Prolog による対象知識とメタ知識の融合とその応用

国藤　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

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　（ICOT）

〔1〕 はじめに

知識情報処理システムの研究開発動向が，第5世代コンピュータ・プロジェクトとの関連で急激に進展する中で，知識ベース管理システムの基礎ソフトウェア技術確立につながる研究が注目されている．このような研究のうち，最も魅力的ではあるが未開拓の研究分野〔1〕が，知識の獲得や学習に関する研究とメタ推論や高次推論の機構に関する研究である．

論理データベース・システム，すなわち関係データベースへの演繹的質問応答システム，の研究開発にあたって，われわれは知識の獲得やメタ推論に関する基本機能を，論理型言語DEC-10 Prolog〔2〕上にインプリメンテーション中である．論理データベース・システムとの関連で考慮したのは，論理データベース（知識ベース）に知識を取込むにはどうすればいいかという「知識の同化（Knowledge Assimilation）」の実現であり，またDEC-10 Prolog上でメタ知識を用いる推論を行うにはどうすればいいかという「メタ推論（Meta Inference）」の実現である．前者については，知識同化の諸相を演繹的同化・帰納的同化・発想的同化という三種のフェーズでとらえ，段階的にインプリメンテーションすることを基本方針とした．前報〔3〕および本報告では演繹的同化のフェーズに，次報〔4〕では帰納的同化のフェーズに焦点を当てて考察する．また後者については，メタ知識では「知識の使い方に関する知識」であることを再認識した上で，その最もプリミティブな概念である証明可能性という概念を直接反映した知識の取扱いについて考察することにした．

演繹的な知識同化とProlog向きメタ推論への基本的考え方を与えたのが，Bowen & Kowalskiの対象言語とメタ言語の融合（Amalgamation）という論文〔5〕である．彼らのアイディアは，現在の知識ベースからある知識が証明可能であるという概念を表わす述語“demo”を導入し，換言すると一階述語論理の証明論での証明可能性という概念そのものを操作対象とし，それを用い種々のメタ知識の表現法・利用法を提案したことにある．しかしながら彼らは“demo”述語そのものの具体的実現法を与えなかったので，その後，プログラム・レベルの実験研究は行われず，見るべき継続的研究成果もあがっていない．そこでわれわれは，DEC-10 Prolog上で“demo”述語を実現するにはどうすればよいかを考察し直し，その結果，本論文で述べるインプリメンテーション技法を見出した．この技法を用いて，論理データベース向きの演繹的な知識同化プログラムを試作した．このことは論理データベース上で肯定的知識と否定的知識をメタレベルで管理するメタ推論プログラムを作成したことにもなる．要約すると，本報告ではDEC-10 Prologによる対象知識とメタ知識の融合を行うメタ推論方式を提案し，その各種応用例について検討を行ったので，インプリメンテーション・レベルでその検討結果を述べる．

〔2〕 知識融合とメタ推論

〔2.1〕 対象言語とメタ言語上の知識

“demo”述語実現にあたって，われわれは処理対象とする対象言語およびメタ言語上の知識，これらを簡単のためそれぞれ対象知識(Object Knowledge)とメタ知識(Meta Knowledge)と呼ぶことにするが，を制限しなければならない．まず対象知識としては，DEC-10 Prolog の適当な部分集合をとることにする．われわれのアプローチによれば，EMACS で取扱うファイル内に，対象言語上の肯定的知識と否定的知識を，それぞれ次のような形式で格納する．

$P :- Q_1, Q_2, \cdots, Q_n.$ （ルール型肯定的知識の場合） (1)

$P.$ （ファクト型肯定的知識の場合） (2)

$fail :- Q_1, Q_2, \cdots, Q_n.$ （否定的知識の場合） (3)

ここにPやQは肯定的あるいは否定的な素命題である．これらはそれぞれ論理的には，$Q_1 \wedge Q_2 \wedge \cdots \wedge Q_n \supset P$，$P, \neg(Q_1 \wedge Q_2 \wedge \cdots \wedge Q_n)$ の意味である．

EMACS ファイル内の対象知識を DEC-10 Prolog の内部データベース内に読込んだ時，ある種の前処理プログラムが作動し，自動的に次に述べるようなメタ言語の形式に変換する．

$pkb((P :- Q_1, Q_2, \cdots, Q_m)).$ (4)

$pkb(P).$ (5)

$nkb((Q_1, Q_2, \cdots, Q_n)).$ (6)

ここに“pkb”と“nkb”はそのファイルから読込まれたかの手掛りを残す述語名で，“pkb”部が肯定的知識ベース，“nkb”部が否定的知識ベースと呼ばれる．これら両者が内部データベース上の処理対象となる現存知識ベースである．各ファイルには，それぞれ固有の肯定的知識と否定的知識が格納されているが，ここでは簡単のため，それらを内部データベース内の単一の肯定的知識ベースと否定的知識ベース内に転送することにする．対象言語上の知識（ルール等）の全てが，メタ言語上のアサーション（ファクトの一種）に変換されていることに注意しなさい．

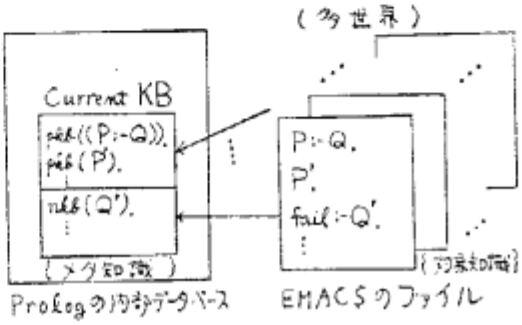

図1 “対象-メタ”知識変換

〔2.2〕 “demo”述語の実現

ある論理データベースに外部世界から入力データがどんどん追加・更新される時，すなわち動的に変化する論理データベース管理システムを設計する際，そのようなシステムは対象知識とメタ知識を明確に区分し，必要に応じて両者を融合管理する必要性が生じた．さてメタ推論とは対象知識とメタ知識を用いる推論であり，メタ知識とは「知識の使い方に関する知識」であった．またProlog処理系そのものは，一階述語論理の部分集合であるホーン論理の定理証明系の一種であった．そこでメタ推論を実行するには，Prologインタプリータ/コンパイラの使い方に関する知識表現技術や知識利用技術に還元しなければならない．このことをProlog処理系の外に飛出し，ジョブ制御言語レベルで実現するのは困難だったので，われわれはProlog処理系の単一内部データベース内に，対象知識とメタ知識の両者を埋込むことにした（本節，図1）．このことをインプリメンテーショ

ン・レベルで実現するため，われわれの着目したのはPrologでPrologインタプリータを作成すると極めて単純明解であるというPereira ら〔6〕の考えである．この考えて“demo”述語のまえの概念的等価性に注目し，拡C-10 Prologのメタロジカル述語を用いて“demo”述語を実現すると次のようになる．ここに述語“demo(KB, Goals)”は“KB⊢P Goals (⊢P: Prologで証明可能)”で定義され，現存知識ベースKBからあるゴールの系列Goalsが Prolog上で証明可能であるという意味である．

```
demo(KB,true):-!,true.
demo(KB,not(P)):-!,metanot(demo(KB,P)).
demo(KB,(P;Q)):-!,(demo(KB,P);demo(KB,Q)).
demo(KB,(P,Q)):-!,demo(KB,P),demo(KB,Q).                    (7)
demo(KB,P):-metacall(KB,(P:-Q),KBPQ),demo(KB,Q).
demo(KB,P):-metacall(KB,P,KBP).

metanot(P):-P,!,fail.
metanot(_).                                                 (8)

metacall(K,X,KX):-KX=..[K,X],KX.                            (9)
```

式(7)の各節はそれぞれ値‘true’，否定，選言，連言，ルール，ファクトの処理を行っている．また述語metanotやmetacallは，それぞれ式(8)や(9)のようなメタロジカル機能を用いて実現される．ここにPrologの否定“metanot”は“Negation as Failure”の意味であり，一階述語論理の否定と異なる〔7〕．

もう論，このような考え方の延長でカット演算子(!)を処理する“demo”述語を実現することも可能であるが，われわれの考えの眼目は，個々の問題向きの“demo”述語を提供することにより，個別問題向きの効率のより必要最小限の証明可能性の概念，および関連派生概念(例えば証明のプロセスの抽出とか制御情報の授受等)を容易に実現ならしめることにある．なお“demo”述語の第2引数は，対象言語上の質問“?-p.”の“p”部にくるPrologのゴール列である．従って“p”内に変数を含む場合は，論理的には「“$\exists X\, p(X)$”を真ならしめるXは何か」ということである．すなわちこの述語で実現されているのは存在記号のみを含むゴールの系列の“demo”であり，全称記号や存在記号を混在して含むより一般的なゴールの系列の“demo”については，次に発表する報告〔4〕を参照されたい．

〔3〕 知識同化プログラム

〔3.1〕 知識の同化

論理データベース・システムにおいて整合的な知識ベースを構築していくには，Kowalski ら〔5,8〕の指摘によれば，次の4概念のチェックと対応する知識ベース更新がこの順序に従って必要である：(C1) 証明可能性，(C2) 冗長性，(C3) 矛盾性，(C4) 独立性．

論理データベースへの漸増的知識同化プログラム作成というわれわれの立場からすれば，次のような順序で，対応する概念のチェックと知識ベース更新を行わなければならない．

(P) 肯定的知識を管理する場合：(P1) 証明可能性，(P2) 矛盾性，(P3) 冗長性，(P4) 独立性〔3〕．

(N) 否定的知識を管理する場合：(N1) 矛盾性，(N2) 証明可能性，(N3) 冗長性，(N4) 独立性．

ここに前述の(C1)あるいは(C3)が成立する場合，知識ベースKBに入力データInputを同化することが不要あるいは無意味であり，(C2)あるいは(C4)が成立する場合，KBにInputを同化することが有意味あるいは必要である．従来からの論理的概念構成によれば，(C1)と(C3)のチェックに失敗したInputは独立であるとして，対応するKBに同化されていた．われわれの立場は，独立性という概念を更に2つに分解し，冗長性のある場合とそうでない場合に分けたことに相当する．ただしこの冗長性チェック・モジュールはメモリ節減をはかるために導入されたものであるが，一般に大量のCPU時間という資源を消費するので省略することも多い．すなわち「メモリ節約をとるか，CPU時間の節約をとるか」という古典的な議論との兼合いで採用をきめるべきである．いずれにせよ，このモジュールを実用的に実現するには，各種の最適化技法や構造的データの最適アクセス技法の並用が必須である．

論理データベース・システム向きに知識同化プログラムを作成するにあたって，われわれは次のような一般的前提の成立する枠組の中で考察した．

（前提1） 対象言語としての入力される新知識は，ファクト型肯定的／否定的知識のみに制限する．この前提は関係データベースに相当する部分のタップル更新を意図したものである．

（前提2） Prologの否定と自己連結的に動くことを保証するため，“Negation as Failure”を仮定する．具体的には対象言語上の否定“not”を，次のようなメタ言語上の形式で定義し導入しておく．

```
pkb((not(P):-P,!,fail)).
pkb(not(_)).                                    (10)
```

（前提3） 知識ベース創成時に，肯定的知識ベースと否定的知識ベースの和集合全体の整合性を仮定する．

〔3.2〕 肯定的知識の同化プログラム

所与の肯定的知識ベースPKBと否定的知識ベースNKBに対して，ファクト型入力知識Pinputが与えられたものとする．するとファクト型肯定的知識を演繹的に同化するPrologプログラムpassimilate([PKB,NKB],Pinput)は，図2で示されるようなメタ知識としてインプリメントされる．以下，各モジュールの説明をする．

(P1) $PKB \vdash_F Pinput$ の場合，$KB \vdash_F Pinput$ ($KB \equiv PKB \cup NKB$)となる．すなわち入力知識Pinputが現在の知識ベースKBから証明可能なので，KBにPinputを同化することは不要である．

(P2) $\exists X(\neg X \in NKB) \wedge (PKB \cup \{Pinput\} \vdash_F X)$ の場合，$KB \cup \{Pinput\} \vdash_F \square$ (□：矛盾)となる．すなわち知識ベースKBに入力知識Pinputを挿入すると矛盾を生じるインスタンシェーションが見出されたことになる．この場合，KBにPinputを同化することは無意味なので，そのようなXが1つでも見出された時点で終了する

(P3) $\exists X(X \in PKB) \wedge (X:\text{ファクト}) \wedge ((PKB-\{X\}) \cup \{Pinput\} \vdash_F X)$ の場合，$(KB-\{X\}) \cup \{Pinput\} \vdash_F X$ となる．すなわち知識ベースKBに入力知識Pinputを挿入した際，ファクト型肯定的知識Xの冗長性が見出されたので，冗長なXを取除いた知識ベース$KB-\{X\}$の同化プログラムを再帰的にコールすればよい．

(P4) (P1)，(P2)，(P3)のどれもが成立しない場合，入力知識Pinputは知識ベースKBに対して独立であるとみなし，KBにPinputを機械的に同化していく．

```
(P1)    passimilate([PKB,NKB],Pinput):-
            demo(PKB,Pinput),
            message(' Pinput is deducible from PKB ! ',PKB).

(P2)    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)).

(P3)    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 KB ! '),
                  (passimilate([PKB,NKB],Pinput);true));
              asserta(PKBX),retractf(PI))).

(P4)    passimilate([PKB,NKB],Pinput):-
            metaasserta(PKB,Pinput,PI),
            message(' Pinput is acquired in PKB ! ',PKB).
```

図2　肯定的知識同化プログラム

〔3.3〕　否定的知識の同化プログラム

所与のPKBとNKBに対して，ファクト型の否定的入力知識Ninputを演繹的に同化していくPrologプログラムnassimilate([PKB,NKB],Ninput)は，図3で示されるようなメタ知識としてインプリメントされる．以下，本プログラムの各モジュールの説明を行う．

(N1)　$PKB \vdash_p Ninput$ の場合，$KB \cup \{\neg Ninput\} \vdash_p \square$ となる．すなわちKBのNKB部にNinputを同化すると矛盾を生じる．この場合，KBにNinputを同化することは無意味である．

(N2)　$\exists X\,(\neg X \in NKB) \wedge (PKB \cup \{Ninput\} \vdash_p X)$ の場合，$KB \vdash_p \neg Ninput$ となる．すなわちKBから $\neg Ninput$ が証明可能なので，KBにNinputを同化することは不要である．

(N3)　$\exists X\,(\neg X \in NKB) \wedge ((NKB - \{\neg X\}) \cup \{X\} \vdash_p Ninput)$ の場合，$(KB - \{\neg X\}) \cup \{\neg Ninput\} \vdash_p \neg X$ となる．すなわちKBにNinputを同化すると否定的知識 $\neg X$ の冗長性が見

```
(N1)    nassimilate([PKB,NKB],Ninput):-
            demo(PKB,Ninput),
            message(' Ninput is inconsistent with PKB ! ',NKB).

(N2)    nassimilate([PKB,NKB],Ninput):-
            metaasserta(PKB,Ninput,NI),
            (metacall(NKB,X,NKBX),
             demo(PKB,X),
             retract(NI),
             message(' Ninput is deducible from KB ! ',NKB);
             retractf(NI)).

(N3)    nassimilate([PKB,NKB],Ninput):-
            metacall(NKB,X,NKBX),
            PKBX=..[PKB,X],
            (metanot(ground(X))->fail;
             ((retract(NKBX),asserta(PKBX),demo(PKB,Ninput))
               ->(retract(PKBX),
                  message0(' Redundancy is found for KB ! '),
                  (nassimilate([PKB,NKB],Ninput);true));
              asserta(NKBX),retractf(PKBX))).

(N4)    nassimilate([PKB,NKB],Ninput):-
            metaasserta(NKB,Ninput,PI),
            message(' Ninput is acquired in NKB ! ',NKB).
```

図3　否定的知識同化プログラム

出された事になる．そこで冗長な $\neg X$ を取除いた知識ベース KB-$\{\neg X\}$ に対して，否定的知識同化プログラムを再帰的にコールすればよい．

(N4) (N1)，(N2)，(N3) のどれもが成立しない場合，入力知識 Minput は知識ベース KB に対して独立であるとみなし，KB に Minput を機械的に同化していく．

〔4〕 応用

〔4.1〕 否定的知識同化プログラムの応用例

論理データベース・システムへの各種応用例を与える．

(例題1) pkb としてファクト(11)とルール(12)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質問(13)に対して，プログラムはメタ言語上で証明可能なことを示している．

```
pkb(father(shimako,susumu)).                                          (11)
pkb(mother(shimako,izumi)).

pkb((parent(X,Y):-father(X,Y);mother(X,Y))).                          (12)

| ?- passimilate([pkb,nkb],parent(shimako,izumi)).                    (13)

 Pinput is deducible from PKB !
```

(例題2) 論理データベースに対する統合性制約(Integrity Constraint)は，否定的知識の典型例である．例えば，母親と子供の間の血液型の遺伝学的検査プログラムはそのような知識の一種なので，(14)のような表現形式で nkb 部に記述できる．この時，剛夫さん(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A)が生まれたとする．そこで陽子さんの父親が剛夫さんという知識を同化しようとした所，(15)に見られるように，統合性制約(14)との間に矛盾を生じてしまうことが示されている．

```
nkb((father(X,F),blood_type(F,FT),married(F,M),
     blood_type(M,MT),genes_match(FT,MT,CBT),                         (14)
     blood_type(X,BT),not(member(BT,CBT)))).

| ?- passimilate([pkb,nkb],father(youko,norio)).                      (15)

 Pinput is inconsistent with KB !
```

(例題3) (11)，(12)に更に(16)，(17)の追加された pkb が与えられているとする．この時，"father(susumu,toshio)" という知識を同化しようとすると，(18)に見られる

```
pkb(ancestor(shimako,toshio)).                                        (16)
pkb(parent(shimako,susumu)).
pkb((ancestor(X,Y):-parent(X,Y);parent(X,Z),ancestor(Z,Y))).          (17)

| ?- passimilate([pkb,nkb],father(susumu,toshio)).                    (18)

 Redundancy is found for PKB !

 Redundancy is found for PKB !

 Pinput is acquired in PKB !

pkb(father(susumu,toshio)).
pkb(mother(shimako,izumi)).
pkb(father(shimako,susumu)).
```

ように，冗長な二つの知識(4)を除去し，独立な知識"father(susumu, toshio)"を挿入している．

〔4.2〕 否定的知識同化プログラムの応用例

(例題4) (18)より"god(zeus)"なのに"¬god(zeus)"という知識を同化しようとすると，(20)に見られるように矛盾を生じ，nkb内に同化されない．

```
pkb(god(zeus)).                                          (19)
| ?- nassimilate([pkb,nkb],god(zeus)).                   (20)

 Ninput is inconsistent with PKB !
```

(例題5) (19)かつ(20)が成立する状態で，"¬man(zeus)"という知識を同化しようと試みた．すると(22)に見られるように，god(zeus)∧∀X ¬(man(X)∧god(X)) ⊢ ¬man(zeus)なので，この知識は同化する必要がないことがわかる．

```
nkb((man(X),god(X))).                                    (21)
| ?- nassimilate([pkb,nkb],man(zeus)).                   (22)

 Ninput is deducible from NKB !
```

(例題6) pkb(23)とnkb(24)が成立する状態で，"¬a(f(a))"という知識を同化しようと試みた．すると(25)に見られるように，∀X(a(X) ⊃ a(f(X)))∧¬a(f(b)) ⊢ ¬a(b)なので，nkb内に冗長性が見出される．するやすその知識が除去された後，入力知識がnkb内に同化されている．

```
pkb((a(f(X)):-a(X))).                                    (23)

nkb(a(b)).                                               (24)

| ?- nassimilate([pkb,nkb],a(f(b))).                     (25)

 Redundancy is found for KB !

 Ninput is acquired in NKB !

nkb(a(f(b))).
```

〔5〕 おわりに

本報告では，DEC-10 Prolog上に作成された証明可能性述語"demo"を用いるメタ推論方式を提案した．またその典型的応用例として，論理データベース・システムに演繹的に知識を同化するプログラムをふした．具体的には論理データベースにファクト型の肯定的／否定的知識を同化する局面に注目し，論理データベース上のメタ知識と対象知識を融合管理する演繹的知識同化方式の提案を行った．ルール型知識同化方式あるいは帰納的知識同化を含む場合については，次報〔4〕を参照されたい．

ここに述べた"demo"述語を用いるメタ推論方式は，知識同化のみならず，他に多くの適用分野をもつ．われわれが試行し，ある程度の具体的成果を得た適用分野に次のようなものがある：

(A1) Prologで関係データベース管理システムのインタフェースをとるための部分評価法を用いる問合せ変形〔9〕．

(A2) 論理プログラムの性質の証明およびその検証への適用〔10〕．

(A3) 多世界知識ベース・モデルの自然な知識表現と知識利用.
(A4) Default Reasoning を含む場合への拡張.

"demo" 述語をベースとするメタ推論・知識獲得方式を確立していくには，今後，以下に述べるような研究課題を検討していかなければならない:

(T1) 一階述語論理での証明可能性"下"を含む各種の問題向き"demo"の提唱.
(T2) メタ推論向き知識表現言語の提案．ただしインヘリタンスの管理を含むフレーム型多世界モデル表現を許容する知識表現言語とする.
(T3) Truth Maintenance System のプロトタイプの試作.

〔謝辞〕 本研究の機会を与えていただいた渕一博研究所長(ICOT)，有益なご討論をしていただいた古川康一主任研究員をはじめとするICOT第2研究室の皆様方，情報システム研究会の諸先生方，ICOTのWG2，WG4 メンバの諸先生方に感謝します.

〔参考文献〕

〔1〕 Cohen, P.R. & Feigenbaum, E.A. (eds.): The Handbook of Artificial Intelligence, Vol. III, Heuris Tech Press & William Kaufmann, Inc., 1982.
〔2〕 Bowen, D.L.: DEC system-10 PROLOG USER'S MANUAL, DAI University of Edinburgh, 15 Dec. 1981.
〔3〕 宮地，岡部，北上，古川，竹内，横田: 論理データベース向きの知識同化方式の一提案，研究集会「モデル表現とその積算に関する理論と実際の研究」，京大数解研講究録，1983.
〔4〕 北上，麻生，岡部，宮地，古川: 知識同化機構の一実現法，情報処理学会 知識工学と人工知能研究会，1983年6月.
〔5〕 Bowen, K.A. & Kowalski, R.A.: Amalgamating Language and Meta-language in Logic Programming, School of Computer and Information Sciences, University of Syracuse, 1981.
〔6〕 Coelho, H., Cotta, J.C. & Pereira, L.M.: How to Solve It with Prolog (2nd. ed.), Laboratório Nacional de Engenharia Civil, Lisboa, 1980.
〔7〕 Clark, K.L.: Negation as Failure, in Logic and Data Bases (Gallaire, H. & Minker, J. (eds.)), Plenum Press, 1978.
〔8〕 Kowalski, R.A.: Logic for Problem Solving, North Holland, Inc., 1979.
〔9〕 Kunifuji, S., Yokota, H., Kitakami, H., Miyachi, T. & Furukawa, K.: How to Interface Logic Programming Language and Relational Data Base Management System, presented by CERT Workshop on "Logic Base for Databases", CERT, DEC. 1982.
〔10〕 坂井，宮地: プログラムの検証と否定的知識，ICOT Tech. Rep., 1983.