽象データ型又はオブジェクトとしてとらえられ、メッセージのやりとりによりお互いの処理が進行していくとみなすことができる。これは Smalltalk 等のいわゆるオブジェクト指向言語の考え方であり、その実現にConcurrent Prolog を利用すれば記述の簡潔性、整合性、さらに効率の向上も期待できる。

以上の要求を考慮したエディタシステムの構成を図2に示す。

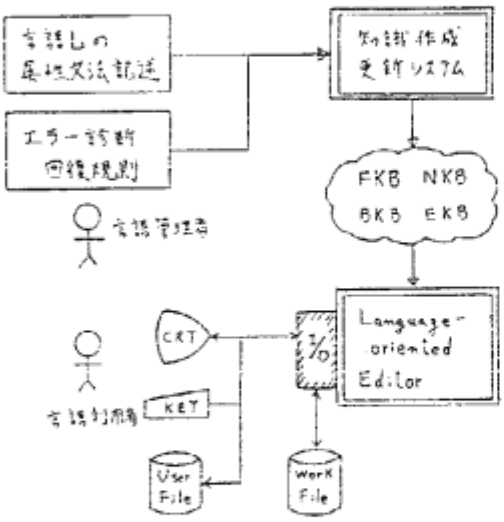


図2．エディタシステムの構成

```
+(demo *KB :- NIL)-(!):
+(demo *KB :- (*P.*Q))-(!)
        -(demo *KB *P)
        -(demo *KB :- *Q):
+(demo *KB *P)-(*KB *P :- *Q)
              -(demo *KB :- *Q):
+(demo *KB (*P.*A))-(*P.*A):
```

図3．demo述語

```
+(KB (append NIL *X *X) :- NIL):
+(KB (append (*E.*X) *Y (*E.*Z))
     :- ((append *X *Y *Z))):

-(demo KB (append (A B) (C) (A B C))):
success
-(demo KB :- ((append (A B) (C D) *X)
              (print *X))):
(A B C D)
success
```

図4．appendの定義と実行例

3．実現に必要な機能のPrologによるインプリメント

2.2節で示した制御機構(1) 及び(2) は、対象言語上の知識（対象知識）とメタ言語上の知識（メタ知識）を1つの言語上で取扱うことにより容易に実現できる[1]。

3．1 対象知識とメタ知識の融合

対象知識とメタ知識の融合をPrologにより実現する方法が提案されている[7]。この節ではその概略を示し、次に制御機構実現のための拡張について述べる。なお以下で用いるPrologの表記法はMarseille Prologに準拠している。

図3に示すProlog述語 "demo" は、与えられた知識ベースのもとでゴールが証明可能であるか否かを判定するメタ述語である。対象知識は次のような形式で表現されているものとする。

ファクト型知識

```
+(KB P :- NIL):
```

ルール型知識

```
+(KB P :- (Q1 Q2 ... Qn)):
```

図3における第1，2番目の節は、bodyの展開手順を与えている。ここで(!) はカットオペレータを意味する。第3番目の節は知識によるゴール証明部であり、ファクトとルールを用いた証明を行う。第4番目の節は組み込み述語による証明部である。

図4には、append述語を対象知識とした例を示す。

3．2 demo述語の拡張

demo述語を拡張することにより、以下に示すような種々の機能を実現することができる。

(1) 証明過程のトレース機能
(2) 多重世界の実現
(3) 知識の動的断片化

(1) 証明過程のトレース機能

demo述語は一種のインタープリターであり、証明過程においてどの知識（ファクト、ルール）が適用されたかをトレースする機能を容易に実現できる。例えば、図5に示すように知識にルール番号情報を付加し、demo述語にルール番号引数を付加すると、証明が終了した時点でその証明過程を表現したルール番号の木が得られる(bottom-upトレース)。また、ルール番号情報をleft- to-right, depth-first に伝搬することで、証明途中で履歴の木を参照できる(top-down トレース)。後者のトレース機能は編集時の構文環境表示やヘルプ機能実現に利用できる。

```
+(KB RULE1 (append NIL *X *X) :- NIL):
+(KB RULE2 (append (*E.*X) *Y (*E.*Z))
           :- ((append *X *Y *Z))):

+(demo *KB NIL :- NIL)-(!):
+(demo *KB (*R1.*R2) :- (*P.*Q))-(!)
        -(demo *KB *R1 *P)
        -(demo *KB *R2 :- *Q):
+(demo *KB (*R1.*R2) *P)
        -(*KB *R1 *P :- *Q)
        -(demo *KB *R2 :- *Q):
+(demo *KB NIL (*P.*A))-(*P.*A):
```

図5．トレース機能の実現

```
+(demo *KB *P)-(getrule *KB (*P :- *Q))
               -(demo *KB :- *Q):

-(getrule NIL *CL)-(!)-(fail):
+(getrule *KB (*P :- *Q))
        -(atom *KB)-(!)-(*KB *P :- *Q):
+(getrule (*KB1.*KB2) *CL)
        -(getrule *KB1 *CL):
+(getrule (*KB1.*KB2) *CL)
        -(getrule *KB2 *CL):
```

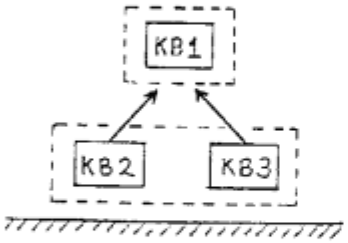
図6．Multiple Inheritance demo 述語

(2) 多重世界の実現

demo述語の知識引数に木構造を許すと多重世界を実現することができる。demo述語の変更は、知識によるゴール証明部である（図6）。

getrule の第1番目の節は、知識が空ならば知識によるゴール証明は失敗となることを表現している。第2,3,4 番目の節により、木構造で与えられる知識をleft-to-right, depth-firstに探索する。この機構により、いわゆる知識のMultipl Inheritance が実現される。

```
-(demo (KB1 (KB2 KB3)) .... ):
```



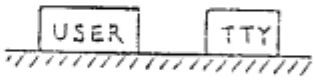
(3) 知識の動的階層化

demo述語の知識引数を証明過程時に変更することにより、知識の動的構成が可能となる。例えば、demo述語に次に示す節を加えることで、動的階層化機構を提供できる。

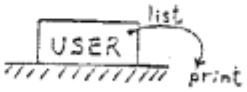
```
+(demo *KB (WITH *NEWKB :- *Q))-(!)
        -(demo (*NEWKB *KB) :- *Q):
```

このWITH述語は、PROLOG/KR におけるWITH述語と同様の知識世界を実現している。動的な世界階層の応用例を以下に示す。

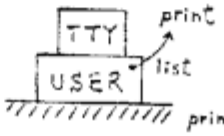
```
+(TTY (print *X) :- ((put-tty *X))):
+(USER (list *X) :- ((print *X))):
```




```
-(demo USER (list DATA)):
```




```
-(demo USER (WITH TTY :- ((list DATA)))):
```



WITH述語によりユーザの知識（プログラム）を変えることなく、特定の述語(print) の機能を変更することができ、入出力デバイスを一時的に切換えてデバックに利用すること等の応用が可能となる。

3．3 Prologによるエディタ制御機構のインプリメント

以下では前節で述べた拡張demo述語を用いて、エディタの制御機構、特にForward/Backtrace 推論機構のPrologによるインプリメントを提案する。

エディタの制御機構と知識の関係を図7に示す。

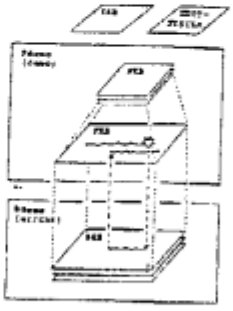


図7．エディタ制御機構と知識の関係

Forward 推論(Fdemo) とBacktrace 推論(Bdemo) の切換えは各々エラー発生とその回復時になされる。EDIT-SYSTEM なる知識は、エディタシステムの組み込み述語（例えば、標準入出力ルーチン、標準エラー処理ルーチン等）から成り、EKB と共にFdemo , Bdemo の共通知識となっている。

Forward 推論では、FKB 知識を中心に前向き処理が進行し、その過程でエラーチェックレベルに応じたNKB 中の禁止条件がチェックされる。禁止条件が成立するとエラーが発生し、Backtrace 推論に制御が移る。Back- trace 推論では、発生したエラーの種類とエラーチェックレベルに応じたBKB 知識に基づきエラー診断・回復処理の支援を行う。

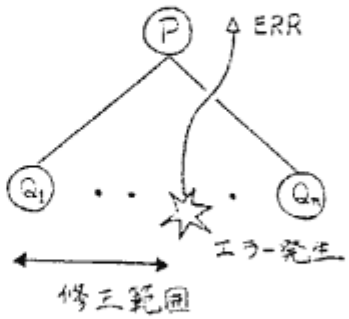
これらの制御機構のPrologインプリメントを図８に示す。Forward 推論を行うFdemo （述語名 "demo" ）は前節で述べた拡張demo述語にF/B 切り換えを制御するためのリターンコードを付加して実現している。Backtrace 推論を行うBdemo （述語名 "errchk" ）では、このリターンコードを調べエラーが発生したか否かを判断している。

```
{ expand body }
+(demo *KB :- NIL *ERR)-(!);
+(demo *KB :- (*P.*Q) *ERR)-(!)
        -(demo *KB *P *NEWERR)
        -(errchk *KB :- *Q *ERR *NEWERR);
+(errchk *KB :- *Q *ERR *NEWERR)
        -(var *NEWERR)-(!)
        -(demo *KB :- *Q *ERR);
+(errchk *KB :- *Q *ERR *ERR);

{ return predicate }
+(demo *KB (RETURN *CODE) *CODE)-(!);

{ apply rule }
+(demo *KB *P *ERR)-(getrule *KB (*P :- *Q))
                   -(execbody *KB *P :- *Q *ERR);

+(execbody *KB *P :- *Q *ERR)
        -(notdemo *KB :- *Q *NEWERR)
        -(!)-(fail);
+(execbody *KB *P :- *Q *ERR)
        -(demo-status *KB :- *Q *NEWERR)
        -(retract (demo-status *KB :- *Q *NEWERR))
        -(errchk *KB *P :- *Q *NEWERR *ERR)
        -(notnil *ERR);
+(execbody.*ARGS)-(execbody.*ARGS);

+(notdemo *KB :- *Q *NEWERR)
        -(demo *KB :- *Q *NEWERR)
        -(assert (demo-status *KB :- *Q *NEWERR))
        -(!)-(fail);
+(notdemo.*ARGS);

+(notnil *ERR)-(atom *ERR)
        -(eq *ERR NIL)-(!)-(fail);
+(notnil *ERR);

+(errchk *KB *P :- *Q *NEWERR *ERR)-(var *NEWERR);
+(errchk *KB *P :- *Q *NEWERR *ERR)-(getBKB *BKB)
        -(*BKB *P :- *Q -> *RECOVER)
        -(demo (*BKB *NEWERR *KB)
               :- ((system-errorhandler).*RECOVER)
               *ERR)
        -(!);

{ call standard predicate }
+(demo *KB (*P.*A) *ERR)-(*P.*A);
```

図８．Prologによる制御機構の記述

エラー発生から診断・回復までの過程を以下に示す。

(1) ある生成規則 P::=Q1...Qn における子(Q1.....Qn) にエラーが発生すると、そのエラー番号をリターンコードとして親(P) へ制御が移る。



(2) 親(P) はBKB を用いてエラー診断を開始し、以下のようにしてユーザに適切な処置を問合わせる。まずこの生成規則内の修正箇所を指摘し、応答を受ける。次に、

(2-1) 指摘された部分を修正するならば修正処置ルーチンを実行し、その後この生成規則内の前向き処理 Q1...Qn を再実行する（エラー回復）。

(2-2) この生成規則外（親側）の修正ならば、（新）エラー番号を持ってさらに親へ制御を移す（エラーの伝搬）。

(2-3) エラーチェックレベルを低く設定し直してこのエラーを無視するならば、前向き処理を再実行する。

```
{ EKB }
+(ERR1 (display-errormessage)
       :- ((print "*** type conflict")));
+(ERR2 (display-errormessage)
       :- ((print "*** ERR2")));
+(ERR1 (error-class ERR1) :- NIL);

{ BKB }
+(BKBn *P :- *Q ->
        ((error-class *ERR)
         (errorcase *ERR
           (ERR1 recover1)
           (ERR2 (RETURN ERR2))
                :
           (ERRk recoverk))));

{ EDIT-SYSTEM }
+(EDIT-SYSTEM (system-errorhandler)
      :- ((display-errormessage....)));
```

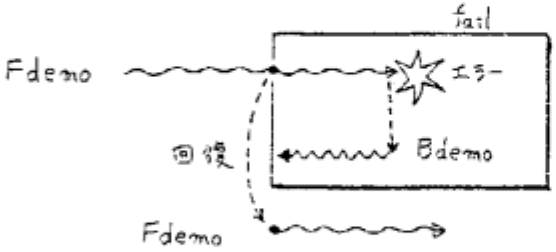
図９．知識定義例

図９は、各種知識の定義例である。BKB 中のエラー回復ルーチン(recover) はユーザに修正箇所を指摘し応答を受ける処理を行い、その結果に応じた処置の後RETURN 述語を呼び出す。

- 修正→（修正ルーチン）(RETURN NIL)
- 親の修正→(RETURN NEWERROR)
- 無視→(RETURN NIL)

このようにして実現された制御機構の特徴は次の点にある。

(a)Backtrace時の処理履歴はテキスト以外全て削除される。即ち、Backtrace 推論中に実行されたエラー診断・回復処理はエラーが回復された時点でfailし、あたかもこれらの処理がなかったかのようにふるまう。この性質はメモリ効率向上にとって重要である。



(b) エラー条件チェックはFKB に基づき起動されるが、エラーチェックレベルに応じてチェックすべきNKB の階層範囲が変化する。条件チェックは与えられた階層範囲の全ての条件を判断する。例えば、エラーチェックレベル１の場合には構文解析チェックだけで意味解析チェックを行わないと仮定すると、これはForward 推論時にNKB の階層を全く用いないことにより実現される。

(c) エラーチェックレベルに応じたBKB 知識の選択が可能である。例えばエラーチェックレベル１の場合には、エラーメッセージのコンソール表示(system-errorhandler により提供される）を中止しワークファイルへ一時的に蓄積させ、Forward 推論を続行すると仮定する。このための知識BKB1は次のように定義される。

```
+(BKB1 *P :- *Q -> ((RETURN NIL))):
+(BKB1 (system-errorhandler) :-
                ((save-file ...))):
```

(d) エラーメッセージの表示ルーチンは各エラー番号に対応した知識ベース中に存在するが、その呼び出しは disply-errormessage なる同一述語により行っている。この機能は知識の動的階層化により実現されている。

4．おわりに

プログラム言語の構文や意味を考慮した編集機能を持つ会話型エディタを設計し、その制御機構を中心にPrologによるインプリメントの一方法を提案した。対象とする言語の構文ならびに編集時に利用される意味規則は、属性文法に基づき表現されるため、これら知識のモジュラリティ、拡張性が高く保たれている。また、Forward/Backtrace 推論と多世界知識ベースを導入することで、構文・意味誤り発見時に、修正候補及びその理由の提示機能が容易に実現できた。これらの機能は、demo述語を用いるメタ推論方式により実現されており、Prologによって複雑な機能を簡潔に記述できることが確認できた。

今後の課題として、属性文法に基づく言語知識からの知識ベース作成方法論、及びプログラミング技法や対象領域の知識の導入を検討したい。

謝辞　本研究の機会を与えて頂いた溝口文雄助教授（東理大）ならびに有益な討論をして頂いたICOT WG4メンバーの諸先生方に深謝いたします。

参考文献

[1] Bowen,K.A. and Kowalski,R.A.: Amalgamating Language and Metalanguage in Logic Programming, School of Computer and Information Sciences, University of Syracuse, (1981).

[2] Knuth,D.E.: Semantics of Context-Free Languages, Math. Syst. Th., 2 (1968), 127-145.

[3] Hoar,C.A.R. and Lauer,P.E.: Consistent and Complementary Formal Theories of the Semantics of Programming Languages, Acta Informatica, 3 (1974), 135-153.

[4] Marcotty,M. and Ledgard,H.F. and Bochmann,G.V.: A Sampler of Formal Definitions, Computing Serveys, 8,2 (1976),191-276.

[5] Pereira,F.C.N. and Warren,D.E.D.: Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, 13 (1980), 231-278.

[6] Stoy,J.E.: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory, MIT Press (1977).

[7] 国藤、麻生、竹内、宮地、北上、横田、安川、古川：Prologによる対象知識とメタ知識の融合とその応用、情報処理学会研究会資料、知識工学と人工知能、(1983).

自動化ガイド・マニュアルの説明機能について
— Pre-study —

満口　徹夫　（三菱電機・情報電子研究所）

概要：コンサルテーションに対する各種の視点の中から、本文ではガイド・マニュアルの自動化（電子化＋構造化）を選び、その技術的問題点について初歩的検討を試みる。
まず、なぜこのような対象を選んだかについて示し、次にこのテーマをどのような枠組で把えようとするのかについて論じ、このようなテーマを取扱うために事例を用いた分析を行い、その後将来の実現法について考えてみる。

1．なぜガイド・マニュアルの説明機能を？

改めて言うまでもなく各種の知識が存在し、各種の知的活動が行なわれているのであるが、ここでは一般的定義や分類を行なう訳ではない。ここではむしろ何をどの範囲で扱いたいのかを示したい。
コンサルテーション・システムというからには人間が介在していることが前提であることは間違いない。また人間が質問を出し、機械がその解をさがして提示し、人間がそれを理解するという一般形が存在しそうである。ここで問題視されるのは質疑応答のすすめ方と内容の質的度合いであろう。
内容の質については機械における知性向上を図った種々の試みがなされ、対話のすすめ方は談話理解等で扱われるのであろう。しかしこれは一般的な話で個々のコンサルテーション・システムがこのような高度な枠組で最初から把えられそうもない。何らかの試み、経験、工夫が必要と思える。
世の中には文書化された知識（完全であるか否かは別として）とそうでないものがあるとすると、前者には一般の書籍も含まれ、特に本文で触れるガイド・マニュアルもこの中に入る。ガイド・マニュアルは本来は内容の増加は行なわず、記述している内容をいかに利用者に理解してもらうかがその目的である。
ガイド・マニュアルへの質問の出し方、分野の相違や目的の相違から来る内部の構成法も種々あるであろうが、利用者への説明をいかにするか、またその説明機構は内部構成法とどのように係わればよいのかを考えることもまた大切と思える。ガイド・マニュアルを含め書籍は利用者の反応のない一方向性の通信といえるが、自動化されたものには利用者の反応に基づく動作が要求されるので、単にマニュアル内の文章の表示ではすまされない一方、文書化された情報を無視する訳にはいかないし、自然語に含まれる全ての情報を構造化できるとも限らない。そこで内容の比較的安定し、充分な情報が記述されているもの（ガイド・マニュアル）を、しかも説明という一方向性の強い、ただし逆方向の通信もある環境へ仕立てるには、どのように再構成すればよいかの検討を本文ではとり上げてみたい。
以下では、通信としての説明機能（通信モデルと説明の内容）
事例study ……計算機システム性能評価の情報分析ステップを入出力関係について行った場合、
及びその考察
について述べる。

2．通信（情報伝達、意味解釈）としての説明機能

通信（communication ）の1つとして説明の持つべき機能は何かを考えてみる。通信には｛man, machine｝←→｛man, machine｝の組合せがあるが、少なくともここでは片方にman が存在するのは確かである。つまりman-machine の組合せが我々の最も興味ある所であるが、man A-machine-man B という形も考えられない訳ではなく、片方がexpert、片方がuserと考えられよう。
通信の役目は今ここで強調する必要もない位で、工学面、組織学面、社会学面、言語学面、心理学面の種々のアプローチがあるようである。通信のモデルを［F］に従った分類でまとめると以下の通りである。

2．1　通信モデル

［F］によれば、通信モデルは次の2つに分類される。

分類	主関心	主役	モデル事例
Process	メッセージの意図	送信者	Shannon
Semiotics	メッセージの意味解釈	受信者	Peirce Saussure

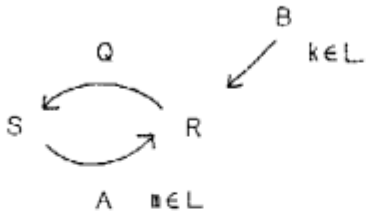
Semiotics では、受信者がどのようにメッセージを解釈するのかが大きな関心事であることが特徴的である。しかもSemiotics はテキスト（言語表現化したもの）の扱いが主体で、non-yerval-communicationは大きく扱われてない。対象は次の3つである。

- sign —— icon, index, symbol からなる。reality との対応（denotation）
- code —— signを体系化したもの。（paradigmとsyntagm ）
- culture —— culture 下での形式による作用（connotation ）と内容による作用（belief ）

一方process モデルは情報の伝達プロセスとしての通信が扱われており、送信者の役割、意図が重要である。通信のモデルとして、単に送信者、あるいは受信者のいずれかの役目を演じるのではなく、フィードバックによって双方の役目を演じる形が大切との指摘もある。［D］ここで論じる説明機能を考える点では注意すべきことと思える。

興味深い通信モデルとしてHarrahのモデル［H］がある。

"to construct a model of a rational procedure for asking questions and giving answers, applicable in situations where receiver wants to acquire information, explore meanings, and evaluate the quality of the sender as an information source"

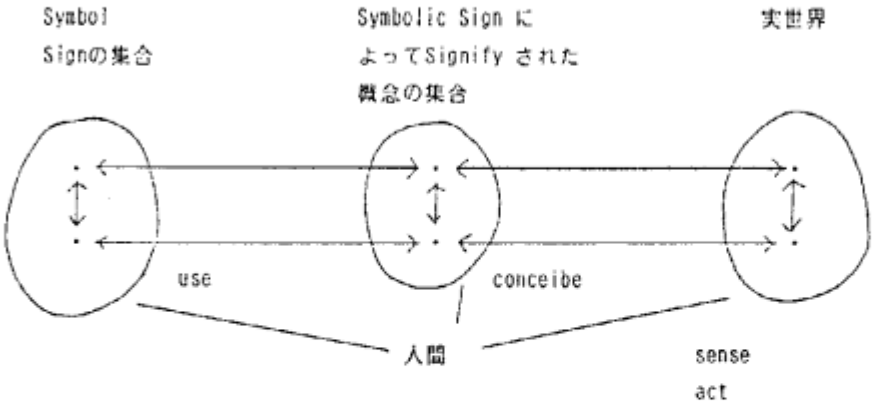


B : Background Knowledge
S : Sender
R : Receiver
L : Language
Q : Question Set
A : Answer Set

Shannon のモデルではメッセージの平均的情報量を扱うのに対し、Harrahは個々のメッセージの重要さを論理的、かつ定量的の扱う試みをしている。質問の形を(n-place) whether questionとwhich questionを扱っている。前者はmultiple choice 、後者はfill in the blank 形である。

2．2　説明したいこと

2．1では抽象的に通信モデルを云々したが、視点をコンサルテーション・システムに向けて、仮に下のような図形式を考えてみたとき、



知識の構造は、‘実世界の反映としての概念集合における要素間の関係づけ’と言うことができるであろうし、コンサルテーション・システムは‘その知識構造を対応するsymbolic sign の集合上の操作に置きかえるもの’と言える。

ここで知識と説明のポイントは何かを考えてみると、

知識は与えられた環境での、入力からある結果を得るときに用いられ、より正確な結果を得るには、実世界と概念とsignがより正確に対応づけられているか否かが重要と思える。一方説明はそのような知識を正確に人間に伝え、かつ正確に解釈、理解してもらうことが重要である。

説明で人間に伝えなくてはならない情報の種類として以下のものがある。

(1) 実世界←→概念の対応づけ

あるいはこれは概念そのものの説明と言えるが、当然説明は他の概念を用いて、言葉で表現される。概念の説明はその概念に含まれないものでもって説明を補強することが多い。

(2) 概念←→symbolic sign の対応づけ

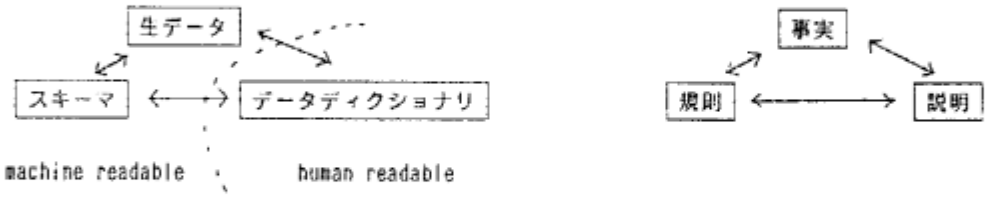
各symbolic sign が何を意味するかを説明する。

(3) 実世界での要素間、即ち近似した概念集合内での要素概念間の関連づけ

また以上の情報を理解してもらうための説明もされなくてはならない。

知識はmachine への‘説明’情報であるとしたならば、ここで説明と言っているのは人間に対する‘説明’情報である。machine にはsymbolic sign によって‘説明’されるが、人間には概念レベルの自然語で行なわれる所がややこしい。

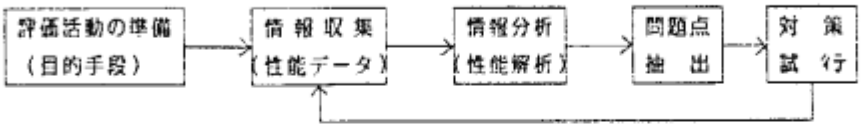
データベースにおける以下の関係は、コンサルテーションにおける関係と似ている所がある。



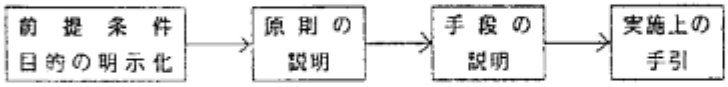
3．事例によるstudy

2．のモデルの話はあまりにも抽象的すぎるとの反省から、事例として「計算機システム性能評価マニュアル」(例MVS Performance Notebook等)を対象としてみた。

計算機システム性能評価活動自体の過程は以下のものであり、



ガイド・マニュアルの構成は以下のものからなる。



例として情報分析ステップでのガイド情報は；（入出力関係に話を限って触れる）

目的の明示化： IOB ：ＩＯボトルネックが発生している。

↓

IOWM ：ＩＯの要求の遅れを最小化する。

IOB とは：ＩＯ要求での遅れが 計算機処理性能 上で障害となっていること。

2つの観点 → システム全体性能

2つの観点 → 特定ユーザ性能

現象は 1. IOA ：ＩＯ装置の活動大 → IORS ：ＩＯの資源利用を調査する。

かつ 2. CPW ：ＣＰＵの待時間大（即ちＩ／Ｏ要求待ち時間大……逆は正しくない）

計算機に十分な負荷がかかっていない。

又は

IOB

IORS → IOWM

原則： 原理的動作モデルの説明

ＩＯ要求の遅れの要因を説明するには、遅れを生じる動作の説明が必要。

ＩＯ要求遅れ要素と遅れの主原因	1．ＩＯ装置利用可待	
	2．ＩＯアクセス路利用可待	：他のＩＯ要求の外乱
	3．シーク終了待	：自ＩＯ要求の性質上
	4．回転待	：自ＩＯ要求の性質上
	5．RPS 再結合終了待	：他のＩＯ要求の外乱
	6．データ転送終了待	：自ＩＯ要求の性質上

手段： 実際に情報分析を行うために、原則に示したもの等に基づいて、

1．評価項目の選定：どのような点に注目するか？　　x
2．評価指標の選定：どのような評価基準を持つか？　　y　　y＝f（x）
3．評価関数の選定：どのように評価値を求めるのか？　f

他に問題点抽出の場合では

問題ＩＯパスの発見 ── なぜか（なぜ最初のこれを検討するのか？
↓　　なぜこれを検討すればよいのか？
：　　：　　）
：　　：
└ どのように……

実施上：評価実施上での判定支援

例　ＩＯ装置の利用率は装置タイプ×××、接続台数×台の時、
（システムorオンラインor etc.）利用をしたらば、20% では
ＩＯ活動大と言うのか？

以上のようにまとまりのない形で表現したが、これをまとめて行くには、

1) ガイド情報をどのように構造化するか。
2) 利用者の問いかけをどのように設けるか。

は最低考えないといけない。

1)については、既に示したガイド情報の分類を、例えばシステム全体、CPU 性能、ＩＯ性能、etc.に階層化する必要がある。2)についても、why とhow だけでよいかどうか考えてみる必要がある。［CM］たとえば

What do you mean by x?
Why did you say that?
Whay do I have to do? (why me?)
How did you get that?
How do I have to do?
Where am I?

また右に示すようにたとえば作ったとして、

a)枚切り問いかけを決った所でしか許せないのをどのようにするか？
b)ずらずらと説明文を表示して、全て読めというのか？

自然語（日本語）による問いを用いることも考えられるが問題はその内容であるし、文章ではなく、ブロック図(icon)による説明の必要もあるかもしれない。

そもそもモデルと事例の関係はどこへ行ってしまったのであろうか？
概念とsymbolic sign の関係は事例でその対応をつけたつもりである。また概念間の関係づけも行なわれたとしよう。

利用者の問いかけの例を考える内で、symbolic sign 、概念、および概念間の関係についての問いかけ方が明らかになると考えられる。

単純に考えて、ガイド・マニュアル内の原文をそのままキーワード対応の形で問合せ、出力するものからガイド情報の構造化とそれに合せた利用者問いかけの自由度提供までのスペクトルの中で、どこに位置づければよいかを考えてみることが必要と言えよう。

モデルで考えられていないのは、対話そのものについてである。対話のモデルとしてどのようなものがよいのか、対話とは何か、その単位は何か等について考えることも必要となる。［DEM］，［M］

参考文献

[F] Fiske, John 'Introduction to Communication Studies' Methven 1982

[H] Harrah, David 'Communication : a logical model' MIT Press 1962

[D] Devito, Joseph A. 'Interpersonal communication book' Harper & Row 1980

[CM] Clark , McCabe 'PROLOG : a language for implementing export systems' TR:DOC80/21 (1980) Univ. of London

[DEM] Dehning, W.,H. Essig, S. Haass 'The adaptation of virtual man-machine interface to user requirements in dialogs' LN in CS, Vol.110 Springer-Verlag 1981

[M] Moran, T. 'The Command Language Grammar : a representation for the user interface of interaction computer systems' Int'l J. man-machine studies 14 (1981)

データベース論理設計支援エキスパートシステム

溝口理一郎　磯本征雄　角所　収
（大阪大学）

1．まえがき

第５世代コンピュータが「使い易い」コンピュータを目差しているということはよく言われることであるが，コンピュータ利用における利用者の知的な支援は重要な問題となっている．コンピュータ利用のなかでプログラミングを支援するシステム，即ち知的プログラミングシステムに関してはよく論じられており，デバッグまでを含めたプログラミング支援さらにはプログラミング教育への応用を考えれば，その重要性は極めて大きいものと考えられる．上述の意味でのコンピュータの利用を「ソフトウェアの生産」とみなせば，コンピュータの利用には「ソフトウェアの消費」，即ち既成品のプログラムを使って結果を出すという，いわゆるエンドユーザ的な利用形態もある．データベースの検索利用などはその典型的な例である．一般に，この種の利用形態に於ける利用者は「素人」であることが多く，特に丁寧に支援する必要があるように思われる．これら「素人」利用者の意見はコンピュータに対する評価に意外に大きな影響を及ぼすからである．

プログラミング言語によるソフトウェアの生産と既成品ソフトの単なる利用の中間に位置するものとして，DBMSを用いたデータベースの構築がある．DBMSはある意味で既成品のソフトウェアであるが，データベースというシステムを作る為のソフトウェアと見ることもできる．データベースの作成，特に学術データベースにおいては，作成者がコンピュータとは全く関係のない分野の人々であり，データベースの知識は勿論，コンピュータの知識すら十分には持っていないという特性がある．学術データベースの有用性はコンピュータの進歩とともに今後益々増加し，データベース構築の潜在的欲求は強くなるものと思われる．しかしながら，DBMSという巨大なソフトウェアを使いこなすことは「素人」にとっては極めて困難であり，データベース構築の大きな障害となっている．しかも，現存するDBMSにはこのような潜在的利用者を支援する機能は用意されておらず，大部なマニュアルが用意されているだけであるのが現状である．

第５世代コンピュータとしてはこのような利用者をも支援して，初めて「使い易い」コンピュータと世に受け入れられるものと思われる．その為に解決すべき課題は多いが，本稿では，ソフトウェアを消費する「素人」の支援を想定して「マニュアルレスシステム」というものを考えてみたい．「マニュアルレスシステム」は，エンドユーザにとっては実に有難いもので，事前にそのシステムに関する勉強をする必要がなく，端末のまえに座りさえすれば目的とするソフトウェアを使うことが出来るというものである．自然言語インターフェースがあり，誤りがあれば診断して直してくれ，判らないところは親切に教えてくれる，そういうシステムである．見方をかえれば，「マニュアルレスシステム」はそのシステムの利用に関するエキスパートシステムということが出来る．マニュアルを読むより，良く知っている人に聞く方が早くて理解も容易であることは誰しもが経験することであるが，その「良く知っている人」をコンピュータ上に実現すれば良いわけである．

次節以降では，「マニュアルレスシステム」に関する研究の一環として現在我々が行っている，データベース構築のための支援システム〔１〕について，その中の論理設計支援システムを中心にその概要を述べる．

2．データベース構築支援システム

大阪大学大型計算センターでは現在６つの学術データベースが運用中〔２〕であり，さらに６つの新しい学術データベースが開発中である．DBMSとしては全て日本電気のＩＮＱが用いられている．筆者らも音声データベース〔３〕〔４〕を作成した経験を持つが，Ｉ

ＮＱは勿論,データベースに接するのが初めてであったので，マニュアルの輪講をしたり，日本電気に問い合せたり色々苦労したものである．本システムは，筆者らが得た，ＩＮＱを用いてデータベースを構築する為の知識を移植したものということができる．

データベース構築において初心者が遭遇する困難な点は次の５つにあると思われる．

１）ジョブ制御言語の使い方．

２）全体の処理の流れの把握．

３）誤りからの回復．

４）論理設計．

５）手持のデータのフォーマット変換．

現在，１）と２）についてはほぼ完成しており，４）の論理設計を支援するシステムの設計を行っている．

3．論理設計支援エキスパートシステム

データベースの構築において，その論理設計は最も重要であることは言うまでもない．しかしながら，初心者にとっては手持のデータに潜在する論理的な関係を瞬に把握して，データベースにおける表現に適したように再構成することは極めて困難なことである．従って，このデータベースの論理設計を行う部分を支援することができれば，初心者にとって極めて有益なことと思われる．

3.1 基本的な考え方

データの論理構造は，データが持つセマンティックスに係わることであるので，正確に抽出することはかなり困難なことと思われる．従来この問題に関しては，実際のデータ（値）を用いて，種々の解析技法により手探りで行うというアプローチがとられていた．ここでは，データの論理構造を知る手掛りとして，自然言語によるデータベースへの検索要求文を用いることにする．一般に検索要求文はかなり制限された形をしており，各検索文に固有の論理構造をもっている．従って，十分な数の検索要求文が与えられれば，それらの検索要求に応じる為に必要なデータベースの論理構造を推定することが可能であると考える．

上述のように，データベースの開発者にとってデータベースの論理設計は困難であっても，開発しようとしているデータベースの使い方に関する考えはもっているものと思われる．即ち，データベースに対する検索要求の例を列挙することは容易である．従って，ある程度制限された自然言語で検索要求の例を列挙することを要求することは開発者にとってそれほど大きな負担にはならない．

特定のデータに関する，システムが持つべき知識は原則として仮定しないものとする．正統的なアプローチとして，データの値に関する情報を用いてそれらの間の関係の同定，バリュウアブストラクション（値の抽象化）による属性の抽出等を行うことにより，論理構造を「学習」する方法が考えられる．このアプローチは学問的に見ても帰納的推論との関連が示唆されて，興味深いところがある．しかしながら，たとえこの方法が理論的にある程度解決されたとしても，次の理由によりそれは現実的なシステムに組み込むことはできない．一般にはどの様なデータが来るか判らない為，多種大量のデータ（値）に関する情報とそれらを操作する知識を蓄積していなければならないが，これは事実上不可能である．従って，このアプローチを採用するシステムでは，利用者は各自が所有するデータのセマンティックスを（各個の関係をシステムが要求する形に）整理する操作が必要となる為利用者の負担は大きく，悪くするとデータベースの論理設計と大差ないことにもなりかねない．

我々のシステムでは，名詞以外の単語にかんする辞書を持ち，未知の単語は全て名詞として扱い，構文情報を中心に処理を行うという立場をとる．このことにより，上述の，システムが持つべき辞書の複雑さの問題は解消される．必要な情報はできるだけ利用者との対話により得ることとし，利用者の負担を最小限に留めるように努めている．

3.2 本システムにおける処理の概略

A）構文解析

本稿では複雑な文の出現の防止と解析の容易さの為，検索文は英語とする．更に，名詞以外の単語の品詞は全て既知（辞書にあるもの）と仮定する．検索要求文はFindで始まる命令形の文に限定する．従って，構文解析は問題ないと考えられる．代名詞及び関係詞が使われたときは，本パーサーはそれらが指示する名詞を同定できないので，利用者に聞い合せる．これはパーサーと利用者とのQ&Aの例になっている．検索文のいくつかの例を以下に示す．

a-1 Find the names of the employees in the toy department.

a-2 Find the items sold by departments on the second floor.

a-3 Find the average salary of the employees in the shoe department.

a-4 Find the departments that sell pens.

a-5 Find the companies that supply the items of type A to all the departments on the second floor.

B）基本関係の抽出

構文解析により得られた構文木をもとに，of,on,in等の前置詞，或いは陽に関係を表す動詞等をキーワードとして名詞間の関係を抽出する．次に上述の例文から得られた基本関係の抽出例を示す．

```
    of (names, employees)                  in (employees, toy department)
  sell (departments, items)                on (departments, floor)
    of (salary, employees)               sell (departments, pens)
supply (companies, items, departments)     of (items, type A)
```

C）属性処理

我々のシステムは名詞に関する情報を一切仮定しておらず，構文情報だけをもとにデータの論理構造の抽出を試みているところに特徴がある．一般に，検索文に現れる名詞は属性名か属性値のどちらかであるが，その区別をしなければならない．以下に例を挙げる．

```
c-1 toy department          c-2  type A, volume B
    Anderson's  manager          length C
c-3 sell items
    sell pens
```

c-1 （N1＋N2）の場合ではN2が属性名であるが，c-2 の場合はN1が属性名である．c-2 のN1が限定された名詞（type, size, hight 等のいわゆる属性名詞）であることから，これらの名詞を予め辞書に持っておくことにより，この2つの場合を区別することができる．更に，属性値となる名詞は修飾を受けることはないので，被修飾の関係の有無を調べることによりある程度属性名と区別することが可能である．しかしながら，属性名と属性値との関係は相対的なものであり，viewが変れば属性名にも属性値にもなることだあるので，このテストは完全ではない．従って，以上のことを検討した後，確からしい方を利用者に提示して確認してもらうようにしている．

c-3 の場合では，"sell pens "は例文a-4 のようにpensを修飾するものはないので，pensは明らかに属性値であることが判る．"sell items "のitems はa-2 のようにFindの目的語になっていることから属性名である．このことより pens はitems の値であることも同時に判るが，念の為利用者に確認する．最終的に得られた属性名を用いて，基本関係の整形・消去を行う．以下に名詞が属性名となる為の構文的な条件をまとめておく．

1）被修飾関係にある．
2）Findの目的語になる．
3）is，equal 等の動詞の主語になる．

D）概念スキーマの形成

概念スキーマの形成はさらに細かく次の4つのステップに分れる．

1）検索主体となり得る属性名の検出
2）階層関係の決定
3）冗長性の除去
4）概念スキーマの形成

1）検索主体の検出

Find N1 of N2 which ・・・，　　Find N2 Which ・・・

という文型における名詞N2は検索主体であり，明らかにN1はN2の1つの属性である．このことから，検索主体の検出は容易に行なえる．

2）階層関係の決定

構文情報だけからでは，名詞間の階層性の決定は殆どの場合不可能であり，利用者との対話により決定を行う．　in（employees，departments）のように　in（A，B）の関係では　A：B　=　1：n　であると予想されるが，他の大部分を占める　of（A，B）では全く不明であるので，利用者に問い合せ　1：1，1：n，m：nの何れであるかを定める．

3）冗長性の除去

以上述べた処理を施した後の基本関係の集合には幾つかの冗長なものがあり得るので，それらを除去しなければならない．

イ）2つの関係において，関係名と項の数が同じで一部の項目名のみが異なっている場合，それらの異なる名詞が同義であるかどうか，そして同義であるならどちらを属性名として採用するかを問い合せる．例を以下に示す．

supply（companies，items）
supply（suppliers，items）

ロ）関係名が異なるが，その項目（属性名）が順序を無視して一致するものをクラスクとして集め，それらが単に自然言語上での言い換えなのか，或いは意味のある区別であるのかを利用者に問い合せる．例を以下に示す．

give（Tom，Mary，book ）
receive（Mary，book，Tom ）

ハ）既に，

of（employees，salary ）
of（department，employees ）

の2つの関係が得られているものとする．この時，

・・・departments having the average（total ）salary greater than ・・・

という検索文の処理を行うと

have（departments，salary ）

が得られる．しかしながら，この関係はdepartments がsalaryという属性を持つということを意味することとなり不都合が生じる．これら3つの関係はemployees，departments，salaryの3つの属性をノードとする完全グラフを形成しているが，この問題は次の様にして解決することができる．

departments とemployees は共に検索主体でありsalaryは属性であること及びdepartments：employees　=　1：n であることはここまでの処理で確かめられているので，今「average（total ）」に注目すれば，平均（総和）はdepartments の中のemployees に基って行われるものであり，have（・）の関係は不適切なものとして除去される．

ここまでの処理により得られた関係のＥＲモデル表現を図1に示す．

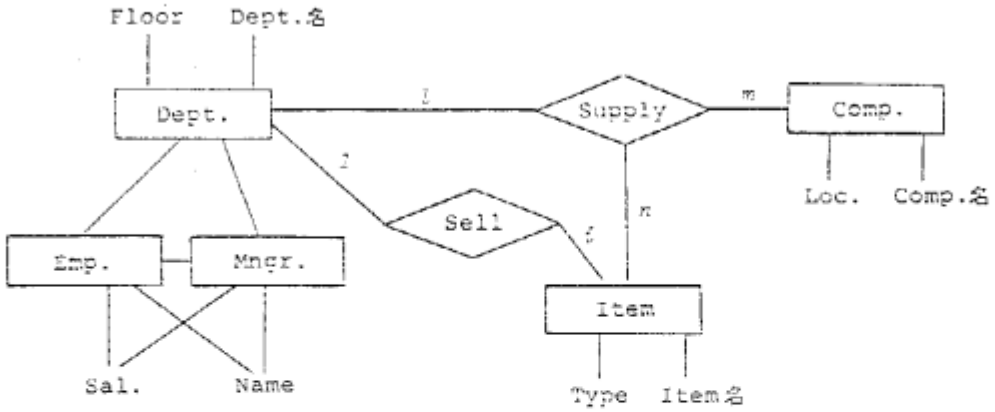


図１．関係のＥＲモデル表現

４）概念スキーマの形成

設計の基本方針としては，陽に表現された関係と検索主体となり得る属性を重視する．

STEP 1：動詞を用いて表現された関係例えばsupply（・），sell（・）等を優先的に使用する．

STEP 2：上で用いられた属性の中で，検索主体となるものから順に取り出し，それと関係のある属性をまとめる．

STEP 3：他の属性に関しても，検索主体を優先的に処理してゆき全ての関係が網羅されるまで繰り返す．

関係をまとめる際に注意しなければならない点を次にのべる．

具体的に概念スキーマを構成する際には，用いられるDBMS，我々の場合ではINQ の特性も考慮しなければならない．INQ では階層的に定義された複数のファイル（関係）を関係データベース的に用いることができるが，この特性を有効に利用することを考える．

ａ）３つ以上の項目からなる関係はそのまま用いる．

ｂ）検索主体と属性の関係で，属性値がユニークに決る場合はその属性を主体に単一項目として付加する．ユニークに決らない場合は不定繰返し項目として付加する．

以下では検索主体同志の関係について考える．

ｃ）１：ｎ（ｎ＝１の場合を含む）の関係では，ｎの方の検索主体に１の方の主体名を属性として付加する．

ｄ）ｍ：ｎの関係でｍ＞ｎのときは，ｎの方はそのままにして，ｍの方に不定繰返し項目としてｎの方の主体名を付加する．

以上述べた処理によって得られる概念スキーマの例を以下に示す．

Emp.

Name	Dept.	Mngr.	Sal.

Mngr.

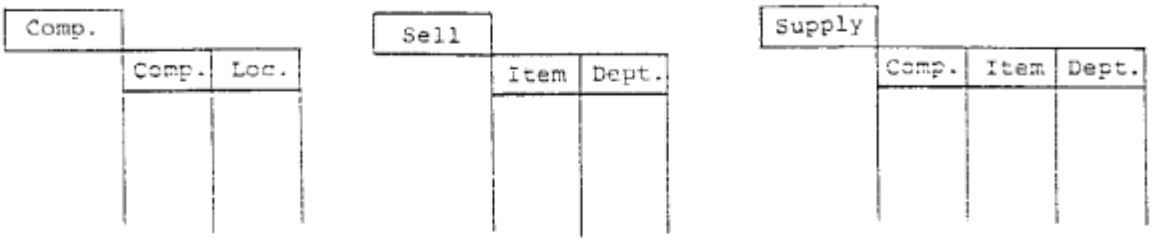
Name	Dept.	Sal.

Dept.

Dept.	Floor

Item

Item	Type



E）ＩＮＱでのＦＤＬ（概念スキーマ）記述の生成

最終的に得られた概念スキーマを生成する為に，ＩＮＱのＦＤＬを用いた論理構造の記述を行う．構造としての両者の表現は完全に一致するする為，変換は一意的であり比較的容易である．

４．インプリメンテーション

本システムの実現には，本学の計算センターで使用可能なプロログshape-upを用いる予定である．shape-upはＣ言語の標準関数systemをcallでき，ＩＮＱとのリンクが考慮されているので，我々の目的には最適な言語となっている．現在，システム全体の概略設計が終了した段階であり，インプリメンテーションはまだ行っていない．

５．むすび

「マニュアルレスシステム」の研究の一環として行っている，ＩＮＱを用いてデータベースを構築する利用者の支援システムの中の，データベースの論理設計を支援するエキスパートシステムについて述べた．本システムを使用するにあたって利用者が用意すべきものは，開発しようとするデータベースへの検索要求文の集合だけであり，簡単な質問に適宜答えるればよく，負担は極めて軽い．データのセマンティックスに関する一切の事前情報を仮定せず，検索文中の係り受けに基づく構文情報を中心として解析を行うことが本システムの特徴である．そのため，検索要求文に現れる，開発者の意図を忠実に反映したデータベースの論理構造の抽出が可能になるものと思われる．

〔参考文献〕

〔１〕國本，石術，溝口，角所："データベース構築・管理のための知的支援システム――Knowledge based DBMS（KDBMS）――"「アドバンスト・データベース・システム」シンポジウム，情報処理学会，昭和57年12月8日，9日，pp.49-58.

〔２〕大阪大学大型計算センターニュース，Vol.13，No.1，p.16，1983.

〔３〕溝口　他："知的アクセス機能を持つ音声データベース「SPEECH-DB」"，情報処理学会誌，Vol.24，No.3，pp.271-280，1983.

〔４〕溝口　他："汎用DBMSを用いた知的マン・マシンインタフェースの実現――音声データベースSPEECH-DBに関して――"情報処理学会誌（投稿中）．

Concurrent Prolog 上の
知識情報処理用プログラミング言語／システム
Mandala （曼陀羅）について

古川康一、竹内彰一、国藤 進 （ICOT）

1. 概要

対象指向プログラミングは、表現力が豊かであることから、最近になって急速にその重要性が認識されてきている。とくに知識表現の観点から見ると、時間と共に変化する個々の対象を自然に記述できることから、従来の静的な事実の集合のみを記述する体制に比べて、その応用範囲は大きく広がってきたと言える。

ところで、これまでの対象指向プログラミングは、その評判とは裏腹に、実際にはあまりその技術が展開されていない。その理由の１つは処理系が広く出回っていない点である。現在よく知られている対象指向プログラミング言語／システムには、 Smalltalk, Flavor, LOOPS [1] などがあるが、日本国内でそれが利用できる環境はごく限られている。

対象指向プログラミングが普及しない理由は、もう１つあるように思われる。それは、言語自身あまり使い勝手が良くないのではないか、という点である。あるいは、言語が十分に洗練されていないので、概念整理をする際に言語を使いこなすのに苦労をするのではないかと思われる。

我々は、Concurrent Prolog の上に対象指向に基づく知識情報処理用プログラミング言語／システム Mandala（曼陀羅）を作り上げる作業[2,3] を開始したところである。Concurrent Prolog 自身、並行事象を記述する言語であり、簡潔であること、Prologとの関連が明確であることなどのいくつかのすぐれた性質を備えている。そのようなすぐれた性質は、その上で対象指向プログラミング言語／システム[2] を作り上げる上で、大きな利点となっている。

本報告では、２章でConcurrent Prolog の概略を紹介し、つぎに３章で Concurrent Prologでどのようにしてオブジェクトを実現するのかを示す。４章では、LOOPS に現れる諸概念が Mandalaでどのように表現されるか、その対応関係について述べる。そして最後に基本的な部分のプログラムと、その実行例を示す。

2. Concurrent Prolog

Concurrent Prologは、並行事象の記述のための論理型プログラミング言語で、Prologとは、図１のような関係にある。

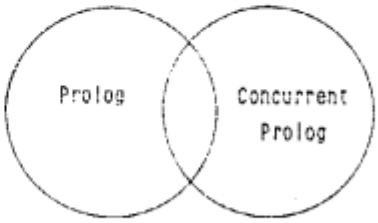


図１． PrologとConcurrent Prolog の包含関係

すなわち、Concurrent Prolog は、Prologの一部の機能をその中に含むと同時に、並行事象を記述する拡張部分をもつ。Prologにあって Concurrent Prolog にない機能は、深い後戻りの機能 (deep badktracking)である。実際、並行事象の記述では、２つ以上のプロセスが互いに通信し合いながら動くプログラムを作るが、１つのプロセスが他のプロセスに通信した後で失敗した場合、通信自身も取り消さなければならなくなる。このような状況は、それを実現するのも困難であるし、そのようなプログラムに意味を持たせるのもむずかしい。そのため、Concurrent Prolog では、一旦他のプロセスへの通信が行われれば、その通信自身にまで及ぶ後戻りは禁止している。

しかしながら、Concurrent Prolog 自身、そのlogic partはHorn logicであり、control partが通常のPrologのそれと異なる。すなわち、つぎの式が成り立つ。

Horn Logic ＋ 深さ優先探索 ＝ Prolog
Horn Logic ＋ ? ＝ Concurrent Prolog

この式の？部分は、つぎのような実行規則である。

(1) Process 間の通信は、共有論理変数によって行われる。

(2) 通信メッセージは、論理変数のとる値である。
(3) メッセージは、送り手と受け手が区別される。とくに受け手は、X？のように変数に？が付加される。X？をもつプロセスは、その変数の値を定めてはいけない。もし、具体的な値との同一化が行なわれたとき、その変数の値が定まっていなければ、その同一化は、一時中断される。
(4) メッセージの送出は、送り手が通信用論理変数の値を定めることによって行なわれる。その値を決定するときにいくつかの選択があるとき、送り手はその決定を“|”によって指示する。“!”はPrologのCut に似た働きをする制御命令でCommitと呼ばれる。

3. Concurrent Prolog によるオブジェクトの実現

Concurrent Prolog を用いてオブジェクトを表すには、プロセスを使うのがよい。特に、プロセスをtail recursion programとすると、同一の名前のプロセスが生きつづける。そして、オブジェクトの保持する状態は、そのプログラムの引数に持たせることができる。

［例］ 計数器

```
counter([clear | S],State):- | counter(S?,0).
counter([up | S],State):-
  NewState is State + 1 | counter(S?,NewState).
counter([down | S],State):-
  NewState is State - 1 | counter(S?,NewState).
counter([show(State) | S],State):-
                -       | counter(S?,State).
counter([],state):- |.
```

この例で、手続きcounter の第１引数は通信用論理変数で、第２引数は、カウンタ・オブジェクトの状態である。counter プログラムは、最後のclause以外は、すべてtail recursiveなプログラムになっている。そして、メッセージ"clear" によりState は 0になり、"up"あるいは"down"メッセージにより Stateは+1あるいは -1 変化する。

オブジェクトの生成は、単にその定義プログラムを呼び出すことによってなされる。すなわち、

```
?- instream(X),use_counter(X?,C1),
        counter(C1?,0).
```

により、３つのオブジェクトinstream, use_counter, counter が作られる。ここで、instreamは端末からの入力を受けつける組込みオブジェクトで、 use_counter はcounter の値を参照して走るプログラムである。

use_counter プログラムの一例を次に示す。

```
use_counter([show(Val) | Input] ,
             [show(Val) | Command]):-
        | use_counter(Input?,Command),
          wait(Val) & print(Val).
use_counter([X | Input], [X | Command]):-
        | use_counter(Input?, Command).
```

4. 対象指向プログラミングシステムとしての Mandala（曼陀羅）

Concurrent Prolog 上の対象指向プログラミングを行うときの基本は、前節で述べたオブジェクトである。本節では、このオブジェクトの囲りにいかにして対象指向プログラミングの諸機能を実現していけばよいか、その基本的な考えを示す。

(1) instance object

LOOPS のinstance object は、Concurrent Prolog のオブジェクトが対応する。

(2) class object

LOOPS ではclass もinstance object と同様に定義されるが、Concurrent Prolog では、class object は状態を持たない静的なプログラム（データベース）が対応する。

(3) property と method

propertyおよびmethodは、Concurrent Prolog では全く同一に扱われ、それぞれ、オブジェクトあるいはclass を表すプログラム定義中の引数上で実現される。そして、それらはそのオブジェクトに付随したlocal world を定める。すなわち、propertyやmethodは節集合となっている。

(4) is_a hierarchy によるproperty inheritance

is_a hierarchy は、local world のhierarchy を

作り出す。そのhierarchy は、1つのオブジェクトに対して親が2つ以上存在してもよい。すなわちmultiple inheritanceが可能である。その場合のサーブ・ルールは、適当なdemo programを作ることによって、容易に変更が可能である。

(5) part_of hierarchyによるproperty inheritance

part_of hierarchyは、それ自身がclass level と instance levelの両方に現れる。part_of hierarchyによるproperty inheritanceは、行なってよい場合とそうでない場合があるが、その基準が明確であればdemo programに組み込むことができる。instance levelのpart_of hierarchyはプログラムの階層的モジュール化に対応する。上の階層のプログラムは、下の階層のモジュールを部品として用いる。そしてその間の関連は共有変数を介したオブジェクト間通信によってとられる。

(6) meta object

LOOPS では、class を管理するものとしてmeta classを用いるが、Concurrent Prolog では、class objectを管理するためにmeta object を作る。meta object は、動的なオブジェクトが対応し、class variableはそこに置かれる。

以上が主要な特徴であるが、以下に、設計上考慮した問題点について述べる。

(a) 何故、class は静的なプログラムとして表現するのか

class の表現方法には、静的なプログラムの他に、動的なオブジェクトを用いる方法も考えられる。しかし、動的なオブジェクトでclass を表現すると、いくつかの不都合な点が生じる。その1つは、is_a hierarchy によるproperty inheritanceの実現が困難な点である。default reasoning の自然な実現法は、demoを用いて「失敗」時にis_a hierarchy を遡上るやり方であるが、動的なオブジェクトでは「失敗」の扱いが困難である。第2に、instance object とclass objectを共有変数を介して結合することを考えると、複数のinstanceからの通信のmerge が必要となり、それは非実現的である。

静的なプログラムとする積極的な理由としては、class が表すものは一般的な概念であり、それ自身計算の進行と共に変化する状態をもつのが不自然な点である。こうするとclass variableが実現できないが、それは、そのclass に相当するmeta object を動的なオブジェクトとして作り、そこに持たせることにする。そのmeta object は、そのclass に属するinstanceの管理を行なう。

(b) 何故、状態はlocal world か

instance object の状態は、local world と考えなくとも良いが、demo述語を考えるとき、上のclass のlocal world と考え方を合わせた方がよい。instance object の状態はそのobjectを表すリテラルの引数上に現れ、それをlocal world すなわち節集合と考えると、そのobject自身メタ的な性質を持つようになることに注意したい。ただし、instance object のlocal world に現れる節は、通常はunit clause で、ルールの形をしたものがそこに置かれることはないであろう。

(c) property inheritanceに何故demo述語を用いるか

property inheritanceは、対象指向プログラミングの要であり、その高速処理が大切な点であるが、ここでは、あえてdemo述語を用いたやり方を採用した。それは、property inheritanceの戦略自身未だ十分に煮つまっておらず、いろいろな戦略を試してみることが必要であると思われるからである。なお、この点については、ハードウェアによる高速化の検討も必要となるであろう。そのような高速化機構は、知識ベース・マシンの一構成要素を成すものと思われる。

［謝辞］ 本研究を行なう機会を与えていただいた渕一博当研究所所長に感謝致します。また WG4におけるディスカッションは本研究を始めるきっかけとなりました。渕口文雄主査を初めとする各委員に感謝致します。

References

[1] D.G.Bobrow and M.Stefik: The LOOPS Manual - A Data and Object Oriented Programming System for Interlisp (Preliminary Version), Memo KB-VLSI-81-13 (working paper), Aug. 1981.

[2] A.Takeuchi, K.Furukawa, E.Y.Shapiro:
Object Oriented Programming using Concurrent Prolog, Proc. of the Logic Programming Conf. '83, Tokyo, March 1983. (in Japanese).

[3] E.Y.Shapiro: A Subset of Concurrent Prolog and its Interpreter, ICOT Technical Report TR-003 (1983).

知識ベースシステムの課題とその実現法

富士通（株）・国際情報社会科学研究所　　新谷虎松

1．はじめに

本研究は知識ベースシステムを構築する際に、主機能となるいくつかの本質的な点に焦点をあてて、その部分解決をはかろうとしたものである。

これら問題点は、ある知識をコンピュータ上に表現し、利用する場合に具体化する。問題点を大きく区分すると以下の２つになる。

1)　知識とその利用から生じる問題

2)　知識の認識から生じる問題

2)の問題は、知識を知的システムの中に取り入れるために知識自身の概念とその有効性を論じる必要があり、どのようにしたら知識を知識ベースとして明確に表現できかという点にある。また、1)の問題は、ある知識をコンピュータ上の知的なシステムにおける知識として具体的に表現し、利用するには知識をどのようにプログラムするかという点にある。

本報告では、特に1)の問題点に関する課題を取り上げて報告する。

2．知識の利用とその課題

知識を分類し、その形態を明らかにするには、心理学上の問題となるが、コンピュータ上で知的システムにおける知識を具体的に表現していくには次の３点が課題になる。

a)　知識の獲得方式

b)　知識の引き出し方式

c)　推論方式

本研究では以上のそれぞれの課題に対して以下の方式・方法を提案する。a)とb)の課題にアプローチするために知識をルール[9] とフレーム[7] をベースにした表現形式で記述するルールをベースにすると次のような利点がある。

1'　知識の均質な表現を提供する。

2'　個々のルールを付け加えることにより知識ベースを成長させることができる。

3'　推論のための制御構造を容易に付加できる。

また、フレームをベースにした処理過程においては、与えられた状態をどのように処理していくべきかという手段をそのつど特別に記述しなくても適用可能な手段や情報を選び出せる利点がある。この利点をさらに詳細に展開すると以下の４点になる。

1'　情報遺伝の実現

2'　暗黙値の実現

3'　付加手続きの実現

4'　フレーム間の干渉によるフレーム間情報の共有の実現

c)の課題に対しては、不十分な知識のもとでの推論を実現するために、非単調な推論方式であるデフォルト推論の方式を導入する。非単調な推論方式は非単調論理[1] をベースにしている。非単調論理は一階述語論理をベースにしており、新しい公理が導かれることにより、古い定理を無効にすることができる論理である。このような論理は不完全な情報のもとでおこなわれる推論において、たいへん重要なものとなる。本研究では、このような非単調な論理にもとづいた推論方式である非単調推論を計算機上に実現するために、Reiterが示したデフォルト推論のための論理[2] を用いる。この論理は論理的に不十分であるということから多くの議論がなされているがその中で示されているデフォルト規則とその推論形態はコンピュータ上で実現しやすいものとなっている。そこで論理的な不十分な点をルールにおける制御機構を用いて補うことにより、デフォルト推論を達成する。

ところで、以上述べてきた方式を実現するためには以下の問題点が解決される必要がある。

問題点(1)　：ルール適用時におけるルールの競合を解消する方式を決定する。

問題点(2)　：知識ベース管理機構を構築する。

問題点(1) は特にシステムの実行中にルールをルールベースに付加できるためには重要な課題であり、このために我々はルールの優先度を動的に決定するためのルール制御構造を提案した[5][6]。

一方、問題点(2) に関しては、非単調な推論を実現するために、知識ベースの一貫性を保持するための機構を提案した[4] 。

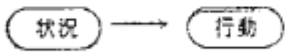
次節より、これら新たに提案した方式について述べる。

3．Rule Dependency

我々は知識を効率良く利用するためにプロダクション・システム（PSと略）に代表されるルールベースシステ

ムの形式を導入する。PSは知識の均質な表現を提供するといった長所がある半面、古い知識（ルール）との関連について何ら考慮することなしに新しい知識の断片をシステムの中に組み込んで行く結果、否定的な推論の結果や、意外な推論結果を導き出す可能性がある。

そこで、ルール制御構造として提案したのがRule Dependency（RDと略）である。RDはルールベースが活性化された場合にどのルールが起動されるべきかを決定する制御構造である。RDは主にルール間の不適切な干渉を取り除くために用いられ、ルール間の優先度を決定するものである。一般に、ルールは図１で示すような状況－行動対になっている。



IF部　　Then部

図１．ルールの形態

ルール間の不適切な干渉は図１で示すIF部の状況が類似するために生じる。例えば、以下のような２つのルールがあるとする。

① IF (:X IS BIRD) THEN (:X CAN FLY)
② IF (:X :Y BIRD) THEN (:X :Y BEANS)

ここで:Xと:Yは変数であり、これらルールに対して次のような事実を与えると

(TARO HAS BIRD) ……(A)

この事実と②のIF部のマッチングが成功し、:XにTAROが、:YにHAS が代入され(TARO HAS BEANS)かの②のTHEN部により生成される。ところが、次のような事実を①、②のルールに対して与えると両方のIF部とのマッチングが成功し、この時点でルール間の競合が生じる。

(TARO IS BIRD)……(B)

そこで、①、②のルールの優先度を決定する必要が生じる。一般的にこの優先度を決定する方式（競合解消の方式）は、ルールを優先度の大きい順に並べておくなどしてあらかじめ決めてしまうことが多い。このことは、ルールをシステムの実行中に随時、ルールベースに付加させることを困難にしている。

我々は、このような困難さを解決するものとしてRDを提案する。

RDはルールのIF部である状況に関してその情報の確からしさに着目し、プログラム間の優先度を記述する。RDは、新たなルールがルールベースに付加された場合、そのルールに付加され、そのルールと競合するルールがある場合には、RDは競合ルールに付加される。

RDは以下のようなアルゴリズムで自動的に決定される。今ここで、ルールのIF部の状況を示すもの（ルールパターン）としてP,Q があり、P とQ をユニファイ(Unify)[8] した結果をR とする。そこで、次のようなP, Q, R, を導入する。

$P = \{\chi \mid MATCH(P,\chi) = T\}$

$Q = \{\chi \mid MATCH(Q,\chi) = T\}$

$R = \{\chi \mid MATCH(R,\chi) = T\}$

ここで、MATCH は、例えば、パターンP の中の変数を任意の指定数でおきかえたχと対になることにより真となる関数である。つまり、P, Q, R は変数を含まないパターンの集合になる。

この時、 $P \subseteq Q$ の場合はP のパターンの方がQ よりも具体的であるとして $P \supseteq Q$ として表わし、P を含むルールの方がQ を含むルールより優先度を大きいものとして決定する。

R は具体的にP とQ の包含関係を決定するために次のように用いられる。

1) $R = P, R \neq Q$ の時（図２参）
　$P \subset Q$ であり、 $\bar{P} \supset \bar{Q}$ となる。

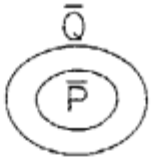


図２．$P \supset Q$ の場合

2) $R \neq P, R = Q$ の時（図３参）
　$Q \subset P$ であり、 $\bar{Q} \supset \bar{P}$ となる。

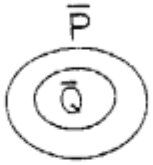


図３．$Q \supset P$ の場合

3) $R = P, R = Q$ のときは（図４参）
　$Q = P$ であり $\bar{Q} = \bar{P}$ である。

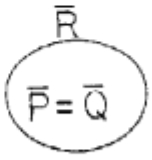


図４．$Q = P$ の場合

4) R ⇏P,R ⇏Q の時（図５参）

この場合、P とQ の間の具体性は決定できない。（ルール間の優先度の決定は自動的にできない。）この時、システムはIF部としてR を有するルールを作成することをユーザーに促すか、もしくは、このようなP とQ の優先順位をユーザに質問する必要がある。我々は、現在のところカレントに入力したルールを優先的に用いている。

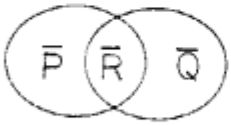


図５．優先度決定が困難な場合

5) R が存在しない時（図６参）

P かQ を含む各々のルールが完全に独立している。（ルール間の競合が生じない。）

図６．ルール間の競合が生じない場合

ここで以上の方法を用いて先ほど示した例における①のルールと②のルールの優先度を決定する。

まずR を求めると、

R ＝Uniyfy((:X IS BIRD), (:X :Y BIRD))

＝(:X IS BIRD)

ここで得られたR は①のIf部と同じになり、図２もしくは図３で示された場合であり、①のルールが②のルールよりも優先度が大きいことになる。

ところで、RDは図７で示されるように各々のルールに付加される。

```
(RULES (AKO (VALUE (ATTACHED!!!)
       (RULE-0001 (METHOD (&!)
                  (PATERN ((GRASP :OBJECT!
                           (:OBJECT SUPPORTS NOTHING!!))
                  (ACTION ! .... ) )
                  (RD (RULE-0002 ***)!!)  - - - - - ②
       (RULE-0002 (METHOD (&!)
                  (PATERN ((GRASP :OBJECT!)))
                  (ACTION ( .... ) )
                  (RD (GROUPED RULE-0001))) - - - ①
   . . .)
```

図７．ルールの表現形式

ルールは、ルール名をスロット名とするフレームで表現され、そのファセットにルールパターンの結合方式、ルールパターン、ルールアクション部それにRDを持ち、それぞれの値が決定される。図７では（イ）と（ロ）がRDに関する記述であり、（イ）によりルールRULE-0001 の方がルールRULE-0002 より優先度が大きいことが表現されている。（ロ）はルールRULE-0002 の優先度決定のための情報がルールRULE-0001 で記述されていることを示している。

我々はこのようなRDを導入することにより、システムの実行中にルールをルールベースに付加することが可能になり、柔軟なルールがベースの構築を実現する。

4．Knowledge Garbage Collector

知識ベースは直接に知識を付加したり消去するといった操作により形成されるほかに、間接的に推論連鎖によっても形成される。推論連鎖の結果、新たに矛盾が生じる場合があり、知識ベースの一貫性を保持するための知識ベース管理機構が必要になる。

我々はこのような知識ベースの一貫性保持のための機構としてKnowlege Garbage Collector(KGCと略す）を提案する。

KGC は我々が導入したデフォルト推論方式を実現する際に、特に重要な機能となっている。デフォルト推論は仮定的な（不明確な）事実を前提として用いることのできる推論方式であり、矛盾が生じた場合にその一貫性をチェックする機能を実現することにより、始めてコンピュータ上に実現可能になる。

KGC を実現するために我々は知識が知識間の関係を保ちながら知識ベースに保存されるように設計する。知識は以下の２つに区別される。

(i)　明確な知識（常に真）

(ii)　仮定的な知識（不明確な知識）

これら、知識の関係構造に、我々は、Data ,dependency [3] の概念を導入する。Data dependency は、主張された知識に付加され、知識間の依存関係を記述するこのような知識間の依存間の依存関係を我々はKnowledge dependency（KDと略）と呼ぶ。

KDを実現するために、上で示した(i) の型の知識を事実ノードとして(ii)の型の知識を仮定ノードとして、知識に固有のノード名を付加する。（図８と図９参照）

図８．Knowledge dependency

矛盾は仮定的な知識が原因で生じるので、矛盾の解決はKBで記述されている知識間の連鎖をたどり、都合の悪い知識を発見し、その反対命題を主張することにより達成する。（前提として閉世界仮説を仮定している。）

KGC の行動の戦略は図９で示す。

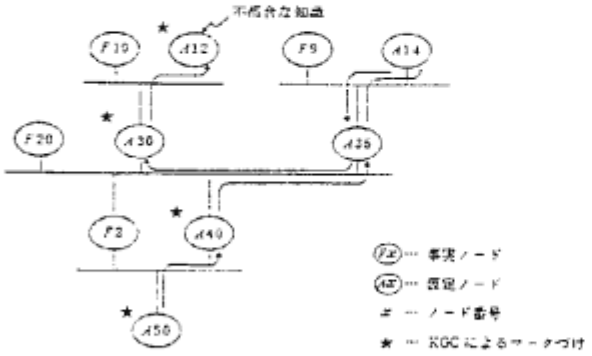


図９．KGC の行動

図９で示すようにKGC は、あるところまでバックトラックして仮定的な知識（図９ではノードA14がこの知識であり、これはデフォルトの値である。）を見つけ、この知識の否定を主張することにより当初の矛盾が可決しない場合、途中まで戻り、他の可能性を調べていく自動バックトラッキングが行なわれる。

現在、KGC はあらゆる形態の矛盾の解決をはかるにはいたっていないが、矛盾の原因となった仮定的な知識などを明示できるので、かなりの程度まで、知識ベースの一貫性のチェックを可能にする。

５．おわりに

我々はRule dependency やKnowledge Garbage Collectionをフレームをベースにしたシステムの中で実現している。Rure dependency を導入したことによりルールベースを柔軟に構築したり、利用したりすることが可能になり、我々はその有効性を確認した。Knowledge Garbage Collector を用いることにより、知識ベースシステムの一貫性を保持するための機能はかなりの程度まで達成されたが、まだ不十分である。しかし、不明確な知識からの推論が可能になる一方、Knowledge dependencyで表現された知識間の連鎖をたどることによる説明機能が容易に実現した。このようなKGC 機能は知識ベース管理機構を構築する際に、重要なや機能となる。

「文　献」

(1) McDermott,D. and Doyle,J. "Non-monotonic logic I", MIT Technical Report Memo 486, 1978

(2) Reiter,R., "A logic for default reasoning " Artificial Intelligence 13, 1980.

(3) Charniak,E., Riesbeck,C.K. and McDermott,V.D. Artifickal Intelligence Programming, Lawrence Erlbaum Associates, Inc. pp193-226

(4) 新谷「知識ベース管理機構について」情報処理学会第26回全国大会、1983.

(5) 新谷「知識ベースシステムにおけるRule dependency の利用」，情報処理学会，知識工学と人工知能研究会 27-2,1982.

(6) Shintani,T., "A Construction Method of Dynamic Rule Base for knowlege based System " ,FUJITSU IIAS, Research Report No.33, 1983.

(7) Roberts,R.B. and Goldstein,I.P. "The FRL Primer" , AI memo No.408, Cambridge, MA:AI Lab., 1977.

(8) Chagn,C. and Lee,R.c, "Symbolic Logic and Mechanical Theorem Proving" , Academic Press.

(9) Waterman,D.A., and Hayes-Roth,F. "Pattern-directed inference systems" , Academic Press.