TM-0004 APR. 1983

# 論理データベース向きの co-op 知識同化方式の一提案

宮地泰造，国藤　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

財団法人
新世代コンピュータ技術開発機構

宮地　泰造
國藤　進
北上　始
古川　康一
竹内　彰一
横田　治夫

概　要　：　本論文は、論理データベース・システムとして、関係データベースへの演繹的質問応答システムを考え、そのようなシステム向きの知識の獲得方式の提案を行うものである。知識獲得方式としては演繹論理志向の知識の同化という概念を、オブジェクト言語とメタ言語の融合という考え方に基いて実現可能な形で定式化した。知識同化の概念は、証明可能性，矛盾性，冗長性，および独立性というサブ概念のチェックと対応する内部データベース更新とからなる。また、論理データベース向きの知識同化のインプリメントを、論理型プログラミング言語:PROLOG により行った。その結果、PROLOGが知識の同化の実現などに非常に適していることがわかった。

Keywords：　Logic Database, Deductive Logic, Knowledge Acquisition, Knowledge Assimilation, PROLOG, Internal Database, Knowledge Base, Integrity Constraint, Cut Operator, Provability, Contradiction, Redundancy, Independency, Object Language, Meta Language, Amalgamation, Database Management.

## １．はじめに

人間は、実世界に存在する様々の知識に対して、その存在価値を認めた場合に、知識として獲得していく。データベースなどの知識を蓄積・管理するシステムは、この人間の知識の獲得や表現・利用の高度化・容易化を実現する役割をになっている。とは言っても、データベースなどに知識を格納するか否かの判断は、最終的には人間が決定する。しかしながら、人間は、知識不足であったり、不注意を生じたりするので、人間の意図に対して明らかに正しくない知識や既に保有している知識の獲得は拒否すべきである。

このように、知識の獲得時に、獲得者の意図に合致する、必要な知識だけをデータベースなどに取込むことを『知識の同化（Knowledge Assimilation）』と呼ぶ。知識の同化は、データベースの知識の妥当性を保証するために、知識の獲得時に不可欠である。

知識の同化の基本的概念はBowen らにより示されている［B&K 81］。しかしながら、彼らはその具体的インプリメンテーション例を示さなかった。

一方、データベースのIntegrity Constraint（以下、IC　と略記する）の研究が、Cadiou[Cadiou 76],Nicolas[Nicolas 78],Beeri ら[B&B&G 78]などにより行われている。これらの研究は、Bowen らが知識の同化の中で取上げている矛盾性の問題の研究に対応すると考えられる。

本論文では，第２章で、知識の同化の概念を実現可能な形で定義することのより、その概念の理論的明確化を図っている。。また第３章では、PROLOGによるそのインプリメンテーションを行ったので、PROLOGによる知識の同化の実現方法について報告する。

なお、本論文で取上げる研究分野は、従来関係データベースへの演繹的質問応答システムの基礎・応用・インプリメンテーション研究と呼ばれている分野であるが、ここでは、簡単のためCERT-Workshop[G&M 78],[G&M&N 81],[Nicolas 82] にならって“論理データベース”(Logic Database)と呼ぶことにする。

## ２．知識の同化

知識の同化の基本的な概念は、Bowen & Kowalskiにより示された［B&K 81］。知識の同化の概念を説明するためには、オブジェクト言語の証明可能性を操作するメタ言語の導入が必要である。そこで、まずメタ言語研究の説明を２．１節で行い、２．２節で知識の同化の概念を説明する。

### ２．１　メタ言語の研究

オブジェクト言語の証明可能性の操作はオブジェクト・レベルの言語の枠の中では出来ない。そこで、オブジェクト・レベルの言語の証明可能性の処理をメタ言語として構築し，オブジェクト言語の中に組込むことにより機能の拡張を実現できることが、［B&K 81］に示されている。すなわち、オブジェクト・レベルとメタ・レベルを１つの言語で取扱うことが可能であることが示されたのであり、これをオブジェクト・レベルの言語とメタ・レ

レベルの言語との融合（Amalgamation）と呼ぶ。

次に示すＤ１，Ｄ２が、上述の証明可能性を処理するメタ述語“demo”のトップ・レベルのプログラムである。“demo”は、Goals がProgから証明可能であるか否かを判定する。すなわち、“demo(Prog,Goals)”は論理学の´Prog ├─ Goals ´（├─ ： 証明可能）に対応する（注１）。

```
Ｄ１）  demo(Prog,Goals) <- empty(Goals)  (注2)
Ｄ２）  demo(Prog,Goals) <- select(Goals,Goal,Rest),
                            member(Proc,Prog),
                            rename(Proc,Goals,Variantproc),
                            parts(Variantproc,Concl,Conds),
                            match(Concl,Goal,Sub),
                            apply(Conds+Rest,Sub,Newgoals)  (注3)
                            demo(Prog,Newgoals).
```

“demo”プログラムを以下に簡単に説明する（詳細は［B&K 81］参照）。

Ｄ１はGoals が空になったかを判定している。つまり、証明が完了するまでＤ２を続行するための制御を行う。

Ｄ２では導出原理による定理証明を次の各ステップで行っている。

（１）解くべきGoals から部分問題Goalを選択し、
（２）このGoalを解く手続きProcをプログラムProgから選びだす。
（３）次にProcに現れる変数の名前換えを行い、
（４）その結果得られたVariantproc を、このVariantproc を成立させるための条件部Conds と結論部concl に分解する。
（５）このConcl とGoalから変数の置換Sub を求め、
（６）Goals の残りRestと(4) の条件部Conds にこの置換Sub を適用し、新しいゴールNewgoalsを得る。
（７）以下、このNewgoalsに同様の手続きを行い、ゴールが空になってＤ１が成立するまで繰り返す。

---

(注1)demoは証明可能なゴールに対しては常に正しい解を見付けて停止するが、そうでない場合は必ずしも停止しないことは注意すべきである。

(注2)ここでPROLOGの':-'ではなく'<-'を使用している場合、PROLOGの手続き的解釈は保存するが、節の順序については特に規定しないという解釈を採用している。

(注3)ここに '+'は和集合演算子(set-union operator)である。

ところで、PROLOGの世界においては、既に1978年に Pereira & Warren によりPROLOGでPROLOG Interpreterを作成すると極めて簡単に実現できることが“PROLOG Interpreter in PROLOG”として実証されている［P&W 1978］。

ここで、我々は次のような事実を見い出した。メタ述語“demo”とPROLOG Interpreter in PROLOGとはメタ言語を実現するという点においてほぼ同一の機能を有すこと、また“demo”をPROLOG Interpreter in PROLOGにより非常に容易に実現できることがわかったのである。すなわち、次に示す６つの節により“demo”の基本機能を実現できる。なお、ここでは、２引数述語: demo(Currentdb, Goals)に対して、Currentdb を固定して１引数述語: demo(Goals) で実現している。

```
demo(true):-!,true.
demo(not(P)):-!,not(demo(P)).
demo((P;Q)):-!,(demo(P);demo(Q)).
demo((P,Q)):-!,demo(P),demo(Q).
demo(P):-  currentdb ((P:-Q)),demo(Q).
demo(P):-  currentdb (P).
```

demoプログラムを簡単に説明すると以下の様になる。第１番目の節はGoals が'true'のときは真とする。第２，３，４番目の節は、ゴールがそれぞれ、否定，選言，連言の形式の場合の証明手順を与えている。第５，６番目の節は実際に証明を実行する。第５番目の節は内包（ルール）によるゴールの証明を行う。第６番目の節では外延（ファクト）によるゴールの証明を行う。

また、上述の“demo”を改版することにより、Cut Operator（！）を取扱えるdemoも容易に実現可能である。

これにより、PROLOGにおいては、オブジェクト・レベルの言語とメタ・レベルの言語とを、同一言語上で容易に実現可能となる。すなわち、知識の同化の処理を比較的容易に実現可能にすることになる。また、別の観点からみて、取扱う知識そのものについてみると、それはオブジェクト知識とメタ知識とのPROLOGによる操作を可能に出来ることを意味している。これらについての詳細は後述する。

## ２．２　知識の同化の概念

論理データベースにおいては知識は一般にデータとも呼ばれるので、本稿でもデータを知識の意味で用いることにする。すると、知識の同化の概念は、つぎのＡ１）～Ａ４）で示される(注) ［B&K 81］。

---

(注) ここで、'+' は和集合演算子(set-union operator),'-'は差集合演算子(set-difference operator) である。

A１) assimilate(Currdb, Input, Currdb) <- demo(Currdb, Input).

A２) assimilate(Currdb, Input, Newdb) <- Info ∈ Currdb,
Interdb = (Currdb - Info),
demo(Interdb + Input, Info),
assimilate(Interdb, Input, Newdb).

A３) assimilate(Currdb, Input, Currdb) <-
demo(Currdb + Input, false).

A４) assimilate(Currdb, Input, Currdb + Input) <-
independent(Currdb, Input).

ここで、assimilateの第１引数(Currdb)が現在のデータベースを示し、第２引数(Input)が入力データを示す。そして、結果として得られる新しいデータベースが第３引数である。A１)～A４)はそれぞれ次のことを表わしている。

（A１）現在のデータベース(Currdb)から入力データ(Input)が証明可能ならば新しいデータベースは現在のデータベースのままである。

（A２）現在のデータベース中のあるデータ(Info)が、データベース中のその他のデータ(Interdb)にInputを加えたものから証明できる場合は、Interdbにを同化して得られるデータベースが新しいデータベース(Newdb)である。

（A３）CurrdbにInputを追加すると(Currdb + Input)に矛盾が発生する場合は、現在のデータベース(Currdb)に入力データ(Input)を追加してはならない。新しいデータベースは、現在のデータベースのままである。

（A４）InputがCurrdbと独立なデータである場合は、CurrdbにInputを追加したデータベースが新しいデータベースである。

以上のA1～A4より、知識を同化するためには、

（A１） 証明可能性の検査

（A２） 冗長性の除去可能性の検査

（A３） 矛盾性の検査

（A４） 独立性の検査

を行い、それぞれに対応してCurrdbの適切な更新を行えばよいことが分る。しかしながら、この考えは、論理としては正しいが、論理プログラム言語でインプリメントする場合A1～A4の処理順序は明確でなく、それらの処理手順は再考すべきだと考える。また、矛盾，独立の定義も具体的には行われていない。

そこで、本稿では、知識の同化の処理順序をも考慮し、矛盾，独立とはいかなるものかを定義し、更に冗長性除去検査の役割をより深く考察することにより、知識の同化のより

明確な概念を定義する。

### ２．２．１　知識の同化の処理順序

知識の同化の処理順序は、次の順に行うのが適切と考えられる。

１）　証明可能性の検査
２）　矛盾性の検査
３）　冗長性の除去可能性の検査
４）　独立性の検査

その理由は、以下のａ）～ｃ）である。

ａ）　証明可能性の検査と矛盾性の検査とは、ともに、入力データがデータベースに追加されるべきである（これを〝同化可能(assimilatable)〟とよぶ）か否かの判定を行う。これに対して、冗長性の除去可能性の検査は、入力データが同化可能である場合に必要となる。よって、冗長性の除去可能性の検査は、証明可能性および矛盾性の検査の後で処理すべきである。

ｂ）　入力データがデータベースから証明可能である状態と、入力データがデータベースに矛盾する状態は背反する。基本的には概念的に単純かつ先行する証明可能性の検査を矛盾性の検査より先に行うべきである。

ｃ）　独立性の検査は、同化可能性の検査をその一部に含むので、同化可能性の検査より後で行われるべきである。また、入力データがデータベースに独立である場合には、冗長性の除去可能性の検査は既になされているので、独立性の検査は冗長性の除去可能性の検査の後に処理すべきである。

### ２．２．２　データベースの矛盾性

個々のデータベースは、それぞれ目的に応じた意味を有する。よって、その意味に反するデータの蓄積は拒否されるべきである。また、個々のデータベースの意味論的制約は個別Integrity Constraint(ICi) によって規定されるべきであり、ICi によって規定された意味に従ってデータベースへの知識の獲得は行われるべきである。

そこで、データベースの矛盾性を次のように定義する。

現在のデータベース(Currdb)に入力データ（新知識,Input）を追加したデータベースが矛盾するとは、適当な　Integrity Constraint (ICi)に対して、ICi を満足しないデータが存在することを言う。

### ２．２．３　入力データの知識の同化可能性

入力データ（新知識）が現在のデータベース(Currdb)に知識の同化可能であるとは、次の条件　ａ），ｂ）を満足する場合である。

ａ）　入力データがCurrdbから証明可能でない。
ｂ）　入力データがCurrdbに 2.2.2項の意味で矛盾しない。

入力データがCurrdbに知識の同化可能の場合、更に次の２つの場合がある。

１）　冗長な場合
２）　独立な場合

これらの場合について、それぞれ 2.2.3.1項，2.2.3.2項で説明する。

### ２．２．３．１　データの冗長性

入力データ(Input) が知識の同化可能性を有す場合、現在のデータベースをCurrdbと表わすと、(Currdb + Input) に冗長なデータが存在する可能性がある。そこで、（Ａ）式で表現される自明でないＫｉが存在する場合、(Currdb + Input)は冗長であると定義する。

$$Ki \sqsubseteq Currdb, \quad (Currdb + Input - Ki) \vdash Ki \quad \cdots \quad (A)$$

ここで、注意すべきこととして、一般に冗長である可能性は、知識として意味のある最小単位に対して派生するということである。すなわち、知識の最小意味単位が存在することまで意識すると、Currdbの単なる各要素以外に、Currdbの部分擬順序集合（擬順序'$\sqsubseteq$'で定義）に対して、冗長である可能性が存在することである。この場合、冗長性の除去の検査は、一般に、Currdbの部分擬順序集合に対して行わなければならない。

また、ここで、Input が(Currdb + Input - Input)に対して冗長な知識でないことは、知識の同化可能性の検査のうち証明可能性の検査ですでに完了しているので省略できる。

### ２．２．３．２　入力データの独立性

入力データ(Input) が現在のデータベース(Currdb)に対して独立であるとは、 2.2.3項のa,b が成立し、かつ、Ａを成立させるデータがCurrdb中に存在しない場合と考える。

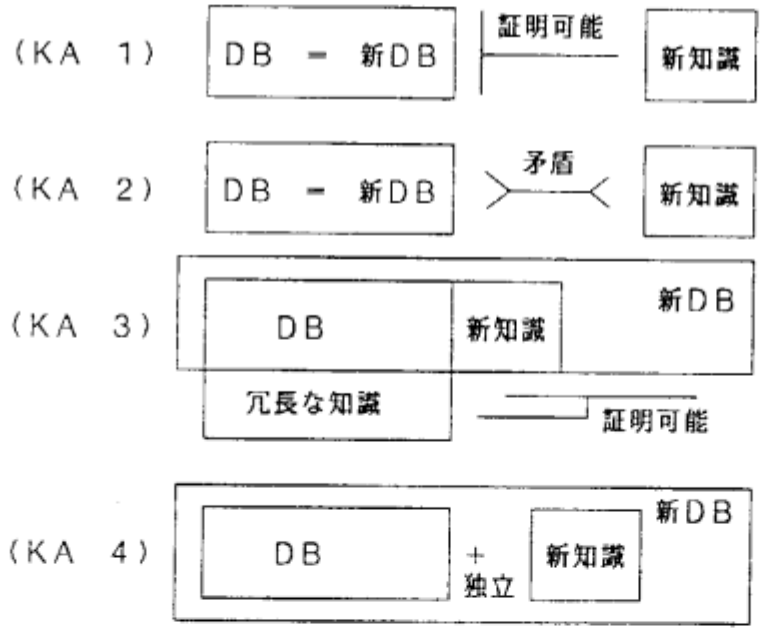
すなわち、Input がCurrdbに同化可能で、かつ、(Currdb + Input)が冗長でない場合、Input はCurrdbに対して独立であると言うと定義する。

### ２．２．４　知識同化概念の明確化

以上のように明確に規定した、知識の同化における矛盾性，同化可能性，冗長性，独立性の概念と知識の同化の処理順序により、知識の同化の概念は、順序をも含めて、次のように表せる（図２．１参照）。

```
（KA　1）　assimilate (Currdb, Input, Currdb)<-
                demo (Currdb, Input).
（KA　2）　assimilate (Currdb, Input, Currdb)<-
                demo (Currdb + Input + ICs, false)  (注)
（KA　3）　assimilate (Currdb, Input,Newdb)<-
                Info ⊑ Currdb,
                Interdb=(Currdb-Info),
                demo(Interdb +Input, Info ),
                assimilate(Interdb, Input, Newdb).
（KA　4）　assimilate (Currdb, Input, Currdb + Input)<-
                independent (Currdb, Input).
```

ここで、false の証明可能性すなわち矛盾性(KA2),独立性: independent(KA4)は、それぞれ 2.2.2項, 2.2.3.2 項の意味である。 また、冗長性の除去 (KA3)は必要に応じて行えばよいと考えられる。(KA3) 手続きを省略した場合、〝independent ＝ 同化可能〟となることは注意すべきことである。




（図２．１　知識の同化の処理）

---

（注）　ＩCs はオブジェクト(Currdb やInput)に対するメタ知識であるが、メタ述語 “demo” を用いることによりオブジェクト言語上に表現できる。

## 3. PROLOGによる知識の同化の実現

第2章において、知識の同化の概念を定義した。また、Prolog Interpreter in Prologの手法によりdemoを実現することにより、知識の同化の処理が容易に実現可能であることも示した。

本章では、PROLOGによる知識の同化の実現方法について報告する。

### 3.1 知識の種類と表現

まず、知識同化の対象となる知識を定義する。

知識は、外延（ファクト）,内包（ルール）, Integrity Constraint(ICs)の3種類に区別して、表現できる（具体例は 3.7節参照）。

ここで、ICs をとくに区別しているのは、ICs はその対応するデータベースのメタ知識だからである。すなわち、ICs はデータベースの保全性制約の条件であり、データベースに蓄積される知識に関するメタ知識であるからである。また、外延・内包が追加されるとデータベース内の知識は増大するが、ICs が追加されて保全性の条件が強くなると、各データベースから証明できる知識は減少する。この意味において、ICs は単なる外延や内包とは区別すべきである。

よって、ICs を外延・内包と異なるシンタックスで表現することにする。 ICs は " check_kb "のワクにおいて、対象リレーション，矛盾検出時のメッセージ，対象データベースとともに表記することにより定義する。

check_db（対象リレーション，ICs,' 矛盾検出時のメッセージ',［対象データベース］），

check dbにおいて、ICs は次のシンタックスで記述する。

```
<ICs>::=<IC>, <ICs>|
        <IC>;<ICs>|
        <IC>
<IC>  ::=<Ls>-><L>
<Ls>  ::=<L>, <Ls>|
        <L>;<Ls>|
        not(<Ls>)|
        (<Ls>)|<L>
<L>   ::=not(<L>)|<I>
<I>   ::=<ゴール>
```

（注） ゴールはPROLOGのゴールである。

このシンタックスにより、ＩＣｓはホーン論理と同等の表現能力を有することができる（例は３．５節参照）。

## ３．２　知識の同化と融合

知識を 3.1節のように規定した場合、知識の同化の処理の実現における知識と言語の融合(Amalgamation)の形態に対しては、以下に示す２通りの捉え方がある（図 3.1, 参照）：

（ａ）　データベース管理（ICs を除く）がメタの世界に存在するとみる場合，

（ｂ）　ＩＣs をメタの世界に存在するメタ知識とみる場合。

(a), (b) について、それぞれ、 3.2.1項， 3.2.2項に説明する。

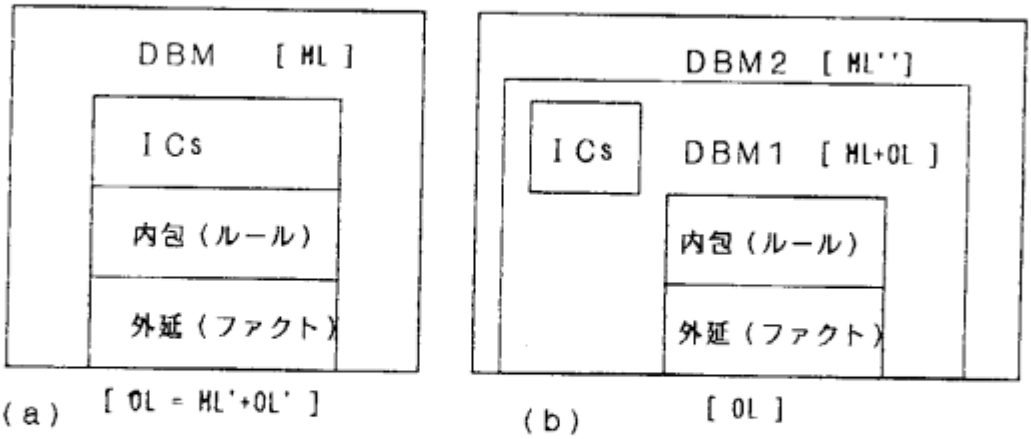
### ３．２．１　データベース管理（ICs を除く）がメタの世界に存在するとみる場合

データベース管理（ICs を除く）がメタの世界に存在するとみる場合、つぎの３つのことが言える（図 3.1 (a)　参照）。

（１）　demoというメタ述語を用いて、データベース管理(DBM) というメタの世界の処理系を作成した（メタの世界をメタ言語(ML)により構築した）。すなわち、固定されているDBM だけがメタの世界に存在し、DBM はICs などを用いてデータベース管理の処理を実行すると考える。ここで〝固定されているDBM〟とは、DBM がオブジェクトの世界からの影響によって変化しないことを意味している。すなわち、ここではメタの世界とオブジェクトの世界とが分離して存在すればよい。

（２）　DBM が対象とするオブジェクトレベルの言語(OL)が存在する。このOLは別のオブジェクト言語(OL') とメタ言語(ML') とを融合した言語である。すなわち、メタの世界とオブジェクトの世界とを融合した世界はオブジェクトの世界に存在している。

（３）　PROLOGの能力では、ML, OL, ML', OL'を実現できる。すなわち、PROLOGの能力によると、“PROLOG = ML = OL = ML' = OL'” となる。




（図３．１　知識と言語の融合（Amalgamation））

### ３．２．２　ＩＣｓ をメタの世界に存在するメタ知識とみる場合

ＩＣｓ がデータベースの知識のメタ知識である点に注目すると、ＩＣｓ はメタの世界に存在するメタ知識である（図 3.1 (b)　参照）。この場合、つぎの２つのことが言える。

（１）　demoというメタ述語を用いて、メタ知識(ICs) とデータベースの知識とを融合させて、それらを同様に処理できる DBM = ( DBM1 + DBM2 )を作成した（メタの世界をメタ言語(ML)とオブジェクト言語(OL)とで、メタのメタの世界を( ML + OL ) をオブジェクト言語とみた場合のメタ言語(ML'')で構築）。すなわち、メタの世界とオブジェクトの世界との融合をDBM というメタの世界とメタのメタの世界とで実現している。

（２）　DBM が対象とするオブジェクトレベルの言語(OL)が存在する。

### ３．３　知識同化の実現の前提

ＰＲＯＬＯＧで知識同化を実現するために、以下の３条件を前提とする。

（１）　入力データを外延とする。
入力データは内包まで拡張可能であるが、今回は入力データを外延（ファクト）に限定する。

（２）　データベースはClosed World Assumption( CWA ) [Reiter 78], すなわち Negation as Failure [Clark 78]を仮定する。簡単に言うと、データベースから証明できない知識は、そのデータベースにおいて“偽”であるとすることである。

（３）　Consistency に対して、次の仮定を置く。

（３．１）　CurrdbとICs は一階論理の意味で無矛盾とする。

（３．２）　２引数demoを用いれば、現在のデータベース（Currdb）とICs とが次の意味で無矛盾(consistent)である(2.2.2項参照）。

consistent (Currdb, ICs)　≡
not (demo (Currdb, not (ICs))) .　(注)

ただしここに、“≡”の右辺の最初の“not ”は証明可能性の否定というメタ言語上の否定である。また２番目の“not ”はオブジェクト言語上のICs 表現の否定というオブジェクト言語上の否定である。“consistent”という概念はメタ言語上の知識とオブジェクト言語上の知識を、まさに融合して使用している。

なお、この前提３）は本稿で示している“知識の同化”を開始する時点で成立していればよいことは注意すべき点である。

---

(注)　ここでは、 2番目の“not ”を“！（カット・オペレータ）”を用いて実現しているので、demoが有限時間内に停止する場合のみを考察対象とする。

## ３．４　矛盾性チェックの効率化

矛盾性チェックの効率化を２つの方法で実現しているので、以下に説明する。

### ３．４．１　Integrity Constraint選択方式

ICs のシンタックスは３．２節で示したが、それは次の形式である。

```
current_db (check_db( 対象リレーション, (ICs), '矛盾検出時のメッセージ',
        [対象データベース])).
```

ここで、" check_db "の第１引数（対象リレーション）が指定可能になっている。これにより、対象リレーションに対応するICs だけを選択する。具体的には、次の３条件を判別することにより、対象リレーションに無関係なICs のチェックによる矛盾性の検査を行わなくて済む。

ａ）　リレーション名

ｂ）　対象リレーションの引数の個数

ｃ）　対象リレーション内に出現する定数値

### ３．４．２　実現値駆動型矛盾検索

３．４．１項のICs 選択方式により選択された個々のICs に対して矛盾性の検査をする方式として、実現値駆動型矛盾検索を考えた。この方式をdemoを用いて表現すると、次のようになる。

```
demo (not (ICi))
```

すなわち、ICi の否定を満足するデータ・オカレンスを検出する方式である。

この方式によると、対象となるデータベースを高々１回検索するだけでよい。しかも、その検索の途中でICi の否定を満足するデータ・オカレンスを検出すると、その時点で矛盾を発見したことになり、矛盾性の検査を終了することができる。

また、検索時に変数をインスタンシエートしながら検索を行うので、検索空間をせばめることができる。

さらに、ICi の否定というゴール列に属する各ゴールの証明処理をPROLOGのバックトラック機能を直接利用して行える。

## ３．５　冗長性の除去可能性の検査効率化

一般に、冗長性の除去の対象は、現在のデータベース（Currdb）の元（但し、元は知識として意味のある最小単位,2.2.3.1参照）である。しかし、同化の対象を外延（ファクト）に限定すれば、Currdbのうちファクトだけを個別に冗長性の除去の対象にすればよい。これにより、冗長性の除去の検査の効率化が図れる。その略証は次のようになる。

〔略　証〕

“Currdb + Input”から冗長性を除去するために、“Currdb + Input”が$A \cup a \cup B \cup b \cup \{c_1, \cdots, c_n\}$ (A,B: ルールの集合; a,b,$\{c_1, \cdots, c_n\}$ : ファクトの集合; それらは互いに素) と表記され、かつ次の条件(1) が成立する場合を考える。

$$A \cup a \vdash c_1, \cdots, c_n$$

(ただし$\{c_1, \cdots, c_n\}$ は全ての冗長なファクト集合)　　(1)

集合$F = a \cup b \cup \{c_1, \cdots, c_n\}$ を考え、それらから任意のファクトを順次選択していく。$f \in a$ あるいは $f \in b$の場合、内部データベースは更新されないので、$f \in \{c_1, \cdots, c_n\}$ の場合のみを考える。

$f = c_{i_1}$の場合、ファクト$c_{i_1}$を除去した内部データベースから、次式(2) が成立する。

$$A \cup a \cup B \cup b \cup \{c_1, \cdots, c_{i_1-1}, c_{i_1+1}, \cdots, c_n\} \vdash c_{i_1} \qquad (2)$$

( ∵ (1) より$A \cup a \vdash c_{i_1}$　∴ (2) の左辺 $\vdash c_{i_1}$ )

以下同様にして、ファクト$c_{i_j}$を除去していくと、最終的に内部データベースに対して次式(3) が成立する。

$$A \cup a \cup B \cup b \vdash c_{i_m} \qquad (3)$$

上式(3) の左辺は、内部データベースから全ての冗長なファクトが除去されたことを意味する。　　( Q.E.D )

## 3．6　PROLOGによるプログラムの実現

PROLOGによると、PROLOG Interpreter in PROLOGの手法を用いて，知識の同化を容易に実現できることを先に述べた。本節では、知識の同化を実現するPROLOGプログラムを説明する（図 3.2参照）。ただし知識の融合に関しては、 3.2.2項のアイデアを採用した。

KA1 は証明可能性の検査プログラムである。入力データに対してdemoを実行して、成功すれば証明できたことになる。入力データを証明できた場合は、新データベースは現在のデータベース(current db)のままであり、知識の同化の処理は終了する。

KA2 は矛盾性の検査プログラムである。最初の節において、最初のゴールで (current db + Input) が生成されたという仮定を行っている。 2,3番目のゴールで、適当なICを１つずつ選択して取ってくる。 4,5番目のゴールで矛盾性の検査のための形式の変換を行い、6番目のゴールで矛盾性の検査を実行する。 7番目以降のゴールは矛盾検出時のメッセージの出力及び後処理である。矛盾を検出した場合は、新データベースは元のデータベースのままであり、知識の同化の処理は終了する。

KA3 は冗長性の除去可能性の検査のプログラムであり、noredun がそれを行う。noredun では、current dbから抽出した１つのデータ(Ki)が (current_db + Input - Ki) から証明可能か否かを検査している。noredun の最後の節は後処理を行っている。

KA4 は、独立性の検査を行うプログラムである。しかし、独立を 2.2.3.2項の定義とすると、KA1 ～KA3 により独立性の検査がCWA を仮定したデータベースに対して完了していることになるので、KA4 では入力データをデータベースに格納することだけを行っている。

```
/* KA1) Deducibility Check ? */

assim(Input,View):-
      demo(Input,View),ttynl,ttynl,
      display('--- Input-Knowledge is deducible from DB !!'),
      ttynl,ttynl.

/* KA2) Contradiction Check ( against Integrity Constraint) */

assim(Input,View):-
      assert(current_db(Input)),
      current_db(check_db(Input,IC,Message,ViewX)),
      co_owner(View,ViewX),
      ic_trans(IC,ICR),
      Check_IC=..[demo,ICR,View],
      Check_IC,ttynl,ttynl,
      display('--- Input conflicts with '),
      display('the Integrity constraint !!'),
      ttynl,ttynl,display('     '),
      display(Message),
      ttynl,ttynl,ttynl,
      retract(current_db(Input)).
assim(Input,View):-retract(current_db(Input)),fail.

ic_trans((ICP-->ICQ),(ICP,not(ICQ))).
ic_trans((ICA;ICB),(ICAM;ICBM)):-
                     ic_trans(ICA,ICAM),
                     ic_trans(ICB,ICBM).
ic_trans(((ICP-->ICQ),ICB),(ICP,not(ICQ);ICBM)):-
                     ic_trans(ICB,ICBM).


/* KA3) Redundancy Check in ( DB + Input ) */

assim(Input,View):-
      current_db(X),
      noredun(X,Input,View),
      fail.
noredun((P:-Q),Input,View):-!,fail.
noredun(X,Input,View):-retract(current_db(X)),
                       assert(current_db(Input)),
                       demo(X,View),
                       retract(current_db(Input)),!.
noredun(X,Input,View):-retract(current_db(Input)),
                       asserta(current_db(X)),!.

/* KA4) Independency Check     */

assim(Input,View):-
      assert(current_db(Input)),
      ttynl,ttynl,
      display('--- New Knowledge is acquired !!'),
      ttynl,ttynl.
```

（図３．２　知識同化のPROLOGプログラム例）

## ３．７　例題

知識の同化の各検査についてその実行例を以下に示す。

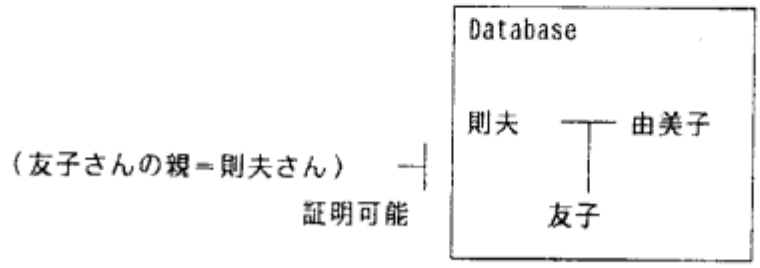
［例 KA1］　証明可能性の検査例

『則夫さんは友子さんの親である』という知識をデータベースに追加することにする。しかし、実行結果からわかるように、データベースではその知識は導出できるので、新しい知識として追加しなくてもよいことが検査結果から示されている。（図 3.3参照）。

－実行結果－

```
?- assimilate(parent(tomoko,norio),[parent]).

--- Input-Knowledge is deducible from DB !!
```




（図３．３　証明可能性の検査）

［例 KA2］　矛盾性の検査例

『Ｈ病院で、則夫さん・由美子さん夫妻に子供（陽子さん）が誕生しました。病院側では、陽子さんを２人の子供として登録します。このとき、遺伝学的に正しいことの検査も行っています。』

実行結果では、遺伝学的に誤っていることが検出されている（「（遺伝学）のメンデル博士が“Ｎｏ”と言っている。」という矛盾検出時のメッセージが出力されている）。これは、血液型がＡ型とＯ型の夫婦からＢ型の子供は生まれないことが検出されているのである（図３．４参照）。

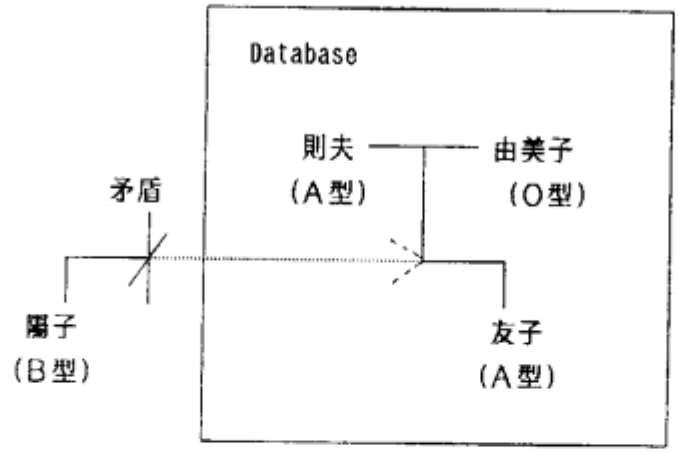
－実行結果－

```
?- assimilate(blood_type(youko,b) ,[parent]).

--- New Knowledge is acqured !!

yes
  ?-assimilate(father(youko,norio),[parent]).

---Input conflicts with the Integrity Constraint  !!
   Dr. Gregor Johann Hendel says  " NO ! "
```




（図３．４　矛盾性の検査）

ここで、矛盾性を検出したICは次の形式をとる。

－ICの形式－

```
check_db(
    father(X,F),
    (blood_type(F,FT),married(F,M),blood_type(M,MT),
    genes_match(FT,MT,CBT),blood_type(X,BT)
      -> member(BT,CBT)),
    'Dr. Gregor Johann Mendel says  " NO ! "',
    [parent] ).
```

ここでは、父親の血液型，母親，母親の血液型，子供の血液型，両親に対する子供の血液型の候補を導出して、子供の血液型がその候補の中に入っていることを関係“father”に対するICとしている。

［例 KA3］　冗長性の除去可能性の検査

データベースに、father,mother の知識が外延で、parent,grandparentの知識が外延と内包とにより表現されている。また、'not( ・・・)'の形式の否定の知識がデータベース中に入っている（図 3.5参照）。

このとき、冗長性のプログラムが実行されると、図 3.6，図 3.7で示されるようにgrandparent(由紀子，康夫）はそれ以外のデータベースの知識から導出されるので、冗長な知識と判断されてデータベースから除去される。もっと一般的に言うと、father,mother の外延とparent,grandparentの内包とから導出される外延が除去される。また、CWA を仮定しているので、'not( ・・・)'の形式の否定の知識もデータベースから除去されている（図 3.7参照）。ここで、" grandparent(yasuo,nizaemon) " という知識は他の知識から導出できないので除去されていない。これは、" grandparent " の知識が外延と内包との２つの形式でデータベースに蓄積されていることを示している。

```
| ?- listing(current_db).


current_db(father(yukiko,asao)).
current_db(father(asao,yasuo)).
current_db(father(yasuo,toshio)).
current_db(father(hiroko,masuo)).
current_db(father(tomoko,norio)).
current_db(father(youko,norio)).
current_db(father(norio,fujio)).
current_db(father(yumiko,haruo)).
current_db(mother(yukiko,tomoko)).
current_db(mother(asao,hiroko)).
current_db(mother(yasuo,akiko)).
current_db(mother(hiroko,setsuko)).
current_db(mother(tomoko,yumiko)).
current_db(mother(youko,yumiko)).
current_db(mother(norio,michiko)).
current_db(mother(yumiko,sachiko)).
current_db(parent(yukiko,asao)).
current_db(parent(yukiko,tomoko)).
current_db(parent(asao,yasuo)).
current_db(parent(asao,hiroko)).
current_db(grandparent(yukiko,yasuo)).
current_db(grandparent(yukiko,hiroko)).
current_db(grandparent(yukiko,norio)).
current_db(grandparent(yukiko,yumiko)).
current_db(grandparent(yasuo,nizaemon)).
current_db(not(father(tomoko,masuo))).
current_db(not(mother(yukiko,yumiko))).

current_db((parent(_1,_2):-father(_1,_2);mother(_1,_2)),[_3]).
current_db((grandparent(_1,_2):-parent(_1,_3),parent(_3,_2)),[_4]).
```

（図3．5　冗長性の除去可能性の検査例（データベースの内容））

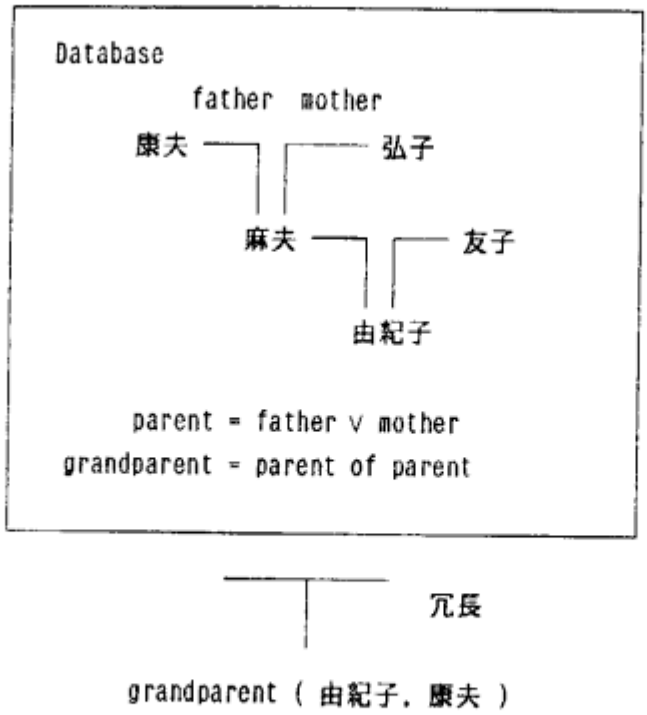
```
| ?- listing(current_db).

current_db(grandparent(yasuo,nizaemon)).
current_db(mother(yumiko,sachiko)).
current_db(mother(norio,michiko)).
current_db(mother(youko,yumiko)).
current_db(mother(tomoko,yumiko)).
current_db(mother(hiroko,setsuko)).
current_db(mother(yasuo,akiko)).
current_db(mother(asao,hiroko)).
current_db(mother(yukiko,tomoko)).
current_db(father(yumiko,haruo)).
current_db(father(norio,fujio)).
current_db(father(youko,norio)).
current_db(father(tomoko,norio)).
current_db(father(hiroko,masuo)).
current_db(father(yasuo,toshio)).
current_db(father(asao,yasuo)).
current_db(father(yukiko,asao)).

current_db(person(harumi)).

current_db((parent(_1,_2):-father(_1,_2);mother(_1,_2)),[_3]).
current_db((grandparent(_1,_2):-parent(_1,_3),parent(_3,_2)),[_4]).
```

（図３．７　冗長性の除去可能性の検査例（冗長性の除去後のDBの内容））




（図３．６　冗長な知識の検出例）

［例 KA4］　独立性の検査例

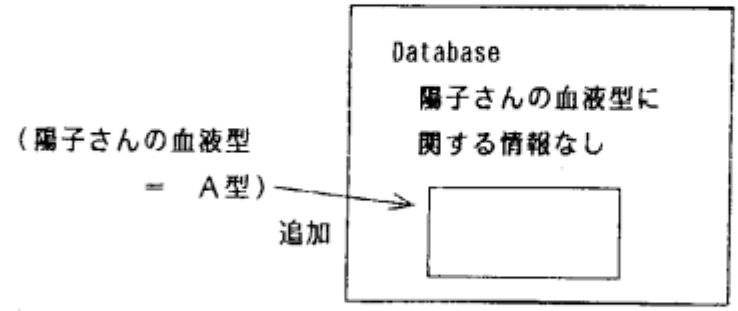
『陽子さんの血液型がＡ型である』というデータベースに追加する。

ここでは、証明可能性，矛盾性，冗長性の検査を通過して来ているので、入力データはデータベースに独立であると判断されて、データベースに追加される（図 3.8 参照）。

－実行結果－

```
?- assimilate(blood_type(youko,a),[parent]).

--- New Knowledge is acquired !!
```




（図３．８　独立性の検査例）

## 4．ま と め

データベースは個々にその目的に応じて意味を有していて、既に蓄積されているデータベース中の知識はその意味において価値を有している。知識の同化は、データベースの意味に矛盾する知識や冗長である知識を獲得しないようにデータベース管理を行うという意味で重要である。

そこで、最初に論理データベースはIntegrity Constraint(IC)によりその意味を定義されるべきであることを述べた。

つぎに、ＩＣにより意味が規定されているデータベースに対して、知識の同化における矛盾性，冗長性，独立性の定義とその検査間の関係の検討とを行うことにより、知識の同化の概念を実現可能な形で構築した。

また、知識の同化を、PROLOG Interpreter in PROLOGの手法によるオブジェクトの世界とメタな世界との融合を実現する方法により、容易に実現出来ることを見い出した。さらに、実現された融合化世界(Amalgamated World) の形態について考察を加えた。

知識の同化のPROLOGによるインプリメントは、CWA を仮定し、入力データを外延に限定して行った。ここで、知識の同化の対象（入力データ）を内包（ルール）に拡張することも可能であるが、本稿では行わず、別の機会に報告することにする。しかし、一方では、入力データを外延に限定したことにより、矛盾性の検出の効率化および冗長性の除去の効率化をも幾つか行うことができた。

なお、この知識同化プログラムは、使用開始時にデータベースとＩＣとがconsistentであれば使用してよいことは注意すべき点である。また、ＩＣの管理はデータベース利用者にその責任をゆだねる形態をとる。

今後の研究課題として、知識の同化のシステムについて、以下の事項を検討中である。

- 矛盾性の検査および冗長性の除去の検査の最適化
- 利用者との柔軟性のある会話形態と易用性の追及
- データベースのビュー機能の利用による多世界データベースに対する知識ベース管理の実現

［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所長と有益な討論をして頂いた麻生盛敏，坂井公研究員，向井国昭主任研究員(ICOT)に深謝いたします。また、数解研の研究集会の席上で適切なコメントをいただいた大須賀節雄教授（東大)，田中譲講師（北大），勝野裕文氏（武通研）らに感謝いたします。

〔参考文献〕


〔B&B&G〕 C.Beeri, P.A.Bernstein, and N.Goodman; "A Sophisticated Introduction to Data Dase Normalization Theory," Proc. of the 4th VLDB Conf., Berlin, 1978.

〔B & K 81〕 K. A. Bowen, R. A. Kowalski ;" Amalgamating Language and Meta - language in Logic Programming," June, 1981.

〔Cadiou 76〕 J.M.Cadiou;"On Semantic Issues in the Relational Model of Data," Math. Found. Comput. Sei.Mazmkiewiez. Vol.45, Berlin Heidelberg New York:Springer, 1976.

〔G & M 78〕 H.Gallaire, J.Minker(eds.); " Logic and Data Bases," Plenum Press , New York London, 1978.

〔G & M & N 81〕 H.Gallaire, J.Minker, and J.Nicolas(eds.); "Advances in Data Base Theory, Vol.1," Plenum Press , 1981.

〔Nicolas 82〕 J.Nicolas (ed.);"Proceedings of Wordshop on 'Logical Bases for Data Bases,', " ONERA-CERT, Toulose, 14-17, Dec.1982.

〔N & G 78〕 J.M.Nicolas, H.Gallaire;"Data Bases: Theory vs. Interpretation," in Logic and Data Bases ( H.Gallaire and J.Minker, Eds.), Plenum Press, New York London, pp.34～ 54, 1978.

〔P & W 78〕 F.Pereira, D.H.Warren; "Definite Clause Grammars Compared with Augmented Transition Network, " DAI, Univ. of Edinburgh 1978.

〔Reiter 78〕 R.Reiter ; "On Closed World Databases," in Logic and Data Bases ( H.Gallaire and J.Minker, Eds.), Plenum Press, New York London, pp.55 ～ 76, 1978.