TR-010

知識同化機構の一実現法

北上　始　麻生盛敏　国藤　進

宮地泰造　古川康一

知識同化機構の一実現法

北上始、麻生盛敏、国藤進、宮地泰造、古川康一
（財団法人・新世代コンピュータ技術開発機構）

1．はじめに

人間の知的な精神活動を計算機システムに行わせるためには、種々の計算機構を備えなければならないが、その一つとして、人間がもつ知識を組織的に蓄積管理する機構があげられる。これは、一般に、知識ベースの研究として知られている(1)。

我々は、この研究の一貫として、知識獲得の分野に属する知識同化の実現法について検討している(2)，(3)。

知識同化とは、知識が蓄積されているデータベースに対して、ユーザの意図に合った知識を、入力(3)，(4)，(5)、または、作成する機能を意味する。

以下では、データベースに蓄積されている知識を、(i)ファクト（個々の事実）(ii)ルール（一般的な規則）、(iii)ＩＣ（ファクトとルールについてのIntegrity Constraint）としてとらえ、データベースの形式を、関係データベースへの演繹的質問応答システムとしてとらえる。以後、このシステムを、論理データベースと呼ぶ。

演繹的機能として、SEQUEL又はQUELなどの言語を、サポートしている関係データベースは、知識として、主にファクトを対象にしていた。論理データベースは、本格的に、ルールをも知識の対象にしている。

本稿では、知識ベースシステム向きの知識同化機構を、ルールの場合も含めて検討し、演繹的かつ帰納的推論機構を採用するという立場で、それを体系的に整理したので報告する。

さらに、そのいくつかを、Prologによりインプリントしたので、その結果についても報告する。

2．知識同化機構の構成

知識同化機構に重点をおいた論理データベースのアイデアは、Kowalskiらにより検討されている(6)。

本章では、そのアイデアに対する考察と、それに基づいて再構成された知識同化機構の概念構成について述べる。ただし、以下では、知識の形式としてホーン節を基本にする。

2．1　論理データベースの考察

Kowalskiらのアイデアでは、次のような手続きを必要としている。

Ａ１）論理データベースへの質問応答
Ａ２）データベースから、入力知識の証明による知識同化の拒否
Ａ３）データベースの冗長性除去
Ａ４）無矛盾なデータベースの管理
Ａ５）データベースと独立な知識の同化
Ａ６）入力知識よりも有益な知識の生成と同化

Ａ１）では、データベースに含まれているファクトとルールに対する演繹的質問応答を提供している。一般に、質問の形態は、存在限定(existential _quantifier) 及び全称限定 (universal _quantifier) が混在する命題f($x_1, \cdots, x_n$) である。

$x_1, \cdots, x_n$ は、それぞれ、どちらかの量限定を受ける変数である。本稿では、任意の質問は、どちらか１種類の量限定から成る命題とする。著者らは、既に、存在限定だけから成る命題（以下では、存在命題と呼ぶ）の証明を、demo述語と呼ばれるPrologインタプリタで実現している(3)。

全称限定だけから成る命題（以下では、全称命題と呼ぶ）の証明については、３章で、その実現方法を述べる。

以後、特に、存在命題を証明する述語を、existential _demo述語と呼び、全称命題を証明する述語を、universal-demo述語と呼ぶ。

Ａ２）では、ある入力知識をデータベースに同化しようとするとき、データベースから、その入力知識を証明できるならば、その知識の同化を拒否している。証明過程そのものは、入力知識がファクトであれば、existential _demo述語により実現されるが、その知識がルールであれば、universal _demo述語により実現される。

Ａ３）では、データベースに入力知識を挿入した際に、冗長性が生ずることがあるので、それを除去している。入力知識がデータベースに同化されて

ることを前提として、データベースから任意の知識を取り出し、その知識が残りのデータベースから証明されるとき、取り出された知識は、冗長な知識であるという。このときの証明過程においても、上記、２種のdemo述語が利用される。

ところで、冗長性除去の処理は、ユーザによって、その処理を希望する場合と、そうでない場合がある。例えば、良く利用される知識が、非常に時間のかかる演繹によって得られるのであれば、ユーザは、直接その知識を、データベース内に蓄積するかも知れない。

Ａ４）では、データベースに入力知識を挿入した際に、意味的な矛盾が生ずると、その入力知識の同化を拒否している。データベースの意味的な矛盾性は、ＩＣにより発見される。一般に、ＩＣは、ファクトやルールを制限するメタな知識である。

Ａ５）では、Ａ２）からＡ４）までの判定により、入力知識が同化できるとわかったとき、入力知識は、データベースと独立であるとしている。明らかに、独立な入力知識は、データベースから証明されない。

Ａ６）では、データベースに仮説を付加することによって、独立な入力知識を証明できれば、その仮説を、入力知識の代わりに同化している。

本稿では、この問題を帰納的なモデル推論(7) の問題としてとらえる。

2．2　知識同化機構の概念構成

図１は、知識ベースシステム向きの知識同化機構を組織的に再構成した概念図である。

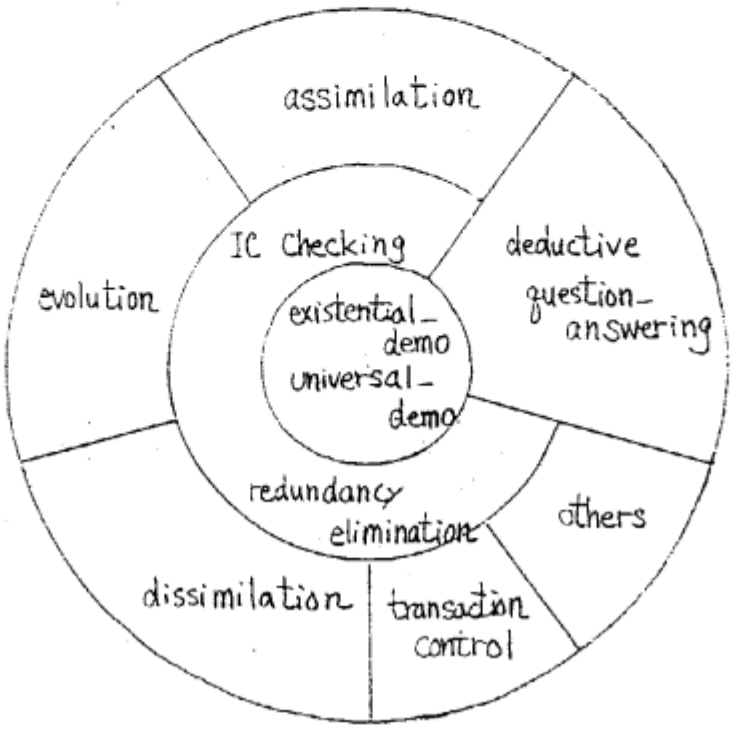

図１．知識同化機構の概念図

前節の考察から、２種のdemo述語が、知識同化の基本機能として、重要であることがわかる。これらのdemo述語は、演繹推論の機構であり、基本的にプログラミング言語Prologをオブジェクトとするメタ言語でもある。オブジェクト言語とメタ言語を融合して、利用する。両言語の融合については、(8) の発表でその詳細が述べられるが、データベースのホスト言語に対応する概念である。

図１のassimilationは、知識の同化に関する各部分の機能であるが、前節のＡ６）の機能を含めていない。ＩＣの判定及び冗長性除去の機能が、含まれている。

evolution は、入力知識をもっと高度な知識としてデータベースに同化したいときに利用される。これを帰納的なモデル推論により実現する。

dissimilation は、同化された知識が、その後の業務で不要になったとき、利用される。詳細は、文献(9) を、参照されたい。

複数のassimilationが、一つの単位として、実行される場合、途中段階で、ＩＣを満たさないことがある。transaction control は、その便宜をはかるために必要である。

deductive question-answeringは、メタ述語によるデータベースへの質問応答機能を提供する。

その他に、データベースのルールに関する包含関係及び同等性を証明する機能があげられるが、これらは、他の分野で作成されたデータベースを有効利用するときに重要な機能である。

３．演繹的推論機構

本章では、知識同化で利用される演繹的推論機構のexistential _demo述語と、universal _demo述語の実現法について述べる。

３．１　存在命題の証明

図２に、この命題を証明するためのexistential _demo述語の実現例を示す。命題は、以下の形を仮定する。

１）Head：-Goals.

２）Head.

ただし、“Head”は、１つの述語であり、“Goals ”は、述語の論理和及び論理積から成る。また、“Goals ”に、カットオペレータ、否定述語“not(・)”システムの組み込み述語“system（・）”を許す。

図２のexistential _demo述語は、２引数をもつ。第１引数は、使用するデータベースの名前ＤＢを与え、第２引数は、証明すべき存在命題を指定する。

図中のdemo述語は、Prologインタプリタとして知られている(3) 。また、clause_ＤＢ述語は、述語Ｐを“Head”とする“Goals ”ＱをＤＢから探索する述語である。

```
existential _demo(Head:- Goals)) :-! ,
    (demo(DB, Head, _) :-not (demo(DB, Goals,_))).
existential _demo(DB,Head):-demo(DB,Head,_) .
demo(DB,true,Cut):-!,true.
demo(DB,!,Cut).
demo(DB,!,Cut).
demo(DB,(P,Q),Cut):-!.
    (demo(DB,P,Cut);demo(DB,Q,Cut )).
demo(DB,(P,Q),Cut):-!,
    demo(DB,P,Value),
    (Value==cut,Cut=cut,! ;demo(DB,Q,Cut)).
demo(DB,not(P),Cut):-!,not(demo(DB,P,Cut)).
demo(DB,P,Cut):-
    system(P) ->P;
    clause_DB (DB,P,Q),demo(DB,Q,Cut ),
    (Cut==cut,!,fail;true).
```

図２．存在命題の証明用述語

３．２　全称命題の証明

ここでは、以下の前提をおく。

１）命題は、３．１節の形式にしたがう。したがって、命題中に現われる変数は、全称限定付きである点だけが違う。

２）命題中の変数は、すべて、有限領域を対象とした変数とする。

すなわち、命題中の変数は、有限のファクトの上に定義されているとする。

以上の前提から、全称命題“Head:-Goals ”を、次のアルゴリズムにより証明できる。全称命題が“Head”のときは、“Head:-true”と考えて、このアルゴリズムを適用すれば良い。

Ｕ１）Prologインタプリタのdemo述語を利用して、“Goals ”を満足するすべてのインスタンスＩを“Goals ”から取り出す。

Ｕ２）Ｉが空のとき、“Head”の述語名がデータベース中に存在すれば、証明結果を真とする。

Ｕ３）Ｉが空でないとき、Ｉのすべてのインスタンスに対し“Head”が真であれば、証明結果を真とする。

Ｕ４）Ｕ２）及びＵ３）のどちらの条件も満さないとき、証明結果を偽とする。

図３に、この全称命題を証明するためのuniversal-demo述語の実現例を示す。この述語も、existential _demo述語と同様に、２引数をもつ。

```
universal _demo(DB,(Head:-Goals)):-
    select_ variabli _list(Goals,X),
    setof(X,demo(DB,Goals,_) ,Set),
    length(Set,N),N=\= 0,!,
    Prove-all _instance(DB,X,Set,Head) .
universal _demo(DB,not(Head)):-
    not(demo(DB,Head ,_)).
universal _demo(DB,Clause):-
    (Clause=(Head:-Goals);Clause=Head),
    \ +( \+(clause_DB(DB,Head, _))).

prove _all _instance(DB,X,[Elem | set],Head):-
    ground(Elem,X,Head,Ground _Head),!,
    demo(DB,Ground_Head ,_) ,!,
    prove _all _instance(DB,X,Set, Head).
prove _all _instance(DB,X,[],Head).
```

図３．全称命題の証明用述語

“select_ variable _list”は、“Goals ”のすべての変数をXに渡す述語である。

“setof ”では、“Goals ”がもつすべてのインスタンスをさがし、変数Set に出力している。“prove _all _instance”は、Ｕ３）に相当する処理を行う述語である。

４．帰納的推論機構

図１のevolution の機構を、帰納的なモデル推論の問題としてとらえる。このモデル推論の考え方は、Shapiro の研究(7),(13)を参考にしている。本章では、このモデル推論の研究を考察し、知識同化への適用方法について述べる。

４．１　モデル推論

図４に、Shapiro が紹介した増殖的なモデル推論アルゴリズムを示す。このアルゴリズムには、ユーザにより、真又は偽の観測文αが与えられる。

観測文αは、ファクトであり、Vは、“true”又は“false ”の真偽値である。ここでは、αとVの組を、観測事実Fnと呼ぶ。推測される仮説の集合は、Prologでプログラム化された手続き（ルールとファクトの集合）である。

L_K から、ファクトαの証明を、３．１節で述べたdemo述語により達成できる。矛盾探索アルゴリズムの実現は、Prologのユーザにとって、非常に簡単である。すなわち、図２に示されている６番目のdemo述語で、ゴール文の最後に。PのインスタンスとS_{false}のファクトαを照合する機構を導入すれば良い。もし、その照合に失敗したら、“P：－Q”は、反ばく仮説になる。

洗練演算子については、帰納関数の理論により構成される。

図４の２つのwhile 文が満足しないとき、次の観測文の読込み待ちになるが、このとき、T_K は、$T_k \nvdash \alpha_{false}$ かつ $T_K \vdash \alpha_{true}$を満足している。

ただし、$\alpha_{false} \in S_{false}$ ・ $\alpha_{true} \in S_{true}$である。

４．２　知識同化への適用方法

入力知識をデータベースに同化しようとしたとき、入力知識よりも有用な仮説を推論するために、モデル推論アルゴリズムが適用される。これは、ユーザの意志にしたがう。

入力知識がファクトのとき、その知識を、本アルゴリズムで作成された仮説におきかえ、同化することができる。

S_{false} = {□}、S_{true}= { }とする。
L_0 = {□}とし、この□に、“false ”をマークする。
repeat.
　次の観測事実F_n = <α, V>を読み込み、αをS_v に付け加える。
repeat.
 while<推測された仮説の集合L_k から、観測文$\alpha \in S_{false}$ を証明できる>do.
　　矛盾探索アルゴリズムを適用し、反ばく仮説に“false ”をマークし、その仮説をL_k から除去する。
while <推測された仮説の集合L_k から、観測文$\alpha_i \in S_{true}$を証明できない>do.
　　K＝K＋１とする。すなわち、α_i を証明できる仮説を、洗練演算子で作成し、それをL_k に付け加えL_{K+1} とする。
until <上記、while ループ条件のどちらも成立しない>
output L_k を出力する。
forever.

図４．増殖的なモデル推論アルゴリズム

もし、この入力知識に関し、さらに多くの情報をユーザが持っている場合は、さらに、正確な仮説を、高速に生成することができる。

入力知識がルールのとき、データベースに、それを同化後、そのルールを拡張する観測事実を与えることによって、さらに有用な仮説を生成することができる。

さて、モデル推論の探索空間を狭めるために、次の情報を与えなければならない。

１）述語の名前と、引数の数
２）引数の構造（例えば、ストリング、リスト）
３）引数の入／出力仕様
４）組込み述語の名前
５）その他

これらは、データベースの辞書情報としてとらえることができる。したがって、これらを管理する辞書機能が必要である。

５．知識同化アルゴリズムの改善

本章では、２章で考察したKowalskiのアイデアに基づき、図１で位置づけたassimilationのアルゴリズムについて述べる。本アルゴリズムでは、独立性の念の定義として、Nicolas が採用した定義を使う(11)。また、本アルゴリズムに関し、次の主な前提をおく。

１）閉世界仮説を仮定する。

２）現在、対象としているデータベースは、無矛盾である（ＩＣも無矛盾であるとする）。

３）データベースのデータ（ルール、ファクト、ＩＣ）は、Prologで記述される。したがって、述語変数は、すべて全称限定付きである。これにより、否定述語not(P)の中に記述される述語Ｐは、存在限定付きの変数となる。

以上からassimilationのアルゴリズムを、次のように整理する。

Ａ１）データベースから入力知識を証明できれば、知識の同化を実施しない。

Ａ２）データベースから入力知識の否定を証明できるならば、知識の同化を実施しない。

Ａ３）データベースに入力知識が同化されたことを前提に、データベースの冗長性を除去する。ただし、冗長性除去の処理は、ユーザの意志にまかせる。

Ａ４）データベースに入力知識が同化されたことを前提に、データベースからＩＣの否定を証明できれば、そのデータベースは、矛盾するので、データベースを最初の状態にもどす（　Ａ３）で冗長性が除去されていることがある　）。

Ａ５）Ａ１）からＡ４）までのいずれの手続きに対しても、入力知識が知識同化の対象になっているならば、入力知識を、データベースに挿入する。

本アルゴリズムでは、Kowalskiが述べている独立性の概念を、Nicolas の概念としてとらえ、その手続きを採用している。それによると、Ａ１）及びＡ２）が共に満足しないとき、入力知識は、データベースに独立であると見做すことができる。

６．知識同化プログラムの実行例

付録Ｉに、実現された知識同化プログラムの中で、assimilateの部分を示す。ただし、５章のＡ４）でのデータベース回復処理は、今後の問題としている。

本章では、よく知られている「積み木」の問題を例に、その実行例を示す。

６．１　問題の設定

図５に「積み木」の配置図を示す。

床ａの上に、ｂ、ｃ、…、ｈ、ｉの積み木を使って、タワーが作られているとする。積み木ｆは長方形の形をしており、それ以外は正方形をしている。その他に、ｊとｋの積み木が手中にあるとする。これは、関係“on”という述語に、ｊとｋに関するデータがないとして特徴づけている。ｊは長方形で、ｋは正方形である。

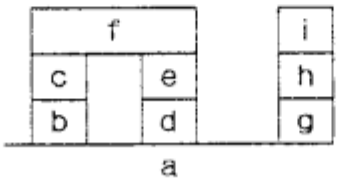

図５．積み木の配置図

図６に、その配置関係をPrologで表現した例を示す。積み木は、“block ”という述語で表現され、積み木の積み重ね関係は、“on”という述語で表現されている。

本データベース中のファクトは、述語“ rectangular_block ”，“square_ block”，“floor ”，“on”であり、ルールとしては、述語“block ”，“tower ”がある。

データベース名は、“build ”と名づけられている。

```
/*  blocks in the rectangular shape   */
floor(a).
rectangular _block(f).
rectangular _block(j).
/*  blocks  in the square shape   */
square_block (b).
square_block (c).
square_block (d).
square_block (e).
square_block (g).
square_block (h).
square_block (i).
/*  block   */
block(x):-
    square_ block(x);
    rectangular _block(x);
    floor(x).
/*  On(X,Y):X is on Y */
on(b,a).
on(d,a).
on(c,b).
on(f,c).
on(e,d).
on(f,e).
on(g,a).
on)h,g).
on(i,h).
/*   tower(X,Y):The  tower consists of block X  */
/*           on top of tower Y.                 */
tower(X,Y):-tower1(X,Y), Y\== [].
tower1(X,[]):-floor(X).
tower1(X,[Y | Z]):-block(X),on(X,Y),tower1(Y,Z).
```

図６．積み木の配置関係のデータベース

ＩＣとして、次の制限をおく。ただし、ＩＣの名前は、“ic”である。

１）“floor(a)”は、決して削除されない。

２）“rectangular _block ”は、２本以上の“tower ”で支えられる。

３）２）の“tower ”には、“square_ block”だけから成る“tower ”が１　本以上存在する。

図７にこの関係を示す。

```
floor(a).
ge(N,2):-
    rectangular _block(X),
    setof(build,(tower(X,Y),ne(Y,[])),[Y]),[Y],S),
    ne(S,[I),length)(S,N).
ge(N2,1):-
    rectangular _block(X),
    setof(build,(tower(X,Y),ne(Y,[]),[Y],S),
    ne(S,[]),length)X,N1),ge(N1,2 ),
    setof(build,'tower(X,V),on)X, W),
          square_ block(W),ne(V,[])),[V],S2),
    length(X2,N2).
le(N,4):-tower(X,Y),length(Y, N).
```

図７．積み木データベースのＩＣ

以上から、次の問題を解いてみる。

Ｐ１）長方形と正方形の積み木から作られるコーナーを認識する述語“corner”の同化

Ｐ２）積み木の組み立てと分解に関するＩＣの問題。

Ｐ３）述語“above ”の同化。

付録Ⅱに、その実行結果を示す。本システムでは、ＩＣのチェックを実施しない知識同化の処理が入っている。すなわち、ＩＣとして［］を与えることにより疑似的なtransaction control を実現している。また、evolutionの処理では、簡単のためＩＣチェックを省略している。

7．おわりに

本稿では、知識データベース向きの知識同化機構について、入力知識がルールである場合を含めて検討し、それを、Prolog (14) でインプリメントした結果を示した。計算機は、DECsystem-2060を利用した。

知識同化の基本機構として、存在命題と全称命題を証明するための演繹的推論機構について述べた。また、知識同化の一環として、さらに有効な知識を同化するために、帰納的推論機構について述べた。

今後の課題として、以下の項目について検討する予定である。

１）本知識同化処理では、１回の処理で入力できる知識を、１つの節に限定したが、これを複数の節としたときの実現方法。

２）冗長性除去の一般化と高速化。

３）universal _demo述語の拡張 *。

これは、全称命題を、skolem化し、“Goals ”の部分を、データベースに付加して“Head”の証明を（demo述語で）実施すれば良い。

４）帰納的なモデル推論の高速化。

５）事実によりＩＣを帰納的に推論する方法。

６）データベースのTrigger 及びActionの検討。

７）データベース間の同等性を証明する問題。

８）並列演算機構の導入による高速化。

［謝辞］

本研究の機会を与えて頂いた当財団法人・新世代コンピュータ技術開発機構の渕一博研究所長と、有益な討論をして頂いた第２研究室の自然言語グループの皆様に深く感謝致します。

---

*これは、グラウンドインスタンスをもたない知識を、データベースで蓄積管理するときに必要である。

［参考文献］

(1) 灘一博：問題解決と推論機構、情報処理学会誌Vol.19 No.10(1978).

(2) 志村正道：知識の獲得と学習、信学技報AL80-46(1980).

(3) 宮地，国藤，北上，古川，竹内，横田：推論データベース向きの知識同化方式の一提案、京大数解研講究録「モデル表現とその構築に関する理論と実際の研究」（1983）.

(4)Bowen,K.A.,Kowalski,R.A.:A malgamating Language and Meta Language in Logic Programming,School of Computer and Information Sciences, Uneversity of Syracuse (1981).

(5)Kowalski,R.A.:Logic as a Database Language,Department of Computing, Imperial College(1981).

(6)Kowalski,R.A.:Logic for Problem Solving,Elsevier North Holland,Inc., PP.239-246(1979).

(7)Shapiro,E.Y.:Inductive Inference of Theories From Facts, Technical Report 192,Department of Computer Sciene,Yale University(1981).

(8) 国藤，麻生，竹内，坂井，宮地，北上，横田，安川，古川：Prologによる対象知識とメタ知識の融合とその応用、情報処理学会研究会資料「知識工学と人工知能」(1983).

(9) 北上，宮地，国藤，古川：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(1983).

(10)Eswaran,K.P.and Chamberlin,D.D.:Functional Specification of a Subsystem for Database Integrity,Proc.1st VLDB Conf.(1975).

(11)Nicolas,J.M.:Logic for Improving,Integrity Checking in Relational Data Bases,Acta Informatica(1982).

(12)Chamberlin,D.D.etal.:A Unified Approach to Data Definition,Manipulation,and Control,IBM J RES.DEVELOP.(1976).

(13)Shapiro,E.Y.: Algorithmic Program Debugging,Research Report 237, Yale University (1982).

(14)D.L.Bowen:DEC-10 PROLOG USER'S MANUAL,University of Edinburgh (1981).

付録Ⅰ. assimilationのプログラム例

```
/* (A1):Is    "Input " derivable from "Current _kb" ? */
assimilate(Current_ kb,IC_ name,Input):-
    demonstrate(Current _kb,Input)  ,
    nl,write)'*Reject.--- >The Input _knowledge " ') ,
    write(Input),write(' "   '),nl,
    write('is derivable from the  Current _Knowledge _Base.'),nl,
    nl,write('*Reject the Input _ Knowledge? (y/n)'),ttyflush,
    ttygeto(X),ttyskip(31),X=:=121.
/*  (A2):Is  "Nega(not(P))" derivable from "Current _kb" ? */
assimilate(Current_ kb,IC_ name,not(P)):-
    demonstrate(Current _kb,P),
    nl,write('*  Reject the Input _Knowledge ? (y/n)'),ttyflush,
    ttygeto(X),ttyskip(31),X=:=121.
/* (A3):Does  "Input " imply information already implicitly
                                  comtained in   "Current _kb" ? */
assimilate(Current_ kb,IC_ name,Input):-
    remove_ redundancy,
    (open _generator(Cnrrent _kb):close _generator,fail),
    generate_ temp _kb(Current_ kb,Input ,[P1,P2,P3,Information]),
    demonstrate(temp_ kb,Information),
    close _generator,
    nl,write('*The Knowledge" '),write(Information),write('  "'),
    nl,write('was removed from Current_ Knowledge_ Base.'),nl,
    delete_ Knowledge(Current_ Kb,[ P1,P2,P3]).
/*  (A4):Is[  "Current-kb"  +"Input]" inconsistent with  "IC" ? */
assimilate(Current_ Kb,IC_ name,Input):-
    IC_name\==[],
    dict(IC _name,[Q1,Q2,Maxptr]),
    get _integrity _constraint(IC _name,IC,Q1),
    negate(IC,not _ic),
    insert_ Knowledge(Current_ kb,  Input,[P1,P2,P3]),
    (set_ backtrace(IC _name):free_ backtrace,fail),
    (demonstrate(Current_ kb,Not _ic):
```

```
    delete_ Knowledge(Current_ kb,[ P1,P2,P3]),fail),
    free_backtrace ,
    delete_Knowledge (Current_ kb,[P1,P2,P3]),
    nl,write('*The Input-knowledge" '),write(Input),write(' " '),
    nl,write('*is inconsist with the Integrity-Constraint.'),nl,
    nl,write('*Reject the Input _knowledge ?(y/n)'),ttyflush,
    ttygeto(X),ttyskip(31),(X=:=110,
    nl,write('*Refine the Current _knowledge _Base?(y/n)'),
    ttyflush,ttygeto(Y),ttyskip(31),(Y=:=121,
    refinement(Current_ kb,IC-name,Input);true);true).
/*   (A5):I   "Input " is logically independent from  " Current_kb"
                                          ->Insert "Input " .  */
assimilate(Current_kb, IC-name,Input):-
    insert_Knowledge (Current_kb, Input ,ptr),
    nl,write('*The Input_ Knowledge" '),write(Input),write(' " '),
    write('was assimiated.'),nl.
```

付録Ⅱ．積み木データベースの実行例

p1) の問題

```
% ?-assim(build,ic,corner(f,[c,b,a])).
*Eliminate Redundancy ?(y/n)n
*The Input-knowledge "corner(f,[c,b,a]" was assimilated.
 4638 msec.
yes
% ?-assim(build,ic,(Corner(X,Y):-tower(X,Y))).
*Eliminate Redundancy ?(y/n)y.
*The knowledge "corner(f,[c,b,a]) "
  was eliminated from Current-knowledge-Base.
*The Input-knowledge "(corner(_0,_1 ):-tower( _0,_ 1))" was assimila
ted.
 13836 msec.
yes
% ?-existential_ demo(build,corner(X,Y)).
corner(b,[a]);
corner(c,[b,a]);
corner(d,[a]);
corner(e,[d,a]);
corner(g,[a]);
corner(h,[g,a]);
corner(i,[h,a]);
corner(f,[c,b,a]);
corner(f,[e,d,a]);
no
% ?-evolution(build,corner(X,Y)) .
*Initialize parameter(y/n)?n.
Next fact(sentence,true/false )or end ?corner(b,[a]),false.
Checking fact(s)...
Error:wrong solution corner(b,[a]).diagnosing...
Query:block(b)?y.
Query:tower1(a,[])?y.
Query:tower1(b,[a]) ?y.
Query:tower(b,[a])?y.
```

```
Error diagnosed:(corner(b,[a] ):-tower(b,[a]))is false.
Listing of corner(X,Y):
Checking fact(s)...no error found.
Next fact(sentence,true/false )or end ?corner(f,[e,d,a])true.
Checking fact(s)...
Error:missing solution corner (f,[e,d,a]).diagnosing...
Error diagnosed:corner(f,[e,d,a])is uncovered.
Searching for a cover to corner(f,[e,d,a])...
Refining:(corner(X,Y):-true)
Refining:(corner(X,[ X | Z]):-true)
Refining:(corner(X,Y):-tower( X,U))
Checking:(corner(X,Y):-tower( X,Y))
Refuted:(corner(b,[a]):-tower (b,[a]))
Refining:(corner(X,[Y,Z | U]):-true
Refining:(corner(X,[ Y | Z]):-tower(X,V))
Refining:(corner(X,Y):-tower(X,Y))
Checking:(corner(X,Y):-tower(X,Y),rectangular-block(X))
Found clause:(corner(X,Y):-tower(X,Y),rectangular _block(X))
    after searching 9 clauses.
Listing of corner(X,Y):
    (corner(X,Y):-tower(X,Y),rectangular_ block(X)).
Checking fact(s)...no error found.
Next fact(sentence,true/false )or end ?corner(i,[h,g,a]),false.
Checking fact(s)...no error found.
Next fact(sentence,true/false )or end ?end.
*purge facts(y/n) ?y.
*Eliminate Redundancy ?(y/n)n.
*Eliminate rejected theory?(y/n)y.
*Construction of the knowledge "corner"
 has been finished.6232 msec.
yes
% ?-existential_ demo(build,corner(X,Y)).
corner(f,[c,b,a]);
corner(f,[e,d,a]);
no
% ?-assim(build,[],on(j,f)).
```

```
*Eliminate Redundacy ?(y/n)n.
*The Input-knowledge "on(j,f) " was assimilated.
 96 msec.
yes
% ?-assim(build,ic,on(j,i)).
*Eliminate Redundancy ?(y/n)n.
*The Input-knowledge "on(j,i) " was assimilated.
 9196 msec.
yes
% ?-existential_demo(build,corner(X,Y)).
corner(f,[c,b,a]);
corner(f,[e,d,a]);
corner(j,[f,c,b,a]);
corner(j,[f,e,d,a]);
corner(j,[i,h,g,a]);
no
```

p2) の問題

```
% ?-assim(build,ic,on(k,j)).
*Eliminate Redundancy ?(y/n)n.
*By thi contradiction backtracing,detected the following knowledge.
---> "(le(-0,4):-tower( _1,_ 2)length(_ 2, _0)) ".
*The Input-knowledge "on(k,j) "
 is inconsist with the Integrity-constraint.
 10486 msec.
yes
% ?-dissim(build,[],on(j,i)).
*The knowledge "on(j,i) "
  was deleted from Current-knowledge-Base.
yes
% ?-dissim(build,ic,on(j,f)).
*The knowledge "on(j,f) "
%  was deleted from Current-knowledge-Base.
yes
% ?-dissim(build,[],on(f,c)).
*The knowledge "on(f,c) "
 was deleted from Current-knowledge-Base.
yes
% ?-dissim(build,ic,on(f,e)).
Searching for a cover to above(i,a)...
Refining:(above(X,Y):-true)
Refining:(above(X,Y):-on(X,U)
Refining:(above(X,Y):-on(X,U) ,on(U,W))
Refining:(above(X,Y):-on(X,U) ,above(U,W))
Checking:(above(X,Y):-on(X,U) ,above(U,Y))
Found clause:(above(X,Y):-on(X,U),above(U,Y))
    after searching 11 clauses.
Listing of above(X,Y):
    (above(X,Y):-on(X,U),on(U,Y)) .
    (above(X,Y):-on(X,Y)).
    (above(X,Y):-on(X,U),above(U, Y)).
Checking fact(s)...no error found.
Next fact(sentence,true/false )or end ?end.
```

```
*Purge Facts(y/n) ? y.
*The knowledge-base "solutions " was droped on Core-space.
*Eliminate Redundancy ? (y/n)y.
*The knowledge "(above( _0,_ 1)-on( _0,_ 2),on( _2,_ 1))  "
 was eliminated from knowledge-Base "build " .
*Eliminate rejected theory? (y/n)y.
*Construction of the knowledge "above "
 has been finished. 12162 msec.
*The knowledge "on(f,e) "
 was deleted from Current-knowledge-Base.
yes
% ? -existential_ demo(build,corner(X,Y)).
no
```

p3) の問題

```
% ?-assimilate(build,ic,(above(X ,y):-on(X,z),on(Z,Y))).
*The Input-knowledge "(above ( _0,_1 ):-on(_0,_2),on(_2,_1)) "
                                                    was assimilated.

yes
% ?-existential_ demo(build,above (X,Y)).
above(c,a);
above(f,b);
above(e,a);
above(f,d);
above(h,a);
above(i,g);
no
% ?-evolution(build,above(X,Y)).
*Initialize parameter(y/n)?n.
*The knowledge-base "solutions " was created on Core-Space>
Next fact(sentence,true/false )or end ?above(b,a),true.
Checking fact(s)...
Error:missing solution above( b,a).diagnosing...
Query:on(a,a) ?n.
Error diagnosed:above(b,a)is uncovered.
Searching for a cover to above(b,a)...
Refining:(above(X,Y):-true)
Refining:(above(X,Y):-on(X,U))
Checking:(above(X,Y):-on(X,Y))
Found clause:(above(X,Y):-on(X,Y))
    after searching 3 clauses.
Listing of above(X,Y):
    (above(X,Y):-on(X,U),On(U,Y)) .
    (above(X,Y):-on(X,Y)).
Checking fact(s)...on error found.
Next fact(sentence,true/false )or end ?above(i,a),true.
Cheking fact(s)...
Error:missing/solution above( i,a).diagnosing...
Query:on(h,a) ?n.
Query:on(i,a) ?n.
```

```
Error diagnosed:above(i,a)is uncovered.
yes
% ?-existential-demo(build,above (X,Y)).
above(c,a);
above(f,a);
above(f,b);
above(e,a);
above(f,a);
above(f,d);
above(h,a);
above(i,a);
above(i,g);
above(b,a);
above(d,a);
above(o,b);
above(f,c);
above(e,d);
above(f,e);
above(g,a);
above(h,g);
above(i,h);
no
```