ICOT Technical Memorandum: TM-0156

推論エンジン
メタプログラミング技法 —

國藤　進

February, 1986

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

推論エンジン　－メタプログラミング技法－

國藤　進

（ICOT第１研究室主任研究員）

１　はじめに

法律エキスパート・システムの有すべき必要機能として、本書第５章で10の機能が示されている。このうち①～③、⑩を除く７つの機能を論理プログラミング言語Prolog上で実現するには、本節で紹介するメタプログラミング技法が有効である。そこで本稿ではメタプログラミング技法の特徴についてそのエッセンスを分り易く解説する。

論理プログラミング言語Prologを用いた知識ベース管理システム、問題解決・推論システム等の研究試作において、著者らは従来の論理プログラミングの枠を超えるメタプログラミングの重要性を認識した。メタプログラミングとは、与えられたプログラミング言語環境において、新たな推論エンジンをその言語自身で作り上げる機能であり、PrologのPrologによるユーザのための推論エンジン定義機能である。

本稿では、メタプログラミング技法によるProlog上での推論エンジン実現法について紹介する。まず第２章でメタプログラミング方法論の基本的考え方を述べる。第３章でメタプログラミング技法の基本について述べる。第４章でメタプログラミング技法を用いた各種推論エンジンの実現例を述べる。最後に、法律エキスパート・システムのシェル実現との関連を述べる。

２　メタプログラミング方法論

2-1 メタプログラミング

第２章で指摘されたようにエキスパート・システムの中核部は、一般に「知

識ベース＋推論エンジン」からなる。与えられた問題に対して、どのような知識表現・利用・獲得形態をとるかによって「知識ベース＋推論エンジン」の構築法は異なる。このうち「知識ベース」の構築法については、溝口の稿に詳しいので省略するとして、ここではユーザ自らが定義できる問題向きの「推論エンジン」の構築法を述べる。

著者らが開発してきたメタプログラミング方法論の特徴[3,5]は、次の３点に要約できる。①（メタインタプリタ、知識ベース、メタ知識ベースの変更が容易なので）高度な機能が容易に実現できる。②（メタ知識とオブジェクト知識が分離しているので）モジュラリティ、メインテナビリティが高い。③（唯一の欠点と思われていた処理性能の遅さも、部分計算法によるコンパイル技術[9,10]の発展により）コンパイラと同程度の性能が保証できる。

2-2 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一つしかない知識、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（いわゆるpure Prolog ）の諸機能でなくプログラミング言語としてのPrologの諸機能（具体的には論理の枠を超える組込み述語）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。

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に関する莫大なノウハウ型知識や発見的知識がある。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に関する知識、あるいはメタ知識に関する知識として再帰的に定義される。するとメタ推論とはメタ知識を用いた推論として定義される。ここでは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に制限する。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・利用ならしめ、それにより問題向けの推論エンジンを提供することである。

2-3 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。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[1,5] によって提案された。

① demo(World, Goal, Result, Control)

このdemo述語は、ある世界World において、与えられた制御情報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。ここにControl とは証明プロセスを制御するために与えられた戦略情報であり、Resultとは証明プロセスの履歴から抽出される与えられたゴールを解決するのに必要な情報を保持していく引数である。

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”あるいは“Prologのメタインタプリタ”として知られている。

④ demo(Goal)

なおPrologのマニュアル等で周知のように、PrologのメタインタプリタはDEC-10 Prolog では、次のように書ける。

```
demo(true):-!.                                          (1)
demo((G1,G2)):-!,demo(G1),demo(G2).                     (2)
demo(G):-clause(G,B),demo(B).                           (3)
demo(G):-system(G),!,G.                                 (4)
```

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、Prologの単一の知識ベースから取出すDEC-10 Prolog 組込み述語である。また式(4) は、G がシステムの組込み述語ならば、直接G を呼出しなさいということを意味する。

3 メタプログラミング技法

3-1 demo述語の実現法

文献[3,5] 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推論エンジンの構築法とその利用法を示したものである。すなわち、上述のメタ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)).                    (5)
kb(P).                                    (6)
```

ここにPやQi （i ＝ 1, …, m ）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)).                    (7)
```

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(7) 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(7) は簡単のため、次のように略記することもある。

```
mkb((Q1,Q2, …, Qn)).                      (8)
```

式(7) （あるいは式(8) ）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(5) 、(6) あるいはメタ知識(8) 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):-!.                          (9)
demo(W,(G1;G2)):- !,(demo(W,G1);demo(W,G2)).   (10)
demo(W,(G1,G2)):- !,demo(W,G1),demo(W,G2).     (11)
demo(W,G):-wclause(W,G,B),demo(W,B).           (12)
demo(W,G):-system(G),!,G.                      (13)
wclause(W,H,B):-X=..[W,C],X ,wclause2(C,H,B)   (14)
wclause2((H:-B),H,B):-!.                       (15)
wclause2(H,H,true).                            (16)
```

式(9) 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(10)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(11)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(12)は、Wにおいて(9) ～(11)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。式(13)は、G がシステムの組込み述語ならば、直接G を呼出しなさいということを意味する。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や②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。

3-2 否定付きdemo述語

古川の稿に指摘されているように、Prologにおける否定は閉世界仮説のもとでの否定であり、論理学の否定ではない。従って、それは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、閉世界仮説のもとでの否定not(_) を、demo述語に導入するのは極めて易しい。例えば、式(9) と式(10)の間に次式(17)を、式(16)の後にnot(_) の定義式(18)、(19)を挿入するだけでよい。

```
demo(W,not(G)):-!,not(demo(W,G)).          (17)
not(P):-P,!,fail.                          (18)
not(_).                                    (19)
```

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ので注意されたい。

3-3 カット付きdemo述語

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,_):-!.                                              (20)
demo(!, _).                                                   (21)
demo(!,cut).                                                  (22)
demo((G1;G2),Cut):- !,( demo(G1,Cut);demo(G2,Cut)).           (23)
demo((G1,G2),Cut):- !,demo(G1,Value),
                      (Value==cut,Cut=cut,!;demo(G2,Cut)).    (24)
demo(G,Cut):- clause(G,B),demo(B,Cut),(Cut==cut,!,fail;true). (25)
demo(G ,_):-system(G),!,G.                                    (26)
```

このdemo述語は、対象言語レベルの"!" の最初の出現時に、節(21)でカットをかける準備を行なう。この時、節(24)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(22)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(25)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　各種推論エンジンの実現例

4-1 計算木抽出用推論エンジン

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。その最も基本的な利用法は証明のプロセスから計算木Treeを抽出し、そこからエキスパート・システムのシェル向きの説明機能を実現することである。この機能を実現するには、次のような計算木抽出用demo述語[8] を構成すればよい。

```
demo(true,[]):-!.                                                    (27)
demo((G1,G2),(T1,T2)):-!,demo(G1,T1),demo(G2,T2).                    (28)
demo(G,node((G:-B),T)):-clause(G,B),demo(B,T).                       (29)
demo(G,node((G:-system(G)),[])):-system(G),!,G.                      (30)
```

このdemo述語に対して、次のような知識ベースが与えられているものとする。

```
grandparent(X,Y):-parent(X,Z),parent(Z,Y).                           (31)
parent(X,Y):-father(X,Y).                                            (32)
parent(X,Y):-mother(X,Y).                                            (33)
father(a,b).                                                         (34)
mother(b,c).                                                         (35)
```

上述のdemo述語と知識ベースに対して、質問“?-demo(grandparent(a,c),Tree).”を実行すると、次のような計算木Treeが得られる。

```
Tree = node((grandparent(a,c):-parent(a,b),parent(b,c)),
            (node((parent(a,b):-father(a,b)),
                  node((father(a,b):-true),[])),
             node((parent(b,c):-mother(b,c)),
                  node((mother(b,c):-true),[])))))                   (36)
```

この計算木Treeは証明のプロセスを最大限に保存しているので、ここから

“How ～? ”や“Why ～? ”といった説明機能を抽出するのは易しい。例えば、あるゴールがどのようにして実行されたかを説明したければ、上述のdemo述語から、次のようにして必要な解釈情報を抽出[8] すればよい。

```
how(G):-demo(G,Tree),interprete(Tree).                    (37)
```

4-2 知識同化用推論エンジン

メタプログラミングによる高度な拡張機能の実現例として知識ベース管理の実現を採り上げる。ICOTでは、知識獲得機能を中核とする知識ベース管理システムを、論理プログラミング言語Prologを用いて研究試作中[3,4,5,6] である。そのようなシステムの研究試作の途上で、与えらえた知識ベースが正しいと仮定した時、外から与えられた知識をその知識ベースに無矛盾かつ系統的に取込むという過程の管理、すなわち知識同化という知識ベース管理の基本技術を明らかにした。

例えば、ファクト型知識の知識同化用推論エンジンは次のようにして実現される。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込みたい新知識とする時、知識同化の基本的考え方[4,5,6] 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).          (38)
assimilate(Currkb,Input,Currkb):- demo(Currkb∪Input,false).    (39)
assimilate(Currkb,Input,Newkb):-Info∈Currkb,Interkb=Currkb-Info,
                                demo(interkb∪Input,Info),
                                assimilate(Interkb,Input,Newkb).(40)
assimilate(Currkb,Input,Currkb∪Input):-
                                independent(Currkb,Input).      (41)
```

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。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。なお本プログラムのコーディング例については、[4] を参照されたい。

本例に典型的に見られるように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。さらに複雑な知識ベース管理問題の解決法については、[3] を参照されたい。

4-3 類推用推論エンジン

メタインタプリタを変形すると、演繹・帰納のみならず類推機能をもつ推論エンジンを実現することができる。著者らの研究は、原口の類推理論[2] にその基礎を置いている。従って、その中核となる部分は彼の類推システム[2] と全く同じであり、次のような推論エンジンとして与えられる。

```
analogy(W,true):-!.                                                    (42)

analogy(W,(G1,G2)):-analogy(W,G1),analogy(W,G2).                       (43)

analogy(W,G):-wclause(W,G,B),analogy(W,B).                             (44)

analogy(W,G):-prematch(W,G,TW,TG,TB),transform(W,G,B,TW,TG,TB,Pair),
              analogy(TW,TB),analogy(W,B),call(Pair),
              message(6,['Transformation has succeeded :']),
              message(10,['resulting analogy is',Pair]).               (45)

analogy(W,G):-message(0,['***Goals',G,'in',W,'fails***']),fail.        (46)
```

原口の類推システムに種々の拡張を施し、次のような仮説生成・検定システム[11]へと機能拡張中である。

(1) 予測推定型類推のみならず問題解決型類推の機能を付与した。例えば、ある種の幾何の定理証明が類推により解けるようになった。

(2) 述語の引数のみならず述語名の１対１対応をも、類推によって発見できるようになった。

(3) 類推の本質を、与えられた観測データを説明する仮説の生成とみなし、与えられた知識表現の世界で合理的な仮説を生成・検定する戦略の検討を行った。

(4) 与えられた知識ベースが事実型のルールのみならず、本当のルールをも取扱う方式を検討中である。

いづれにせよ、この様なアプローチを基礎にすれば、仮説生成・検定や類推の本質の議論をする“思考のワークベンチ”を提供することができる。

5 おわりに

法律エキスパート・システムの有すべき必要機能⑧、⑨は 4-1で述べられた計算木抽出用推論エンジンや 4-2で述べられた知識同化用推論エンジンを拡張修正することによって実現できる。④～⑦は 4-3で述べられた類推用推論エンジンや仮説生成・検定用推論エンジンで採用した考え方を利用することによって実現できる。しかしながら最初に述べたように問題向きの知識表現・利用・獲得形態を確定しなくては、問題向きの「推論エンジン」を構築できない。従って自明のことであるが、与えられた法律の問題（法体系）を分析し、それを表現する論理（知識表現言語）を定めて、初めて本稿で述べたメタプログラミング技法が有効であるということを強調しておく。

今後の研究課題としては、(1) Prologの否定では表現できない論理的な否定の取扱いの問題、(2) 事実認定の問題、(3) 特定の法体系向きの論理の設定問題、(4) 必要機能②、③で指摘された大規模知識ベースの利用問題、等の多くの解決すべき課題を抱えている。それらのあるものは法律エキスパート・システム特有の課題であり、法律の専門家と工学者の密接な協力が要請される。

参考文献

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

2) 原口　誠：ルールの変換による類推の逆向き変換について、Proc. of the Logic Programming Conference '85,ICOT,1985.

3) 北上　始、國藤　進、宮地泰造、古川康一：論理型プログラミング言語Prologによる知識ベース管理システム、情報処理、Vol.26 No.11,Nov. 1985.

4) 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.

5) 國藤　進、北上　始、宮地泰造、古川康一：知識工学の基礎と応用［第４回］－Prologにおける知識ベース管理－，計測と制御，Vol.24, No.6, 53-62, 1985.

6) T. Miyachi, S. Kunifuji,H. Kitakami,K. Furukawa : A Knowledge Assimilation Method for Logic Databases, New Generation Computing, 2-4, 385/404 ,1984.

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

8) L. Sterling: Explaining Explanations Clearly,Case Western Reserve University CES-85-03,May 1985.

9) 竹内彰一、近藤浩康、大木　愛、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.

10) 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.

11) 鶴巻宏治、國藤　進、古川康一：メタプログラミングによる類推システムの試作について、日本ソフトウェア科学会第２回大会論文集、1985年11月.