知識工学と人工知能　36－7
（１９８４．９．１９）

大規模な知識ベース管理システムをめざして

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。

それは、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である。この問題は、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についての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] 。

以下、本論文では、大規模な知識ベース管理システムをめざして、その大枠と基礎を述べる。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、インプリメンテション用言語は、DEC-10 Prolog を前提としている[Bowen 81]。

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[Kitakami 84-4] の基本構成を示す。

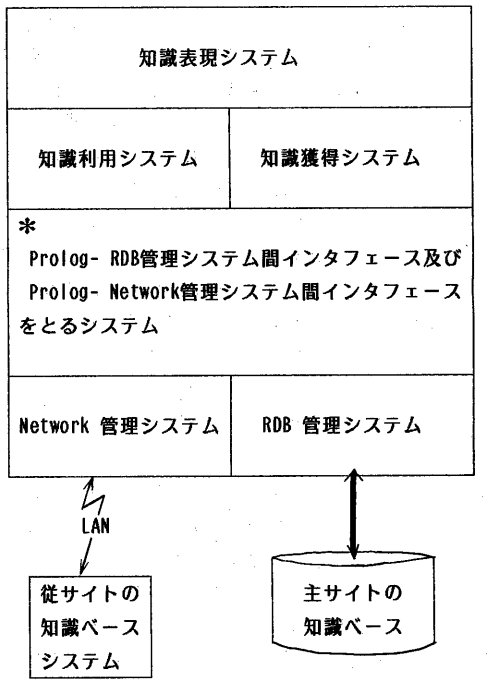

図１．知識ベース管理システム

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 ,Kitakami 84-3] 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 [Makinouchi 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 83-2] 、(2) 論理型言語とRDB との親和性が良い事[Kunifuji82,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] 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。

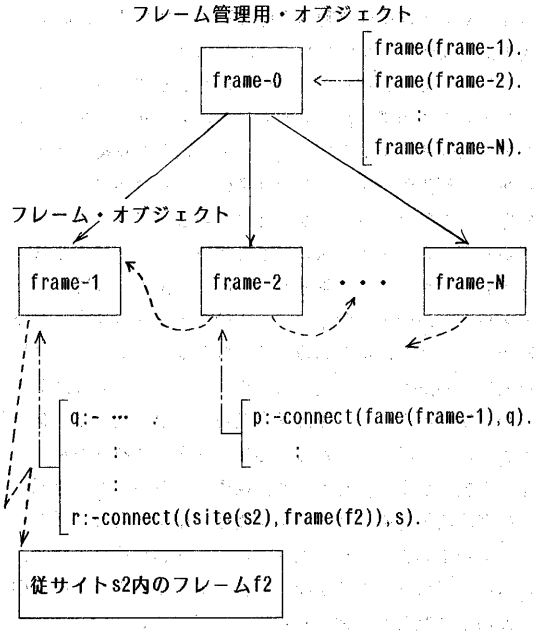

図２．知識ベース内フレームの階層関係

［ 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2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。

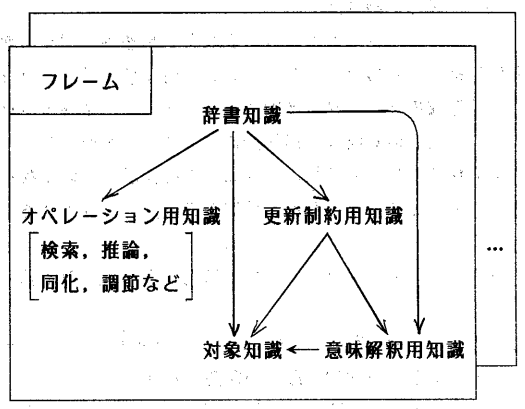

図3．フレーム内の知識群（矢印は、制御関係を表わす）

```
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

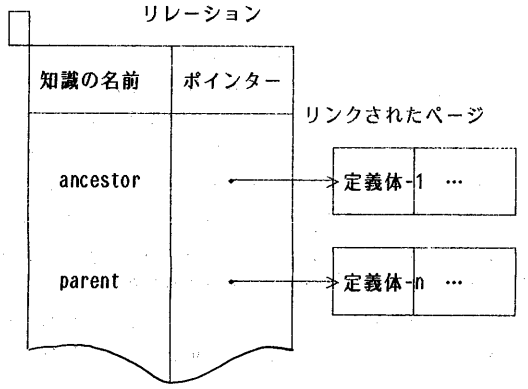

図４．関係データベースによるテキスト・データの管理

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。
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。

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 プログラムとしての知識よりも、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 83-2] 。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。

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 述語の引き数を増し、その引数に exception-false を返すようにすれば良い[Kunifuji 83-2,Kitakami 84-2] 。

最後に、他の一つの更新制約用知識として、動的な制約を与えるメタ知識のシンタックスを以下に示しておこう。

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

これは、トリガーと呼ばれ、実行の方式は、前述の静的な制約を与えるメタ知識と同じである。 Revise-Specification には、既存知識をいかに更新するべきかを、知識ベースに対する更新用述語 (insert,delete など) を利用して記述する。

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 対する特性を捜し、その特性知識名を返す述語である。

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。

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。

8．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。

9．信念の管理

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[Doyle 79,Charniak 80]。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、 9.1 , 9.2 , 9.3 の各節で述べてみたい。

9.1 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。

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。

```
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

9.2 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。

9.3 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。

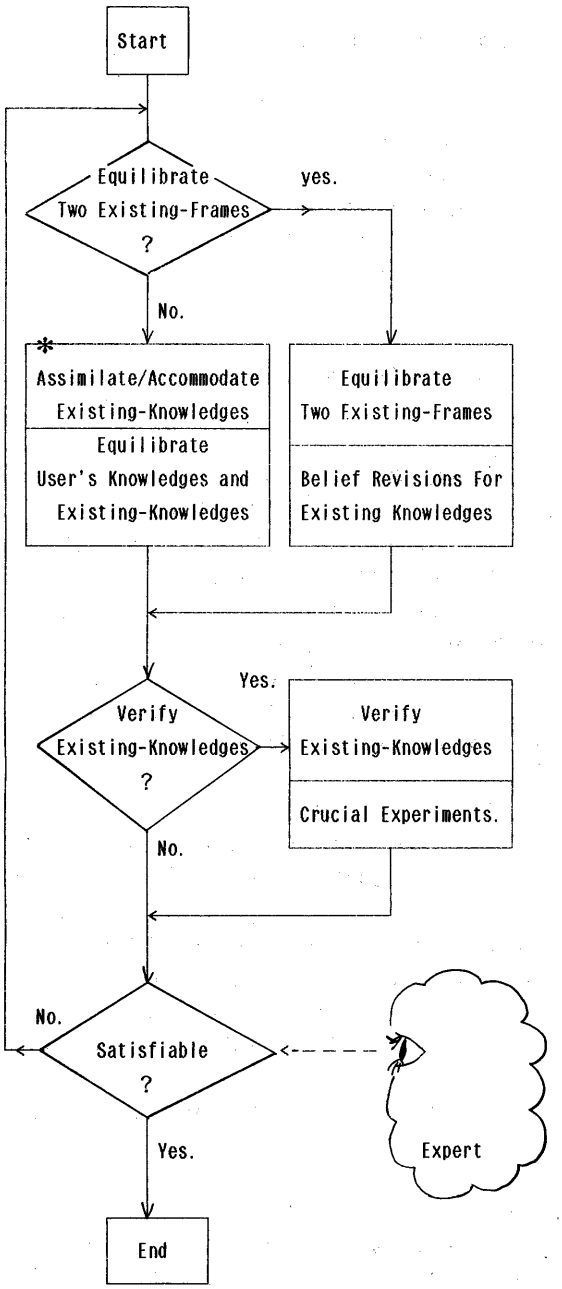

図５．知識ベースの知識獲得プロセス
＊印は、同化（Assimilate）又は
調節（ Accommodate）による知識の
適応プロセスである。

(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) 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。

１０．知識適応プロセス

著者らは、知識適応プロセスを、認知心理学の分野で知られる人間（システム）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。この知識適応プロセスは、知識獲得プロセスの特殊な場合である。ところで、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[Hatano 82] 。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 Genaral 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 83-3,84-1,Miyachi 83,84,Kunifuji 83-1］で報告した。

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 ）。

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。

以上から、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を考えてみよう。このとき、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）。

以上、知識適応および均衡化プロセスの一般的な説明を行なってきたが、具体的な例題については、著者らが既に発表した文献 [Kitakami 84-4] を参照されたい。

１１．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構築をめざして、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。これらの機能は、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。

今後の課題としては、信念を、確信度 (Certainty Factor) という尺度を利用して、系統的に管理する方式の検討がある。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。また、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へと発展させていく上で重要な研究課題である。

［謝辞］

本研究の機会を与えて下さったICOT研究所長・渕一博氏および、有益な討論に参加していただいたICOT第二研究室の各研究員に深謝いたします。

［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 79〕 Doyle,J., Truth Maintenanence System, Artificial Intelligence,Vol. 12, No.3 1979.

〔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 84 〕 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，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 -3〕 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回全国大会，名古屋大学，1983年10月(TM-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 TR-037 1984.

〔Kitakami 84-2 〕 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 84-3 〕 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.

〔Kitakami 84-4 〕 北上 始，国藤進，古川康一，宮地泰造，大規模な知識ベース管理システムのアーキテクチャ，ICOT Techinical Memo. TM-0070 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 83 -1〕 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 -2〕 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).

〔 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 〕 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 84 〕 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.

〔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).

〔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 Dissertation 1982,The MIT Press 1982.

〔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〕 田口昭仁，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).

〔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).