ICOT Technical Report: TR-025

TR-025

A Knowledge Assimilation Method for Logic Databases

by

T. Miyachi, S. Kunifuji,
H. Kitakami, K. Furukawa,
A. Takeuchi, H. Yokota

September, 1983

A KNOWLEDGE ASSIMILATION METHOD FOR LOGIC DATABASES

T. Miyachi, S. Kunifuji,

H. Kitakami, K. Furukawa,

A. Takeuchi, H. Yokota

Institute for New Generation Computer Technology (ICOT)

Abstract

In this paper we consider a deductive question answering system for relational databases as a logic database system. We propose a knowledge assimilation method suitable for such a system. The concept of knowledge assimilation oriented to deductive logic is formulated in an implementable form based on the notion of amalgamating object languages and meta languages. The concept of knowledge assimilation consists of checks on subconcepts called provability, contradiction, redundancy, independency, and corresponding internal database updates. We have implemented a logic database-oriented knowledge assimilation program in PROLOG, a logic programming language. As a result, we found PROLOG suitable for knowledge assimilation implementation, among other uses.

1. Introduction

Humans acquire knowledge from the real world when they recognize the value of its existence. Knowledge-base management systems incorporating a relational database management system take on the role of knowledge acquisition, knowledge representation, and knowledge utilization. Nevertheless, it is humans that make the final decision as to whether or not to store knowledge in databases. Since,however, humans can lack the necessary knowledge and are careless, acquisition of knowledge should be refused if the knowledge is implicitly inconsistent with human intent or has already been acquired.

Storing into databases only the necessary knowledge which is also consistent with the intent of the acquisition is called knowledge assimilation. To assure the validity of knowledge stored in databases, knowledge assimilation is indispensable at the time of knowledge acquisition.

The basic concept of knowledge assimilation was presented by Bowen et al [B & K 81]. But, they did not show any specific method for its implementation. Research into integrity constraints in databases has been conducted by Cadiou [Cadiou 76], Nicolas [Nicolas 78], Beeri et al. [B & B & G 78], and others. This research corresponds to research on the problem of contradiction addressed by Bowen et al. in the context of knowledge assimilation.

In this paper, section 2 defines the concept of knowledge assimilation in an implementable form to clarify it theoretically. Section 3 reports on the PROLOG-based knowledge assimilation implementation method we have developed.

The research field addressed in this paper is what has so far been called research on fundamental theory, application, and implementation of deductive question answering systems for relational databases. For simplicity, however, it is called a "logic database" here as, in CERT-Workshop [G & M 78], [G & M & N 81] and [Nicolas 82].

2. Knowledge assimilation

The basic concept of knowledge assimilation was presented by Bowen & Kowalski [B & K 81]. To explain the concept it is necessary to introduce a meta language to operate on the provability of object languages. Therefore, we first explain research on meta languages in 2.1 and then discuss the concept of knowledge assimilation in 2.2.

2.1 Research on meta languages

Operations on the provability of object languages cannot be performed within the framework of object-level languages. [B & K 81] shows that it is possible to expand the power of an object language by incorporating a meta language to deal with the provability of object-level languages. It shows that the object level and meta level can be dealt with by a single language. This is called amalgamation of object-level languages and metal-level languages. The following D1 and D2 are the top-level program of the meta predicate "demo" for dealing with the said provability. "Demo" judges whether Goals are provable from Prog. "demo(Prog, Goals)" corresponds to 'Prog $\vdash$ Goals' ($\vdash$: provability).(Note 1)

```
D1)  demo (Prog, Goals) <- empty (Goals)    (Note 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 + Reset, Sub, Newgoals),   (Note 3)
            demo (Prog, Newgoals).
```

Programming this 'demo' predicate just as it is written here is not easy, and the program is not efficient.

(Note 1)Note that "demo" stops whenever it finds the right solution for a provable goal but does not necessarily stop in other cases.
(Note 2) The use of '<-' instead of ':-' in PROLOG indicates that the procedural interpretation of PROLOG is preserved but the order of clauses is not specified.
(Note 3) Here '+' represents the union set operator.

Meantime, in the world of PROLOG, Pereira & Warren developed a "PROLOG Interpreter in PROLOG" as early as 1978, demonstrating that a PROLOG interpreter can be easily prepared using PROLOG itself[P & W 1978].

In this connection we have found the following facts: The meta predicate "demo" and the PROLOG interpreter in PROLOG have almost the same function in respect to meta language implementation; and "demo" can be easily implemented by the PROLOG Interpreter in PROLOG. The basic function of "demo" can be achieved by the following six clauses.

```
demo(DB_name,true) :-!.
demo(DB_name,not(P)):-!,not(demo(DB_name,P)).
demo(DB_name,(P;Q)) :-!,(demo(DB_name,P);demo(DB_name,Q)).
demo(DB_name,(P,Q)) :-!,demo(DB_name,P),demo(DB_name,Q).
demo(DB_name,P):-  meta_call(DB_name,(P:-Q)),demo(DB_name,Q).
demo(DB_name,P):-  meta_call(DB_name,P).
meta_call(DB_name,P):- EP=..[DB_name,P],EP.
```

The demo program is briefly explained below. The first clause is assumed to be true if Goals is 'true'. The second, third, and fourth clauses give the proving procedures to be followed when goals are in the form of negation, disjunction, and conjunction, respectively. The fifth and sixth clauses actually do the proving: The fifth clause proves goals by rules(intension), while the sixth clause proves goals by fact(extension).

The above "demo" can be easily modified to handle the Cut Operator (!) as well.

Thus PROLOG makes it possible to easily amalgamate object-level and meta-level languages into one, which means that it is possible to implement knowledge assimilation with relative ease. It also means that PROLOG can operate on object-level knowledge and meta-level knowledge. These will be discussed in detail later.

2.2 Concept of knowledge assimilation

As knowledge is generally called data in the field of logic databases, data is used in the sense of knowledge also in this paper. Then the concept of knowledge assimilation can be shown by the following A1) through A4) [B & K 81].

(A1) If input data (Input) can be proved from the current database (Currdb), the new database is identical with the current one.

(A2) If one item of information (Info) in the current database can be proven from the rest of the database (Interdb) together with Input, the new database (Newdb) represents the database obtained by assimilating Input into Interdb.

(A3) If addition of Input to current database (Currdb + Input) results in a contradiction, input data (Input) must not be added to the current database (Currdb). The new database is identical with the current one.

(A4) If Input is independent of Currdb, the new database represents the database obtained by adding Input to Currdb.

From the foregoing A1 through A4, it can be reasoned that knowledge assimilation requires performing (A1) provability check, (A2) redundancy check, (A3) contradiction check, (A4) independency check and updating Currdb accordingly.

Although this reasoning is logically correct, there is no clear-cut definition of the processing sequence of A1 through A4 when implemented in a logic programming language. We believe their processing sequence should be reconsidered. Besides, no specific definition is given to contradiction or independency.

In this paper, therefore, we consider the processing sequence of knowledge assimilation and, define contradiction and independency. Furthermore, we make an in-depth examination of the role of redundancy checks, thereby giving a clear-cut definition to the concept of knowledge assimilation.

2.2.1 Processing sequence of knowledge assimilation

We consider it appropriate to process knowledge assimilation in the following sequence:

1) Provability check
2) Contradiction check
3) Redundancy check
4) Independency check

The reasons are given in a) through c) below.

a) The provability check and contradiction check are both intended to judge whether input data should be added to the database (which is called 'assimilable'). By contrast, the redundancy check is needed when input data is assimilable. Therefore, the redundancy check should take place after the provability and contradiction check.

b) Cases where input data can be proved from the database are contrary to cases where input data is inconsistent with the database. Basically, the provability check, which is conceptually simple and primary, should precede the contradiction check.

c) The independency check should take place after the assimilability check, because it includes the assimilability check. If input data is independent of the database, the redundancy check is already completed. Therefore, the independency check should be performed after the redundancy check.

2.2.2 Database contradiction

Individual databases have meanings consistent with their respective purposes and, therefore, should refuse to store data contrary to the meanings. Furthermore, semantic constraints on individual databases should be defined by individual integrity constraints. Knowledge acquisition into databases should be performed according to the meanings defined by integrity constraints.

Thus we define a database contradiction as follows: That the database generated by adding input data (New knowledge, Input) to the current database (Currdb) involves a contradiction means that there exists data which does not satisfy the pertinent integrity constraint (ICi).

2.2.3 Assimilability of input data

Input data (new knowledge) is assimilable into the current database (Currdb) if the following conditions a and b are satisfied:

a) Input data is not provable from Currdb.

b) Input data is not contradictory to Currdb in the sense of 2.2.2.

Furthermore, there are the following two cases where input data is assimilable into Currdb:

1) Redundant

2) Independent

These cases are discussed in 2.2.3.1 and 2.2.3.2, respectively.

2.2.3.1 Redundancy of data

When input data (Input) is assimilable, there is a possibility of redundant data existing in (Currdb + Input), where Currdb represents the current database. Thus we define (Currdb + Input) to be redundant if there exists non-trivial Ki represented by formula (A):

$$Ki \mathrel{-\!\!\ni} Currdb, \quad (Currdb + Input - Ki) \vdash Ki \qquad (A) \quad (note)$$

Note that generally the possibility of redundancy exists with respect to the minimum significant constructs of knowledge. If we recognize the existence of the minimum significant constructs of knowledge, the possibility of redundancy exists with respect to the partially pseudo-ordered set of Currdb besides mere elements of Currdb. In this case, the redundancy check generally must be performed on the partially pseudo-ordered set of Currdb.

It need not be demonstrated here that Input does not represent redundant knowledge against (Currdb + Input - Input) because this is already proved by the provability check.

2.2.3.2 Independence of input data

We define input data (Input) to be independent of the current database (Currdb) if a and b in 2.2.3 hold and no data that makes A hold exists in Currdb.

Input is independent of Currdb if Input is assimilable into Currdb and (Currdb + Input) does not constitute a redundancy.

(Note) The symbol '$\mathrel{-\!\!\ni}$' in the notation '$A \mathrel{-\!\!\ni} B$', indicates that A is a partially pseudo-order set of B.

2.2.4 Clarification of the concept of knowledge assimilation

Based on the foregoing definitions of the concepts of contradiction, assimilability, redundancy, and independence in knowledge assimilation and the processing sequence of knowledge assimilation, the concept of knowledge assimilation, including its processing sequence, can be formulated as follows.

```
KA1) assimilate(Currdb,Input,Currdb) <- demo(Currdb,Input).
KA2) assimilate(Currdb,Input,Currdb) <- demo(Currdb + Input, false).(Note)
KA3) assimilate(Currdb,Input,Newdb) <-   Info ∈ Currdb,
                                         Interdb = (Currdb - Info),
                                         demo(Interdb + Input, Info),
                                         assimilate(Interdb,Input,Newdb).
KA4) assimilate(Currdb,Input,Currdb + Input) <- independent(Currdb,Input).
```

Here the provability of false or contradiction (KA2) and independent (KA4) are as defined in 2.2.2 and 2.2.3.2, respectively. And it is considered sufficient to perform redundancy removal (KA3) as needed. Note that if the (KA3) procedure defaults, 'independent = assimilable'.

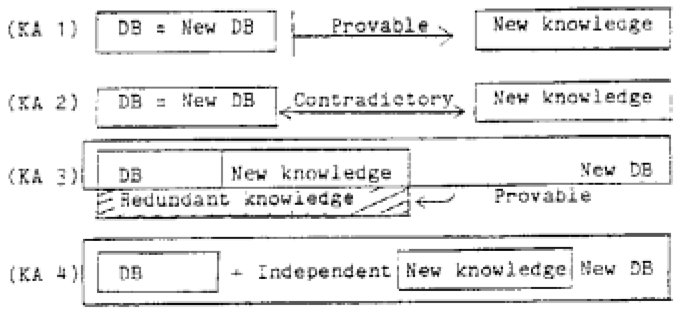


Figure 2.1 Processing of knowledge assimilation

3. PROLOG-based implementation of knowledge assimilation

3.1 Knowledge categorization and representation

(Note) Integrity constraints are meta-level knowledge concerning objects (Currdb and Input) but can be represented in object languages using the meta predicate "demo".

To begin with, we will define the knowledge being considered for knowledge assimilation. Knowledge can be represented by classifying it into three categories: facts(extensions), rules(intensions), and integrity constraints.(For specific examples, refer to 3.6.)

Here integrity constraints are separated from the other two because they represent the meta-level knowledge of extensions and intensions. Although addition of extensions or intensions leads to an increase in the knowledge in the database, addition of integrity constraints, or tightening of integrity constraints, results in a decrease in the knowledge that can be proved from the database. In this sense, integrity constraints should be separated from mere extensions and intensions. Therefore, we will represent integrity constraints with a different syntax than that for extensions and intensions. Integrity constraints will be defined by describing them within the framework of "check_db", together with a target relation, message on detection of contradiction, and target database:

```
check_db (target relation, integrity constraints,
          message on detection of contradiction: [target database]).
```

In check db, integrity constraints are described with the following syntax:

```
    <ICs>::= <IC>,<ICs> ¦ <IC>;<ICs> ¦ <IC>
    <IC> ::= <Ls> -> <L>
    <Ls> ::= <L>,<Ls> ¦ <L>;<Ls> ¦ not(<Ls>) ¦ <L>
    <L>  ::= not(<L>) ¦ <l>
    <l>  ::= <Goal>  (Note)
  (Note)    "Goal" represents a goal in PROLOG.
```

This syntax gives the expressive power, including that of Horn logic. (For an example, refer to 3.5.)

3.2 Preconditions for knowledge assimilation implementation

The implementation of knowledge assimilation in PROLOG is predicated on the following three conditions:

(1) Input data is an extension. Input data could be expanded to include intensions (rules), but for now is limited to extensions (facts).

(2) The database assumes Closed World Assumption (CWA) [Reiter 78], namely, Negation as Failure [Clark 78]. To put it simply, knowledge not provable from a database is assumed to be "false" in the database.

(3) The following assumptions are made on consistency:

(3.1) Currdb and integrity constraints are consistent in the sense of first-order logic.

(3.2) If the two-place demo predicate is used, the current database (Currdb) and integrity constraints are consistent in the following sense (Refer to 2.2.2):

Consistent (Currdb, ICs) ≡ not (demo (Currdb, not (ICs))). (Note)

Here the first "not" on the right side of "≡" is a negation of provability, or a negation in meta language; and the second "not" is a negation of the representation of integrity constraints in the object language, or a negation in object language. The concept of "consistent" uses knowledge in meta language and knowledge in object language by simply amalgamating them.

Note that condition 3 only has to hold true at the starting point of the "knowledge assimilation".

(Note) Since the second "not" is implemented by using "! (cut operator)",we are considering here only those cases in which demo stops within a finite time.

3.3 Improvement in efficiency of contradiction check

We will explain below two methods for improving the efficiency of the contradiction check.

3.3.1 Integrity constraint selection

The syntax of integrity constraints, as shown in 3.1, takes the form:

```
    current_db( check_db( target relation, (ICs),
                  'message on detection of contradiction', [target database])).
```

The first argument (target relation) of "checkdb" can be specified. Thus, only the integrity constraints relevant to the target relation specified are selected. Specifically, the contradiction check against integrity constraints irrelevant to the target relation is dispensed with by identifying the following three conditions:

a) Relation name

b) Number of arguments in the target relation

c) Constants appearing in the target relation

3.3.2 Instantiation-driven contradiction search

As a method for performing the contradiction check against individual integrity constraints selected by the integrity constraint selection method in 3.4.1, we have devised an instantiation-driven contradiction search approach, which is represented using demo as demo (currentdb, not (ICi)). This method is intended to detect a data occurrence which satisfies the negation of the related ICi. The approach requires searching a target database once at most. When a data occurrence which satisfies the negation of ICi is detected in the course of the search, the detection of contradiction can finish.

3.4 Improvement in efficiency of redundancy check

Redundancy removal is generally targeted at elements (minimum significant constructs of knowledge as referred to in 2.2.3.1) of the current database (Currdb). But if the knowledge to be assimilated is limited to extensions (facts), only facts in Currdb need to be individually considered for redundancy removal. This helps to improve the efficiency of the redundancy removability check. The proof is omitted.

3.5 Programs in PROLOG

We stated earlier that knowledge assimilation can be easily implemented in PROLOG using the PROLOG Interpreter. Here we will give an explanation of PROLOG programs to implement knowledge assimilation (Refer to Figure 3.2). Here, as contrasted with the two-place predicate: demo(Current_db, Goals), the one-place predicate: demo (Goals) is used for fixed current_db. And for checking Views, we add the term (View) to it.

KA 1 is a provability check program. If the execution of demo on input data is successful, it means that the data has seen proved. If input data has been proved, the new database remains the same as the current database (current_db), and the processing of knowledge assimilation is terminated.

KA 2 is a contradiction check program. In its first clause, the program assumes that (Current_db + Input) has been generated at the first goal. At the second and third goals it selects an appropriate integrity constraint. At the fourth and fifth goals it transforms the integrity constraints for the contradiction check. At the sixth goal it executes the contradiction check. At the seventh and succeeding goals it performs message output at the time of contradiction detection and post-processing. When contradiction is detected, the new database remains the same as the old database, and the processing of knowledge assimilation is terminated.

KA 3 is a program for the redundancy removability check, which is implemented by noredun. The last clause performs post-processing.

KA 4 is a program to perform the independency check. But given the definition of independency in 2.2.3.2, KA 1 through KA 3 have completed the independency check against the database which assumes CWA. Therefore, KA 4 does nothing but store input data in the database.

```
/* 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)),
     cc_owner(View,ViewX),
     ic_trans(IC,ICR),
     Check_ICs..[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.
```

(Figure 3.2 Example of PROLOG programs for knowledge assimilation)

3.6 Implementation examples

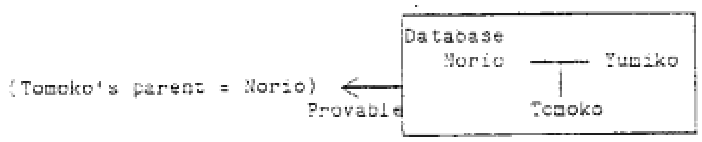
Examples of the implementation of the checks for knowledge assimilation are given below.

(Example KA 1) Example of provability check.

Let us suppose that the knowledge "Norio is Tomoko's parent" is being added to the database. Results of the check, however, show that the knowledge need not be added as new knowledge because it can be deduced from the database. (Refer to Figure 3.3)

```
- results of check implementation -

    ? - assimilate (parent (tomoko, norio), (parent)).
    --- Input-knowledge can be deduced from DB.
```



(Figure 3.3 Provability check)

(Example KA 2) Example of contradiction check

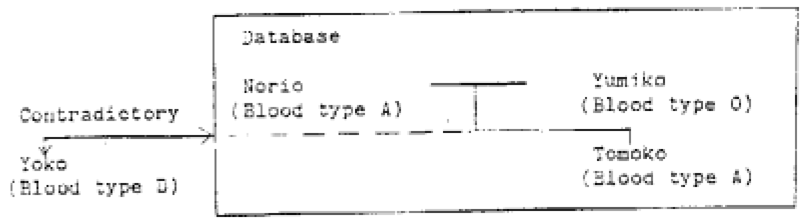
Knowledge to be added: "At hospital H, a child (Yoko) was born to Norio and Yumiko. The hospital registers Yoko as a child of the two. It has made tests to establish that this is genetically correct." Results of the contradiction check reveal that the knowledge is genetically wrong. ("Dr.Gregor Johann Merdel (geneticist) says No", a message given on detection of contradiction, is output.) This shows that no child with blood type B can be born to a couple with blood types A and O. (Refer to Figure 3.4)

```
- Results of check implementation -

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

 ---New Knowledge is acqured !!
 yes
  ?-assimilate(father(yoko,norio), [parent]).

 ---input conflicts with the Integrity Constraint !!
    Dr. Gregor Johann Mendel says " NO ! "
```



(Figure 3.4 Contradiction Check)

Here the integrity constraint that has detected the contradiction takes the form

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

Here the integrity constraint on the relation "father" requires that the blood type of the child be included in the set of possible blood types of the child deduced from the combination of the blood type of the father and the blood type of the mother.

(Example KA 3) Example of redundancy oheck

In the database, knowledge of father and mother is represented by extensions and knowledge of parent and grandparent, by extensions and intensions. (Refer to Figure 3.5.) In this case, if the redundancy check program is executed, the knowledge "grandparent (Yukiko, Yasuo) is judged redundant knowledge and removed from the database because it is deducible from other database knowledge as shown in Figures 3.6 and 3.7. To put it more generally, the extension deducible from the extension of father, mother, and the intension of parent, grandparent is removed from the database.(Refer to Figure 3.7.).

Here the knowledge "grand_parent (Yasuo, Nizaemon)" is not removed because it is not deducible from other knowledge. This indicates that knowledge of "grandparent" is stored in two forms--extension and intension--in the database.

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

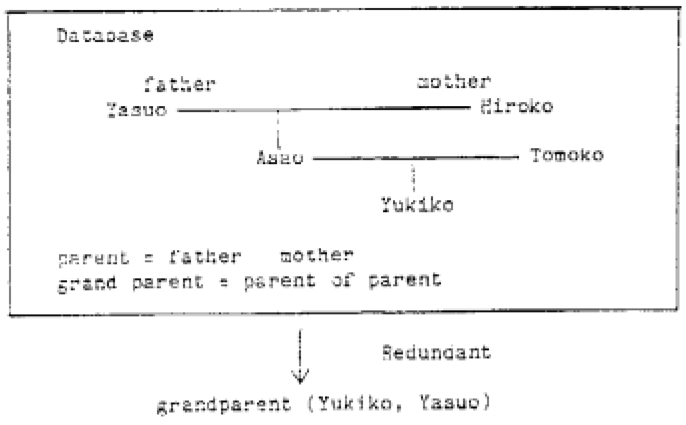
(Figure 3.5 Example of redundancy check (contents of database))

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

(Figure 3.7 Example of redundancy check (contents of DB after removal of redundancy))



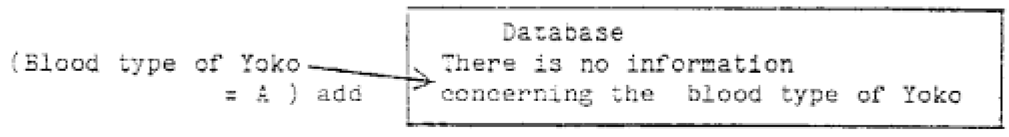
(Figure 3.6 Example of redundant knowledge detected)

(Example KA 4) Example of independency check

Suppose the knowledge "The blood type of Yoko is A" is being added into the database. Here the input data is judged independent of the database through provability, contradiction, and redundancy checks and added into the database. (Refer to Figure 3.8).

- Results of check implementation -

```
 ?-assimilate(blood_type(yoko,a),[parent]).
--- New Knowledge is acqured !!
```


Database
(Blood type of Yoko = A) add
There is no information concerning the blood type of Yoko

(Figure 3.8 Example of independency check)

4. Summary

Each database has a meaning consistent with its purpose. The knowledge already stored in a database has value in terms of that meaning. Knowledge assimilation is important for the purpose of managing a database so as to avoid acquiring knowledge inconsistent with the meaning of the database or redundant knowledge. At the beginning of this paper, we stated that the meaning of a logic database should be defined by integrity constraints. Then we constructed the concept of knowledge assimilation in an implementable form by defining contradiction, redundancy and independency in the assimilation of knowledge into the database whose meaning is defined by integrity constraints and discussing the relationships among these checks. We found knowledge assimilation could be easily achieved by amalgamating object and meta worlds using a PROLOG interpreter written in PROLOG.

Our PROLOG-based implementation of knowledge assimilation was conducted by assuming CWA and limiting input data to extensions. Input data to be assimilated could be expanded to include intensions (rules) as well, but we will report this on another occasion. Limiting input data to extensions, however, enabled us to make improvements in the efficiency of contradiction and redundancy removability checks. Note that this knowledge assimilation program can be used if the database and integrity constraints are consistent when at the start. Also, the program leaves the responsibility for integrity constraint management to the database user.

For the future, we are studying the following possibilities related to knowledge assimilation systems:

. Optimization of contradiction and redundancy checks

. Enhancement of flexibility in interfaces with users and ease of use

. Knowledge base management of multi-worlds databases using a database view function

(Acknowledgements)

The authors wish to express their thanks to K. Fuchi ,director of the Institute for New Generation Computer Technology (ICOT) for providing the opportunity to conduct this research. The authors thank M. Aso, H. Sakai, and K. Mukai of the Research Center for their helpful discussions. The authors also gratefully acknowledge the useful comments made by Professor. S. Ohsuga of Tokyo University, Lecturer Y. Tanaka, of Hokkaido University, H. Katsuno of Musashino Telecommunications Laboratory of NTT, and others at the RIMS symposium held at Kyoto University.

[Referrences]

[B&B&G 78] C.Beeri,P.A.Bernstein,and N.Goodman; "A Sophsticated Introduction to Data Base Normalization Theory,'Proc. of the 4th VLDB Conf., Berlin, 1978.

[B & K 81] K.A.Bowen,R.A.Kowalsky; "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 Workshop on 'Logical Bases for Data Bases,'," ONERA-CERT,Toulose,14-17,Dec.1982.

[N & G 78] J.Nicolas,H.Gallaire; "Data Base: Theory vs. Interpretation," in Logic and Data Bases(H.Gallaire,J.Minker,Eds.), Plenum Press, New York London,pp34-54,1978.

[P & W 78] F.Pereira,D.H.Warren; "Definite Clause Grammers 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, pp55-76, 1978.