ICOT Technical Report: TR-120

TR-120

# 論理型言語 Prolog による知識ベースの管理

國藤　進，北上　始
宮地泰造，古川康一

June, 1985



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

# 論理型言語Prologによる知識ベースの管理

國藤　進、北上　始*、宮地　泰造、古川　康一
（財）新世代コンピュータ技術開発機構

[Abstract]　In order to develop a knowledge information processing system, it is necessary for a knowledge base management system to support functions such as knowledge representation, knowledge acquisition, and knowledge utilization. This paper describes some results of our research on the knowledge acquisition functions. Using a meta-programming technique in Prolog, several kinds of basic knowledge acquisition functions are proposed. They are object-level knowledge assimilation, object-level knowledgeaccommodation, and meta-level knowledge assimilation. All functions are implemented in DEC-10 Prolog and such typical examples which show knowledge acquisition features are presented in this paper.

## 1．はじめに

論理型プログラミング言語Prologを用いると、既に知られているように[1),2),3)]、簡単に知識ベースの記述ができ、知識の表現や利用に関する基本問題の解決が比較的容易にできる。人工知能分野、特に知識工学と認知科学、の基礎と応用にとって、最も重要な研究課題は知識の表現・利用・獲得問題である。このうち前二者については、既に多くの著者によって議論されている。本稿は著者らのグループによって行われている論理型言語Prologによる知識ベースの管理の仕方を分り易く要約し、論理型言語での知識獲得問題に対する一接近法を述べる。

## 2．知識獲得とメタ知識

### 2.1　知識獲得問題

計算機システムの知識情報処理システム指向への発展の中で、知識の表現・利用・獲得を支援する人工知能ツールの整備が急速になされつつある。現実世界で人間が知的な問題解決活動を行う際、人間の扱う知識は極めて多様かつあいまいである。そのような知識を人工知能ツールの入力情報として利用するには、知識の表現や利用に関する問題の解決以前に、知識の獲得に関する基本問題が解決されなければならない。知識獲得問題の研究は、知識の表現・利用問題の研究に比べて比較的遅れているが、最も重要な研究課題である。知識獲得問題は、基本的に次の三つの局面からなっている。

1)　知識獲得の第一局面（知識表現同定問題）：
エキスパートの頭の中にあって具体的な問題解決に役立ってはいるものの、それをはっきりとした言葉（記号）で表現したことがないような知識をどう結晶化・表現するかという局面。

2)　知識獲得の第二局面（知識抽出格納問題）：
第一局面を通して得られた新たな知識を、エキスパートからエキスパタイズ（専門的知識）として抽出し、それをどのようにして計算機システムで処理できるような形（記号）に変換・加工し、格納・蓄積していくかという局面。

3)　知識獲得の第三局面（知識ベース管理問題）：
第二局面を通して得られた新たな知識の追加に際し、格納・蓄積されている知識体系の全体を無矛盾かつ系統的に管理していく局面。

エキスパートシステムを作成する過程全体をとってみると、第一局面や第二局面の方がより大切な局面かもしれないが、その方法論もいまだ定まっていない状態である。ここでは第三局面に焦点をあてて、その基本問題を明らかにし、その解決の指針を述べることにする。知識獲得の第三局面において、知識獲得問題は次のような知識体系の管理問題として出現する。第一に、新たな知識の獲得に際し、どのようにして知識体系の全体の矛盾を検査し、その整合性を維持していくかという問題がある。第二に、得られた知識体系の論理体系としての完全性あるいは健全性を維持するにはどうするかという問題がある。第三に、外の世界から付与される知識に適合するように、知識ベースそのものを修正・調節するにはどうするかという問題がある。

---

*1985年 6月 1日より（株）富士通研究所に畑属

その他にももろもろの問題が存在するが、ここではICOTでの知識ベース管理システムの研究開発の途上で得られた知見をもとに、上述のような問題を解決するために導入した最もプリミティブなメタプログラミング技法を中心に知識ベースの管理問題を議論する。

### 2.2 知識とメタ知識

論理型言語Prologで処理可能な知識は、基本的に一階述語論理の部分クラスであるホーン節である。これは論理的には帰結部の不確かさを含まない知識、すなわち結論が高々ひとつしかない知識、の知識表現に適している。このようなホーン節を知識表現の礎とする時、人間のもつ多様かつ不確かな知識をも処理対象とするには、論理型言語としてのPrologでなくプログラミング言語としてのPrologの諸機能（具体的にはメタロジカルな組込み述語[4]）を用いて、そのような知識を実証（demonstrate)あるいは模倣(simulate)する機能を実現してやればよい。これはプログラミング言語としてのPrologのメタロジカル機能を用いて、そのような知識のインタプリタを作ることに相当する。そのようなアプローチの第一歩として、ここではメタ推論機構に基づく知識獲得支援機能の実現法を述べる。

Prologでは、現実世界の対象に関する知識はファクト（事実）またはルール（規則）として取扱われる。ここではファクト型知識やルール型知識の容物を知識ベースと呼ぶことにする。しかしながら、現実世界にはファクトやルールといった対象世界に関する知識以外に、そのような対象知識の使い方に関する莫大なノウハウ型知識がある。このような知識をメタ知識と呼ぶことにすれば、メタ知識は対象知識の使い方に関する知識、あるいはメタ知識の使い方に関する知識として再帰的に定義される。著者らはメタ知識に関する容物としてメタ知識ベースというものを提案し、その知識に基づき知識ベースの管理・更新を行う知識ベース管理システムを研究試作中である。

換言すると、著者らはメタ推論機構を用いて知識ベースの管理を行う。ここに、メタ推論とはメタ知識を用いた推論のことである。メタ知識の基底をなす対象知識としては、論理型言語Prologで表現可能なホーン論理の節集合が利用される。メタ推論機構の提供とは、具体的にはProlog処理系の動きをモニタし、処理系の使い方に関する知識を表現・利用する技術を確立することにある。

## 3．メタ推論

### 3.1 理論的根拠

知識ベースの管理や知識獲得、あるいは協調問題解決といった高度の機能を通常の推論機構の上に実現するには、メタ推論機構の理論的枠組が必要であり、そのことを簡単に述べる。

Lを与えられた論理体系、PをLにおける論理式の有限集合、QをLにおけるある論理式とする。LにおいてPからQが証明可能である時、

$$P \vdash_L Q \qquad (1)$$

と表わすことにする。式(1) が成立する時、QをLの定理と呼ぶこともある。

二つの論理体系L，Mにおいて、LからMへの一対一の対応付け関数（coding function）′が定められているものとする。このときメタ推論を行うためのプリミティブ述語demo[5]を、次のようにして導入する。

$$Pr \vdash_M demo(P',Q') \equiv P \vdash_L Q \qquad (2)$$

ここにPr はMにおける論理式の集合、P′やQ′はそれぞれPやQの関数′の適用結果である。

Weyhrauch [6]は、式(2) を反射原理（reflection principle）と呼んでいる。式(2) は、対象言語上の定理をメタ言語上で実証あるいは模倣するための基本原理とみなすこともできる。Bowen ら[5]は二つの論理体系LとMが同一の形式的言語で表現できる場合に着目し、知識ベースの更新、すなわち知識の同化（knowledge assimilation）におけるdemo述語の重要性を指摘した。彼らはこれを対象言語とメタ言語の融合（amalgamation）と呼んだ。メタ推論方式が有効なのは、論理式の有限集合Pの無矛盾性を保てる範囲であり、基本原理(2) の性質が常に成立する範囲を明確化することが必要である。

### 3.2 メタ推論方式

論理型プログラミング言語Prologには、逐次型のものと並列型のものがある。ここでは最も良く使われており、かつ利用価値の高い逐次型Prologでのメタ推論方式について議論する。

逐次型Prolog（DEC-10 Prolog [4]あるいはその部分集合）上における知識ベース管理機能は、基本的にdemo述語を利用したメタ推論方式を用いて達成される。以下の考察では、前述の論理体系L、Mをホーン論理に制限する。するとdemo述語とは、ホーン論理あるい

はその部分集合での証明可能性（provability ）を判定するためのメタ述語となっている。

メタ推論用demo述語としては次のような形式のものが最も一般的であり、著者ら[7]によって提案された。

①demo(World, Goal, Control, Result)

このdemo述語は、ある世界World において、与えらえた制御情報Control に従って、与えられたゴールGoalを証明していくプロセスを実際に示し、その証明のプロセスから必要なメタ情報Resultを抽出する。

上述のdemo述語①に対して、次のような省略した形式のdemo述語がしばしばある種の応用では有効である。その理由は、主としてそれらの実現の容易さと性能の向上のためである。

② demo(World, Goal, Result)
③ demo(World, Goal)

なお対象世界として、Prologの内部データベース自身を用いる場合、①～③の第１引数World は省略できる。例えば、次のメタ述語④は"Prolog interpreter in Prolog"[4]として知られている。

④ demo(Goal)

### 3.3 実現法

著者らは、Prolog上でのメタ推論用プリミティブdemo述語のDEC-10 Prolog 上での実現法を示し、それを用いた各種のメタ推論エンジンの構築法とその利用法[8),9)] を示した。すなわち、最も簡単なPrologインタプリタである純粋Prologインタプリタをヒントにし、次のような対象知識やメタ知識の表現法とそれに対するdemo述語の実現法を提案した。

まずPrologの節集合に対して、その対象知識の世界を識別するユニークな名前、例えばkb、を付与するものとする。この対象世界の名前kbをプリンシプル・ファンクタとして、その対象世界に属するルール型知識やファクト型知識のそれぞれを、Prologの内部データベースに次のような形式で表現・登録するものとする。

```
kb((P:-Q1,Q2, …, Qn)).        (3)
kb(P).                        (4)
```

ここにPやQi （i = 1, …, n ）は肯定的あるいは否定的な素命題である。

メタ知識としては、次の形式のみを扱うものとする。

```
kb(( :-Q1,Q2, …, Qn)).        (5)
```

ここにQj （j = 1, …, n ）は肯定的あるいは否定的な素命題である。このメタ知識(5) は、“特定の対象世界kbにおいて、Q1 かつQ2 かつ…かつQn が成立すれば矛盾する”と、宣言的に解釈できる。式(5 ) は簡単のため、次のように略記することもある。

```
mkb((Q1,Q2, …, Qn)).          (6)
```

式(5) （あるいは式(6) ）は、通常はPrologのゴールとしてしか表現可能でないヘッドが空のホーン節を、Prologの知識として表現し利用可能とするために導入したものである。

以上の準備をした後、対象知識(3) 、(4) あるいはメタ知識(6) を処理対象とする２引数demo述語③を、DEC-10 Prolog で次のようにして実現する。

```
demo(W,true):-!.                              (7)
demo(W,(G1;G2)):-
  !,(demo(W,G1);demo(W,G2)).                  (8)
demo(W,(G1,G2)):-
  !,demo(W,G1),demo(W,G2).                    (9)
demo(W,G):-wclause(W,G,B),demo(W,B).          (10)
wclause(W,H,B):-
  X=..[W,C],X ,wclause2(C,H,B)                (11)
wclause2((H:-B),H,B):-!.                      (12)
wclause2(H,H,true).                           (13)
```

式(7) は、対象世界Wにおいて恒等的に真なゴールtrueは常に成功することを意味する。式(8) は、Wにおいて選言ゴール（G1 ；G2 ）が成功するためには、WにおいてゴールG1 が成功するか、あるいはWにおいてゴールG2 が成功することを意味する。式(9) は、Wにおいて連言ゴール（G1 , G2 ）が成功するためには、WにおいてゴールG1 が成功し、かつWにおいてゴールG2 が成功することを意味する。式(10)は、Wにおいて(7) ～(9) が成立せず、一般にゴールGが成功するには、WにおいてGとユニファイ可能なヘッドをもつホーン節がある限り、そのボディBを取出し、WにおいてBをゴールとして成功すればよいことを意味する。ここに対象言語の知識とメタ言語の知識とを同一レベルで取扱うことを可能とするために、ホーン節（プログラム）を述語の項（データ）に変換・逆変換するメタロジカルな組込み述語“=..”を利用して

いるのに注意されたい。

このメタ述語demo③をベースに①、②、および④のようなメタ推論用プリミティブを構築していくのは易しい。4章ではそのようなプリミティブの構築法をメタプログラミング技法としてまとめることにする。

## 4．メタプログラミング

### 4.1 否定の導入

Prologにおける否定は、閉世界仮説（CWA:Closed World Assumption）[10]のもとでの否定であり、与えられた知識ベースから証明できないものは否定されたと解釈される。従って、CWAのもとでの否定not(_)を、demo述語に導入するのは極めて易しい。例えば、式(7)と式(8)の間に次式(14)を、式(13)の後にnot(_)の定義式(15)、(16)を挿入するだけでよい。

```
demo(W,not(G)):-!,not(demo(W,G)).          (14)
not(P):-P,!,fail.                          (15)
not(_).                                    (16)
```

このようにして導入されたnot(_)は、実は非単調論理としての諸性質を満たす。非単調論理は論理体系として現在急速に整備中であるが、一般にその取扱いは極めて慎重を要する。このあたりの議論については、5.4節を参照されたい。

### 4.2 カットの導入

Prologを使って、実際的問題を解くには、カットオペレータ"!"を処理するdemo述語を提供しなければならない。カットオペレータの動きをメタ言語上で模倣するdemo述語は、プログラムの状態を保持するための制御用ダミー変数Cutを導入することによって、次のようにして実現される。ただし、ここでは簡単のため、対象世界としてPrologの内部データベースそのものを使用することにする。

```
demo(true,_):-!.                           (17)
demo(!,_).                                 (18)
demo(!,cut).                               (19)
demo((G1;G2),Cut):-
  !,( demo(G1,Cut);demo(G2,Cut)).          (20)
demo((G1,G2),Cut):-
  !,demo(G1,Value),
  (Value==cut,Cut=cut,!;demo(G2,Cut)).     (21)
demo(G,Cut):-
  clause(G,B),demo(B,Cut),
  (Cut==cut,!,fail;true).                  (22)
demo(G,_):-G.                              (23)
```

ここにclause(G,B)はゴールGとユニファイするヘッドをもつホーン節のボディBを取出すDEC-10Prologの組込み述語である。

このdemo述語は、対象言語レベルの"!"の最初の出現時に、節(18)でカットをかける準備を行なう。この時、節(21)の変数Valueはまだ例示されていない。そこでその後、対象言語レベルでfailすると、メタ言語レベルのバックトラックが起こり、demo(G1,Value)のValueにcutが節(19)により例示される。すると呼出した節のdemo(G,Cut)のCutにcutが例示される。その結果、節(22)によってバックトラックがカットされる。

このdemo述語の自然な拡張として、DEC-10 PrologのインタプリタをDEC-10 Prologそのもので書くことができる。実際、DEC-10 Prologインタプリタは DEC-10 Prologコンパイラの基本命令を用いて書かれている。

### 4.3 証明のプロセス情報の抽出

demo述語を用いると証明の結果であるtrueやfail情報のみならず、証明のプロセスをメタ情報Resultとして抽出できる。その最も簡単な利用法は、証明のプロセスからトレース情報を抽出する機能[7]の実現である。この機能を実現するには、前述の知識ベース(3)、(4)のデータ構造を若干修正[11]すればよい。すなわち、それらに対して、ルールやファクトの通し番号RN、FNを付与した次のようなデータ構造(24)、(25)をトレース情報抽出用demo述語(26)～(30)の処理対象とする。

```
kb(RN,((P:-Q1,Q2,…,Qm)).                  (24)
kb(FN,P).                                  (25)
demo(W,true,[]):-!.                        (26)
demo(W,(G1;G2),T):-
  !,(demo(W,G1,T);demo(W,G2,T)).           (27)
demo(W,(G1,G2),T):-
  !,demo(W,G1,T1),demo(W,G2,T2),
  append(T1,T2,T).                         (28)
demo(W,G,T):-
  tclause(W,G,B,T1),demo(W,B,T2),
  append([T1],T2,T).                       (29)
demo(W,G,[]):-G.                           (30)
```

```
tclause(W,H,B,T):-
  X=..[W,T,C],X ,wclause2(C,H,B)          (31)
```

例題1[14]　次のような知識ベースが与えられているものとする。

```
kb(n1,append([],X,X)).                    (32)
kb(n2,(append([U|X],Y, [U|Z]) :-
        append(X,Y,Z))).                  (33)
```

この知識ベースに対して、次のようなゴールを与えると、その実行結果は下記のようになる。

```
| ?- demo(kb,append([a,b,c],[d,e],X),
          Trace).                         (34)
X = [a,b,c,d,e]
Trace = [n2,n2,n2,n1]
yes                                       (35)
```

上述の例に見られるようにユーザが定義する推論エンジンに相当するdemo述語や知識ベースの定義を、多少変更するだけで、demo述語に高度な機能を簡単に付与できる。論理型言語における知識表現の基本問題である多重世界のもとでの多重継承（multiple inheritance）の実現と知識の動的陳腐化の実現は、demo述語をベースにすればきわめて容易であることを指摘しておく。しかしながら、そのような話題は本稿の主題である知識獲得の問題から離れるので、それらの実現については省略したい。

## 5．知識同化機構の実現

### 5.1　知識同化

知識同化とは、与えられた知識ベースに外の世界から新しい知識を無矛盾かつ系統的に取得する過程である。説明の便宜上、まずファクト型知識の同化機構の実現を述べる。ルール型知識の同化については 5.4節で述べる。

さてCurrkbを与えられた現存の知識ベース、Inputをその知識ベースに取り込みたい新知識とする時、知識同化の基本的考え方[8),9)]は次のような抽象的プログラムとして実現できる。

```
assimilate(Currkb,Input,Currkb):-
  demo(Currkb,Input).                     (36)
assimilate(Currkb,Input,Currkb):-
  demo(Currkb∪Input,false).               (37)
assimilate(Currkb,Input,Newkb):-
  Info∈Currkb,Interkb=Currkb-Info,
  demo(Interkb∪Input,Info),
  assimilate(Interkb,Input,Newkb).        (38)
assimilate(Currkb,Input,Currkb∪Input):-
  independent(Currkb,Input).              (39)
```

ここに∪，－，∈ は集合論の通常の記法である和集合、差集合、メンバシップの意味である。

この知識同化述語の宣言的解釈は次の通りである。式(36)は、もし現知識ベースCurrkbから入力知識Input が証明可能ならば、その知識Input を知識ベースCurrkbに同化しないことを意味する。式(37)は、入力知識Input を知識ベースCurrkbに同化すると矛盾が生じる場合、矛盾を生じる知識Input は知識ベースCurrkbに同化しないことを意味する。式(38)は、現知識ベースCurrkbからある知識Infoを取出し、残りの知識ベースInterkb に入力知識Input を挿入したものから取出された知識Infoが証明可能であり、かつその知識ベースInterkb に入力Input を同化した知識ベースがNewkb となる場合、そのような知識Infoは冗長な知識として除去し、知識ベースはNewkb に更新することを意味する。式(39)は、入力知識Input が現知識ベースCurrkbと独立ならば、その知識Input を知識ベースCurrkbにそのまま同化することを意味する。

本述語は、知識ベースに新知識を同化するには、証明可能性、矛盾性、冗長性、独立性という四つの概念のこの順序での検査と対応する知識ベース更新の必要性を示している。Bowen らの考え[5)]と異なり、矛盾性検査を冗長性検査より先に評価することの意義は、冗長性検査手続き(38)は知識ベースの更新を含み、更新された知識ベース自体が矛盾していると、この手続きが無意味となるからである。

### 5.2　知識同化プログラム[8)]

知識同化述語(36)～(39)のDEC-10 Prolog での実現例を与える。ルール型知識(3) とファクト型知識(4) とからなる知識ベース（肯定的知識ベース）とメタ知識(6) からなるメタ知識ベース（否定的知識ベース）とを与える。それらの名前を、それぞれPKB とNKB とする。与えられた肯定的知識ベースPKB や否定的知識ベースNKB に対して、ファクト型入力知識Pinputを同化するためのPrologプログラムpassimilate ([PKB,NKB],Pinput)は、次のようにして実現される。

```
passimilate([PKB,NKB],Pinput):-
  demo(PKB,Pinput),
  message(' Pinput is deducible from PKB ! ',
          PKB).                                    (40)
passimilate([PKB,NKB],Pinput):-
  metaasserta(PKB,Pinput,PI),
  (metacall(NKB,X,NKBX),demo(PKB,X),
   message(' Pinput is inconsistent
            with KB ! ',PKB),
   retract(PI);
   retractf(PI)).                                  (41)
passimilate([PKB,NKB],Pinput):-
  PI=..[PKB,Pinput],metacall(PKB,X,PKBX),
  ((X=(Xh:-Xb);X=not( _ ))->fail;
   ((retract(PKBX),asserta(PI),demo(PKB,X))
     ->(retract(PI),
        message0(' Redundancy is found
                   for  PKB ! '),
        (passimilate([PKB,NKB],Pinput);true));
    asserta(PKBX),retractf(PI))).                  (42)
passimilate([PKB,NKB],Pinput):-
  metaasserta(PKB,Pinput,PI),
  message(' Pinput is acquired
            in PKB ! ',PKB).                       (43)
metaasserta(K,X,KX):-
  KX=..[K,X],asserta(KX).                          (44)
metacall(K,X,KX):-KX=..[K,X],KX.                   (45)
retractf(X):-retract(X),fail.                      (46)
message(M,KB):-
  ttynl,display(M),ttynl,listing(KB).              (47)
message0(M):-ttynl,display(M),ttynl.               (48)
```

この肯定的知識同化プログラムの実行例を示す。

例題2　肯定的知識ベースpkb にファクト(49)、(50)とルール(51)が与えられているとする。この時、対象言語上の質問"?-parent(shimako,izumi )." に対応するメタ言語上の同化問合せ(52)に対して、実行結果(53)はメタ言語上で証明可能なことを示している。

```
pkb(father(shimako,susumu)).                       (49)
pkb(mother(shimako,izumi)).                        (50)
pkb((parent(X,Y):-
      father(X,Y);mother(X,Y))).                   (51)
| ?- passimilate([pkb,nkb] ,
                 parent(shimako,izumi)).           (52)
 Pinput is deducible from PKB !                    (53)
```

例題3　論理データベースに対する統合性制約(integrity constraint)は、典型的なメタ知識の使用例である。例えば、両親と子供の間の血液型の遺伝学的検査手続きは、そのような知識の一種であり、否定的知識ベースnkb 部に(54)のような表現形式で記述できる。

```
nkb((father(X,F),blood_type(F,FT),
     married(F,M), blood_type(M,MT),
     genes_match(FT,MT,CBT),
     blood_type(X,BT),
     not(member(BT,CBT)))).                        (54)
```

上式(54)は、子供X の父親がF であり、F の血液型がFTであり、F がM という人と結婚しており、M の血液型がMTであり、FTとMTとから可能な子供の血液型のリストがCBT であり、X の血液型がBTであり、かつBTがCBT の要素でないことはありえないことを表わしている。さて、則夫さん（血液型ａ）、由美子さん（血液型ｏ）ご夫妻に、女の子供陽子さん（血液型ｂ）が生まれたとする。そこで(55)に見られるように、陽子さんの父親が則夫さんという知識を同化しようとすると、実行結果(56)は統合性制約(54)との間に矛盾が生じてしまうことを示している。

```
| ?- passimilate([pkb,nkb],
                 father(youko,norio)).             (55)
 Pinput is inconsistent with KB !                  (56)
```

例題4　肯定的知識ベースに(49)～(51)、および(57)～(59)の知識が付与されているとする。この時、"father(susumu,toshio)" という知識を同化しようとすると、(61)～(66)に見られるように、冗長な二つの知識(57)、(58)を除去し、独立な知識"father(susumu,toshio)" をpkb に挿入している。

```
pkb(ancestor(shimako,toshio)).                     (57)
pkb(parent(shimako,susumu)).                       (58)
pkb((ancestor(X,Y):-
       parent(X,Y);
       parent(X,Z),ancestor(Z,Y))).                (59)
| ?- passimilate([pkb,nkb],
                 father(susumu,toshio)).           (60)
 Redundancy is found for PKB !                     (61)
 Redundancy is found for PKB !                     (62)
 Pinput is acquired in PKB !                       (63)
pkb(father(susumu,toshio)).                        (64)
```

```
pkb(mother(shimako,izumi)).                    (65)
pkb(father(shimako,susumu)).                   (66)
```

以上示してきたのは肯定的ファクト型知識同化プログラムである。否定的ファクト型知識の同化プログラムも、プログラム(40)～(43)を多少修正[8]すればできる。

### 5.3 例外値の扱い[7]

知識表現言語や知識表現システムの分野では常識であるが、is_a 階層による性質継承という知識表現力の導入は必然的に例外値(exception value) の記述をも許容しなければならない。すなわち例外の記述を許容して初めて、暗黙推論が有意なものとなる。そこで本項では、前述の知識同化プログラムに例外値を操作する機能を追加する方法を述べる。

肯定的知識同化プログラムは、前述の(40)～(43)に見られるように証明可能性、矛盾性、冗長性、独立性の順で知識ベースを管理していく。例外値の存在は、実は論理体系としては非単調論理なので、基本的にそのような論理での証明可能性概念をdemo述語に反映しなければならない。そのための最も基本的なアイディアは、対象言語上のfailがメタ言語上のfailに波及しないようにすることである。そのようにして構成したdemo述語が、(67)～(75)に与えられている。このdemo述語demo(KB,Goal,Cut,Exception) は、その知識ベースKBにおいてゴールGoalを証明し、アクセスした例外値リストを第４引数Exception にリターンする。第３引数Cut はカットオペレータ様作用の補助変数である。

```
demo(KB,true,Cut,[]):-!.                      (67)
demo(KB,false(X),Cut,[X]):-!.                  (68)
demo(KB,!,Cut,[]).                             (69)
demo(KB,!,cut,[]).                             (70)
demo(KB,(P;Q),Cut,L):-
  !,(demo(KB,P,Cut,L);demo(KB,Q,Cut,L)).       (71)
demo(KB,(P,Q),Cut,L3):-
  !,demo(KB,P,Value,L1),
  (Value==cut,Cut=cut,!;
   demo(KB,Q,Cut,L2),append(L1,L2,L3)).        (72)
demo(KB,P,Cut,L):-
  clausekb(KB,P,Q),demo(KB,Q,Cut,L),
  (Cut==cut,!,fail;true).                      (73)
demo(KB,P,Cut,[]):-P.                          (74)
clausekb(KB,P,Q):-wclause(KB,P,Q).             (75)
```

ここに対象言語上のfailが，式(68)に見られるように、メタ言語上ではfalse(X)と成功値として取扱われているのに注意されたい。

前述の知識同化プログラム(40)～(43)に出現するdemo述語を、このdemo述語(67)～(75)に置換え、かつ式(40)を次式(76)に更新すれば、例外値を取扱う知識同化プログラムができる。

```
passimilate([PKB,NKB],Pinput):-
  demo(PKB,Pinput,_,Exception),
  (Exception==[]->
    message(' Pinput is deducible from PKB !',
            PKB);
   message(' Pinput is exception !',PKB)).(76)
```

例題５　次のような知識が肯定的知識ベースに与えられていたとする。

```
pkb((can_fly(X):-
     bird(X),(cannot_fly(X),!,false(X);
              true))).                         (77)
pkb(bird(condor)).                             (78)
pkb(bird(ostrich)).                            (79)
pkb(cannot_fly(ostrich)).                      (80)
```

式(77)は、“X が飛ぶことができるためには、X が鳥であり、かつX が飛ぶことができなければそのX を例外値として処理し、そうでなければ無条件に成功する”ことを意味している。

上述の知識(77)～(80)に対して、例外値を含む知識同化の入出力例を示す。

```
| ?- passimilate([pkb,nkb],
                 can_fly(ostrich)).            (81)
 Pinput is exception !
yes                                            (82)
| ?- passimilate([pkb,nkb],
                 can_fly(condor)).             (83)
 Pinput is deducible from PKB !
yes                                            (84)
| ?- passimilate([pkb,nkb],
                 can_fly(bat)).                (85)
 Pinput is acquired in PKB !
yes                                            (86)
```

### 5.4 ルール同化[12]

Prologの知識ベースに蓄積し利用すべき知識は、多くの場合、ファクト型知識だけでなくルール型知識を含んでいる。そこで更新される知識ベースを処理対象とするような応用においては、ルール型知識の知識同化プログラムを作成しなければならない。そのためには基本的に、ルール型知識をゴールとするようなdemo述語を構築する必要がある。残念ながら、ほとんどのdemo述語の研究は、Prologのゴールが存在束縛されていたせいもあり、存在束縛型問合せに対するメタ推論方式の研究であった。しかしながら、上記問題を解決するには全称束縛型問合せに対するメタ推論方式の研究が必須となる。Prologの否定やカットが単調論理の枠を越えるので、そのようなメタ推論用demo述語は慎重に取扱わないと、すぐ反射原理(2) の枠をはみでてしまう。ルール型知識をゴールとするメタ述語demoが論理体系としておかしな振舞いをしないためには、反射原理(2) が成立する範囲で使用すべく、知識ベース、メタ知識ベース、および入力知識に対してそのクラスを厳しく制限しなければならない。

ルール同化問題を一般的に解くことの困難さを認識した上で、最も簡単な場合についてルール同化方式を述べる。より一般的な場合のルール同化については文献[12] を参照されたい。Prologで最も関心のあるCWAの場合については、最近CWA のもとでのルール同化に役立ちそうな結果[13),14)] が出始めた段階である。

Pure Prolog with Negation(PPN)[15)] という論理体系を前提として、ルール同化方式を述べることにする。PPN は一階ホーン論理の最も素朴なクラスで、肯定的知識ベース(3) 、(4) や否定的知識ベース(6) に出現するリテラルが全て肯定的リテラルという制限をもつものである。これに対応し知識ベースに同化する新知識Input も、次のような形式のものに制限する。

$$\exists X1 \cdots \exists Xn\ h(X1, \cdots, Xn) \qquad (87)$$

$$\forall X1 \cdots \forall Xn(p(X1, \cdots, Xn) :- b(X1, \cdots, Xn)) \qquad (88)$$

ここに h (X1, …, Xn) や b (X1, …, Xn) は肯定的リテラルの連言、 p (X1, …, Xn) は肯定的リテラルそのものである。PPN 用demo述語は、次の手続き[12)] によって実現される。

〈手続き１〉

(a) Input が形式(87)なら、Pure Prolog の場合のdemo述語(7) ～(10)に従う。

(b) Input が形式(88)ならば、各変数X1, …, Xnに、この肯定的知識ベースや否定的知識ベースに含まれない定数を割当てる。この割当てを$\theta \equiv <(X1, c1), \cdots, (Xn, cn)>$とする。c1, …, cnはいわゆるスコーレム定数である。

(c) 肯定的知識ベースに b (c1, …, cn) というファクトの連言をアサートし、否定的知識ベースに p (c1, …, cn) というファクトをアサートする。

(d) 否定的知識ベースから任意の否定的知識をとってきて、それを肯定的知識に反転し、肯定的知識ベースに対する形式(87)の問合せとして証明してみる。この時、もし成功すれば、上述の知識(88)は与えられた肯定的知識ベースと否定的知識ベースとから証明可能となる。

知識同化プログラム(36)～(39)のdemo述語を、上述の〈手続き１〉で置換すれば、PPN の場合の知識同化プログラムとなる。

例題６　一階述語論理ではルール(89)と(90)とからルール(91)は簡単に証明できる。しかしながら、Prologでは〈手続き１〉の技法を要する。すなわち(89)、(90)、(92)に対して、ゴール"?-c."が証明できることにより、(91)が証明できることが分かる。

```
b:-a.                                   (89)
c:-b.                                   (90)
c:-a.                                   (91)
a.                                      (92)
```

例題７　ルール(93)と(94)が与えられているとする。

```
above(X,Y):-on(X,Y).                    (93)
above(X,Y):-on(X,Z).above(Z,Y).         (94)
```

この時、ルール(95)が冗長であることを証明するためにファクト(96)と(97)を肯定的知識ベースにアサートし、ゴール(98)を実行してみる。

```
above(X,Y):-on(X,Z),on(Z,Y).            (95)
on(c1,c3).                              (96)
on(c3,c2).                              (97)
?- above(c1,c2).                        (98)
```

するとゴール(98)の評価結果は、確かにyes となる。

なお〈手続き１〉に関連したルール同化プログラムが文献[16),17)] に見られる。

## 6．知識調節機構の実現

### 6.1 知識調節

知識調節とは外の世界からの入力知識に適合するように、知識ベースそのものを無矛盾かつ系統的に修正していく過程である。著者らは知識調節機構[16]を、外部世界からのファクト型入力知識に応じて、そのような入力知識を証明しうるルール型知識（推測モデル）を、帰納的な推論(inductive inference)によって構築する機構と位置付ける。本機構の最も中核とする部分は、Shapiro のモデル推論アルゴリズムのdemo述語による改良版である。Shapiro のモデル推論システム(MIS: Model Inference System)[18]のユーザは、誤りのないファクト型知識をMIS に与えることが要請される。すなわちMIS は完全教師付きの学習システムの一種であることに注意されたい。

Shapiro のモデル推論アルゴリズムの概略をFig. 1に示す。本アルゴリズムはどのようなホーン型の知識も、矛盾"□"を出発点とし、精密化オペレータ（Refinement Operator)を適用し、次々と仮説を精密化することにより、目標とするモデルに接近できると言っている。推測Tは仮説の集合である。Tからユーザが与えた否定的知識を証明できた時、推測Tが強すぎると呼び、逆にTからユーザが与えた肯定的知識を証明できない時、推測Tが弱すぎると呼ぶ。本アルゴリズムはTが強すぎもなく弱すぎもなくなった時、その時点までに読み込まれた全ての知識に対して、適切な推測モデルTを推論したとして、Tを出力するアルゴリズムである。

Tを{□}と設定する。
Repeat
　つぎの事実を読む。
　Repeat
　　While 推測Tが強すぎる　do
　　　矛盾探索アルゴリズムを適用し，
　　　Tから反証を与える仮説を取り除く。
　　While 推測Tが弱すぎる　do
　　　先に反証を与えた仮説を，さらに
　　　精密化したものを，Tに加える。
　Until 推測Tが強すぎることもなく弱すぎることもない。
　　(読み込まれた事実に関する限り)
　推測Tを答えとして出力する。
Forever

Fig. 1　Model Inference Algorithm

### 6.2 知識調節プログラム

モデル推論アルゴリズムをベースに、DEC-10 Prolog で知識調節プログラムを実現した。そのプログラムの一部を、式(99)～(102) に掲げる[16]。

```
model _ inference(KBN, IC_name, Concept):-
  nl,ask_for('Next fact(sentence,true/false)
              or end ',Fact),
  (Fact=end;
  (Fact=check->
   model_inference1(KBN, IC_name, _);
   (Fact=(P,V),verify(P=Concept);
   nl,write('* Error Concept  Form.'),fail),!,
   Fact=(P,V),(V=true;V=false )->
   assert _ fact(P,V),
   model_inference1(KBN, IC_name, P);
    write('!Illegal input'),nl),!,nl,
  model _ inference(KBN, IC_name,
                    Concept)).              (99)
model _ inference(KBN, IC_name, Concept):-
  nl,ask_for('Next fact(sentence,true/false)
              or end ',Fact),
  (Fact=end;
  (Fact=check->
   model_inference1(KBN, IC_name, _);
   Fact=(P,V),(V=true;V=false )->
   assert _ fact(P,V),
   model_inference1(KBN, IC_name, P);
    write('!Illegal input'),nl),!,nl,
  model _ inference(KBN, IC_name,
                    Concept)).              (100)
model _ inference1(KBN,IC_name,P):-
  write('!Checking fact(s)...'),ttyflush,
  (fact(P,false),model_demo(KBN,P)->
   nl,false _ solution(KBN,IC_name,P),
   model_inference1(KBN, IC_name, _);
                    % (Too Strong)
   fact(P,true),not(model _ demo(KBN,P))->
   nl,missing _ solution(KBN , IC_name,P),
   model_inference1(KBN, IC_name,P);
                    % (Too Weak)
  write('no error found.'),nl).             (101)
model _ demo(KBNl,Goals):-
  demo(KBNl,Goals,[[],[],Cut,50],
       [Result,Stack]),
```

```
(Result \==true,!;true),
(Result=overflow->
 nl,stack _ overflow(KBNL, Goals,Stack),
 model_demo(KBNL,Goals );
 true).                                    (102)
```

ここに式(101)が前述のモデル推論アルゴリズムに対応する。また式(102)は③型のdemo述語で、demo(World,Goal,[C1,C2,C3,C4],[R1,R2])という構造をしている。R1は証明の結果としてtrueかoverflowの表示、R2はoverflow時の証明木を表わす。またC1はWorldに追加してアクセスする知識、C2はC1では使われない知識をマスクする識別子、C3はカットオペレータ制御用補助変数、C4は最大分解ステップ数である。このdemo述語のインプリメンテーションの詳細については、Appendix 1を参照されたい。

北上らのシステム[16]の特徴は、メタ知識としての統合性制約の導入により否定的知識の記述力が強まったことと、反ばく仮説保持による再計算の禁止に基づくモデル推論の高速化、等の工夫にある。

### 6.3 知識同化や知識調節の実行例

ICOTオープンハウスで行われたデモンストレーション[19]を例題にとり、知識同化や知識調節の機能をもつ知識獲得支援システムの実行例を示す。

例題8 Prologでは自然言語の文法や辞書はDCG(Definite Clause Grammar)の形式で記述されることが多い。そこで簡単な英文の文法辞書を下記のように与えるものとする。

```
np(X,Y):-n(X,Y).                          (103)
vp(X,Y):-vi(X,Y).                         (104)
vp(X,Y):-vt(X,Z),n(Z,Y).                  (105)
pp(X,Y):-p(X,Z),n(Z,Y).                   (106)
n([hermia | X],X).                        (107)
n([forest | X],X).                        (108)
n([lysander | X],X).                      (109)
vi([walks | X],X).                        (110)
vt([loves | X],X).                        (111)
p([in | X],X).                            (112)
```

ここに(103)～(112)はそれぞれ名詞句の定義、動詞句の定義、動詞句の別定義、前置詞句の定義、名詞hermiaの辞書登録、名詞forestの辞書登録、名詞lysanderの辞書登録、自動詞walksの辞書登録、他動詞lovesの辞書登録、前置詞inの辞書登録である。

ついで(113)で与えられるようなメタ知識を統合性制約として与える。このメタ知識は、文法知識を適用して得られる任意の文の解析結果に例示されていない変数が存在するようなことがあると矛盾していることを意味している。

```
is_inconsistent(insert,_):-
  s(X,[]),exist _ variable(X).            (113)
```

(103)～(113)の知識のもとで、文に関する誤った文法規則(114)は拒否され、正しい文法規則(115)は受理される。ここにメタ知識(113)がなければ、両規則とも受理される。

```
s(X,Y):-np(X,Z),vp(Z,W).                  (114)
s(X,Y):-np(X,Z),vp(Z,Y).                  (115)
```

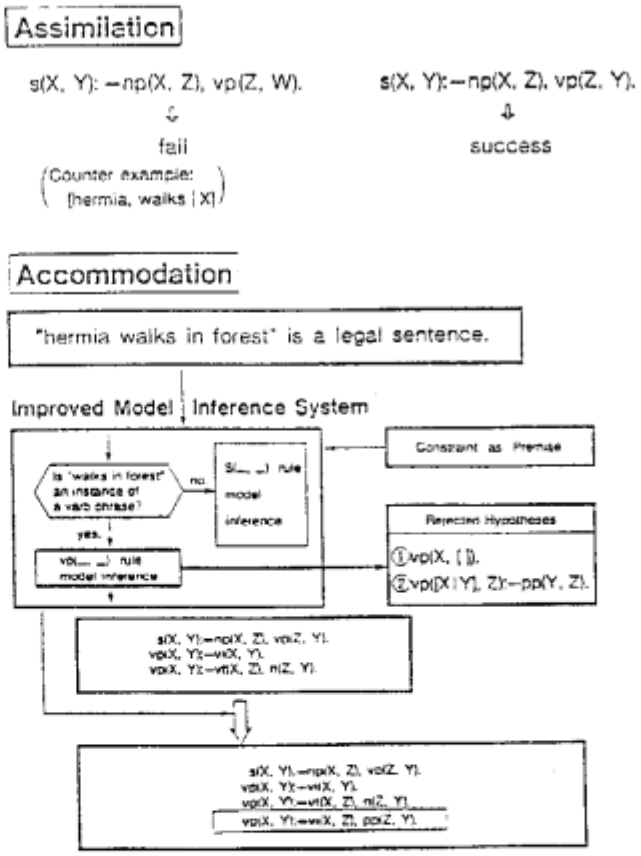


Fig. 2 Object-level Assimilation and Accommodation

例題９　(103) ～(113) 、(115) の知識が与えられているとして、知識調節プログラムの実行例をFig. 2に与える。

まず正しい文の実例 "hermia walks in forest" がプログラムの入力データとして与えられる。この文は不幸なことに文の定義(115) ではパーズできない。そこでシステムは"walks in forest" は動詞句の実例かと聞き返してくる。これに対してユーザが"yes" と答えると、システムは"walks in forest" を動詞句として認識する文法規則を生成し始める。この場合、先程のメタ知識(113) が極めてすぐれた制約条件となっており、改良されたモデル推論アルゴリズムは若干の仮説生成の後、直ちに文法規則(116) を見出すことに成功する。

```
vp(X,Y):-vi(X,Z),pp(Z,Y).                    (116)
```

この文法規則(116) を知識ベースに追加すると、確かに先程の例文"hermia walks in forest " は、前置詞句を含む文として受理される。

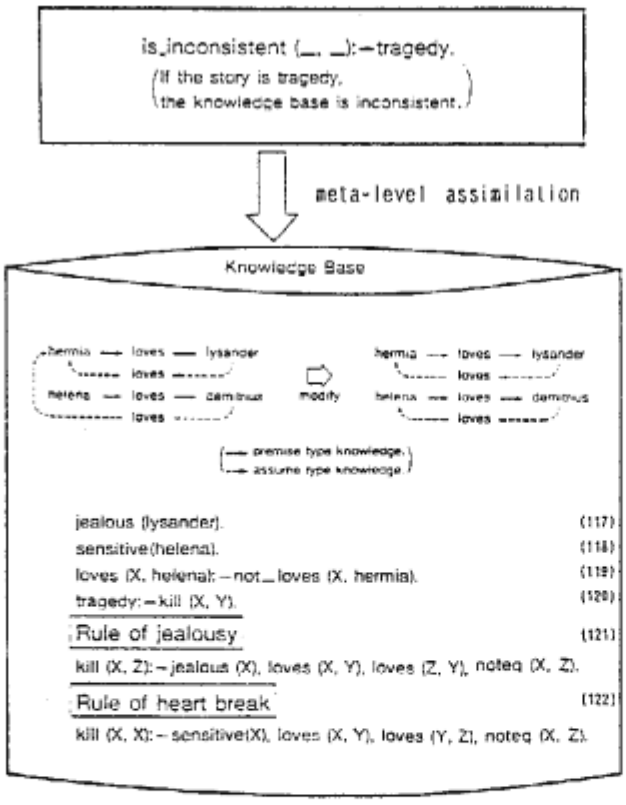


Fig. 3　An Example of　Belief Revision

例題１０　demo述語によるメタ推論方式を用いると、不確かな知識と確かな知識の混在する知識ベースの全体を管理することができる。シェークスピアの“真夏の夜の夢”に登場する人物の恋愛関係を素材に、次のように不確かな知識（人工知能分野では、しばしば信念(belief)と呼ばれる）が翻意していく有様を模倣した。この過程は信念の翻意(belief revision) とも呼ばれる。

最初、Fig. 3に示されているように、(117) ～(122) や(123) ～(124) で示される確かな知識、および(125) ～(126) で示される不確かな知識が与えられているとする。

```
loves(hermia,lysander).                       (123)
loves(helena,demitrius).                      (124)
loves(lysander,hermia).                       (125)
loves(demitrius,hermia).                      (126)
```

ここに(121) は“嫉妬深い人は、恋敵を殺す”というルールであり、(122) は“神経過敏な人は、恋敵が出現すると自殺する”というルールである。

この時、シェークスピアの悲劇だから当然のことであるが、“（この物語においては）悲劇が起こると矛盾している”というメタ知識を追加したとする。すると(121) や(122) の故に、この知識ベースの全体には矛盾が生じる。システムは矛盾の原因を不確かな知識(125) 、(126) に求め、これらを修正し、無矛盾な知識ベースにしようと試行錯誤する。最終的にシステムは、不確かな知識(126) を(127) に修正する。

```
loves(demitrius,helena).                      (127)
```

Fig. 3を見れば直ちに分かるように、得られた恋愛関係(123) ～(125) 、(127) は互いに“相思相愛”の関係にある。

## 7．おわりに

論理型言語Prologを用いた知識ベースの管理について、知識工学の主要課題のひとつである知識獲得問題との関連で論じた。特に知識ベース管理の基本的考え方である知識同化機構や知識調節機構に

ついて、Prologによるプログラミング例や分り易い例題を用いて解説した。知識同化プログラムや知識調節プログラムの実現の過程で、demo述語によるメタプログラミング技法が生まれた。

メタプログラミング技法は知識ベースのエディタやデバッガの開発、論理プログラムの自動合成、知識プログラミング言語の設計、Prologと関係データベースとのインタフェース機構の実現、等においても、その有用性が確認されつつある。しかも並列型Prologへのメタプログラミングの適用例も、最近急激に増えつつある。

メタプログラミング技法の唯一の欠点は、インタプリーティブに走る所が多いので、処理性能が遅いことである。しかしながら最近、竹内らが部分計算法(partial computation method)によるメタ推論システムのコンパイル化技術[20),21)]を開発し、二～三の例題でその有効性を確かめた。論理ベースのエキスパート・システム、ボトムアップパーザ、トレースを行なうPrologインタプリタ等に部分計算プログラムを適用し、メタ推論の高速化に成功した。この方法を適用すれば多くのメタ推論システムの最適化が行える。

最後に本稿で述べたことはPrologによる知識ベース管理問題のうちで、割合に論理的に性質のよいものについて、その解決の指針を述べたに過ぎないことを強調しておく。実際の知識ベースの管理問題は、より難しくより複雑なたくさんの問題をかかえている。知識ベースシステム作成の途上において、知識の同化過程と知識の調節過程が互いに協調しあい、知識の均衡化状態に到達する過程[22)]の基本原理についてすら、全くといっていい程、分っていない。知識ベース管理問題や知識獲得問題の全体からみれば、なすべきことの余りに多き、なせしことの余りに少なきが現状である。本稿で述べたことを手掛りとして、自らの眼前にそびえたつ知識ベース管理問題の問題解決の山に挑戦することを期待したい。

[ 参考文献 ]


1) 小林重信：知識工学の基礎と応用　第1回　知識の表現と利用(1)、計測と制御、Vol.24 No.2 (昭和60年 2月)

2) 小林重信：知識工学の基礎と応用　第2回　知識の表現と利用(2)、計測と制御、Vol.24 No.3 (昭和60年 3月)

3) 畝見達夫：知識工学の基礎と応用　第3回　Prologによる知識ベースシステムの実現、計測と制御、Vol.24 No.5 (昭和60年 5月)

4) D.L. Bowen :DECsystem-10 PROLOG USER'S MANUAL, DAI, University of Edinburgh (1981)

5) K.A. Bowen and R.A. Kowalski :Amalgamation Language and Meta-language in Logic Programming, TR 4/81, Syracuse University (1981)

6) R.W. Weyhrauch : Prolegomena to a Theory of Mechanized Formal Reasoning, Artificial Intelligence, 13, 133/170 (1980)

7) 國藤　進、竹内彰一、古川康一、上田和紀他：核言語第1版概念仕様書（案）、ICOT (1983)

8) 國藤　進、麻生盛敏、竹内彰一、坂井　公、宮地泰造、北上　始、横田治夫、安川秀樹、古川康一：Prologによる対象知識とメタ知識の融合とその応用、情報処理学会知識工学と人工知能研究会30-1 (1983)

9) T. Miyachi, S. Kunifuji,H. Kitakami,K. Furukawa : A Knowledge Assimilation Method for Logic Databases, Proc. of the 1984 International Symposium on Logic Programming, Atlantic City, U.S.A., 118/125(1984)

10) R. Reiter : On Closed World Databases, in Logic and Data Bases (H. Gallaire and J. Minker (eds.)), Plenum Press, New York / London, 55/76 (1978)

11) 武田　正之：知識ベースに基づくLanguage-oriented Editorの設計、（溝口文雄、古川康一（編）：WG4 ワークショップ'83 "知識表現"、ICOT TR-070 (1984))、20/25 (1983)

12) 國藤　進、坂井　公、宮地泰造、北上　始、古川康一：Prologによるルール型知識の知識同化について、日本ソフトウェア科学会第1回大会論文集1B-2, 21/24 (1984)

13) J.C. Shepherdson : Negation as Failure : A Comparison of Clark's Completed Data Base and Reiter's Closed World Assumption, J. Logic Programming, 1-1, 51/79 (1984)

14) J.W.,Lloyd and R.W. Topor : Making Prolog More Expressive, J. Logic Programming, 1-3, 225/240 (1984)

15) K. Sakai and T. Miyachi : Incorporating Naive Negation into Prolog, ICOT TR-028 (1983)

16) H. Kitakami, S. Kunifuji, T. Miyachi, K. Furukawa : A Methodology for Implementation of A Knowledge Acquisition System, Proc. of the 1984 International Symposium on Logic Programming, Atlantic City, U.S.A., 131/142 (1984)

17) T. Miyachi, S. Kunifuji,H. Kitakami,K. Furuk

awa : A Knowledge Assimilation Method for Logic Databases, NewGeneration Computing, 2-4, 385/404 (1984)

18) E.Y. Shapiro : Algorithmic Program Debugging, ACM Distinguished Dissertations, The MIT Press (1982)

19) H. Kitakami, S. Kunifuji,T. Miyachi,K. Furukawa,A. Takeuchi,T. Miyazaki,S. Ishii,T. Takewaki,H. Ohki : Demonstration of the KAISER System at the ICOT Open House in FGCS'84, ICOT TM-0081 (1984)

20) 竹内彰一、近藤浩康、大木　優、古川康一：部分計算のメタプログラミングへの応用、情報処理学会ソフトウェア基礎論研究会(1985)

21) 竹内彰一、古川康一：Prologプログラムの部分計算とメタ・プログラムの特殊化への応用、Proc. of the Logic Programming Conference '85 , ICOT, 1985年7 月.

22) 北上　始、國藤　進、宮地森造、古川康一：大規模な知識ベース管理システムのアーキテクチャ、知識理解システム・夏期シンポジウム報告書、富士通（株）国際情報社会科学研究所(1984)

# Appendix 7

```
demo(KBML,true,[Res,Cond,Cut,Count],[true,[]]):-!.
demo(KBML,Goals,[Res,Cond,Cut,0],[overflow,[]]):-!.
demo(KBML,!,[Res,Cond,Cut,Count],[Result,Stack]).
demo(KBML,!,[Res,Cond,cut,Count],[Result,Stack]).
demo(KBML,(P;Q),[Res,Cond,Cut,Count],[Result,Stack]):-!,
    (demo(KBML,P,[Res,Cond,Cut,Count],[Result,Stack]) ;
     demo(KBML,Q,[Res,Cond,Cut,Count],[Result,Stack])).
demo(KBML,(P,Q),[Res,Cond,Cut,Count],[Result,Stack]):-!,
    demo(KBML,P,[Res,Cond,Value,Count],[Result1,Stack1]),
    (Value==cut,Cut=cut,Result=Result1,Stack=Stack1,! ;
     Result1=true->demo(KBML,Q,[Res,Cond,Cut,Count],[Result,Stack]) ;
     Result=Result1,Stack=Stack1).
demo(KBML,P,[Res,Cond,Cut,Count],[Result,Stack]):-
    system(P)->P,Result=true,Stack=[] ;
    Count1 is Count-1,
    clause_kb(KBML,P,ID,Q,[Res,Cond]),
    demo(KBML,Q,[Res,Cond,Cut,Count1],[Result1,Stack1]),
    (Cut==cut,!,fail ;
     Result1=true->Result=true,Stack=[] ;
     Result=overflow,Stack=[(P:-Q)|Stack1]).

clause_kb((KBN,KBML),Head,ID,Goals,[Res,Cond]):-!,
     (clause_kb(KBN,Head,ID,Goals,[Res,Cond]) ;
      clause_kb(KBML,Head,ID,Goals,[Res,Cond])) .
clause_kb(KBN,Head,ID,Goals,[Res,Cond]):-
     next_clause(KBN,ID,Clause,Cond),
     ( Clause=(Head:-Goals) ;
       Clause=Head,Goals=true  ) .
clause_kb(KBN,Head,[],true,[Res,Cond]):-
     Res\==[],
     unify(Res,Head).

next_clause(KBN,ID,Clause,Cond):-
     next_clause1(KBN,ID,Clause,ID1),
     Clause\==[],check_condition(ID,Cond).

next_clause1(KBN,[P1,P2,P3],Clause,[P1,P2,P3]):-
     not(var(P1)),X=..[KBN,P1,P2,P3,Clause],X.
next_clause1(KBN,ID,Clause,[P1,P2,P3]):-
     var(P1),!,dictionary(KBN,[P1,_,_],_),
     X=..[KBN,P1,P2,P3,_],X,!,
     next_clause1(KBN,ID,Clause,[P1,P2,P3]).
next_clause1(KBN,ID,Clause,[P1,P2,P3]):-
     X=..[KBN,P2,Q2,Q3,_],X,!,
     next_clause1(KBN,ID,Clause,[P2,Q2,Q3]).

check_condition(ID,[]).
check_condition(ID,[ID|Y]):-!,fail.
check_condition(ID,[X|Y]):-check_condition(ID,Y).

unify((C;CL),P):-!,
     unify(C,P),unify(CL,P) .
unify((C,CL),P):-!,
     (unify(C,P);unify(CL,P)) .
unify(P,P).
```