ICOT Technical Report: TR-137

TR-137

メタプログラミングによる論理プログラミングと知識情報処理技術の融合

國藤　進, 武脇敏晃, 世木博久, 竹内彰
大木　優, 古川康一 (ICOT)
鶴巻宏治 (NTT)

September, 1985

©1985, ICOT

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

メタプログラミングによる
論理プログラミングと知識情報処理技術の融合

関聡　進、武脇敏晃、世木博久、竹内彰一、大木　良、古川康一（ICOT）、溝巻宏治(NTT横通)

1．はじめに

近年、人工知能や知識工学といった研究分野の急激な進展の刺激をうけ、知識の表現、知識の利用、知識の獲得に関する基礎・応用・開発研究の成果を集大成し、知識情報処理システムのプロトタイプを構築しようという動きがさかんである。知識情報処理システム構築の気運の胎動は、歴史的に見ると、学界や産業界における人工知能研究の見直しの動きあるいは勃興と無縁ではない。周知のように人工知能研究の歴史は、1960年代前半まではゲームとパズルの時代、1960年代後半は知能ロボットの時代、1970年代は言語と知識の時代、そして1980年代は知識工学と認知科学の時代といわれている。

著者らの所属するICOTは、いわゆる第五世代コンピュータの研究開発を目途として設立された財団法人である。第五世代コンピュータは1990年代における知識情報処理システムのプロトタイプ実現をめざすコンピュータであり、その実現のためICOTでは知識ベース管理システム、問題解決推論システム、知的インタフェースシステム、知的プログラミングシステムに関連する各種基礎ソフトウェアの研究開発（図１参照）を行なっている。

知識情報処理システムのプロトタイプ実現にあたって、ICOTでは逐次型／並列型の論理プログラミング言語Prologを用いた各種の基礎ソフトウェアの実験的試作を行っている。なかでも核言語第１版、数式処理システム、知識プログラミング言語、知識ベース管理システム、類推システム等の研究試作において、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が認識された。メタプログラミングは、与えられたプログラミング言語環境において新たな拡張された言語機能を、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言語機能拡張機能である。

本稿で取り上げる話題は、逐次型／並列型論理プログラミング言語Prologを用いたメタプログラミング技法による知識情報処理システムのプロトタイプ構築の方法論についてである。まず第２章で人間による問題解決・推論の記号論的本質を明らかにし、第３章で知識情報処理に対する著者らの考え方を述べ、第４章で本論文で紹介するメタ推論の基本的考え方を述べる。ついでメタプログラミングによる知識の表現・利用・獲得機能のラピッド・プロトタイピングの成功例について述べる。すなわち、第５章でメタ推論による知識利用機能の実現、第６章でメタ推論による知識表現機能の実現、第７章でメタ推論に基づく知識獲得機能の実現を述べる。最後に、第８章でメタプログラミングによる種々のシステムのプロトタイプ構築という方法論の有効性を、性能面からも保証する部分計算によるメタ推論の高速化技法について述べ、メタプログラミング・アプローチが論理プログラミングと知識情報処理技術を融合する一つの方法論であることを実証する。

2．問題解決と推論

2.1　問題解決のプロセス

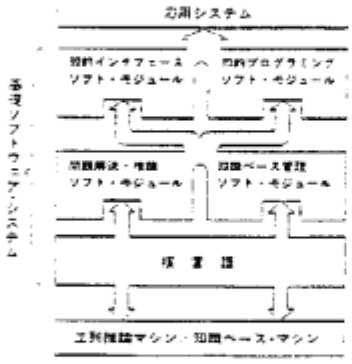

図１　基礎ソフトウェア・システム

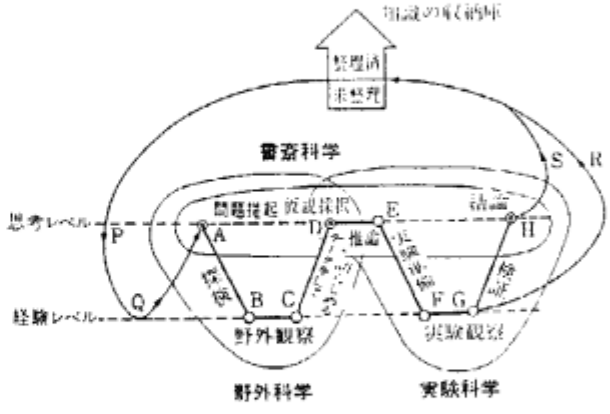

図２　W型問題解決学[1]

TR-137

人間の問題解決プロセスの本質を、近い将来において機械において実行／模倣できるということを前提に、その本質を分析してみる。人間の知的問題解決活動を「問題提起、現状把握、本質追及、問題評価、決断、構想計画、具体策、計画評価、手順化、実施、検証、総括」といった一連のプロセスの中でとらえるのが、川喜田らの「問題解決学」である[1]。問題解決学は、異質のデータ・情報を統合することによって新しい発想とアイデアを産む方法論であるKJ発想法を中心とする知的生産の技術を駆使して、人間が毎日遭遇する問題を問題として意識して、自らの課題としてとりあげ、解決に導いていく方法論を与えるものである。問題解決学を野外科学・書斎科学・実験科学をつなぐプラグマチックな方法論（図２）とみるとき、前半のV（A～D部）を発想法のプロセス、真中のD～E部を演繹法のプロセス、後半のV（E～H部）を帰納法のプロセスとみなすことができる。

2.2 推論のプロセス

人間の問題解決のプロセスを、最も厳密な論理学の体系の中で、初めてふれたのはアリストテレスの分析論である。彼はその中で問題解決の論理的方法として、演繹法・帰納法・発想法という３つの推論プロセスを挙げている。

そのうち演繹法の体系は、その後中世に至るまで、学問の明晰な方法論として、順調すぎるほど順調に成長していった。他の２つの方法論は、長い間まどろみの時代を過ごした。近世になってベーコンやミルが帰納法の体系の再発見を行い、20世紀に入って推測統計学という学問の世界で帰納論理の急速な整備が行われた。一人取残されていた発想法の体系化については、19世紀後半から20世紀初頭にかけて生存した独創的な哲学者パースの活躍を待たねばならなかった。

2.3 パースの記号学

19世紀後半から20世紀初頭にかけて幾多の著作を残したプラグマティストC.S.パースは、先駆的な記号学者として人間の探求活動の全過程を演繹論理、帰納論理、発想論理[2]の立場から体系化した。

彼の記号学によれば、人間の探求過程とはある未知の問題（驚くべき事実）に遭遇した人間が、その問題を説明する仮説を発見し（発想）、その仮説から導き出される合理的な帰結を推論し（演繹）、そして導出された帰結を検証する（帰納）過程である。この探求の３プロセスは、実はそれぞれ前述のW型問題解決学におけるA～D部分、D～E部分、E～H部分にほかならない。

以上から、人間の知的問題解決活動を支援する知識情報処理システムを構築するには、究極的には演繹・帰納・発想というプロセスを支援するシステムを作らなければならないことが分る。

3．知識情報処理への道

3.1 処理対象の拡大

情報処理装置としての計算機システムの近年の発展は、その処理対象をマシーンよりのものからヒューマンよりのものに急激に拡大しつつある。例えば初期の計算機システムは０、１という二進数の処理中心のものであったが、次第に数値処理やデータ処理向きの計算機システムが作られた。それに伴いユーザの使用する計算機言語も機械語、アセンブラ語、Fortran 、Cobol 、PL/Iへと発展してきた。1970年代、1980年代になって新たに記号／知識処理向きの計算機システムを要求するニーズが高まり、それに伴いユーザもLispやPrologといった知識情報処理用の思考のツールとなりうるような使い易い処理系を求めるようになった。

3.2 推論エンジンの変遷

このような計算機システムの知識情報処理システム志向への発展の中で、計算機システムのガソリンやエンジンにあたる部分も新たなる変容を遂げつつある。ガソリンにあたる部分は知識そのもので、エンジンにあたる部分は知識を処理する機構である。知識とそれを処理する機構の間には密接な関連がある。著者らの立場は、知識を容れる容物を、次のような三種のモデルに分類する。第一のものがファクト型個別知識を格納するデータベースと呼ばれるもので、第二のものがルール型一般知識を格納する知識ベースと呼ばれるもので、そして第三のものが個別知識や一般知識の使い方に関するノウハウ型知識を格納するメタ知識ベースと呼ばれるものである。これら三種のモデルに対応し、著者らはそれぞれ、データベースからの検索機構、知識ベースからの推論機構、メタ知識ベースからのメタ推論機構といった知識処理機構の必要性を喚起してきた。ソフトウェア的にみると、それぞれデータベース管理システム、知識ベース管理システムおよびメタ知識ベース管理システムとして実現できるものである。

3.3 アルゴリズム化の道

２章で述べたような観点から、計算機による知識処理の本質を分析するに、著者らは知的問題解決過程における人間の推論の本質である演繹、帰納、発想[3,4]の計算機システム上での実現方式に注目したい。既存の計算機システム上で演繹的あるいは帰納的推論機構を実現するには、限られた論理の世界を前提とすれば、分解証明法あるいはモデル推論法というアルゴリズムを用いればよいことが知られている[3,4]。発想的推論機構実現のための統一原理は、現時点ではほとんど知られていないが、その中核をなす類推については最近、原口[5]が部分同型に基づく類推の理論を構築中であり、その機械化のための突破口がようやく

見えてきたところである。また著者らによっても類推システムのプロトタイプ研究が行われているが、これについては 7.3節で述べる。

4．メタ推論

4.1 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。

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。

さて著者らの提案する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インタプリタ／コンパイラの使い方に関する知識表現・知識利用技術を確立することにある。次節でメタ推論の実現法を与えることにする。

4.2 demo述語によるメタ推論

第五世代コンピュータ・プロジェクトでは知識情報処理指向の各種アプリケーションを研究開発するために、論理プログラミング言語Prologを土台とする核言語ファミリィを提供しつつある。逐次型（並列型）推論マシン上の機械語がKL0 （KL1 ）で、そこにおけるメタ推論用プリミティブがdemo(simulate)述語といわれる。demo述語とsimulate述語は概念的には等価であり、その相違は主として逐次型計算環境と並列型計算環境という計算環境の相違に従属している。

ここではまず、逐次型Prologでのメタ推論方式について要約する。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[7] 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)

4.3 simulate述語によるメタ推論

次に並列型Prologでのメタ推論方式について要約する。並列メタ推論方式として著者らの提案している最も一般的なメタ推論用simulate述語は、次のような形式[6] をしている。

⑤ simulate(World, NewWorld, Goal, Result, Control)

このsimulate述語は、ある世界World において、与えられたゴールGoalを与えられた制御情報Control 下で解くプロセスを模倣する。模倣の結果、世界World は新世界NewWorldへ更新されると同時に、ゴールを証明していくプロセスそのものから必要なメタ情報Resultが順次抽出されていく。

上記のsimulate述語⑤に対して、次のような省略した形式のものが、しばしばある種の応用では有効である。

⑥ simulate(World, NewWorld, Goal, Result)
⑦ simulate(World, NewWorld, Goal)
⑧ simulate(World, Goal)

なお、対象世界World として、内部データベース自身を用いる場合は⑥～⑧の第一引数World は省略できる。例え

ば、次のメタ述語⑨は“KL1 Interpreter in KL1”あるいはKL1 のメタインタプリタとして知られている。

⑨ simulate(Goal)

5．知識利用機能の実現

5.1 数式処理の制御

メタ推論による知識利用機能の実現例として、数式処理の制御への適用例をあげる．一般にPrologによるメタプログラミング手法を用いると、論理プログラムの制御をかけることが容易である。そこで著者らは、メタプログラミング手法を用いて、高校教科書や大学受験程度の問題を対象とした数式処理システム[7,8] を試作した。その理由は、人間のもっている数式解法のテクニックやノウハウをメタ知識として活用しないと、基本的に問題解決のための探索空間が広がりすぎるからである。一般に方程式を解くために使用可能な各種の規則を無制限に使用すると、探索空間が大きくなりすぎるため、探索空間を絞り込み、無駄な探索を押える必要がある。そこで著者らは、数式処理の制御にメタ推論方式を利用した。その特徴は、次の三つである。

(1) 複雑な処理制御の決定を容易に行える。
(2) 各数式処理規則のモジュール化が図れ、規則の追加変更が容易になる。
(3) 探索空間の絞り込みにより、無駄な探索を押える。

このようなメタ推論に基づく制御を実現するために、demo述語を利用した。これにより、上記の特徴の他に、メタ知識とオブジェクト知識が明確に分離し、拡張性に富むプログラムの実現が可能になった。

図3 メタプログラミングによる数式処理

試作されたシステムは、図３に示されているように、方程式の型の判定を行なう型制御部と適用可能規則の選択を行う規則制御部の二種のメタ推論部をもつ。型制御部では、①代数方程式型、②無理方程式型、③分数方程式型、④指数方程式型、⑤対数方程式型、⑥三角方程式型の６つの型の制御を行っている。。また規則制御部では①公式規則②分離規則、③収集規則、④誘引規則、⑤置換規則、⑥変換規則⑦因数分解規則、⑧展開規則の８つの規則の制御を行う。

5.2 規則制御の実際

demo述語による制御の実際を示すため、規則制御部の処理を行なうプログラムを示す。

```
MD1)  method_demo( Eqn,Ctrl,Ans, _ ,[]):-
        end _check(Ctrl,Eqn,Ans),!.
MD2)  method_demo( E1 && E2,Ctrl,
                   Ans1 && Ans2,His,H3):-
        !,method_demo(E1,Ctrl,Ans1,His,H1),
        method_demo(E2,Ctrl,Ans2,His,H2),
        append(H1,H2,H3).
MD3)  method_demo( Eqn,Ctrl,Ans,His,N_His):-
        strategy(Eqn,His,World, Ctrl,Ctrl1),
        demo(World, solve(Eqn,Ctrl1,Int,
             N_Ctrl, H1),Control ),
        append(H1,His,H2),
        method_demo(Int,N_Ctrl,Ans,H2,H3),
        append(H1,H3,N_His).
```

述語“method_demo”の引数は順に、方程式、制御情報、方程式の解、strategyのための規則使用履歴、トレースのための（解法の難易度を含む）規則使用履歴を示す。MD1)は、方程式Eqn が終了条件及び制約条件Ctrlを満す時に適用され、解をAns に返す。MD2)は、方程式E1 && E2が分割可能を示す演算子&&で結合されている時に適用され、分割したそれぞれの方程式E1,E2 にmethod_demoを適用し、それぞれのトレースのための履歴H1,H2 をappendし、その値をH3に返す。MD3)は、上記以外の時に適用され、strategyによって規則制御部の８種の規則の一つWorld とその時の制御情報Ctrl1 を抽出し、それらに基づき、規則の適用をdemoによって行う。そしてdemoの結果Int を再びmethod_demoに適用し、適用した規則の履歴H1をstrategyのための履歴His に追加しH2とする。そしてトレースのための履歴の追加登録を行う。

数式処理システムの制御にメタ推論を導入することにより、複雑な数式の処理制御が容易に実現でき、数式処理の推論の過程の説明や問題の難易度を容易に与えることができる。解法規則として与えられた８種の規則と６種の方程式の型により処理を行っているが、この枠組の決定は、処

理効率の面からいっても、システムの最適処理実行の構成上重要である。高次の代数方程式を解くための整係数の代数方程式に関する因数分解アルゴリズムなどを、有効に利用する方法の検討も重要である。数式を解く推論過程から得られた難易度の結果を学習し、これにより難易度が最小になるように規則適用順序を制御することも可能である。Prologによる数式処理において、メタ推論に基づく基本機能の容易な拡張性を確認できた。学習機能をもつ数式処理システムへの展開について、今後、検討していく予定である。

6．知識表現機能の実現[19]

6.1 知識表現機能の拡大

メタ推論による知識表現機能の実現例として、知識プログラミング言語Mandala [18,19] の知識表現力を拡大する方法を提示する。人間は複雑な問題に直面すると、もっている膨大な知識および種々の戦略や推論法を使ってその問題を解決しようとする。そのため、人間の知的活動の模倣を計算機上に実現する場合には、人間と同様に様々の知識表現法や対応する推論法を自由に実現することが望ましいと考えられる。核言語第１版のたたき台となった並列型論理プログラミング言語Concurrent Prolog (CP)上に構築された知識プログラミング言語Mandala は、人間の知的活動を計算機上に実現するために考案された言語である。

さてここでは、知識表現力を拡大する機能の一つとして実現されたMandala におけるユーザ定義推論エンジンの組込み機能について述べる。ユーザ定義推論エンジンの組込み機能とは、ユーザ自身で記述した推論エンジンをMandala の推論エンジンとして組込むことを可能とする機能である。ユーザ定義推論エンジンの組込み機能の利点は次の通りであると考えられる。

(1) ユーザ自身で、ユーザ専用の推論エンジンを組込むことが可能となる。
(2) システムとして複数の推論エンジンを容易に提供することが可能となる。
(3) 推論エンジン自体の機能拡張が容易になる。

6.2 Mandala におけるsimulate述語の役割

Mandalaの基本要素は問題解決機である実体（ instance ）とその実体が使用する公理の集合である単位世界（World ）からなる。Mandala の実体は，実体への入力列をゴールとみなして解くCPのプロセスと考えることができる。それは次の形で表される。

```
instance（<名前>，<入力列>，<世界>）.
```

ここで、<名前>は実体の識別子であり、<入力列>は実体が受け取るメッセージ列である。<世界>は、実体が使用する単位世界名と実体固有の値や状態を保持する。このinstance自身はCPの永続的プロセスで、次のように定義される。

```
instance(Name,[Goal | Input],World):-
  simulate( World,NewWorld,Goal),
  instance(Name,[Input?, NewWorld?),
instance(Name,[],World).
```

第１節のsimulateは、入力メッセージGoalをWorld という単位世界で解き、新しい単位世界NewWorldを返す述語で、その結果、instanceはNewWorldを単位世界とする新しいinstanceへと更新される。Goalの解き方はsimulate述語の定義に従っており、この意味でsimulate述語を推論エンジンと見ることができる。simulate述語の定義を変えることにより、それぞれの知識表現向きの推論を行う実体を作ることが可能となる。すなわち、与えられた問題向きのsimulate述語を導入することにより、ユーザはその問題解決に適した種々の知識表現機能を提供される。

図４　Mandala の構成

図４を用いて、実体の生成方法について説明する。実体を生成するためには、ある単位世界のマネージャにcreateメッセージを送る。マネージャの単位世界には、create述語、instance述語やsimulate述語などの定義が記述されている。create述語は実体の識別子としてName、入力列としてInput 、その実体が使用する単位世界としてWorld をもつ一つの実体を生成する。その定義は次の通りである。

```
create(Name,Input,World):-
  instance(Name,Input?,World).
```

上記定義から分るように、create述語はinstance述語を実行するプロセスを生成する。すなわち、生成された実体

はこのinstance述語のプロセスに相当し、マネージャの単位世界に記述されているsimulate述語に従ってゴールを解く。ユーザが、マネージャの単位世界にsimulate述語を記述することができれば、ユーザ定義推論エンジンの組込みが可能となる。以上述べてきたことから分るように、ユーザ定義推論エンジンの組込み機能により、メタレベルのsimulate述語をユーザが組込むことが可能となった。メタレベルで概念の階層構造を表すis_a リンクが使えるので、ユーザはsimulate述語だけを記述すれば良い。節を選ぶための述語はシステム提供の述語を使うことができる。すなわち、is_a リンクによりsimulate述語の機能の拡張も容易となる点に注目されたい。

7. 知識獲得機能の実現

7.1 知識同化による知識獲得

メタ推論による知識獲得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。著者らは、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[9,10,11] である。そのようなシステムの研究試作の途上で、次のような知識ベース管理に関する要素技術が明らかになった。

(1) 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 [11,12,13,14,15] という知識ベース管理の基本技術を明らかにした。
(2) その際、論理型言語Prologでの無矛盾性管理の仕方を明らかにした。これは、統合性制約 [11,12,13,14,15] に基づく論理データベースの管理の考え方の自然な拡張となっている。
(3) 外から与えられた知識が正しいと仮定した時、そのような知識を論証しうるモデルを構築するように、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という過程の管理、すなわち知識調節[16]という知識ベースの管理の基本技術を明らかにした。
(4) 知識同化や知識調節という要素技術の抽出を通じて、論理型言語での知識獲得のパラダイム[10]を提案した。これは従来から人工知能で言われている学習のパラダイムの統合であり、演繹的知識獲得と帰納的知識獲得の自然な統合である。
(5) その際、demo述語によるメタ推論の方式を明らかにし、多くの適用事例[9] を与えた。これにより論理型言語におけるメタプログラミング技法の有用性を実証した。

例えば、ファクト型知識同化機構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[13,14] 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).
assimilate(Currkb, Input, Currkb):-
  demo(Currkb ∪ Input, false).
assimilate(Currkb, Input, Newkb):-
  Info ∈ Currkb, Interkb=Currkb-Info,
  demo(Interkb ∪ Input, Info),
  assimilate(Interkb, Input, Newkb).
assimilate(Currkb, Input, Currkb ∪ Input):-
  independent(Currkb, Input).
```

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。 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。なおCP上のsimulate述語を用いて、assimilate述語を定義することも可能であるが、その場合、並列計算環境特有の困難な問題が派生する。

7.2 知識調節による知識獲得

一方、知識調節プログラムはShapiro によって提案されたモデル推論アルゴリズム（図5参照）を用いて達成された。Shapiro はモデル推論アルゴリズムをホーン論理に適用できるように洗練し、PrologのPrologによるPrologのための知的デバッガを開発した。モデル推論アルゴリズムは矛盾を発見するプログラムと反証を与えた仮説を精密化するプログラムからなる。これら両プログラムは前述のdemo述語を用いてもインプリメントできる[10,16] 。さて著者らは、前述の知識同化機構とShapiro のモデル推論システ

```
Tを□と設定する.        □：空節
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
```

図5　モデル推論アルゴリズム

ムに基づく知識調節機構の両者を中核にし、Prologによる知識獲得支援システムを構築中 [11,12,17] である。知識獲得支援システムにおいては、知識の同化と調節の機構を用いて、知識ベースへの知識の獲得の管理を行っている。この両機構を併用することにより、著者らは知識ベースへの知識獲得支援システム[11,12,16,17] へと統合中である。知識同化と知識調節という考え方を統合する際、最も注意を払ったのは帰納的推論によって得られた論理プログラム（仮説）が正しいかどうかをユーザによって確認する作業である。現在はこの仮説確認実験をユーザの判断に基づいて行っている。この部分は、概念的には、仮説の検定を行っていることを留意しておきたい。

著者らは知識ベース管理について、更に次のような課題を解決した[17]。

(6) 構造化知識としての概念の階層関係を簡潔な形で表現し、この関係の関係に関する（本来は高階述語で表現される）諸性質を系統的にルール化し、それをメタプログラミングによって実現した。

(7) メタ推論を利用して制約用メタ知識を容易に評価実行できることを実証した。なかでも、自然言語処理で良く使われる因果関係の表現に、知識ベースの更新オペレーションを管理するメタ知識であるトリガー述語が有効に適用できることを示した。

(8) 制約用知識の評価実行のタイミングを明確にするために、即時型メタ知識と遅延型メタ知識という評価実行のタイミングを表わす概念を取り入れた。特に遅延型メタ知識の実行管理を行う場合、トランザクションという処理単位で統合することの重要性を具体例をあげて実証した。

(9) メタ・レベルの知識を同化する場合、オブジェクト・レベルの知識のうち信念と呼ばれるものの調節を実現するための基礎技術として、矛盾知識の抽出法とその効率的な修正法を明確にした。

7.3 類推による知識獲得

メタプログラミングを用いると、演繹・帰納のみならず類推を用いた知識獲得支援システムを試作[20]することができる。著者らによって研究試作中の類推システムは、原口の類推理論[5] 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[21]と全く同じであり、次のようなメタプログラムで与えられる。

```
analogy(W,true):-!.
analogy(W,(Goal1,Goal2)):-
  analogy(W,Goal1), analogy(W,Goal2).
analogy(W,Goal):-
  wclause(W,Goal,Body),analogy(W,Body).
analogy(W,Goal):-
  prematch(W,Goal,TW, Tgoal,Tbody),
  transform(W,Goal, Body,TW,Tgoal,Tbody,Apair),
  analogy(TW,Tbody),
  analogy(W,Body),
  call(Apair),
  message(6,['Tranformation has succeeded :']),
  message(10,['resulting analogy is',Apair]).
analogy(W,Goal):-
  message(0,['***Goals',Goal, in,W,'fails***']),
  fail.
```

著者らは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システム[20]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[22]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する戦略の検討を行った。
(4) あた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、仮説生成や発想の本質の議論をするたたき台を提供することができる。

8．部分計算による推論エンジンの高速化

8.1 部分計算

前章までに述べたようにメタプログラミング技法は数式処理における推論の制御、知識プログラミングにおけるユーザ定義推論エンジンの導入、知識ベース管理における知識同化や知識調節といった要素技術の実現等において極めて有効であった。メタプログラミング技法は知識ベースの

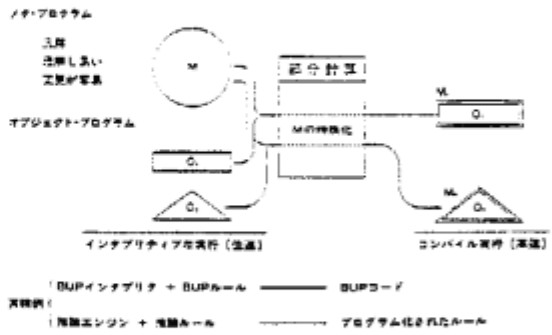

図6 メタプログラムの高速化

エディタやデバッガの開発、論理プログラムの検証や自動合成、Prologと関係データベースとのインタフェース機構の実現、など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、インタープリティブに走る所が多いので、処理性能が遅いことである。しかしながら最近著者らは、部分計算によるメタプログラミング技法の高速化技術を確立した。本章では、その成果を要約する。

プログラムの部分計算とは一般に『稼働環境に関する情報を利用して、汎用のプログラムをより効率の良いプログラムに特殊化すること』と定義される。部分計算の原理は任意のプログラミング言語で書かれたプログラムに適用でき、応用上の意義もコンパイラの自動作成など非常に大きい。部分計算の原則は与えられたプログラムの中で、計算できる部分を計算し、計算できないものは元のままに残して置くことと言える。一般に関数型言語のような値指向の計算においては、変数の値の有無が計算できる、できないにつながり、値がないままに計算する場合には遅延評価などの特殊な評価方式をとらなくてはならない。しかし、Prologの計算は、ユニフィケーションをベースにしているので、基本的に部分計算時に特殊な評価方式を必要としない。このことはPrologでの部分計算の重要な特徴である。

8.2 推論エンジンの高速化

著者らの提案する Prolog プログラムの部分計算法 PEVAL[23,24] はメタインタプリタ的なプログラムを効率の良いプログラムへと変換するのに極めて有効である。実際、図6に示されているように、メタインタプリタにとってオブジェクト・プログラムは入力データ的なものであるので、メタプログラムにオブジェクト・プログラムを与えて部分計算し、特殊化することができる。この特殊化されたプログラムは、機能的にはメタインタプリタ的なものを用いずにすべてをオブジェクト・レベルのプログラムに埋め込んだものに極めて近くなる。従って効率についても、特殊化されたメタプログラムは、メタインタプリタを使わなかったものと同等のものになる。

試作されたPEVAL の効率の向上について述べる。ここにp1は、確信度（CF）を扱う最も簡単なプロダクション・システムでPrologメタインタプリタとその解釈実行されるオブジェクト・レベルのプログラムとからなる。p2は部分計算の前向き進行制御を行うtype述語をデータとして付与した場合の、部分計算の結果であり、この問題向きの特殊化されたプログラムとなっている。p3は更に、部分計算の後向き進行制御を行うinhibit _unfolding 述語を付与した場合の部分計算の結果であり、p2よりオブジェクト・プログラムに良く似た構造をした特殊化されたメタプログラムとなっている。p4はCFなしの場合の結果である。以上述べたp1～p4プログラムの実行効率の比較を表１に示す。表１よりコンパイル実行ではp2プログラムの効率がp1プログラムの約３倍であり、またp4プログラム単体の実行効率の約２／３であることが分る。なお第５章で述べた数式処理システムをPEVAL で部分計算した結果[25]、人間がハンド・コンパイリングした理想的な状態より、５％程度の効率の損失で済むことが分かった。このことはメタプログラミングの解り易さ、開発のし易さからいっても、本手法の有効性を実証している。

	p1	p2	p3	p4
インタプリティブ実行	1674	901	1157	95
コンパイル実行	110	39	46	26

(CPU Time)

表１　実行時間比較[23]

メタプログラミングは Prolog プログラミングの中でその表現力の強力さ故に重要な役割を果しつつあるが、部分計算はこのメタプログラムの実行効率を向上させることができ、メタプログラミングをより実用的なプログラミング技法にすることができる。このことはPrologによる推論システム構築の新しい方法を示唆するものと考えることができる。また本手法はPrologの重要な特徴の一つである段階的プログラム開発の容易さとうまく調和でき、メタインタプリタの段階的特殊化の実現可能性の高いことも示している。このことは推論システムが段階的に推論ルールを獲得する場合においても、本方法が有効に働く可能性を示している。今後の改良点としては、PEVAl をより広い範囲のPrologプログラムを扱える強力なものにすること、およびプログラミング環境の中に部分計算をツールとして有機的に取組む方法について研究することである。

9．おわりに

論理プログラミングにより知識の表現・利用・獲得の機能をもつ知識情報処理システムのプロトタイプを構築するという一大目標を達成するには、解決すべき多くの課題をかかえている。この一大目標を攻略する一つの方法論として、メタプログラミングによるアプローチについて論じた。論理プログラミングにおけるメタプログラミング・アプローチは、理論的にも技術的にも、現在急激にその方法論が整備されつつある。

著者らのアプローチは、まず論理プログラミングに対してdemo述語やsimulate述語に基づくメタ推論方式を確立し、ついでそれを用いた知識の表現・利用・獲得に関する各種アプリケーションを開発し、更にユーザにとって満足する

性能を提供するために部分計算によるメタ推論の高速化方略を確立してゆくという三段階の手順を経た。今後、ESPやGHC 用の部分計算のツールの整備とともに、メタプログラミング技法は知識の段階的コンパイル技法と結び付き、その実用性がますます増大することと思われる。

メタプログラミングを論理型言語Prolog上で実現するに当って、メタ知識に基づくメタ推論機構実現の重要性を喚起してきた。メタ推論機構を用いた知識ベース管理機構の研究が着実に前進しつつあることも述べ、これが前述の演繹・帰納・発想的推論の機構化研究とも密接に関連しており、演繹・帰納・発想的推論機構を持つ知識情報処理システム構築の道筋が見えてきたことも強調した。

以上述べてきたように著者らの研究は、論理プログラミングによる知識情報処理システム構築への第一歩を与えるものである。著者らの研究はいまだ実験段階にあるとはいえ、論理プログラミングと知識情報処理との間に横たわる深くて広いギャップを埋め、両者の橋渡しをとる一本の道を見い出している。この道は、方法論的にはメタプログラミングによる論理プログラミングと知識情報処理技術の融合ということである。

［謝辞］　本研究を行なう機会を与えていただいた渕　一博ICOT研究所長に感謝いたします。また今回報告した内容について、ICOT出向時代をふくみ、有益な討論をしていただいている北上　始（現富士通研）、宮地泰造（現三菱情電研）の両氏に感謝いたします。

［参考文献］

1)　川喜田二郎、牧島信一編著：問題解決学－KJ法ワークブック、講談社、1970.

2)　米盛裕二：パースの記号学、勁草書房、1981.

3)　國藤　進：知識情報処理システムから創造科学へ、オペレーションズ・リサーチ、 261-268、1981年 5月号.

4)　國藤　進：演繹・帰納・発想の推論機構化をめざして、日本創造学会編、創造性研究第３巻、共立出版、1985.

5)　原口　誠：類推の機械化について、同上.

6)　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)

7)　武脇敏晃、宮地泰造、國藤　進、古川康一：Prologによる数式処理システム試作の構想、日本ソフトウェア科学会第１回論文集, 18-4, 29-32, 1984.

8)　Takewaki,T.,Miyachi,T.,Kunifuji,S.,Furukawa,K.: An Algebraic Manipulation System Using Meta-level Inference Based on Human Heuristics ,to appear in ICOT TR,1985.

9)　古川康一、國藤　進、北上　始、宮地泰造：知識情報の取得と管理、昭和59年電気四学会連合大会32-3、1984.

10)　北上　始、國藤　進、宮地泰造、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)

11)　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.

12)　國藤　進、北上　始、宮地泰造、古川康一：論理型言語Prologによる知識ベースの管理、Proc. of the Logic Programming Conference '85, ICOT, 1985.

13)　國藤　進、麻生英樹、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)

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

15)　宮地泰造、國藤　進、古川康一、北上　始：Constraintに基づく論理データベースの管理について、情報処理学会知識工学と人工知能研究会、36-8, 1984.

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 Progrramming, Atlantic City, U.S.A.1, 131/142 (1984)

17)　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、ICOT TR-130,May 1985.

18)　K. Furukawa, A. Takeuchi, S. Kunifuji, H. Yasukawa, H. Ohki, K. Ueda: Mandala : A Logic Based Knowledge Programming System, Proc. of the International C

conference on FGCS'84 , 613-622, 1984.

19) 大木　殷、古川康一、竹内彰一、國藤　進、安川秀樹、三浦敏彦：Mandala Ⅱの拡張機能－ユーザ定義推論エンジンの実現方式－，日本ソフトウェア科学会第１回大会論文集10-2, 53-56, 1984.

20) 諏巻宏治、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回論文集，1985 年11月.

21) 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 LogicProgramming Conference '85, ICOT, 1985.

22) 有川節夫：帰納推論と類推、日本創造学会編、創造性研究第３巻、共立出版、1985.

23) 竹内彰一、近藤浩康、大木　殷、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)

24) 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.

25) 武智敬晃、竹内彰一、國藤　進、古川康一：数式処理システムAMIEへの部分計算の応用と評価、日本ソフトウェア科学会第２回論文集，1985 年11月.