TM-0095

Constraint-based Logic Database Management : Structuring Meta-knowledge in Database Management

Taizo Miyachi, Susumu Kunifuji, Koichi Furukawa
and Hajime Kitakami

February, 1985

ICOT

Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03) 456-3191~5
Telex ICOT J32964

Institute for New Generation Computer Technology

Constraint-based Logic Database Management : Structuring Meta-knowledge in Database Management

T. Miyachi, S. Kunifuji, K. Furukawa and H. Kitakami

Institute for New Generation Computer Technology (ICOT)

ABSTRACT

The semantic representation of knowledge and active utilization of structured meta-knowledge are very important for constructing intelligent knowledge assimilation functions and database management functions. This paper focuses on the profiles of the functions 'constraint' for the representation of the semantics of objectes in the real world. A database model is proposed; Constraint-based Semantic Model for a logic database (CSM) using Horn logic expressions. CSM easily enables users to describe static and dynamic semantics of the objects and assimilate knowledge according to users' purposes. Furthermore, users can also manage objects in the real world by managing the logic databases. Easily implementation of a prototype system of CSM logic databases using Prolog was achieved and it was confirmed that a logic programming language is suitable for building an intelligent database system.

1. INTRODUCTION

Advanced intelligent functions are required for the database management systems necessary to build intelligent database systems. These intelligent functions are classified into three large groups; 1) Knowledge Utilization Functions, 2) Knowledge Acquisition Functions, and 3) Knowledge Representation & Management Functions. Several studies have been conducted on Knowledge Utilization functions [L84,W81,Su83] and Knowledge Acquisition functions [D79,Sh82,K84,BK82,M84]. In this paper, we propose a method and a database model for Knowledge Representaion & Management functions.

The Knowledge Management & Representation functions are essential for accumulating knowledge and extending knowledge utilization. One of those functions is as follows. The database management system obtains and manages knowledge according to purposes or aims declared by a user for maintaining the database consistently. So users can know the contents of all the databases and the semantics, derivations and functions of knowledge in the databases. They can also assimilate requisite knowledge from other user's databases according to their purposes semi-automatically. Meta-level knowledge is used in such functions. Thus, we need to structure meta-level knowledge to build the intelligent knowledge management functions mentioned above. Some research pertaining to 'constraint' has been carried out [Ca76, NG76, SS80, BP83]. These projects were based on the notion that using 'constraint' in semantic expressions is effective.

In this paper, Section 2 discusses semantic expressions necessary for a logic DB system to reflect the real world. Section 3 proposes a 'Constraint-based Semantic Model' for an object-oriented logic database which treats 'constraint's as objects, and discusses its knowledge assimilation procedure. Section 4 reports on examples of execution runs of programs written in Prolog.

Research reports referenced in this paper are : M84, which established the suitability of using Prolog with a logic DB system for knowledge assimilation processing; SM84, which proposed an easy method for designing a Prolog-supported negative knowledge processing system; and Ki84, on related knowledge accommodation processing. We also refer to the research on integrity constraints by Cadiou[Ca76] and Nicolas[NG78], as well as some research on DB model [Co70],[Ch76],[S81], and [HM81].

2. EXPRESSING SEMANTIC RELATIONSHIPS IN A LOGIC DATABASE

Things existing in the real world are called "objects." Objects change and act continuously. Actions are also regarded as objects. A world in which objects exist is called an "object world;" knowledge in the object world is called "object knowledge." The data managed in usual databases is object knowledge. Necessary conditions on objects and relationships between objects covering action are called "meta-knowledge." Meta-knowledge differs from object knowledge in describing the controls on object knowledge.

The meaning of an object and its relationships with other objects are important when it changes or acts. The value of an object is determined according to the results of evaluating completed actions. Therefore, it is very important to describe an object's history and meaning together with its environment, conditions, and the degree of importance under which it changes or acts. A wide variety of meta-knowledge is incorporated based on the aims and intentions of the users, so the meta-knowledge must be carefully structured for describing, managing, and utilizing it effectively. The standard descriptions of aims and intentions can be facilitated for the users by structuring the meta- knowledge. Moreover, intelligent logic database management systems can contain opinions on each object and opinions for managing the logic databases.

In this paper we propose a method for structuring meta-knowledge by clarifying defined objects according to constraints which form part of the meta-knowledge itself.

2.1 Contents Stipulated by Constraints

A constraint is a restrictive condition on the validity of object knowledge covering actions. Thus descriptions of constraints must cover state changes caused by actions. More precisely, they have to describe the worlds, environments, times, conditions for changes, and actions before and after the change occurs. An advantage of state change descriptions over procedure descriptions is that the former are more declarative and more comprehensive. The 'basic constraint' is expressed in the following format which allows descriptions of the necessary conditions for state changes.

<Constraint>::=<Pre-Conditions>, (<Pre-State> ->> <Post-State>), <Post-Conditions>

<Pre-Conditions>, <Pre-State>, <Post-State>, and <Post-conditions> above are goal strings

written in a logic programming language equivalent to DEC 10 Prolog. The procedural meaning of the basic constraint is presented below. The statement "An action satisfies the corresponding <Constraint>." means that the following have been carried out in an object world:

1) Check whether <Pre-Conditions> are satisfied.
2) If <Pre-Conditions> are satisfied, fetch <Pre-State> determined by evaluating <Pre-Conditions>.
3) Re-evaluate <Pre-Conditions> for determining <Post-State> using the information of <Pre-State>.
4) Create <Post-State> and replace <Pre-State> with <Post-State>.
5) Check whether <Post-Conditions> are satisfied in the new object world.

Using the basic constraint <Constraint>, an object constraint <OC> which stipulates the validity of a piece of object knowledge is defined as follows:

```
<OC> ::= <Pre-Constraints> '('<Constraint>')' <Post-Constraints>
<Pre-Constraints> ::= nil | <OC>
<Post-Constraints> ::= nil | <OC>                          ... (S1)
```

<Pre-Constraints> above represents the descriptions of necessary conditions or constraints which are checked before a particular <Constraint>. <Post-Constraints> represents the descriptions of necessary conditions or constraints following a particular constraint. Therefore, the statement "<OC> is satisfied" means that the conditions of a particular constraint and of its preceding and subsequent constraints are satisfied. By allowing necessary limiting conditions to be described before and after <Constraint> like this, the relationships between elements of object knowledge, or indirect causality, or embedded causality can be described.

(Example 1)

Pre-Constraints: An employee is promoted from rank A to the manager class (MC) and his salary increases. (a)
Constraint: His jurisdiction is extended. (b)
Post-constraints: His property such as telephone sets increases. (c) (See 4.2 (B))

The possibilities for causal relations can be described here. They are the possibilities that (a) causes (b) and that (a) and (b) causes (c). The causalities are (1) (a) causes (b), (2) (a) and (b) cause (c) in example 1.

2.2 Constraint types

Constraints are broadly classified into the following two types:

a) **Existential Constraint (EC)**

There are conditions for preventing the existence of an object in an object world from generating a contradiction in that world.

b) **Action Constraint (AC)**

The conditions to be satisfied by an object world for a change in the state of an object existing in that world.

(1) Existential Constraint (EC)

An EC is a necessary condition for preventing the existence of an object in an object world from producing a contradiction in that world [M84]. Since it is a condition for the static existence of an object in an object world, an EC is defined as follows.

<EC> ::= <State>, <Consistency Conditions>

<State> is the description of the state of an object existing in an object world. <Consistency Conditions> is a set of necessary conditions for preventing the generation of contradictions in an object world. An EC can be regarded as the basic constraint independent of <Pre-Condition> and <Pre-State>. It represents necessary conditions for defining the framework of an object world and stipulates the static state of an object in that world. The concept of an EC is identical to that of an 'integrity constraint' used in logic database.

(2) Action Constraint (AC)

An AC can be described as <OC>. We believe that the following four profiles of constraints are indispensable for managing logic databases.

(a) Transition Constraint (TrC)

Generally speaking, time-sequenced multiple actions are generated in an object world. A TrC stipulates conditions for the validity of the sequence of these actions. A TrC is specified in an object constraint by the order of constraints. Thus, the TrC is identical to <Constraint> when the actions in an object world are independent of other constraints and time-bound sequences.

(Example 2) Example 1 is also an example of a TrC.

(b) Dependency Constraint (DeC)

When multiple actions take place in an object world, some of the actions may depend on their new instances determined by other actions. A DeC stipulates the conditions for the validity of dependency between these actions. A DeC is specified by shared variables in the object constraint.

(Example 3) When an automobile is replaced with a new one, the gearbox changes from manual to automatic; thus the driving method also changes.

(c) Class Constraint (ClC)

A ClC is a constraint which is effective for all the members of a real or virtual class existing in an object world. Unlike a TrC, the ClC is not applied to respective members of an object class but to the entire class. A ClC is specified in the description of actions according to pre-state and post-state.

(Example 4) When the salary of an employee at rank A increases 10%, the salaries of all the rest of employees at rank A also increase 10%.

(d) Time Constraint (TiC)

A TiC stipulates the validity of an absolute or a relative time for actions [Sh84,85].

(Example 5) Constraint C2 is applied 3 minutes after constraint C1 is applied.
(Example 6) Constraint C8 is applied at 8 o'clock on every week day.

It is some possibile to characterize constraints as is_a relations or part_of relations. But we do not regard them as special relations to be supported by logic database management systems. We take the view that users can use them in the same way as other general relations.

3. Compound Worlds and Semantic Expressions in a Logic Database

It is very important to represent objects in the real world accurately in the logic database. If this is achieved, we can manage the objects in the real world by managing the logic database. In this section, we draw a comparison between the real world and a logic DB reflecting it. Generally speaking, a DB is used for multiple purposes and consequently has one world corresponding to each purpose. The DB worlds corresponding to purposes are called 'unit worlds,' and a set of unit worlds is called a 'compound world.' A compound world is created for a characteristic shared by many unit worlds. A compound world represents a sub-real-world containing objects that will be interrogated by users. (See Figure 3.1.) Objects in the real world are expressed as reflected objects (RO) in a DB. The meanings and aspects of ROs in a particular unit world differ from those in other unit worlds, and the characteristics of a unit world depend on the meanings of ROs and on the relationships between ROs. To specify each unit world, therefore, means to specify the conditions to be satisfied by ROs.

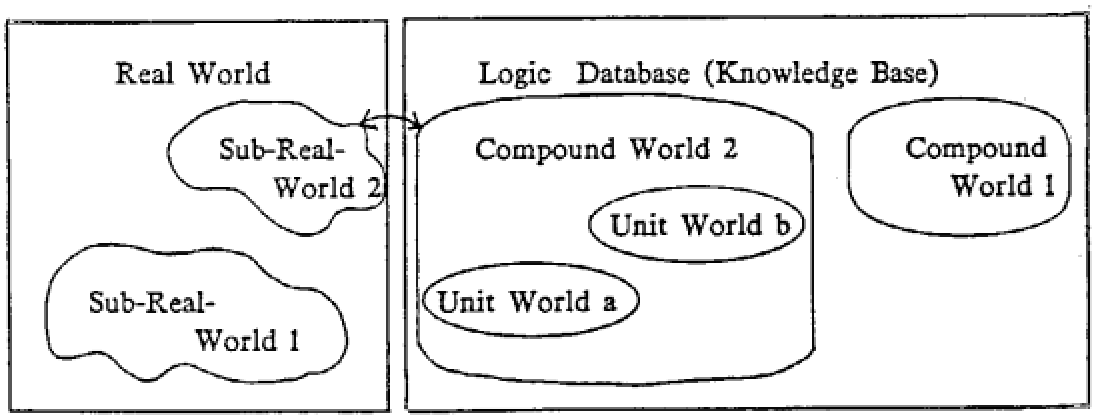

Fig. 3.1 Correspondence between real world and logic database

3.1 Constraint-based Semantic Model

A constraint-based database model is useful for representing the real world semantically in a logic database.

This section proposes a Constraint-based Semantic Model (CSM) for logic databases. Using constraint OC, the CSM expresses not only relationships between the contents of a logic database but also relationships between the semantics of these relationships to reflect sub-real-worlds in compound worlds in the database. The CSM is a database model which can express the static and dynamic semantics of relationships between objects.

[Definition of CSM]

Let Dl, D2, ..., Dn be n (n>0) domains (not necessarily distinct). Relation R of degree n is defined as a subset of the Cartesian product X{Di: i = 1, 2, ... ,n}. As an interpretation model, this relation R is used with the well-formed formulae of first-order Horn logic to define a logic database L. The constraint-based semantic model of a logic database is stipulated by the binomial relation:<{L1, L2, ..., Lp}, {OC1, OC2, ..., OCq}>, where L1 to Lp is a set of logic databases. (OCi is (S!) described in Section 2.1, and Lj's interpretation model is Rj respectively.)

The CSM can express semantically the following three types of objects:

1) Horn logic expression: This represents the semantics of objects in a sub-real-world as static semantic relationships between relations in the logic database.
2) Scenes, actions, and sequences of scenes: These represent the semantics of changes or actions which extend over many objects in the real world, expressing OCs declaratively to provide necessary conditions for compound worlds in a logic DB.
3) Dynamic semantics change based on real values in unit and compound worlds in a logic DB: These are also expressed as OCs.

Figure 3.2 is a conceptual diagram showing semantic relationships (SR1 to SRq)stipulated by expressing attributes and their values.

To express these variously changing objects existing in the real world, the CSM has the following five functions:

1) Expressing meta-knowledge (OCs) declaratively, using first-order Horn logic.
2) Adding OCs to a logic DB to expand the represented real world.
3) Changing relationships between OCs with ease.
4) Combining freely extensions with intensions into a semantic network.
5) Expressing abstract concepts by using relations, attributes, and instances.
(Expressing relations, attributes, and instances as objects.)

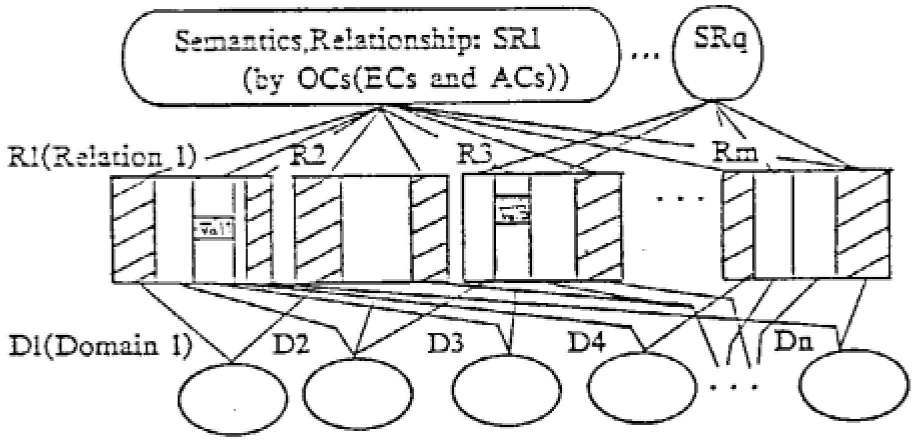

Fig. 3.2 The concept of descriptions of semantics and relationships between relations in CSM.

The results of prior research and our CSM are compared below in terms of ability to express constraints. The prior research refers to work on the "Integrity ; Constraint," "Integrity Checking," and the "Trigger." In "Integrity Checking" [MG78], by the Nicolas group, Existential Constraint (EC) and Transition Constraint (TrC) were studied as 'State Law' and 'Transition Law.' Part of the research on the "Trigger" [Ca76] is on Time Constraint (TiC). The CSM has the means EC and AC (TrC, DeC, ClC, TiC) explained in Section 2.2 to express the real world. These means include the above functions. However, when using Prolog, the CSM cannot have the Time Constraint (TiC) expressibility.

3.2 Consistency

DB management to prevent contradictions in a DB is important for the knowledge assimilation procedure. In a DB designed by using the CSM, the presence of non-contradiction under Clark's conditions (sufficient conditions for guaranteeing consistency of Negation as Failure) can be defined in detail. If non-contradiction is detected in a DB, the DB is said to be 'consistent' and satisfies the conditions shown below. Each demo(W,G) statement in the conditions corresponds to a 'W ⊢ G' defined in first-order Horn logic and means that G is proved from W in first-order Horn logic.

<Consistency check>

```
 non_contradiction(Compound_world,R-objects) ->
   demo(Finished_Constraints_list, Pre_Constraints),
   demo(Compound_world, Pre_Conditions),
   demo(Compound_world, Pre_State),
   demo(Compound_world, Pre_Conditions).
   substitute_state(Compound_World, Pre_State, Post_State),
   demo(Compound_world, Post_Conditions),
   non_contradiction(Compound_worldl  , Following_Constraints),
```

not(demo(Compound_world, not(Existence_Constraint))).
Compound_World1 is defined corresponding to Following_constraints in the OC.

Some dependent characteristics between constraints stipulating the consistency of a logic DB exist globally to be checked for DB consistency. These characteristics are called "Consistency Checking Dependencies (CCDs)." When new knowledge is stored in a DB, the Action Constraint to be first checked (AC(a)) is stored, and AC(a) specifies the AC to be checked next (AC(b)). Since checking AC(b) is requested by requesting checking AC(a) in this situation, AC(b) is said to be 'dependent' on AC(a). This relationship is expressed as "AC(a) => AC(b)." Constraints expressed by OCs make up a tree structure and are checked by the 'depth-frist' method. Each EC is checked after the corresponding AC is checked. (See Figure 3.4.) Here, it is assumed that unique AC is determined corresponding to new knowledge. By referring to CCDs, the user can determine what meaning an added OC has in the DB and how this OC is related with other OCs. CCDs can be used to evaluate the applicability of OCs to the real world.

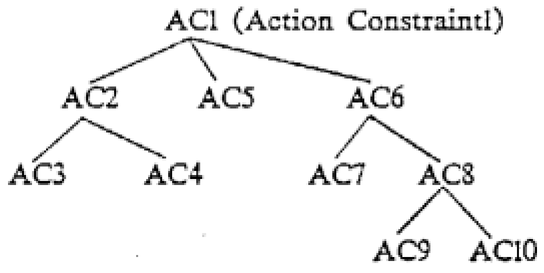

Fig. 3.4 Checking action constraints

3.3 Assimilation management of a logic database

Knowledge can be assimilated to a DB determined by the CSM-defined semantics of objects and inter-object relationships semi-automatically according to the aims of the users. There are two kinds of knowledge: input knowledge and knowledge about this input knowledge and stored knowledge. This section explains how knowledge is assimilated to a logic DB determined by CSM-defined object semantics. Basically, the knowledge assimilation procedure consists of the following three steps:

1) Detecting new knowledge which will produce a contradiction if added to the DB, and eliminating it from knowledge to be assimilated.
2) Adjusting knowledge in a compound world to keep them consistent when new knowledge is added to them.
3) Adusting the DB (a set of compound worlds) to keep it consistent when new knowledge is added to it.

These steps are devided into the 10 substeps below. (Step n is detailed into substep ni.) Procedures consisting of combinations of these are also possible. (See Figure 3.5.)

1a) Contradictory new knowledge is detected (by a check using an EC) and this knowledge is not added to the DB. (Ia)
2a) New knowledge is assimilated to the DB and no other changes occur in the DB.
2b) New knowledge is assimilated to the DB and then propagating changes occur in a compound world. (Ic)
2c) New knowledge is assimilated to the DB and then a change a occurs in all members of a class in a compound world. (Id)
2d) New knowledge is assimilated and then a change occurs in a compound world at a specified time.
2e) New knowledge is assimilated to the DB and then a combination of the changes mentioned in 2b) to 2d) occurs in a compound world.
3b) to 3e) New knowledge is assimilated to the DB and then the propagating, class-dependent, time-dependent, and compound changes mentioned in 2b to 2e occur in the DB. (Ie)

In Figure 3.5, the nodes outside Database 1 represent user's requests for knowledge assimilation. Changes resulting from the requests can be retrieved by the user. On the other hand, each node inside Database 1 corresponds to a change or an action. An arrow between nodes indicates the flow of propagation or influence of a change or an action. The check in step 1) is done on the flow indicated by each arrow. The user should confirm the results of this check if he needs to know about changes made by knowledge assimilation in the DB. In this way, consistent knowledge can be added to the DB.

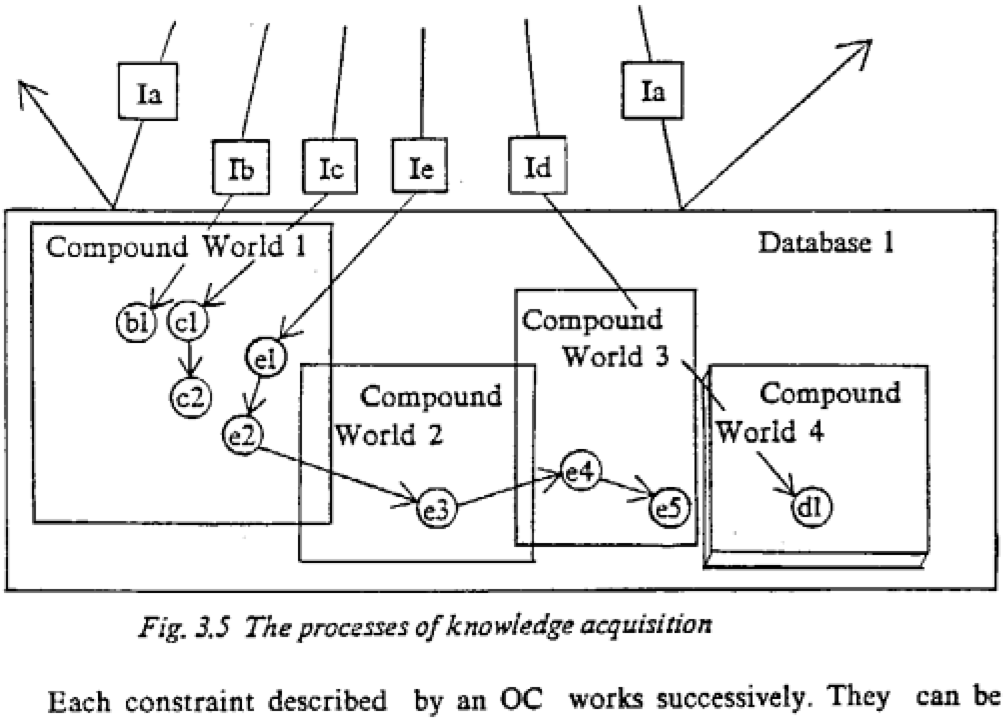

Fig. 3.5 The processes of knowledge acquisition

Each constraint described by an OC works successively. They can be easily described and changed and can clarify the semantics of a object.

(Example 7) In 2.2 (Example 1), the management world of the employee's state corresponds to compound world 1, each of the management worlds of money, of authorities, and of furnitures corresponds to compound worlds 2, 3, and 5 respectively.

3.4 Advantages of a CSM-Controlled Database System

A CSM-controlled DB system has functions to reflect users' purposes or aims thus supporting their intelligent activities and knowledge management. The functions are: a) The users can make brief declarative descriptions of the semantics of each object in the real world in the logic database using OCs. b) The database system can assimilate new knowledge and its related knowledge according to the users' intention and aims. c) The users can manage objects in the real world by managing ROs in the logic database. This section discusses the two interesting functions: 1) the knowledge acquisition function, 2) the function for managing the design and life cycle of a logic DB system.

(1) Knowledge acquisition function

In a CSM-controlled DB system users need only describe the meanings of knowledge items corresponding to applications by using constraints (OCs). Then the DB system acquires semi-automatically the knowledge according to users' aims. (See Figure 3.6.) The same constraint is used to control not only addition but also deletion of knowledge. Knowledge acquired by the DB system includes knowledge users have learned unconsciously and knowledge they have failed to learn. These two types of knowledge may be used to make up common sense. Learning like this is possible because learning stimulated by the assimilation of element knowledge is repeatedly stipulated by multiple constraints (OCs).

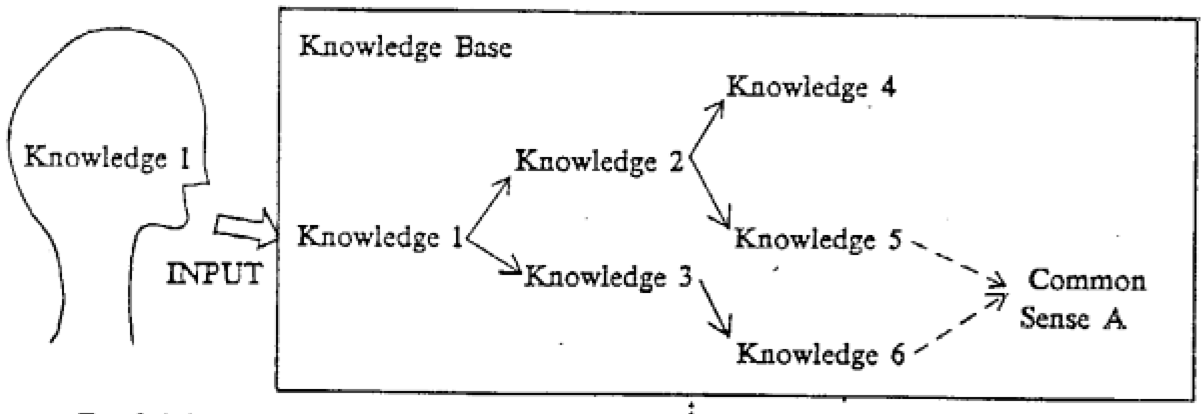

Fig. 3.6 Semi-automatic assimilation of knowledge and common sense

(2) DB design and life cycle management function

An intelligent activity support system should have a function to support not only the DB manager but also to support users in DB system (DBS) design, DBS management, and applications programming. (See Fig. 3.7) Users supported by such a function need only describe the meanings of objects in a logic DB, then the life cycle management function edits and outputs the descriptions of users' aims and the contents of a DB to facilitate DBS redesign and management evaluation.

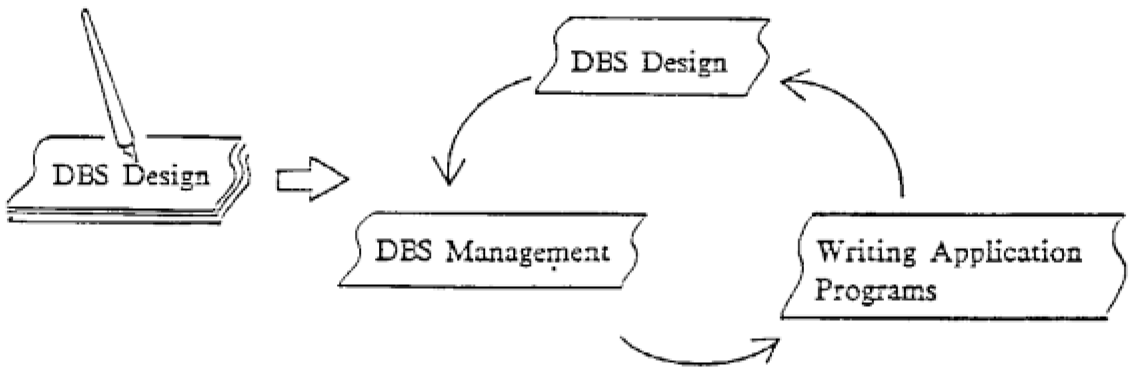

Fig. 3.7 Unifying design and management of DBS and creation of application programs

4. KNOWLEDGE ASSIMILATION BY THE CSM

The CSM can assimilate knowledge about input knowledge to a logic DB where the meanings of objects and semantic relationships between objects have been defined. This chapter explains how to use the CSM with the logic programming language Prolog to manipulate a logic DB.

4.1 Syntax of semantic expression

In a logic DB, knowledge is expressed by facts (extensions), rules (intentions), and constraints (OCs: ECs and ACs). (See Section 3) Constraints are meta-knowledge items expressing the meanings of objects and inter-object relationships and thus are expressed according to syntactic rules different from those for extensions and intensions.

To specify OCs, the following conditions must be satisfied:

a) Each object must be expressed independently.
b) A declarative expression format must be used.
c) Procedural interpretation must be possible.
d) Expressions must be brief.

These conditions must be satisfied because:

- If a) is satisfied, OCs can easily be added, modified, and deleted.
- If b) is satisfied, OCs can easily be interpreted and expression misses can be detected.
- If c) is satisfied, operations executing OCs can easily be interpreted.
- If d) is satisfied, the users' burden of describing OCs is reduced.

(A) Existential Constraint (EC) syntax

ECs are defined in statements including references to the relevant compound worlds and objects as well as a message indicating the detection of a contradiction as follows: check_EC(Compound-world, Object, EC, 'contradiction indication message')

ECs in check_EC frames are defined according to the following syntax:

```
<ECs> ::= <EC>,<ECs> | <EC>;<ECs> | <EC>
<EC> ::= <Ls> --> <L>
```

```
<Ls> ::= <L>,<Ls> | <L>;<Ls> | not(<Ls>) | <L>
<L>  ::= not(<L>) | <G>
<G>  ::= <goals of Prolog>
```

(B) Action Constraint (AC) syntax

An AC is defined in a check_AC frame according to the syntax below. The details of an AC are described as goal strings in Prolog.

```
check_AC( AC_ID,  Input,
        [actions( Pre_State ->> Post_State ),
         local_conditions( Class_Constraint_Attributes,
                           Pre_Conditions, Post_Conditions ),
         compound_world(Unit_World_Name_List), time(Time_Constraints) ],
        global_conditions(Global_Pre_Conditions, Global_Post_Conditions),
        action_constraints(Pre_Action_Constraints, Post_Action_Constraints),
        Importance ).
 Global_Pre_Conditions ::= [[World_Name1, Post_Conditions]|PrCR]
 Global_Post_Conditions ::= [[World_Name2, Post_Conditions]|PoCR]
 Pre_Action_Constraints ::=
       [[Pre_World_Name, Pre_Action_Constraints]|PrAR]
 Post_Action_Constraints ::=
       [[Post_World_Name, Post_Action_Constraints]|PoAR]
```

In a check_AC frame, the first argument specifies the identifier of an AC. The second argument specifies new input knowledge. For updating knowledge, the second argument should specify new relations and old relations in the format "update('oldtuple','newtuple'). For requesting the dissmilation of knowledge, old relations should be specified in format "remove ('tuple')".

The third argument describes the contents of an action as:

a) Actions
   1) List of pre-action states of objects
   2) List of post-action states of objects
b) Local conditions: necessary conditions in each unit world
   1) Class constraint attribute list
   2) Pre-action environment attribute list
   3) Post-action environment list
c) Unit world name list
d) Necessary constraints related to time.

The fourth argument specifies a list of necessary conditions which extends over many worlds before or after the action in the database. The latter term specifies final conditions the database must satisfy. This term provides means to check whether an action has been completed appropriately and the resulting changes made. An example of this type of constraint is the limit of the total budget over the whole database.

The fifth argument specifies conditions related to other actions. The former term specifies a list of necessary preceding actions and a list of prohibited preceding actions for an action using corresponding constraints. The latter term specifies a list of constraint names representing knowledge assimilation objects which should take place successively. As a result, the successive occurrence of actions is described, so is the shift of a scene. A transition constraint is specified by their order and a dependency constraint is specified by the shared variables they contain.

The names of constraints applied to preceding actions should be written to represent the preceding actions. The scene or propagation process before the object action is conditioned by this AC.

The former term in the fourth argument and elements a-1), b-2), c) and d) in the third argument specify the scene before an action. Class constraint attributes specified by b-1) in the third argument themselves become the objects of a class constraint. The elements in a) in the third argument specify changes made by an action, and b-3) provides necessary conditions after these changes.

The sixth argument specifies whether an action assimilates important new knowledge. If it is important, the history of the knowledge can be described and stored for later reference. The history description format is "sys-memory (ID, history)".

A question-answering module for inserting constraints into databases was easily constructed.

4.2 Knowledge assimilation Prolog program execution examples

This section presents examples of executing knowledge assimilation programs under the following three conditions:

1) Facts are input.
2) The DB satisfies Clark's conditions (sufficient conditions for Negation as Failure) [Cl78].
3) Consistency has the meaning explained in Section 3.2.

Since Prolog allows declarative expressions, the programs are written in this language. ECs and ACs can easily be expressed according to Prolog-determined syntax.

Examples of programs applying an EC and an AC are presented below together with necessary consistency check functions for knowledge assimilation.

(a) EC application example

Necessary function: Preventing the assimilation of contradiction-producing new knowledge to a knowledge base.

New knowledge: A baby (Yoko) was born to the couple Norio and Yumiko Yamada at hospital H. The hospital has registered Yoko as their second daughter after making a genetic check on her. Execution result: The result of the check is found erroneous, and the message 'Dr. Gregor Johann Mendel says "No!"' is output. This is because a baby having blood type B can not be born to a couple one of whom has blood type A and the other type O. (See Figure 4.1.)

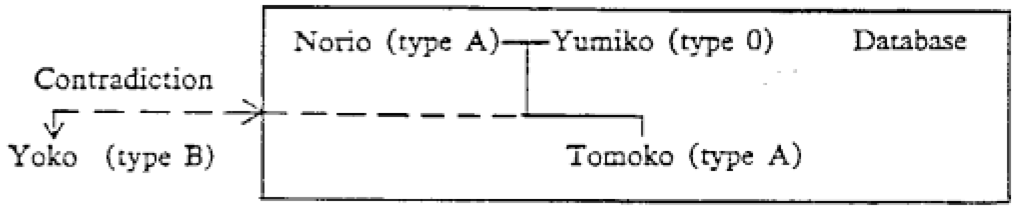

Fig. 4.1 Contradiction check using an EC

Input inquiry and output message:

```
| ?- assimilate([family],blood_type(yoko,b),[parent]).
---- A New knowledge is assimilated !!!
yes
| ?- assimilate([family],father(yoko,norio),[parent]).
--- Input conflicts with the Integrity constraint !!
    Dr. Gregor Johann Mendel says " NO ! "
```

EC format:

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

An EC defined in this format specifies the blood types of the parents and the baby as well as possible blood types of the baby. The EC to be applied to the relation "father" is based on the fact that the baby must have one of certain possible blood types.

(b) AC application example

old knowledge: - Relations: employee(E , ENAME, Rank, SAL, DEPT), aveSAL(Rank, AVERAGE_Salary), rate(DEPT, Rate)
- Yamada is an employee at rank A.
- AC stipulating that "if an employee is promoted from rank A to the manager class (MC), his salary increases to 'average salary of SMC: SMC= MC $\times$ departmental_rate,' and his jurisdiction expands."

New knowledge: Yamada is promoted from rank A to MC.

Execution results: Yamada's position is updated to MC, his salary to SMC, and he is given jurisdiction MC (including permission to enter RoomX1). Then he gets a telephone as his texture.

In this example, the management system automatically performs the necessary actions for retaining consistency in the corresponding worlds when Yamada is promoted and his jurisdiction expands. The user need only check the updated results.

Input inquiry and output message:

```
| ?- assimilate([employees],rank_up(Rank,emp(EN,n_yamada,Rank,SAL,DEPT),mc),[A]).
--- New Knowledge is  emp(4,n_yamada,mc,1176,researcher)
--- New Knowledge is  authority(mc,4,n_yamada,researcher)
--- New Knowledge is  fixtures(telephone,4,n_yamada)
--- AC is equipments(34,35,33,check_AC(8,equipments_request(4,n_yamada,mc),[[acti
ons([]->>[fixtures(telephone,4,n_yamada)]),local_conditions([],[],[equipments_ch
eck(4,n_yamada,mc)]),compound_world([equipments]),time([])]],global_conditions([
],[]),action_constraints([],[]),1))
--- Constraints are checked !!
--- AC is authority(32,33,31,check_AC(5,authority_check(4,n_yamada,mc),[[actions(
[]->>[authority(mc,4,n_yamada,researcher)]),local_conditions([],[[[employees],em
p(4,n_yamada,mc,1176,researcher)]],[]),compound_world([authority]),time([])]],gl
obal_conditions([],[]),action_constraints([],[[[equipments],[equipments_request(
4,n_yamada,mc)]]]),1))
--- Constraints are checked !!
--- AC is employees(29,30,28,check_AC(4,rank_up(a,emp(_69,n_yamada,a,_110,researc
her),mc),[[actions([emp(4,n_yamada,a,700,researcher)]->>[emp(4,n_yamada,mc,1176,
researcher)]),local_conditions([],[dept(3,researcher,98)],[1176 is 1200*98/100])
,compound_world([employees]),time([])]],global_conditions([],[]),action_constrai
nts([],[[[authority],[authority_check(4,n_yamada,mc)]]]),1))
--- Constraints are checked !!
```

AC formats:

```
check_AC(4,
        rank_up(RANK, emp(ENO,Ename,a,SAL,DEPT),mc),
        [[actions([emp(EN,Ename,a,SALpre,DEPT)] ->>
                  [emp(EN,Ename,mc,SALpos,DEPT)]),
          local_conditions([],
             [dept(DN, DEPT, DeptRate)],
             [(SALpos is 1200 * DeptRate / 100)]),
          compound_world([employees]), time([]) ]],
        global_conditions([], []),
        action_constraints([], [[[authority],[authority_check(EN,Ename,mc)]]]),
        1)

check_AC(5,
        authority_check(EN,Ename,mc),
        [[actions([] ->> [authority(mc,EN,Ename,researcher)]),
          local_conditions([], [[[employees],emp(EN,Ename,mc,SAL,researcher)]], []),
          compound_world([authority]),time([]) ]],
        global_conditions([], []),
        action_constraints([], [[[equipments],[equipments_request(EN,Ename,mc)]] ]),
        1)

check_AC(8,
        equipments_request(EN,Ename,mc),
        [[actions([] ->> [fixtures(telephone,EN,Ename)]),
          local_conditions([], [], [equipments_check(EN,Ename,mc)]),
          compound_world([equipments]),time([]) ]],
        global_conditions([], []),
        action_constraints([], []),
        ])
```

5. SUMMARY

We stated that structuring meta-knowledge is indispensable to build an intelligent logic database management system which can assimilate new knowledge and manage logic databases according to user's purposes and aims. We offered a Constraint-based Semantic Model (CSM) for structuring meta-knowledge based on "Constraint." Users of a CSM-controlled DB system need only describe knowledge and each object-associated scene semantically and declaratively. Then, the system assimilates and manages data and knowledge for them. If users express the design of a DB in CSM-supported formats, the CSM not only designs a DB but also manages it and creates application programs. A DB can easily be modified when a new purpose or meaning is generated, because users can easily look up the semantics of a reflected object (RO) in the database. The CSM also allows the flexible expressions of abstraction using relations, attributes, inter-instance relations, and their meanings. We investigated the constraint types required by CSM and the expressive power of CSM. The knowledge assimilation and management functions use the "depth-first" method to check Consistency Checking Dependencies (CCDs) between tree OCs (ECs and ACs). By this method, the functions can determine which new knowledge is to be rejected and perform processing to retain consistency when new knowledge is input into each compound world and each database.

Our future targets are:
- Implement the controls of time constraint using a logic programming language like ESP.
- Improving the expressibility of OCs
- Heightening usability

*** Acknowledgements ***

Our hearty thanks go to ICOT Research Center Director Mr. K. Fuchi who gave us the opportunity to conduct this research, researcher Mr. H. Kondo who helped us so much, and to the staffs of 2nd Research Laboratory.

* References *

[BBG78] C. Beeri, P.A. Bernstein and N. Goodman; "A Sophisticated Introduction to Database Normalization Theory," Proc. of the 4th VLDB Conf., Berlin, 1978.

[BK82] K.A. Bowen, R.A. Kowalski; "Amalgamating Language and Meta-language in Logic Programming," Logic Programming (K.L.Clark and S.-A.Taernlund eds.), Academic Press, pp.153-172, 1982.

[BP80] J. Barwise and J. Perry; "Situations and Attitudes," MIT Press, 1983.

[Ca76] J.M. Cadiou; "On Semantic Issues in the Relational Model of Data," Math. Found. Comput. Sci. Mazmkiewiez. Vol.45, Berlin Heidelberg New York, Springer, 1976.

[Cl78] K.L. Clark; "Negation as Failure," in Logic and Data Bases, H. Gallaire and J. Minker (eds.), Plenum Press, New York, London, pp.293-322, 1978.

[Co70] E.F. Codd; "A Relational Model of Data for Large Shared Data Banks," Comm. ACM 13,6, pp.377-387, Jun. 1970.

[Ch76] P.P. Chen; "The Entity-Relationship Model—Toward a Unified View of Data," ACM TODS, Vol.1, No.1, Mar. 1976.

[D79] R. Davis; "Interactive Transfer of Expertise: Acquisition of New Inference Rules," Artificial Intelligence 12, pp. 121-157, 1979.

[K84] H. Kitakami, S. Kunifuji, T. Miyachi and K. Furukawa; "A Methodology of Knowledge Acquisition System," Proceedings of 1984 International Symposium on Logic Programming, Atlantic City, pp.131-142, Feb. 6-9, 1984.

[HM81] M. Hammer and D. McLeod; "Database Description with SDM: A Semantic Database Model," ACM TODS, Vol6, No.3, Sep. 1981.

[L84] D. Li; "A Prolog Database System," Research Studies Press, 1984.

[M84] T. Miyachi, S. Kunifuji, H. Kitakami, K. Furukawa, A. Takeuchi and H. Yokota; "A Knowledge Assimilation Method for Logic Databases," New Generation Computing, Vol. 2, No. 4, pp. 385-404, 1984, also in Proceedings of 1984 International Symposium on Logic Programming, Atlantic City, pp.118-125, Feb. 6-9, 1984.

[NG78] J. Nicolas and H.Gallaire; "Data Base: Theory vs. Interpretation," in Logic and Data Bases (H. Gallaire and J.Minker,eds.), Plenum Press, New York London, pp.34-54, 1978.

[R78] 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.

[Sh84] Y. Shoham; "Facts and Counterfacts in Temporal Reasoninng," Technical Report, Yale University, 1984.

[Sh85] Y. Shoham; "Notes on Temporal Reasoning," Technical Report, Yale University, Submitted to IJCAI85, 1985.

[S 81] D.W. Shipman; "The Functional Data Model and the Data Language DAPLEX, ACM TODS, Vol6, No.1, Mar. 1981.

[SM84] K. Sakai and T. Miyachi; "Incorporating Naive Negation into Prolog," Proceedings of Logic and Conference, Monash Univ. Jan. 1984.

[St80] G.L. Steele; "The Definition and Implementation of A Computer Programming Language Based on Constraint," MIT AI Lab. AI-TR-595, 1980.

[SS80] G. Sussman and G. Steele; "CONSTRAINTS — A Language for Expressing Almost- Hierarchical Descriptions," Artificial Intelligence 14, pp.1-39, 1980.

[W81] D.H. Warren; "Efficient Processing of Interractive Relational Database Queries Expressed in Logic," Proc. of VLDB, pp. 272-281, 1981.

m