ICOT Technical Report: TM-0070

TM-0070

大規模な知識ベース管理システムのアーキテクチャ

北上　始，国藤　進，
宮地泰造，古川康一

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

大規模な知識ベース管理システムのアーキテクチャ

北上　始，国藤　進，宮地泰造，古川康一
（（財）新世代コンピュータ技術開発機構）

1．はじめに

エキスパートがもつ問題解決能力をコンピュータシステムにもたせるためには、問題解決のために使用する知識を知識ベースに蓄積しなければならない。そのためには、エキスパートがもつ知識を系統的に獲得し、獲得された知識を容易に利用できる機能を実現する必要がある。この様なシステムは、知識ベース管理システムとして知られている。

ところで、エキスパートが問題解決のために、頭の中で利用している知識には、頭の中で明確な形に表現されたことがない知識も含まれている。既に明確に表現されている知識ばかりでなく、このような知識をエキスパートから少しでも多く計算機内で利用可能な形式で獲得するために、次のような問題を解決しなければならない。

(1) エキスパートに、自分のもつ知識をコンピュータ処理できる知識に表現する試みを達成させるために、知識ベース管理システムにとって、どのような支援方法があるかという問題。　(2) エキスパートのモデルをさらに微視的にとらえて、エキスパートがもつメンタルモデルの深い構造的側面から、知識ベース管理システムの知識獲得方式を考えてみようという問題。

深い知識を相手にするようなエキスパートにとっては、(2) の方が(1) よりも見のがせない問題になってくると思うが、著者らは、現在、(1) に関心をもっている。

(1) の問題は、いくつかの異なった分野で別々のアプローチがなされている。データベースの分野では、データベースに対する更新（挿入，削除も含まれる）を行う時に、一貫性を自動的に保持する問題として考えられ、制約条件の記述法，そのチェックの仕方などが研究されてきた。認知心理学の分野では、知識の獲得や学習という問題として研究されてきた。また、この分野で古くから知られている、概念階層関係，CD（概念依存構造）理論などは、深い意味としての知識を獲得する問題を考える上で重要な項目である。形式論理学の分野では、暗黙推論の理論的根拠を与える非単調論理，概念学習の基礎となるモデル推論アルゴリズムなどの研究が論じられてきた。さらに、プログラム方法論の分野では、このような種々の理論および概念を統合するかのように、モジュラープログラミング，抽象データタイプなどの研究が行なわれてきた。

このような研究の背景下で、著者らは、それらのアイデアを統合し、論理型言語 Prolog を使用した大規模な知識ベース管理システムのアーキテクチャについて検討をすすめている。その中でも、著者らは、知識獲得のアーキテクチャに重点をおいて研究・試作を進めている[Kitakami 83-1]。

本論文では、最初に、大規模な知識ベース管理システムのアーキテクチャの大枠とその基礎を述べるために、２章では、知識ベース管理システムの基本構成、３章では、その物理的な容器となる知識ベースのデータ構造について述べる。以下、４章では、知識の否定に対する本論文の考え方について述べ、５章で、信念の管理方式について述べる。６章では、本システムが、どのようにしてエキスパートから知識を獲得するのかについて述べてみたい。そして、この知識獲得の機能を実現するに当り、重要な役割を果たしている諸機能について述べるために、７章で知識の同化、８章で知識の調節、９章で知識の均衡化、１０章で知識の検証、１１章で集合オペレションについて述べる。以上の議論に対する具体例を示すために、１２章では DCG の構文ルールを獲得する問題と知識の均衡化の問題について述べる。最後の１３章では、今後の展開として、分散型知識ベース管理について述べてみたい。なお、本論文で取り扱う知識は、すべてホーン節形式の知識であり、インプリメント用言語は、DEC-10 Prolog を前提としている[Bowen 81]。また、本論文は、著者らが、既に発表ずみの文献 [Kitakami 84-1] に対する、新しい方向への拡張版である。

最後に、本論文の基本構成を下記に示しておく。

はじめに　・・・・・	第１章
全体構成　・・・・・	第２章
基本方式の検討　・・	第３，４，５，６章
各論　・・・・・・・	第７，８，９，１０，１１章
実行例　・・・・・・	第１２章
今後の展開　・・・・	第１３章
おわりに　・・・・・	第１４章

2．知識ベース管理システムの基本構成

図1．に、著者等が考察中の知識ベース管理システムのアーキテクチャについて、その基本構成を示す。

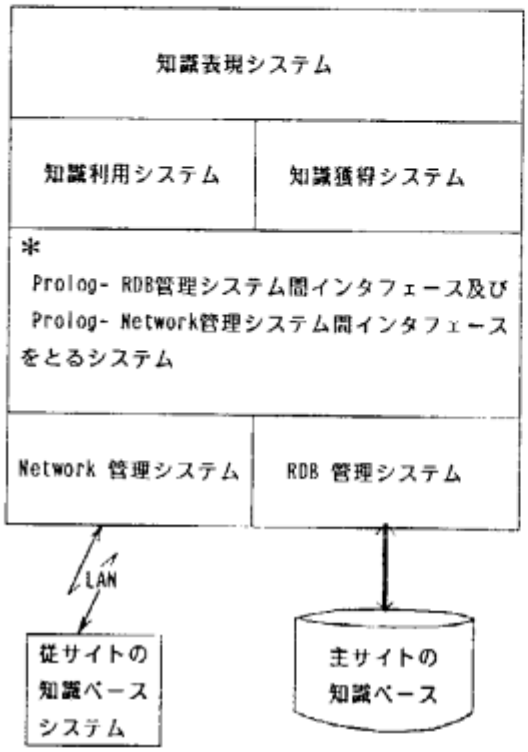


図1．知識ベース管理システム

本システムは、分散処理を意識したシステムアーキテクチャをとっており、 LAN(Local Area Network) [Taguchi 84-1,Taguchi 84-2,Kitakami 84-10] を通してサイト間の知識の交流が行なえるように構成されている。知識ベースの知識は、関係データベース (RDB)管理システム [Hakinouchi 81,Kitakami 83-4] を経由して、二次記憶からの入／出力が行なわれる。この階層では、知識は、リレーションのタップル（ストリング・データ）として扱われる。この入／出力に利用される知識操作コマンドは、関係代数レベルの言語[Furukawa 79] である。関係代数レベルの言語を利用する理由には、(1) 知識の入／出力処理に高速性を期待できる事[Kitakami 83-4] 、(2) 論理型言語とRDB との親和性が良い事[Kunifuji 82,Tanaka 83,Yokota 84]などがあげられる。更に、この関係データベース管理システムは、主サイトがもつ知識の入／出力操作をするために利用するばかりでなく、サイト間のもつ知識を統合するためにも利用される。知識ベースの知識が、問題解決のための推論用知識として扱うのは、＊印のインタフェース用システムから上位のシステムである。＊印のシステムは、 Prolog と同じレベルのインタフェースを、上位システムに提供する。上位のシステムには、三つのシステムがある。その一つは、知識獲得システムであり、エキスパートから知識を無矛盾かつ系統的に取得するためのシステムである。その二つ目は、知識利用システムであり、エキスパートから獲得した知識を使い、ある種の問題解決を行うときに利用される。最後の三つ目は、知識ベース管理システムを利用するエキスパート又は、普通のユーザにとって利用しやすい知的インタフェースを提供する知識表現システムである。以下では、知識の利用・表現システムに関する本格的な議論は別稿に譲り、それらのシステムを実現する上で、最も基本的と思われる幾つかの機能[Furukawa 84-4] について述べてみたい。

3．知識ベースのデータ構造

知識ベースのデータ構造は、知識利用システムを作成するためにも簡単な構造であり、知識ベースに知識を獲得していくうえでも、拡張性に富んだ柔軟な構造をしていなければならない。ここでは、 Minsky のフレーム理論[Minsky 75] に習い、知識ベースはフレームと呼ばれる知識の集合から構成されるとする。フレームは、ある種の限定された領域に適合する知識のかたまりである。ここでは、このフレームをフレーム・オブジェクトと呼ぶ。知識ベース内の各フレームは、有機的な相互関係をもち、種々の問題解決が遂行できる構造をもっているとする。即ち、知識ベースの知識は、このフレームの中に格納されることになる。知識ベースに格納されている知識には、変化し得る知識と変化し得ない知識の二種類があると考えられる。ここでは、前者を、仮定（assume）型知識と呼び、後者を前提（premise)型知識と呼ぶ。以下では、前者の仮定型知識を、信念と呼ぶことにする。

3.1 フレームの階層関係

知識ベースの中には、多数のフレーム・オブジェクトが存在するので、それらのフレーム・オブジェクトを一元管理するような構造が必要である。このフレーム・オブジェクト自身に与えられているフレーム名を、知識のかたまりに付けた知識名と見做し、各フレーム名を知識（ファクト）として管理できる。その管理は、フレーム内の知識の管理と同じ構造で達成できるので、各フレームの管理のために同じフレーム構造を使える。そのためには、そのフレーム名の管理及び、あるフレームから他のフレーム内知識を使用できるような機能が、必要である。これらは、フレーム名管理用述語 " frame（フレーム名）" 及び外部インタフ

ェース用述語 "connect(接続フレーム名，外部の述語)" を用意すれば可能になる。接続フレーム名には、frame(フレーム名) とsite (サイト名) の論理積を記述する。 site 述語を記述しないときは、主サイトと接続する事を意味する。

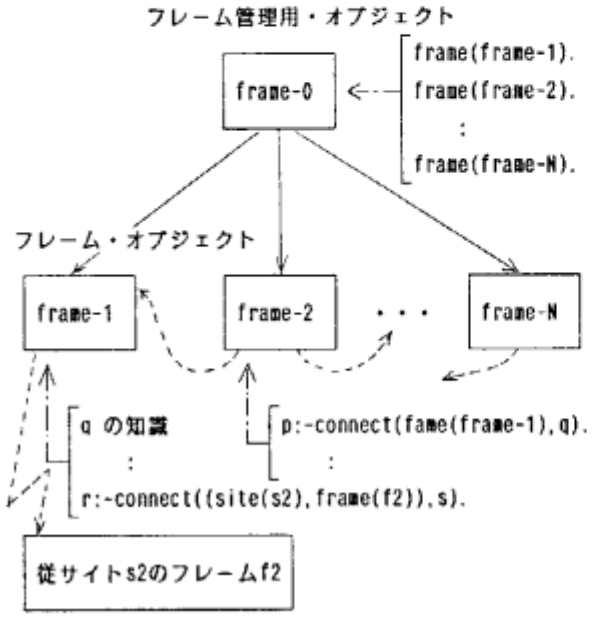


図２．知識ベース内フレームの階層関係
[実線矢印および点線矢印は、各々、制御関係および接続関係を表わす。]

このような状況で、著者らは、以上のフレームを、再帰的な構造をもったシステムにより管理しなければならないと考えている。図２は、知識ベース内のフレーム階層関係を示した図である。この図では、フレーム・オブジェクトを一元管理するためのフレームを、フレーム管理用・オブジェクトと呼んでいる。フレーム・オブジェクトは、エキスパートから獲得した知識を蓄積するフレームを指す。

更に、今まで、フレーム管理用・オブジェクトとして利用していたものをフレーム・オブジェクトと見做し、その上に新しくフレーム管理用・オブジェクトを定義する事も考えられる。

3.2 フレーム内の知識群

次に、このフレームの中を、さらに詳細に検討してみよう。これは、フレーム内の知識を分類することから検討を進めていかなければならない。DEC-10 Prolog でインプリメントしたプログラムを見ると良くわかるように、プログラムを、ある対象世界が記述されているルールとファクトの集合と見做せる。この集合は、ある対象領域の知識を適切に表現している。以後、この知識の集合を、対象知識と呼ぶ。この対象知識に対する (1)意味解釈のためのオペレーションおよび (2)更新処理のためのオペレーションをできるだけ高度なものになるようにするために、対象知識に対する制約条件をメタ知識として表現し、蓄積・管理しておいた方がよい。

(1) の意味解釈については、概念階層関係という立場で対象知識を意味づける方法がある。概念階層関係には、is-a ,has-a,property関係などが良く知られている。どの関係も二引数をもつ概念であり、ファクトとして登録できる。これらの知識をフレーム内に獲得すると、対象知識に対しての意味処理が簡単になる。以後、この知識を、意味解釈用知識と呼ぶ。

(2) の更新については、対象知識に対する静的な制約条件を is-inconsistent述語で表現し、動的な制約条件を trigger述語で表現できる。この知識は、対象知識だけでなく意味解釈用知識にも適用することができる。これらの知識の詳細は、後述する。以後、これらの更新に関して制約を与える知識を、更新制約用知識と呼ぶ。

ここまでの説明では、フレーム内に存在すべき知識として、対象知識，意味解釈用知識，更新制約用知識の三種類の知識群が登場したことになる。これらの知識の種類，個々の知識がもつ構文上の枠組み，使用上の制限（述語引数のデータ・タイプおよび入／出力仕様など）などを辞書知識として管理することによって、さらに高度な処理を本システムのユーザに提供できる。例えば、抽象データタイプの考え方 [Ong 84] を採用し、述語引数のデータタイプおよび本システムが提供する各種のオペレーションを新しく定義するときは、すべてこの辞書知識を通して実施できる。辞書知識は、知識ベース管理システムが独自に管理するものであり、エキスパートには、その参照だけを許すことができる。以上から、フレーム内の知識群を図示すると、図３のようになる。

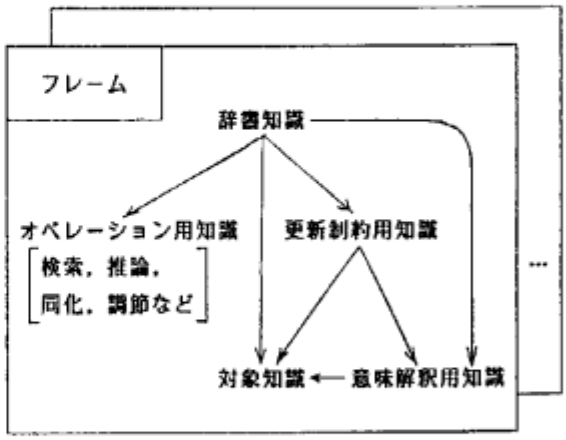


図３．フレーム内の知識群（矢印は、制御関係を表わす）

図３のオペレーション用知識は、フレーム内の知識を利用又は更新に使用するオペレータである。フレーム内に格納する知識の形式に、ある種の制限があるときは、その形態に合ったオペレーション用知識を選択または定義することによって、効率的な処理を期待できる。例えば、あるフレーム内の全知識を論理和もカット・オペレータも使わずに表現する場合は、そのような知識を推論評価するための demo 述語は、次のように非常に簡単な形式になる。

```
demo(true) :- !.
demo((P,Q)) :- demo(P),demo(Q).
demo(P) :- get-clause(P,Q),demo(Q).
```

この demo 述語は、 Prolog で記述された Prolog のインタプリターになっている。この第１行目は、ゴールを証明していったときの停止条件である、第２行目では、論理積の展開証明を行っている。第３行目では、P と同じ帰結部をもつホーン節(P:-Q)をさがし、Q の証明を行っている。また、この get-clause 述語は、DEC-10 Prolog の clause 述語（システム述語）と等価である。以下では、フレーム内に蓄積する知識は Prolog で記述されるような形式の知識であるとして、話を進める。したがって、推論用述語及び検索用述語は、各々、この demo 述語及び get-clause 述語を機能拡張した述語である。

3.3 知識の格納構造

ここでは、フレーム内の各知識群を、物理的に別々の場所に蓄積するものとし、どの知識も、前提型知識あるいは、仮定型知識のどちらかに区別するものとする。本報告で扱うフレーム内の知識は、形状的には、すべて区別がなく、以下の構造をしているものとする。この知識を、いかなる知識群として利用するかは、辞書知識に記述されている。

フレームの名前（ KID, 確信の種別, ホーン型の知識）

ここで、確信の種別を、premise 又は assume とし、ホーン型の知識を、 Prolog 形式の知識で示す。
"KID" は、知識の識別子であり、(1) "KID" に基づく直接検索, (2) 冗長知識間の識別, (3) 知識の格納管理上の順序関係の維持など、を実現するために必要不可欠である。

図３の五種の知識群の中で、対象知識内のルール, 更新制約用知識, オペレーション用知識を、Prologの General Clause または前提部分のある Ground Clause で表現できる。それ以外の知識を、前提部分のない Ground Clause で表現できる。両 Clause ともに、主記憶上で使用しない限り、二次記憶上に蓄積しなければならない。

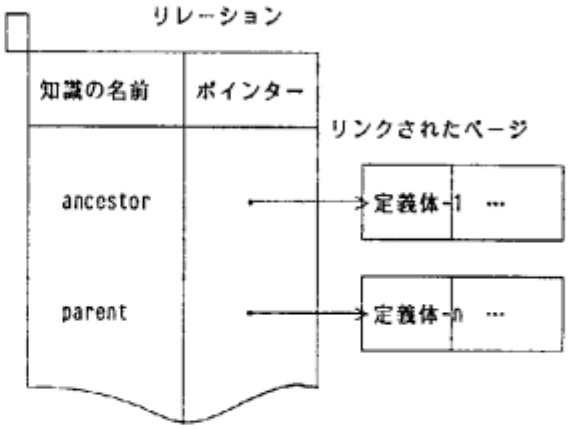


図４．関係データベースによるテキスト・データの管理

前提部分のない Ground Clause（ファクト）を、関係データベースのタップルとして蓄積できるが、それ以外の Clause （ルール）は、その構造が可変長の構造になっているので、テキスト・データと同じ扱いをしなければならない。テキスト・データを関係データベースで実現するための一つの方法としては、図４に示す方法がある。この方法によると、可変構造をもつ知識を、必要な数だけリンクしたページ内に蓄積できる。また、テキスト・データを簡単にアクセスするためには、更に知的なインタフェースを考える必要がある。

このような事情から、同じ知識名をもつファクトは、一つのリレーションに蓄積され、その知識名は、リレーション名と一致させることができる。また、ルール（ファクトを混在させても良い）を、図４に示すリレーションに蓄積できるが、このリレーション名をシステムのユーザに見せないようにしなければならない。

3.4 メタ推論

これまでにも述べてきたように、知識ベースの知識は、通常の Prolog プログラムとして知識よりも多くの制御構造をもった複雑なデータ構造の中に格納されている。この複雑な構造は、知識ベースそのものが拡張性に富んだ柔軟な構造をし、多くの問題解決に耐えられなければならないという現実的な制約から生じている。したがって、単純な証明機能だけしかもたない demo(Goals) 形式の述語だけでシステムをインプリメントするのは、不十分であり、demo(Frame,Goals,Control,Result) 形式の demo 述語を作成・使用しなければならなくなる[Kunifuji 83-7] 。以下、この述語による推論をメタ推論と呼ぶことにする。
この demo 述語は、そのフレーム Frame の中で、与えられた制御 Control に従って、与えられたゴール列 Goals

を証明し、その証明プロセスで必要なメタ情報 Result を抽出することができる [Kitakami 84-1] 。この多種の機能をもつ demo 述語は、知識ベース管理システムをインプリメントするために利用できる以外に、メタ知識を表現するためのツールとしても利用できる。以下の説明では、説明を簡単にするために、この多種の機能をもつ demo 述語をもちだすことを避け、問題毎に特殊化（抽象化）した demo 述語で解説を試みる。ここで、メタ知識とは、"既存知識をいかに使用するか" を表現する知識であり、エキスパートは、このメタ知識を表現するために demo 述語を使用することができる。

3.5 更新オペレーションと一貫性保持

知識ベースの更新に際しては、必ず更新制約用知識と矛盾しないように、一貫性を保持しなければならない。ある対象知識 Object に対する更新オペレーション Access-Operation と一貫性保持をするためのタイミングの間には、深い関係がある。本システムでは、更新オペレーションと一貫性の判定は、次のように実施すべきであると考えている。

```
access-operation(Object):-
    apply(Access-Operation-for-the-Object),
    check-consistency(Access-operation,Object).
access-operation(Object):-
    restore(The-Object).
```

これは、更新オペレーションとそれに対する一貫性保持に関する基本的な枠組みを与えるものであり、この枠を越えない範囲でインプリメントを実施しなければならない。また、更新オペレーションそのものに assert,retract 等のような副作用をもつ述語が使用されているときは、最後の access-operation を見るとわかるように、知識ベースをもとの状態に戻さなければならない。しかし、副作用のない assert,retract 相当の述語がインプリメントされかつそれを使用すれば [Warren 84]、その最後の行の処理は、不要になる。

次に更新制約用知識の中で、静的な制約を与えるメタ知識のシンタックスを説明しておこう。

```
is-inconsistent( Access-Operator, Access-Object) :-
    member( Access-Operator, Operator-List),!,
    Inconsistent-Condition.
```

Access-Operator には、insert,delete,update等の基本命令をはじめとして assimilate,accommodate 等があげられる。エキスパート自身が、独自の更新 Operator を作成・定義したときは、この Access-Operator として記述できる。Access-Object には、更新対象の知識の呼び出し形式を記述する。この制約条件により、この名前の知識についての矛盾判定を実施していることになる。矛盾判定条件は、Inconsistent-Condition で記述される、その判定は、次のゴールの true/false で判定される。

```
demo(Frame ,is-inconsistent(Operator,Object)).
```

この実行結果が "true" のときこの知識Objectに矛盾が発生したことになる。通常のルールを前提としてゴールの証明をする場合、偽の証明の通知は、demo述語そのものが fail する事により通知される。一方、例外値をもつルールも扱えるようにするためには、偽の通知の仕方に工夫がいる。即ち、この場合は、 demo 述語が、それ自身、 fail して通知するのではなく、正常終了し例外値の "false" であることを通知したい。そのためには、この demo 述語の引数を増し、その引数に exception-false を返すようにすれば良い[Kunifuji 83-7,Kitakami 84-4] 。

3.6 意味解釈用知識について

ここでは、意味解釈用知識の一つとして概念階層関係をとりあげてみよう。前述したように、この関係には、三種類の関係がある。

一つ目は、ある知識概念のもつ特性を記述する関係であり、 property と呼ばれる。その形式を、次のように表現できる。

```
propety(知識名 ,その知識の特性を表す知識名).
```

例えば、 "X が鳥ならば、X は空を飛べる" を表現するルール (canfly(X):-bird(X)) には、" 鳥は、空を飛べる" という意味 (property(bird,canfly))を与えることができる。

二つ目は、 is-a 関係と呼ばれ、この関係を、上位知識概念の特性 (property) を下位知識概念に継承する (inherit)ために利用する。逆に言うと、エキスパートが、 is-a 関係を定義するときは、上位知識概念の特性 (property) が下位知識概念に継承するか否かについて、十分に検討しなければならない。 is-a 関係は、次の形式の知識として表現される[Ong 84]。

```
is-a (下位知識名 ,上位知識名).
```

この is-a 関係は、上位の特性を下位に継承するというルールがあるので、そのルールを記述すると、次のようになる。

```
super-concept(Frame,X,Y) :-
    demo(Frame,is-a(X,Y)).
super-concept(Frame,X,Y) :-
    demo(Frame,is-a(X,Z)),
    super-concept(Frame,Z,Y).

inherit(Frame,X,Y) :-
    demo(frame,super-concept(X,Z)),
    demo(Frame,property(Z,Y)).
```

ここで、super-concept(Frame,X,Y)は、そのフレーム Frame の中で、与えられた知識名 X に対する上位知識を捜し、その一つを知識名 Y に返す述語である。inherit(Frame,X,Y) は、そのフレーム frame の中で、与えられた知識名 X に対する特性知識を捜し、その特性知識の名前を返す述語である。

最後の三つ目は、has-a 関係と呼ばれ、この関係は、上位知識概念と下位知識概念の部分関係を表現しており、下位から上位への特性の継承が許す場合と許さない場合がある。この上下関係を、次の形式で表現する。

```
has-a(上位知識名 ,下位知識名).
```

ある知識概念に対する全ての部分知識概念を調べるために、次のルール(sub-concept) を定義する事ができる。

```
sub-concept(Frame,X,Y) :-
    demo(Frame,has-a(X,Y)).
sub-concept(Frame,X,Y) :-
    demo(Frame,has-a(X,Z)),
    sub-concept(Frame,Z,Y).
```

ここで、sub-concept(Frame,X,Y)は、そのフレーム Frame の中で、与えられた知識名 X に対する部分知識を捜し、その一つを知識名 Y に返す述語である。

3.7 辞書知識について

辞書知識には、フレーム内に存在する知識群の各知識に対して、その枠組みを与える知識をファクトの形式で登録している。ここでは、その中でも代表的なものについてだけ論じてみよう。辞書知識の代表的なものを以下に示す。

(1) 知識を構成する属性情報は、
"constitute(知識名 ,属性番号 ,データタイプ ,属性名)" で、表現できる。
(2) 知識を定義するのに使われる知識（組込み述語）については、"build（知識名 ,利用する知識名）" で、表現できる。
(3) 知識群の種別を、
"describe(知識群の分類名 ,定義されている知識名)"で、表現する。知識群の分類名には、辞書知識，オペレーション用知識，更新制約用知識，意味解釈用知識，対象知識などがあり、各々の分類名として、dictionary,operation,constraint,semantics,object を与える。
例えば、この表現例として、

```
describe(semantics,is-a).
describe(constraint,is-inconsistent).
describe(object,append).
```

等を、列挙することができる。

さて、この辞書知識の使い方に対する一例を示してみよう。ここでは、知識名を与え、その知識名に対する呼び出し形式を作成する事にする。例えば、二本のリストを一本のリストに結合する append 述語は、append(X,Y,Z) という呼び出し形式を持っている。これは、次のルール(build-mgt) により作成できる。

```
build-mgt(Frame,X,Y) :-
    setof(A,demo(Frame,constitute(X,A, -,-)),Alist),
    Z=..[X| Alist],get-most-general-term(Z,Y).
```

この、"build-mgt" 述語は、その X に、求めるべき知識名を与えると、その知識名に対する知識を Y に返す述語である。

4．知識の否定

本論文では、否定知識の一般的扱いが非常にむずかしいことから、次の解釈にもとずく知識を導入するだけにとどめ、否定知識の代りに使用する。ある知識に対する否定は、その知識のホーン節の帰結部に示されている述語名に"not-"を付けるか取り除くかで表現されるとする。従って、両知識間のデータ依存関係は、何もないとする。例えば、canfly(ostrich) の否定は、not-canfly(ostrich) であり、この形式以外に推論上の関係は、何もないとする。前者は、“ダチョウは、飛べる”と読み、後者は、“ダチョウは、飛べない”と読む。また、not-canfly(ostrich) の否定は、canfly(ostrich)で表現する。即ち、両知識は、記号的に異るというだけの意味しか持たないので、その本来の意味は、ユーザが管理する事になる。

特に、ユーザが矛盾する知識の共存（例えば、not-canfly(X) とcanfly(X) の共存）を許したくない場合は、次の更新制約用知識として定義しておけばよい。

```
is-inconsistent( X, -):-
                member(X,[insert,update]),!,
                canfly(Y), not-canfly(Y).
```

更新制約用知識に対する否定は、特に、この知識の前提部に記述されているゴールを　\+（ゴール）とすることにより実現できる。" \+ "は、カッコの中の評価結果が "true" であれば、失敗（ fail ）し、失敗すれば、成功（ success）するシステムの組込み述語である。

5．信念の管理

古くから議論されている信念システム(Belief System)は、TMS,RMS と呼ばれ [Doyle 78,79,Chariak 80] 、Lispでインプリメントされていたが、Prologでインプリメントすると、 Data Dependencyの扱いを、あまり意識する必要がなく自然に吸収する事ができる。即ち、ある証明をする際に、 Data Dependencyを使用し、前提と帰結の関係を利用する推論問題を、Prologのインタプリンタそのものを動かすことにより達成できる。また、もしその証明問題を解くために、ある種の制御[Goodwin 82]又は、証明プロセスの記録取得を必要とするときは、Prologの処理系の上にdemo述語と呼ばれる述語を作成すれば、容易にそれが可能になる[Kitakami 84-4] 。

知識ベースには、明確な知識を示す premise 型知識と不明確な知識を示す assume 型知識がある。この assume 型知識は、その知識自身が、知識ベースに獲得された時点から持ち続けている知識ベースの信念であると見ることができる。したがって、 assume 型知識は、知識ベースの知識を収集するにつれ、ある時点で、新しい信念に修正されるかも知れないという側面をもっている。この様な収集プロセスでは、知識ベースに、無矛盾な知識も矛盾する知識も収集されることになる。次章で述べるように、信念の修正は、知識獲得における二種の均衡化プロセスで実施される。本章では、信念を管理するうえで、最も重要な信念の修正方式について述べてみよう。信念の修正は、(1) 矛盾の原因となる修正信念の候補を収集し、(2) その候補を不明確な順に並べ、(3) その順番で、信念を妥当な信念に置換することにより矛盾の解消が達成される。(2) の処理については、現在、その実現方針を検討中であり、今後の研究成果が期待されるが、ここでは、そのアイデアを述べるだけにとどめたい。

以下、(1),(2),(3) の各項目について、 5.1 , 5.2, 5.3 の各節で述べてみたい。また、 5.4 節では、例外値の扱いについて述べてみることにする。

5.1 信念の修正候補

ある状況下で、矛盾が発生したとき、その矛盾を解消するためには、最初に、既存知識群から信念候補を抽出しなければならない。矛盾は、種々の知識群の組合せで生じるが、ここでは、更新制約用知識と対象知識との間に発生する矛盾解消問題について述べてみる。

更新制約用知識に矛盾する知識は、 3.2 節の述語に、 assume 型の知識を収集する機構を付加すればよい。図６は、その結果、得られた demo 述語である。ここでは、信念管理の本質を明らかにするために、フレーム内の知識を単純（論理和、カットオペレータ、その他の複雑な処理を省く）なものとし、できるだけ単純なdemo述語の上で、説明を試みる。

```
demo(Frame, true,d(X,X)) :- !.
demo(Frame, (P,Q),d(X,Z)) :-
    demo(Frame, P,d(X,Y)),
    demo(Frame, Q,d(Y,Z)).
demo(Frame, P,d(X,Y)) :-
    system(P)->call(P),X=Y;
    get-clause(Frame,P,Q,Certainty),
    (Certainty==premise->X=Z;
     X=[(P:-Q)|Z]),demo(Frame, Q,d(Z,Y))).
```

図６．単純なdemo述語の例

更新制約用知識に矛盾する対象知識を修正する時、修正すべき assume 型の対象知識を、取得するために、 demo 述語の第三引数を、 d-list 表現し、証明プロセスで情報を取得する機能を与えている。図中の get-clause は、Clause の格納順に帰結部が P のClause (P:-Q) を捜す述語である。

5.2 真実性(Verisimilitude)比較オペレータ

矛盾の原因となる修正信念の候補が収集された後、それらの候補の中から本当に修正しなければならない信念を出来るだけ効率的に選択しなければならない。そのためには、二つの信念の真実性を比較するためのオペレータ（ルール）が必要である。これを、真実性比較オペレータと呼ぶ事にする。t1,t2 を二つの信念とするとき、"t1 が t2 よりも真実性をもつ信念である" という主張は、次の事実を満足するときに可能である[Popper 68] 。

(1) t2 は t1 よりも精確な主張を行うことができ、これらの主張は、多くのテストに耐えられる。
(2) t2 は t1 を説明でき、t1よりも多くの事実をもっている。
(3) t2 は t1 よりも詳細に多くの事実を説明出来る。
(4) t2 は t1 でパスしなかったテストにパスした。
(5) t2 はそれ自身が設計される前に考えられていなかった多種の新しい実験的テストを提案し、それらのテストにパスした。
(6) t2 は今まで無縁だった種々の問題を、統合又は結合した。

以上のオペレータを t1,t2 に適用しても、どちらが確信のある信念か区別できないときは、エキスパートに、どちらかの信念を選択させることも考えられる。

5.3 信念修正オペレータ

信念の修正は、知識獲得プロセスで知識体系の矛盾を解消するために実施されるが、ある幾つかの信念が矛盾の原因になっていることがわかると、次の信念修正オペレータのいずれかで、それらの信念を修正し、矛盾を解消することができる。

(1) エキスパートが事前に信念の別解を与えていたとき、その別解を新しい信念とする。
(2) 信念の否定をとる。
(3) 修正信念が例外知識であると宣言されたとき、ルールに対しては、 except 述語をゴール節に付加し、ファクトに対しては、負のファクトを知識ベースに登録する。
(4) 矛盾検出時に、エキスパートが信念の別解を直接与える。

5.4 例外値の扱い

最初に、修正される知識を、対象知識としてみよう。修正される対象知識がファクトのとき、その否定を作成し、修正対象の知識と置換すれば、信念の修正が達成される。ただし、修正の対象となる知識は、d-listにより取得された知識(assume 型知識) に限られる。対象知識がルールのときは、一般に、エキスパートが事前に与えた別解がそのルールにたいする新しい信念となるが、最も簡単には、そのルールの否定をとれば良い。ここで、修正対象ルールが例外値処理をするルールであることがわかっている場合は、次の例のように修正すれば良い。

例えば、既存知識が

```
bird (ostrich).
canfly(X):- bird(X).
```

であれば、

```
bird(ostrich).
canfly(X):- bird(X),
  except( Frame , not-canfly(X)).
not-canfly(ostrich).
```

と修正する。

また、既にルール中に例外値処理をする述語がある場合は、単に例外知識（ファクト）の否定だけを追加すれば良い。このように、知識の修正に際し、修正知識を否定する処理は、例外値処理をする知識があるなしにかかわらずわず同じである。

修正される知識が d-list の中に存在しない時、更新制約用知識が修正の候補になる。この修正もその否定をとることにより達成できる。

6. 知識適応プロセス

著者らは、知識適応プロセスを、認知心理学の分野[Hatano 82] で知られる人間（生態）と外界（環境）の間の相互作用と等価なモデルとしてとらえている。即ち、知識ベース管理システムにおける知識適応プロセスを、コンピュータがもつ知識ベースと、エキスパート間の相互作用であると見做している。ここでは、この相互作用の説明を行うために、認知心理学の用語を使用する。これにより、以後の説明では、知識ベースおよびエキスパートは、それぞれ上記の人間および外界に対応させる。

ところで、著者らは、この知識適応プロセスが、知識獲得プロセスの特殊な場合であるととらえている。したがって知識獲得プロセスによる学習の問題から説明をしていくことにしよう。知識ベースが知識獲得による学習を行う為には、知識の同化，調節，均衡化等のプロセスを支援する機能が重要である。図5に、これらの機能を使用した知識獲得プロセスの概略フローを示す。

知識の同化とは、エキスパートがもつ入力対象の知識を、知識ベースへ無矛盾かつ系統的に取得するプロセスである。ここでは、知識ベースの知識に誤りがないことを前提にしている。また、ここでの入力知識には、 Ground Clause ばかりでなく General Clause も含まれる。著者らは、この機構の具体的なインプリメントの方法について、既に、文献 [Kitakami 83-5,84-1,Miyachi 83-1,84-1,Kunifuji 83-5]で報告した。

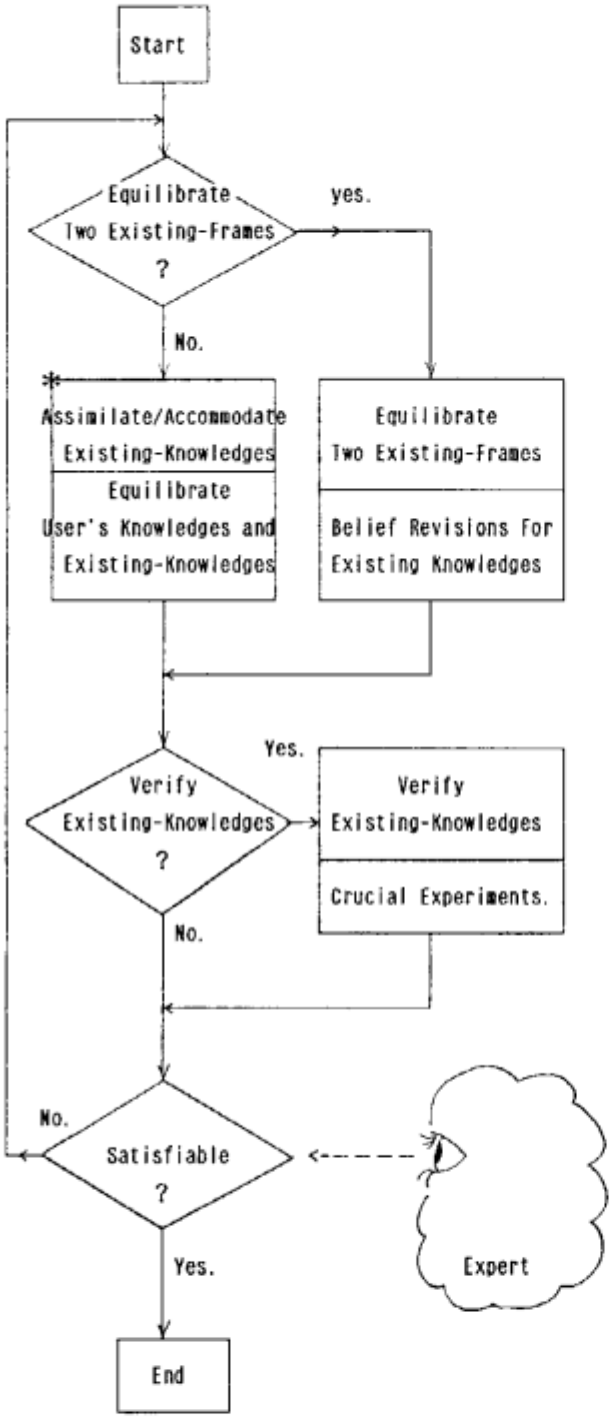


図５．知識ベースの知識獲得プロセス
＊印は、同化（Assimilate）又は
調節（ Accommodate）による知識の
適応プロセスである。

知識の調節とは、エキスパートがもつ知識（ファクト）が、正しいものとして、知識ベース内の知識を無矛盾かつ系統的に更新するプロセスを意味する。ただし、エキスパートが直接、知識ベース内の知識の更新方法を知っているときは、無矛盾性のチエックだけをシステムにまかせれば良いことになる。この具体的なインプリメントの方法についても、著者らは、既に、文献［Kitakami 84-1]で報告ずみである。

知識の均衡化には、一般に、二種類の均衡化が考えられる。一つは、エキスパートがもつ知識と知識ベース化されて既存知識間の均衡化である。これは、知識ベース管理システム自身に“どのような知識が不足しているのか”とか“どのような知識を必要としているのか”などを判断するメタ認知的メカニズムも含めて検討する必要があるが、現在のところ知識の同化，調節，検証の組み合せだけによる均衡化に限定している。ここでは、このプロセスを、特に、知識適応プロセスと呼ぶことにする。図５によると、エキスパートと知識獲得システムとの間で行なわれる会話は、知識獲得システムがエキスパートのもつ知識に適応しようとするために生じる行為であると見做せる。獲得された知識に対する検証については、エキスパートが、適切なゴール列を入力し、その結果を観測することによって達成できる（Crucial Experiments ）。

次に、知識の均衡化として、他の種類の均衡化について述べてみよう。これは、二つの既存知識群に対する均衡化であり、異なるフレーム間の知識群ばかりでなく同じフレーム内の知識群について、それらの整合性をとる時に重要な機能となる。

ところで、エキスパートが、誤った知識を知識ベースに蓄積してしまう場合がある。本システムの設計思想によると、次のどちらかのプロセスで、誤った知識が知識ベースから除外され、正しい知識が追加される事になる。

(1) 図５の検証を終え、ここで検証された誤りの知識を調節プロセスで修正する時（知識の適応プロセス）。
(2) 既存知識群の均衡化を行っている時(知識の均衡化プロセス）。

7．知識の同化

ここでは、知識同化プロセスで利用する知識同化のアルゴリズムを紹介する。このアルゴリズムは、Bowen［Bowen 82 ］のアイデアに基づいてインプリメントされてきた[Hiyachi 84-1,Kitakami 84-1] が、ここでそのアルゴリズムを、もう一度、整理してみよう。

但し、知識同化の処理を、 assimilate(フレーム名 ,入力知識）の述語で達成するものとし、インプリメントに

際して、次の前提条件を置く事にする。

(1) エキスパートがもっている入力対象知識は、知識ベースの辞書知識に登録ずみの概念から構成する。
(2) 更新制約用知識群は、それ自身、矛盾していない。
(3) 入力対象の知識は、Prolog で記述されたホーン節である。
(4) 知識同化が許された知識は、既存知識の次に格納される。
(5) 知識同化のアルゴリズムで処理される入力知識はすべて assume 型の知識である。

以上の前定条件から、知識同化のアルゴリズムを整理してみると、次のようになる。

(A1) エキスパートが入力しようとしている知識(General-Clause又はGround-Instance)が、既存知識(フレーム)から演繹できるか否かを、deduce(フレーム名 ,入力知識) 述語で判定し、その結果が、

true のとき
　入力知識が、既存知識にとって冗長であることをエキスパートに通知し、次の手続に進むか否かの判断をエキスパートにまかせる。
fail のとき
　次の手続に進む。

deduce 述語の具体的インプリメント方法は、著者らの文献 [Kitakami 84-1] を参照されたい。

(A2) 入力知識を知識ベースに挿入し、更新制約用知識に反するか否かを
demo(フレーム名,is-inconsistent(insert,その知識)
でチェックし、その結果が、

true のとき
　そのフレームは、矛盾をかかえることになるので、この段階で挿入した入力知識を削除し、エラー終了する。
fail のとき
　次の手続へ進む前に、この入力知識を既存知識に追加することによって、他の知識の概念が拡大する場合は、他の知識について、無矛盾性を上記方法で判定する。

(A3) この入力知識を既存知識に挿入することにより、他の概念に冗長性が発生したとき、システムは、その概念について冗長性を除去するか否かをエキスパートに問い合せ、エキスパートの指示にしたがって処理を進める。これは、
eliminate-redundancy (フレーム名,KID)
述語により達成される。第二引き数の"KID" は、(A2)で挿入された入力知識に対して割り当てられた知識の識別子であり、冗長性の除去は、この知識を除く他の知識について実施しなければならない[Kitakami 84-1] 。

(A4) 上記の冗長性除去の処理が、矛盾を発生させたことになるか否かについて、冗長性が除去された全ての概念を対象に demo(フレーム名 ,is-inconsistent(delete, 冗長性が除去された知識)) 述語を実行する。

true のとき
　その概念についての冗長性除去が行なわなかった状態に戻す。
fail のとき
　この概念の冗長性除去を正しいものと見做す。

このアルゴリズムの中で、知識を既存知識の中に直接的に挿入するオペレータが利用されているが、これは、次の形式の述語である。
insert (フレーム名 ,知識の種別 ,入力知識, KID)
ここで、"KID" は、入力知識を挿入後、割り当てられる知識の識別子である。この知識同化では、知識の種別は、assume 型知識に限っているが、このオペレータ insert を使用すると、premise 型知識を直接挿入できる。

8. 知識の調節

この調節プロセスでは、エキスパートは、既存知識を調節する上で、完全教師に近い存在であり、このエキスパートの指示に従い、既存知識が無矛盾かつ系統的に修正される。但し、エキスパートがこの場で与える知識は、 assume 型のファクトであるとするが、このファクトは、知識ベース内のいかなる assume 型知識(信念)よりも確信が高いとする。このファクトを assume 型知識にした理由は、教師も間違うことがあるという観点にたっている。知識ベース内に蓄積されている assume 型知識は、それは、繰返し利用し、正当性の確認が何回も行われたときに、 premise 型知識となる。また、この調節に必要とするパラメータとしては、辞書知識の情報を利用するものとする。

以上の検討に基づく知識の調節を実現するために、著者らは、Shapiro のモデル推論システム [Shapiro 82] を機能拡張し、その正当性を研究している [Kitakami 84-1]。

知識の調節アルゴリズムの大枠を示すと、次の様になる。但し、知識調節の述語を、 accommodate (フレーム名, 正/負のファクト) の形式で利用する。負のファクトは、正のファクトに "not-" を付けて表現される。このファクトは、調節対象知識である。

(R1) エキスパートにより与えられた正または負のファクトに基づき、既存知識を修正する。既存知識の修正は、(1) 新知識を既存知識に付加するか、(2) 既存知識を新しい知識に置換するかで実施される。

(R2) 新しく修正された既存知識が、更新制約用知識に矛盾しているか否かを、

上記 (1)のとき
demo(フレーム名,is-inconsistent(insert,知識)

上記 (2)のとき
demo(フレーム名,is-inconsistent(delete,知識)

で判定し、もし矛盾すれば (その結果が "true" になった場合)、 (R1) に戻り、次の新知識を捜す。逆に、矛盾しなければ、次の処理に進む。

(R3) エキスパートが次のファクトを与える場合は、(R1) の処理にもどり、これ以上ファクトを与えない場合は、最後に冗長性の検出を行い、もし、その冗長性を除去したいときは、その指示にしたがって、所定の概念の冗長性を除去する。

以下では、 (R1) を実現するアルゴリズムの詳細を 8.1 節で述べ、その中で中心的な役割を果たす精密化オペレータについて、 8.2節で述べる。最後に、 8.3 節で、 (R1) の処理を高速化するために、どのようなストラテジが必要であるかについて述べる。

8.1 モデル推論アルゴリズム

図7. は、Shapiro により紹介されたモデル推論アルゴリズムである。この図で、 F_n =< OBS_n , TF > は、観測データ (正又は負のファクト) であり、観測文 OBS_n は、ファクトである。 TF には、 "true" 又は、 "false" の真偽値を与える。このアルゴリズムによる処理は、外部から観測データを与えている限り継続される。

本アルゴリズムによると、全ての仮説は、矛盾 "□" を出発点として、その精密化を実施していくとしている。最終的には、作成された仮説 H_i の集合として定義される推測 L_p を、得ることになる。その中で、 "false"印が付けられてない仮説の集合が、解である。 "false" 印が付けられている知識 (仮説) は、 "not-" が付けられている知識 (仮説) と同じ意味をもつ。図7で記述されている演繹結果 (演繹可能/演繹不可) を、 demo 述語により判定できる。また、矛盾探索アルゴリズムを、 demo 述語を少し変形することにより実現できる。即ち、負のファクトが、推測 L_p から演繹されるとき、その原因となる Clause が反駁仮説になるので、 demo 述語の証明過程で、その Clause を抽出する機構をつければよい。

```
S_false = {□}, S_true = { }.
L_0 = {□} とし、□に "false"印を付ける。
Repeat
  次の観測データ F_n =< OBS_n , TF >を読み、
  OBS_n を、 S_TFに追加する。
  Repeat
    While { S_false に属する OBSが L_p から演繹可能 }
      矛盾探索アルゴリズムにより、反駁仮説を見つけ、
      その仮説に "false"印を付ける。
    While { S_trueに属する OBS_i が L_p から演繹不可 }
      "false"印を付けてある仮説に、精密化オペレータを適用し、 OBS_i が L_q ( =L_p かつ H_i ) から
      演繹可能となるような新仮説 H_i を、見つける。
      そして、その新仮説を、 L_p に付け加える。
  Until {上記、While ループのどちらも満足しない}
  L_p の中で "false"印を付けてない仮説を出力する。
Forever.
```

図7. モデル推論アルゴリズム

[S_{true} / S_{false} は、各々、正 /負の観測文を保管するファクトの集合である。]

8.2 精密化オペレータ

図7の精密化オペレータについて説明してみよう。このオペレータは、反駁仮説として "p" をとり、その仮説に対する新仮説の候補として "q" を作成する。そのために、次のいずれかのルールを "p" に適応することになる。

(1) p ::= □ならば、q ::=a(X1,X2,X3,...,Xn) とする。ここで、"a" は、帰納的に一般化される述語名(概念名)であり、n 個の引き数をもつ。

(2) p(X1,X2,X3,...,Xn) に対して、 Xi と Xj をユニファイし、その結果を q とする。

ここで、 Xi と Xj は、異なる変数である。

(3) p(X1,X2,X3,...,Xn) に対して、 Xi に、t(Y1,Y2,Y3,...,Ym)を割り付け、その結果を q とする。ここで、 t は、 m 引き数の functor である。

(4) p ::=(Head :- Goals) に対して、q ::=(Head :- Goals,Gn) とする。ここで、 Gn は、辞書知識などに登録されているユーザ定義述語か又は、再帰に定義される述語である。

8.3 高速化ストラテジー

本アルゴリズムを効率良く実行させるためのストラテジーをまとめると、次のようになる。

(1) 辞書知識に登録されている引き数のデータ・タイプを利用し、精密化オペレータ(8.2 節)の (2)で示されるユニファイの組合せ数を低減する。即ち、このユニファイは、同じデータ・タイプ同志だけしか許されない。

(2) 辞書知識に登録されている組込み述語名を利用し、出来るだけ早く新仮説を作成する。適切な組込み述語を事前に定義していないときは、あらゆる可能性を尽くして新仮説を作成することになるので、多くの作成時間を必要とする。

(3) 辞書知識に登録されている引き数の入／出力仕様を利用し、出力引き数が、入力引き数に対して可制御な構造だけを妥当な仮説と見做すことができる。

(4) 仮説（ホーン節）の条件部の中に同じ振舞いをするゴールを存在させないようにする。これは、むやみに長い条件部をもつ仮説の作成を禁止しているので、新仮説を見つけるための探　索空間を狭めることに役立っている。

(5) ループを許さない仮説にだけ着目する。

(6) 反駁仮説を有効利用し、再計算を防止する[Kitakami 84-1] 。

9. 知識の均衡化

ここでは、二つの既存知識群に対して、整合性をとる問題について述べたい。著者らは、これを知識の均衡化の問題としてとらえている。これを、equilibrate(フレーム名-1 ,フレーム名-2)の形式の述語で達成するものとする。同一フレーム内の均衡化を実施する場合は、フレーム名-1とフレーム名-2は、一致する。同一フレーム内の均衡化を、多くの対象知識を獲得した後で、更新制約用知識が収集されたときに実施する。

さて、整合性のとり方には、三つの問題が含まれている。一つは、更新制約用知識群と対象知識群または、意味制約用知識群とに対する一貫性保持の問題であり、この問題は、既に述べた方法で解決できる。二つ目は、両フレームがもつ共通の概念について、演繹されるインスタンス集合の差を比較し、お互いにその集合が一致するように両知識群を修正する問題である。ここで修正される知識は、信念として位置づけられている知識であり、この修正方法については、５章でのべた。最後の三は、対象知識または意味解釈用知識から更新制約用知識を合成することによって、整合性を保つという高度な処理をあげることができる。

１０．知識の検証

この検証には沢山の方法があるが、ここでは、次に述べる二つの方法を採用している。一つは、エキスパートがもっている知識の真偽値と、知識ベースが演繹結果として結論づける真偽値を比較する方法である。この方法による検証は、 deduce 述語に基づいてインプリメントすると、次の形式の述語により達成できる。

verify-knowledge(フレーム名 ,ルール／ファクト)

他の一は、知識ベースから演繹されるインスタンスを確認することである。この方法による検証については、そのインスタンスとエキスパートがもつファクトを照合することで実施できる。このインスタンスを演繹するための述語は、次の形式をしている。

get-instance(フレーム名 ,ゴール列)

これは、 demo 述語で実現できる。

１１．集合オペレーション

これまでの議論では、すべて、単一の Clause に基づいた知識ベースへのアクセスだけに注目してきた。これは、 Prolog がもつ言語の特殊性から生ずるものであるが、もっと操作性の高い機能を実現するためには、集合操作ができるユーザインタフェースを用意しなければならない。たとえば、あるフレーム frame 内に存在する特定の概念 p(X1,X2,...,Xn) について、そのインスタンスをすべて出力したい時は、

```
?- setof([X1,X2,...,Xn],
        get-instance(frame,p(X1,X2,...,Xn)),Set),
   display-list(Set).
```

とすればよい。ただし、 display-list は、リスト Setの要素をすべて出力する述語である。

ところで、この例でみられるように、 setof と display-list の間では、リストでデータの授受を行っているが、

Cocurrent Prolog [Shapiro 84]などの並列処理用言語が実用化されると、このデータの授受をストリームで行う事ができる。

１２．実行例

ここでは、シェークスピアの物語で知られる「真夏の夜の夢」の登場人物を利用して、 DCG の構文ルールを獲得する問題と、知識の均衡化問題について考えて見よう。

12.1 DCG の構文ルールを獲得する問題

ここでは、問題を単純化するために、次のような制限をおく。

(1) DCG の構文ルール s(X,[]) の形式は、CFG を対象としており、知識ベースの"dcg-frame" フレームには、次のような対象知識が既に存在するとする。

```
n([ hermia | X],X).      ・・・・・・・・・  名詞
n([ lysander | X],X).    ・・・・・・・・・  名詞
n([ forest | X],X).      ・・・・・・・・・  名詞
vi([ walks | X],X).      ・・・・・・・・・  自動詞
vt([ loves | X],X).      ・・・・・・・・・  他動詞
pp([ in | X],X).         ・・・・・・・・・  前置詞
vp(X,Y):-vi(X,Y).        ・・・・・・・・・  動詞句
vp(X,Y):-vt(X,Z),n(z,Y).
preps(X,Y):-pp(X,Z),n(Z,Y).      ・・・  前置詞句
```

(2) 構文ルール s(X,[]) から演繹される文章は、全て、知識ベース内にある既存知識（単語及びルール）から構成されるとする。そのために、更新制約用知識として、次のメタルールが既に与えられているとする。

```
is-inconsistent( insert,s(-,-) ) :-
                    s(X,[]),exist-variable(X).
```

以下、この例に基づいて、各実行プロセスの解説をすることにしよう。ここで、 exist-variable 述語は、文章の中に変数があれば、 "true" を返し，無ければfailする述語である。

（プロセス１） 自然言語処理のエキスパートが DCG の構文ルールの一つを既に知っていたとしよう。このような状況では、"assimilate"命令を使用し、その知識を知識ベースに登録することができる。但し、下記の % 印は、知識獲得システム上のインタープリタと Prolog 上のインタープリタのプロンプテングを区別するために付けられている記号である。

```
%?- assimilate( dcg-frame,
                (s(X,Y):-n(X,Z),vp(Z,W))).

* The Input Knowledge"(s(X,Y):-n(X,Z),vp(Z,W))"
is inconsistent with the Constraint Knowledge
" is-inconsistent( insert,s(-,-) ) ) :-
                  s(X,[]),exist-variable(X) ".

* Reject the Input Knowledge.

yes
```

ここでは、入力知識を見ると分るように"vp"の第二引き数を"Y" とせず"W" にしてしまったので、更新制約用知識に違反し、知識の同化が禁止された。

（プロセス２）前のプロセスの結果に注意し、正しい入力知識を登録することにしよう。

```
%?- assimilate(dcg-frame,(s(X,Y):-n(X,Z),vp(Z,Y)))

* Eliminate Redundancy (y/n) ? n.

* The Input Knowledge "(s(X,Y):-n(X,Z),vp(Z,Y))"
 was acquired.

yes
```

このプロセスで、はじめて、エキスパートがもつ知識の一つが知識ベースに登録されたことになる。

（プロセス３）登録された構文ルールが正しいかどうかを検証するために、獲得された構文ルール s(X,[]) から演繹される全ての文章を、出力してみよう。

```
%?- setof(X,get-instance(dcg-frame,s(X,[])),Set),
    display-list(Set).

[[forest,loves,forest]]
          :
[[hermia,loves,lysander]]
          :
[[lysander,walks]]

yes
```

ここで出力された全ての文章（１２個）は、構文的に正しいが、前置詞句の表現がないことに気がつくであろう。

（プロセス４）そこで、前置詞句をもつ文章として、"hermia walks in forest"が正しい文章であることがわかっているとしよう。このときは、次のようにして、"accommodato" 命令を使って、知識ベースを調節し、より正しそうな構文ルールを作成することができる。

```
%?- accommodate(dcg-frame,
      s([hermia,walks,in,forest],[])).
          :
          :
found Clause (s(X,Y):-n(X,U),vp(U,W),preps(W,Y))
    after searching 29 clauses.

Listing of s(X,Y):
  (s(X,Y):-n(X,U),vp(U,Y)).
  (s(X,Y):-n(X,U),vp(U,W),preps(W,Y)).

Checking fact(s) ... no error found.
          :
yes
```

このプロセスでは、正しい構文ルールを見つけるまでに、次の三つの仮説が更新制約用知識により除去された。

- s(X,[]).
- s([X,Y|Z],U):-preps(Z,U).
- s([X|Y],Z):-vp(Y,V),preps(V,Z).

この更新制約用知識が登録されていない場合は、この三つの仮説を除去するために、エキスパートは、更に三つの事実を入力しなければならなくなる。

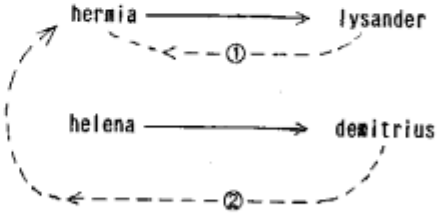
（プロセス５）これまでに獲得された知識に対する正当性を検証するために、プロセス３と同じ事を試みる。

```
%?- setof(X,get-instance(dcg-frame,s(X,[])),Set),
    display-list(Set).

[[forest,loves,forest]]
          :
[[hermia,loves,lysander,in,forest]]
          :
[[hermia,walks,in,forest]]
          :
[[lysander,walks,in,forest]]

yes
```

ここでは、４８個の構文的に正しい文章が出力された。このプロセスで、エキスパートが、構文ルールの獲得結果に対し十分な満足をもてば、構文ルールを獲得するための試みを止める事になるであろう。

12.2 「真夏の夜の夢」の物語における、登場人物の愛情関係の均衡化問題（これは、Jon Doyle がとりあげた問題である）。

loves(X,Y)という述語をＸ→Ｙというグラフで表現すると、ここで登場する４人の人々の愛情関係は、次の様になっている。ただし、assume型の知識は、点線で表現する。



この関係を、次に示すファクトとして、表現することができる。

```
loves(hermia,lysander).      …  premise
loves(helena,demitrius).     …  premise
loves(demitrius,hermia).     …  assume
loves(lysander,hermia).      …  assume
```

この他に、" lysanderは、ジェラシーの持ち主である (jealous(lysander)) " 、" ジェラシーをもつ人は他人を殺すことがある (kill(X,Y) :- jealous(X),loves(X,Y),loves(Z,Y),not-equal(X,Z)) " 、" ある人に愛情をもつ人にとって相思相愛が成立していなければ、その人は自殺する (kill(X,X) :- loves(X,Y),loves(Y,Z),not-equal(X,Z))" 、" hermiaを愛していない人は、helenaを愛している(loves(X,helena) :- not-loves(X,hermia))" 、というルールが与えられている。 また、この物語は、人が１人でも死ねば悲劇になるので、この物語に関係しているエキスパートは、悲劇になる事を避けるための更新制約用知識(is-inconsistent(insert, -) :- tragedy) を、メタ知識としてこのフレームに蓄積するものとする。対象知識、更新制約用知識は、フレーム名"story-frame" に蓄積されることを前提にする。

この場（対象知識が表現する場）の設定によると、"story-frame" フレームが、矛盾をかかえる事になるので、エキスパートは、システムに、それを避けるために、対象知識の修正を実施させる事になる。これは、フレーム内に存在する知識の均衡化問題として、とらえられる。以下に、その実行トレースを示す。

```
%?- setof([X,Y],
    get-instance(story-frame,loves(X,Y),Set),
    display-list(Set).

[demitrius,hermia]
[helena,demitrius]
[hermia,lysander]
[lysander,hermia]

yes
%?- insert(story-frame,premise,
    (is-inconsistent(insert,-):-tragedy),KID).

1 clause inserted in the "story-frame" Frame.

yes
%?- equilibrate(story-frame,story-frame).

        :

*end equilibration.

yes

%?- setof([X,Y],
    get-instance(story-frame,loves(X,Y),Set),
    display-list(Set).

[demitrius,helena]
[helena,demitrius]
[hermia,lysander]
[lysander,hermia]

yes
```

この結果、②のパスがhelenaに修正されることで、信念の修正が終了した事がわかる。

１３．分散型知識ベース管理に向けて

第五世代コンピュータ・プロジェクトでは、知識情報処理システムのプロトタイプの構築を目標としている。現在、パーソナル型逐次推論(PSI) マシンと関係データベースマシン (Delta)の試作・検討を実施している。PSI マシンは、ESP と呼ばれる論理プログラミング言語 (Prolog レベルの言語) で書かれたプログラムを実行するマシンであり、Delta は、関係代数レベルのコマンド列(Queryなど) を実行するマシンである。

さらに、 ICOT では、数台のPSI マシンと１台の関係データベースマシンが LAN(Local Area Network)により接続する予定である。PSI マシンは、ネットワークサブシステムをもつ。このサブシステムには、LAN を経由したPSI － Delta マシン間およびPSI －PSI マシン間の通信を可能にするコマンドが備えられている。

図８に、本プロジェクトの第１ステージ（前期３年間）におけるハードウェア構成を示す。
この図の破線は、本プロジェクトの中期（第２ステージ）に達成予定の PSIマシンと、Delta 間の Direct Coupling を、表わしているが、 Direct Coupling の議論は、今回の報告には含めない。

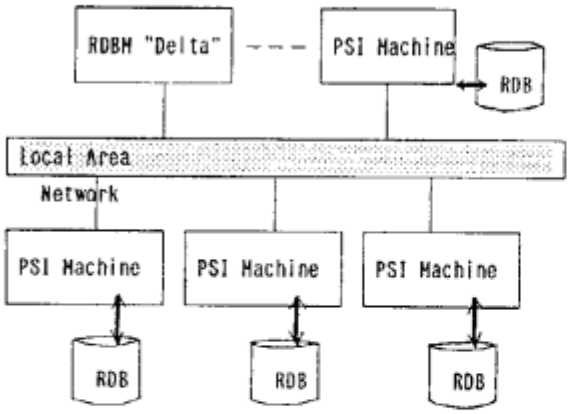


図８．第５世代コンピュータのシステム構成

関係データベースは、Delta ばかりでなく、各々の、PSI マシンにも存在している。

この様な、環境下において、PSI マシンが備えているPrologベースの論理プログラミング言語（ESP)と、PSI マシン及びDelta がもつ関係データベース操作用言語（関係代数レベルの言語）の各々とインタフェースをとり、サイト間の情報交換を円滑に遂行するソフトウェアが必要である。即ち、このソフトウェアは、大規模な知識獲得システムの分散処理を実現するうえで重要な機能を提供する。以下では、この機能を提供するシステムを、知識インタフェースシステムと呼ぶ。

図９は、知識インタフェースシステムの前後に存在するソフトウェアシステムの階層関係を図示したものである。

本論文では、このようなソフトウェア環境下で位置づけられている知識インタフェースシステムのアーキテクチャについて述べる。

13.1 知識インタフェースシステムの概略構成

図１０に、本システムの概略構成図を示す。

本システムにPrologのゴール列が渡されると、 System manager は、次の手順に従い、Prologのゴール列を実行する。

図１０の上から順に、各サブシステムが実行される。Separator は、ゴール列を User Program の上で部分評価し、他の場所で評価すべきゴール列を抽出する。部分評価は、 demo 述語を利用することによって達成される。抽出されるゴール列は、評価される場所単位に分類されている。Translatorは、Separator の分類に従って抽出されたゴール列を、それぞれのインタフェース仕様（ネットワーク側とデータベース側の仕様）に合った関係代数レベルのコマンド列に変換する。変換されたコマンド列が、主サイト実行か従サイト実行かによって、そのコマンド列は、 Database Connector か Network Connector に送られる事になる。Database Connectorは、主サイトのデータベース管理システムを使用するための前処理を行い、その結果を知識操作システムに送る。その結果、一時リレーションが取得される。Network Connector は、従サイトの仕様に合わせて変換されたコマンド列をネットワーク経由で、送信又は、その結果を受信する機能をもつ。ここで、Network Connector の一機能である Cache Managerは、ネットワークシステムと知識蓄積システムの間でデータの受け渡しを行うサブシステムである。ネットワークシステムから取得されたパケットデータ（検索結果のデータ）は、知識獲得システムが内部機構の一つとして持つバッファ管理サブシステムに、直接送られる。ユーザは、このデータを、一時リレーションとして利用できる。

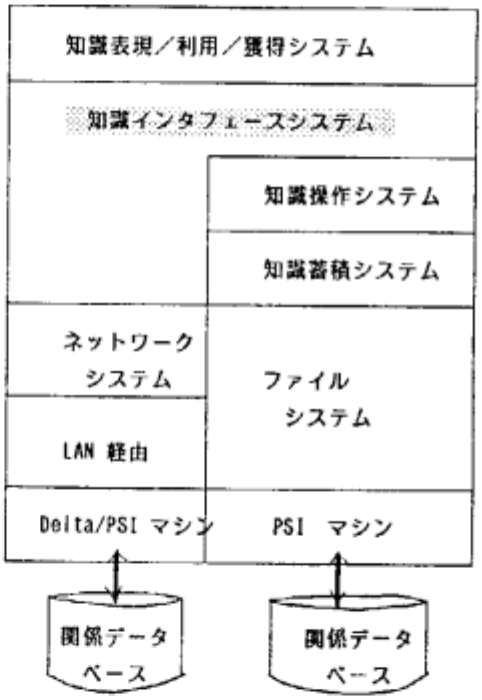


図９．知識表現／利用／獲得システムを支えるソフトウェアシステム（知識操作システム，知識蓄積システムは、各々、関係代数，タップル操作レベルのインタフェースを提供する）。

以上のプロセスを経由して、一度に複数の一時リレーションが取得された時、知識操作システムは、それらをカルテジアン演算またはジョイン演算等を実施し、一個の一時リレーションにまとめ上る。これらの演算は、ユーザが与えたゴールの形式に従って選択される。最終的にまとめあげられた一個の一時リレーションのデータ (Facts)は、ユーザが与えたゴール別の別解の集合であり、これは、格納順に割り付けられる。

13.2 分散知識に対する更新

主サイトに存在する手続き (general clause) が、水平成分および、垂直成分とするリレーションで、定義されている時、その手続きをとおしてデータの更新を行う方法についてのべる。ここで、その手続きの定義に使われているリレーションは、従サイトのみならず主サイトにも存在するものとする。

どのサイトのリレーションに対してもデータの直接的な更新は、insert,delete の命令だけで達成できるが、上記、手続きを利用して、その手続きの定義に使われている分散知識を更新する方法としては、trigger の概念を導入する方法がある。trigger は、以下に示される更新制約用知識である。

```
trigger( Access-Operator, Access-Object) :-
    member( Access-Operator, Operator-List),!,
    Revise-Condition, Revise-Specification.
```

この方式によると、更新に対する不確定性が含まれないように、更新の方法を詳細に記述できる。

主サイト側に工場部門 a があり、従サイト側に営業部門 b がある場合の更新問題を例にあげ考えてみよう。

主サイト及び従サイトには、各々、次のような知識があるとする。

(1) 主サイト

サイト名 = site1

フレーム名 = factory

知識データ = この部門で生産された製品を管理する為に、次のようなリレーション (ファクト) がある。

product(製品名 ,製造年月日)

(2) 従サイト

サイト名 = site2

フレーム名 = business

知識データ = この部門で販売管理をする為に、次のようなリレーション (ファクト) がある。

stock(製品名 ,製造元名)

sales(製品名 ,納品先名)

この様な状況で、生産された製品をすぐに販売のための在庫品としてよい場合は、 product としてファクトが登録された時に、自動的に stock を更新するためのメタ述語が必要になる。このメタ述語は、次のように記述される。

```
trigger(insert,product(X,Y)) :-
  connect(site(site2),
          insert(business,premise,stock(X,a),-) ).
```

例えば、主サイトで、一台のコンピュータが製造されたときは、次のような手順で trigger 述語が実行される。

```
%?- X-computer,Y="1984/7/24",
    insert( factory, product(X,Y),- ),
    demo( factory, trigger( insert, product(X,Y))).
```

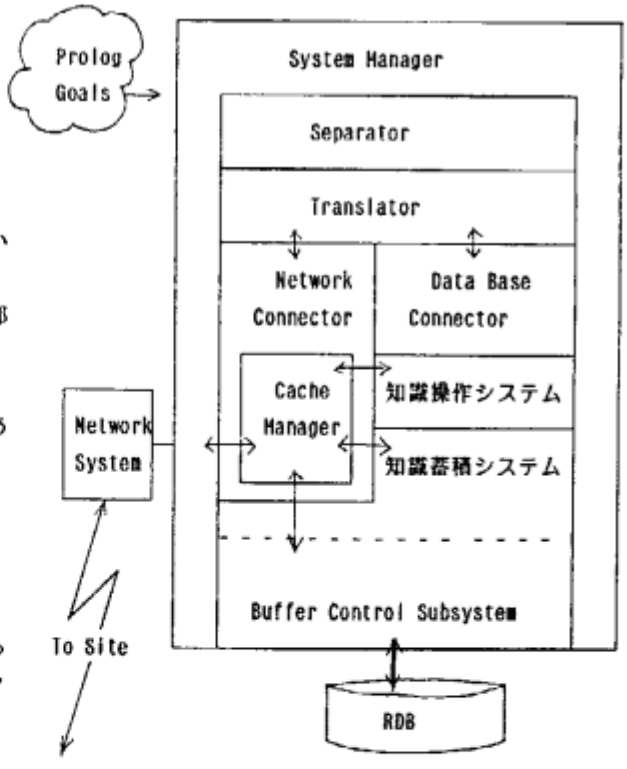


図10. 知識インタフェースシステムの概略構成

14. おわりに

本論文では、大規模な知識ベース管理システムのアーキテクチャについて、その基本的な考え方について述べた。設計目標としては、知識ベースの拡張性および知識ベース管理が容易になる事とし、知識ベースの基本的なデータ構造について述べた。また、エキスパートがもつ知識を系統的かつ無矛盾に獲得するために、知識の適応プロセス、知識の均衡化プロセスで利用される重要な機能について述べた。これらの機能が、認知心理学の分野で知られる知識の同化、調節、均衡化に対応するものであり、これは信念を上手に管理し、成長させていくための基本機能であると見做すことができる。

また、本システムは、ネットワーク環境下でも、分散型の知識ベース管理システムとして、十分な機能を提供する土台にもなる事を示した。

一方、このモデルは、エキスパートと知識ベース間の談話モデルまたは分散型知識ベースにおけるサイト間の知的通信モデルなどを実現するための基礎であると考えられる。

信念の表現方法を、確信度 (Certainty Factor) という尺度を利用して、系統的な管理方式の実現問題は、今後の課題である。そこでは、知識のアクセス回数などの統計情報を系統的に取得する問題が含まれている。この知識のアクセス回数は、信念の真実性を上げる為の基礎知識であると考えられる。更に、自然言語インタフェースにつながりやすい知識表現言語の設定、並列演算メカニズムの導入、意味概念の系統的な獲得方式、時相論理の組込み方式など、も今後の大きな課題である。

更に、本報告では、排他制御，リカバリー，機密保護などの方式については、述べなかった。これらは、本格的な分散型知識ベース管理システムを構築する上で重要な課題である。

［謝辞］

本研究の機会を与えて下さったICOT研究所長・淵一博氏および、有益な討論に参加していただいたICOT第一研究室・村上国男　室長をはじめとする柴山茂樹，横田治夫，田口昭二，宮崎収兄の各氏、ICOT第二研究室・服部隆　氏、三菱電気・溝口徹夫，ニ石章純　各氏（ICOT- KBTGのメンバー）、富士通研究所・手塚正義　氏に深謝いたします。

［参考文献］

（Bowen 81）　Bowen,D.L.,DECsystem-10 PROLOG User's Manual,Dept of AI,University of Edinburgh (1981).

（Bowen 82）　Bowen,K.A., Kowalski,R.A.　Amalgamating Language and Meta-Language in Logic Programming (K.L.Clark and S.-A.Taernlund eds.),Academic Press,pp. 153- 172, 1982.

（Charniak 80 ）　Charniak, E.,et.al, Artificial Intelligence Programming, Lawrence Erlbaum　Associates 1980.

（Doyle 78）　Doyle,J.,　Truth　Maintenance System for　Problem　Solving, MIT Technical Report AI-TR-419 1978.

（Doyle 79）　Doyle,J.,　Truth Maintenanence System, Artificial Intelligence,Vol. 12, No.3 1979.

（Goodwin 82）　Goodwin,J.W.,　An Improved Algorithm for Non-Monotonic Dependency Net Update, Linkoping Institute of Technology, Technical Report 1982.

（Furukawa 79 ）古川康一，データベースの知的アクセスに関する研究，電子総合研究所研究報告　第 804号 1979

（Furukawa 83 ）　Furukawa,K., Takeuchi,A., and Kunifuji,S.,　Handala : A Knowledge Representation System in Concurrent Prolog, Information　Society of Japan,Preprints of WG on Knowledge Engineering and Artificial Intelligence,Nov.1983.

（Furukawa 84 -1）　古川康一，竹内彰一，國藤　進，Handala ,　A unified system for modular programming and knowledge representation on Concurrent Prolog,京都大学数理解析研究所，「情報の構造化と意味に関する研究」研究集会，1984年 2月．

（Furukawa 84 -2）　古川康一，竹内彰一，國藤　進，Handala : Concurrent Prolog上の知識プログラミング言語／システム，情報処理学会第28回全国大会，電気通信大学，1984年 3月．

〔Furukawa 84-3〕 古川康一，國藤 進，竹内彰一，上田和紀，核言語第１版概念仕様書，新世代コンピュータ技術開発機構，ICOT TR-054 ，March 1984.

〔Furukawa 84-4〕 Furukawa,K., Takeuchi,A., Kunifuji,S., and Yasukawa,H., Mandala : A Logic based Knowledge Programming System, Institute for New Generation Computer Technology, ICOT TR-069, April 1984.

〔Hatano 82〕 波多野誼余夫，認知心理学講座 ,第４巻 ,東京大学出版会 (1982).

〔Kitakami 83 -1〕 北上 始，宮地泰造，國藤 進，古川康一，知識ベースシステムKAISERの構想，情報処理学会第26回全国大会，東京工業大学，1983年 3月.

〔Kitakami 83 -2〕 北上 始，宮地泰造，國藤 進，古川康一，KAISERにおける関係代数的言語の特徴，同上.

〔Kitakami 83 -3〕 北上 始，麻生盛敏，國藤 進，宮地泰造，古川康一，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資料 30-2,1983年 6月 (IR-010)

〔Kitakami 83 -4〕 北上 始，牧之内顕文，手塚正義，安達 進，関係データベース RDB/V1 の最適化技法 ,情報処理学会論文誌 (1983).

〔Kitakami 83 -5〕 北上 始，宮地泰造，國藤 進，古川康一，知識獲得システムの一実現法，情報処理学会第27回全国人会，名古屋大学，1983年10月(IM-0017).

〔Kitakami 84-1〕 Kitakami,H., Kunifuji,S., Miyachi,T., and Furukawa,K.,A Methodology for Implementation of A Knowledge Acquisition System, Proc. of the 1984 International Symposium on Logic Programming,Atlantic City, U.S.A., Feb. 6-9, 1984, also available from ICOT as ICOT Technical Report IR-037 1984.

〔Kitakami 84-2〕 北上 始，古川康一，プログラムのデバッキングについて，ICOT Technical Memo.,TM-0033 , 1984.

〔Kitakami 84-3〕 北上 始，"Inductive Inference of Theories from Facts"の日本語訳，ICOT Techical Memo.,TM-0050,1984.

〔Kitakami 84-4〕 北上 始，国藤進，古川康一，宮地泰造，PrologによるBelief System の実現方法，ICOT Technical Memo., TM-0052,1984.

〔Kitakami 84-5〕 北上 始，平川秀樹，古川康一，知識獲得と DCG ,情報処理学会第２８会全国大会 1984.

〔Kitakami 84-6〕 北上 始，宮地泰造，古川康一，國藤 進，古川康一，KAISER/KIM Architecture, ICOT Technical Memo.,TM-0063 1984.

〔Kitakami 84-7〕 北上 始，知識の適応過程を支援する知識獲得システム，ICOT Technical Memo.,TM-0064 1984.

〔Kitakami 84-8〕 北上 始，古川康一，国藤進，宮地泰造，知識獲得システムと知識適応プロセス(1) ，情報処理学会第２９回全国大会， 1984.

〔Kitakami 84-9〕 北上 始，国藤進，古川康一，宮地泰造，知識獲得システムにおける信念の修正について(2)，情報処理学会第２９回全国大会， 1984.

〔Kitakami 84-10〕 北上 始，宮地泰造，古川康一，国藤進，知識獲得システムの分散処理にむけて(3) ，情報処理学会第２９回全国大会， 1984.

〔Kowalski 83 〕 Kowalski R.,Logic and Belief,Department of Computing,Imperial College,London (1983).

〔Kowalski 84 〕 Kowalski R.,Logic as a Database Language,Department of Computing,Imperial College,London (1984).

〔Kowalski 84 〕 Kowalski R.,Logic Database,Department of Computing,Imperial College,London (1984).

〔Kunifuji 82〕 国藤 進，加藤昭彦，安達統衛，竹島卓，沢村 一，ロジック・プログラミングとデータベース，情報処理学会記号処理研究会 18-4 1982.

〔Kunifuji 82 〕 Kunifuji,S. and Yokota,H., PROLOG and Relational Data Bases for Fifth Generation Computer Systems,Proc.of CERT Workshop on "Logic Base for Databases" , DEC. 1982(TR-002).

〔Kunifuji 83 -1〕 國藤 進，問題解決と推論，計測と制御，Vol.22,No 1,昭和58年 1月，pp.153-159.

〔Kunifuji 83 -2〕 國藤 進，横田治夫，古川康一，北上 始，宮地泰造，論理型言語と関係データベース管理システムとのインタフェース(1) 基本的考え方，情報処理学会第26回全国大会，東京工業大学，1983年 3月.

〔Kunifuji 83 -5〕 國藤 進，麻生盛敏，竹内彰一，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資料30-1，1983年 6月(TR-009).

〔Kunifuji 83 -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），新世代コンピュータ技術開発機構，1983年 8月.

〔Kunifuji 83 -7〕 國藤 進，宮地泰造，北上 始，古川康一，知識獲得とメタ推論，情報処理学会第27回全国大会，名古屋大学，1983年10月(TM-0016).

〔Kunifuji 84 -1〕 國藤 進，北上 始，宮地泰造，竹内彰一，横田治夫，古川康一，論理型言語によるメタ推論とその応用，京都大学数理解析研究所，「情報の構造化と意味に関する研究」研究集会，1984年 2月.

〔Kunifuji 84 -2〕 國藤 進，竹内彰一，古川康一，上田和紀，メタ推論とその応用－並列メタ推論用メタ述語simulateについて－，情報処理学会第28回全国大会，電気通信大学，1984年 3月(TM-0039).

〔Kunifuji 84 -3〕 國藤 進，北上 始，宮地泰造，古川康一，論理型言語Prologに基づく知識ベースの管理について、第２３回計測自動制御学会学術講演回招待セッション，1984年 7月.

〔Makinouchi 81〕 Makinouchi A.,Tezuka M.,Kitakami H. and Adachi S.,The Optimization Strategy for Query Evaluation RDB/V1,VLDB , France (1981).

〔Minsky 75 〕 Minsky,M., Framework for Representing Knowledge, In Winston(ed.): The Psycholoy of Computer Vision, McGraw-Hill 1975.

〔 Miyachi 83 -1〕 宮地泰造，國藤 進，北上 始，古川康一，竹内彰一，横田治夫，論理データベース向きの知識同化方式の一提案，京都大学数理解析研究所，「モデル表現とその構築に関する理論と実際の研究」研究集会，1983年 2月(TM-0004).

〔 Miyachi 83 -2〕 宮地泰造，北上 始，國藤 進，古川康一，竹内彰一，横田治夫，KAISERにおける知識獲得について，情報処理学会第26回全国大会，東京工業大学，1983年 3月.

〔 Miyachi 83 -3〕 宮地泰造，北上 始，國藤 進，古川康一，KAISERにおける応用分野の検討，同上.

〔 Miyachi 83 -4〕 Miyachi,T., Kunifuji,S., Kitakami,H., Furukawa,K., Takeuchi,A., and Yokota,H., A Knowledge Assimilation Method for Logic Databases, Institute for New Generation Computer Technology, ICOT Tech. Rep. TR-025, Sept.1983.

〔 Miyachi 84 -1〕 Miyachi,T., Kunifuji,S., Kitakami,H., Furukawa,K., Takeuchi,A., and Yokota,H., A Knowledge Assimilation Method for Logic Databases, Proc. of the 1984 International Symposium on Logic Programming,Atlantic City,U.S.A.,Feb. 6-9, 1984.

〔 Miyachi 84 -2〕 宮地泰造，國藤 進，北上 始，古川康一，知識獲得システムにおける一貫性保持について，情報処理学会第28回全国大会，電気通信大学，1984年 3月(TM-0040).

〔 Miyachi 84-3 〕 Miyachi,T., Kunifuji,S., Kitakami,H., and Furukawa,K., A Constrait-based Dynamic Semantic Model for Logic Databases, Institute for New Generation Computer Technology, ICOT TR-0056 , April 1984.

〔Norman 81 〕 ドナルド A. ノーマン ,認知心理学の展望 ,佐伯胖 監訳 ,産業図書 (1984).

〔Ong 84〕 Ong J.,Fogg D. and Stonebraker M.,Implementation of Data Abstraction in the Relational Database System INGRES,ACM SIGMOD RECORD (1984).

〔Popper 68 〕 Popper K.R.,Conjecture and Refutations,Herper TorchBooks,New York (1968).

(Shank 75) Shank,R.C.,Conceptural Information Processing,Amsterdam,North Holland (1975).

(Reiter 78) Reiter, R., On Closed World Databases, in Logic and Data Bases (H.Gallaire and J.Minker,eds.),Plenum Press New York London,pp.55-76,1978.

(Sakai 84) Sakai,K. and Miyachi,T.,Incorporating Naive Negation into Prolog, Proceedings of Logic and Computation Conference, Melbourne, Australia, Feb. 1984.

(Shapiro 81) Shapiro,E.Y., Inductive Inference of Theories From Facts, Yale University Research Report 192, 1981.

(Shapiro 82) Shapiro,E.Y., Algorithmic Program Debugging, An ACM Distinguished Dessertation 1982, The MIT Press 1982.

(Shapiro 84) Shapiro,E.Y., Systems Programming in Concurrent Prolog, 11th Annual ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages, Salt Lake City, Utah, Jan. 1984.

(Shirai 83) 白井英俊, RMS(理由保持機構)を用いた知識表現システム, 情報処理学会第２７回全国大会 1983.

(Shintani 82) 新谷虎松, 知識ベース管理機構について（その２）, 情報処理学会第２７回全国大会 1983.

(Tanaka 83) 田中 譲 他：推論とデータベースシステムとの部分評価機構による結合, 第１回自動制御学会知識工学シンポジウム資料 1983.

(Taguchi 84) Taguchi,A., Miyazaki,N., Yamamoto,A.,Kitakami,H.,Kaneko,K.,and Murakami, K., INI:Internal Network in ICOT and Its future, in Proc. of ICCC '84,also available from ICOT as ICOT Technical Memo.,TM-0044 1984.

(Taguchi 84) 田口留仁, 金子勝蔵, 宮崎収兄, 北上 始,山本 明, 村上国男, 複合ローカル・エリア・ネットワーク INI,情報処理学会 LAN/マルチメディアの応用と分散処理シンポジウム資料 (1984).

(Takeuchi 84) 竹内彰一, 古川康一, 國藤 進, Mandala, Concurrent Prolog 上の知識プログラミング言語／システムーその２ Implementation －, 同上.

(Yokota 83-1) 横田治夫, 國藤 進, 古川康一, 北上 始, 宮地泰造, 論理型言語と関係データベース管理システムとのインタフェース(2) インプリメンテーション, 情報処理学会第26回全国大会, 東京工業大学, 1983年 3月.

(Yokota 83-2) 横田治夫, 國藤 進, 柴山茂樹, 宮崎収兄, 角田健男, 村上国男, Prologによる推論機構と関係データベースの結合, 情報処理学会知識工学と人工知能研究会資料 32-2,1983年11月.

(Yokota 84) Yokota,H., Kunifuji,S., Kakuta,T., Miyazaki,N., Shibayama,S.,and Murakami,K., An Enhanced Inference Mechanism for Generating Relational Algebra Queries, Proc. of Third ACM SIGACT-SIGMOD Symposium on Principles of Database Systems,Waterloo,Canada, April 2-4, 1984.

(Warren 84) Warren D.S., Database Updates in Pure Prolog,Computer Science Department,SUNY at Stony Brook (1984).