数理解析研究所講究録
第 593 巻 1986 年 119-134

A Model-based Connection of KAUS to RDBMS

山内　平行　　　大須賀　節雄
H. YAMAUCHI and S. OHSUGA

Institute of Interdisciplinary Research
Faculty of Engineering, The University of Tokyo
4-6-1 Komaba Meguro-ku, Tokyo 153 Japan

ABSTRACT
We discuss about the connection of logical inference system KAUS to the existing conventional RDBMSs, where knowledge representation of views and integrity rules in S-model expressions, and P-models of user requests, which are generated from S-models in the knowledge base by the inference engine in KAUS are proposed for the purpose. A P-model is a semi-logical program model of user requests which is addressed to be transformed to target data sublanguages in remote RDBMSs or to be evaluated by the interpreter provided by the local database system. It is shown that the existing conventional relational databases are put into deductive use via S-models and P-models described in this paper. They provide us with vast classes of data definition and manipulation facilities including transitive closure operations.

1. Introduction

In the last decade since the Codd's proposal of the relational data model, the research and development in database fields have been greatly advanced. The study of deductive databases is one of them. A deductive database is a logic database which can make logical inference onrelations among facts [1]. It is now getting possible for us to implement a practical system of the deductive database according to the recent rapid progress in the computer hardware and software technology. Although the conventional relational database systems (RDBMS) are still in practical use, we think it would be more profitable that the existing conventional RDBMSs, which have no deductive power, could be put into the deductive use.

With this motivation, we aim to show that the conventional relational databases are put into the deductive use by conjoining logical inference system KAUS [15] to their management systems. More strictly speaking, we try to make the existing conventional RDBMSs turn to definite deductive database systems under the closed world assumptions (CWA) by conjoining KAUS to them. We remarks here that KAUS is a knowledge processing system based on multi-layer logic (MLL), which is an extended version of the first order predicate logic [13,14]. The name of the system is the abbreviation of 'Knowledge Acquisition and Utilization System'.

We conjoin KAUS to conventional RDBMSs through S-models in the knowledge base and P-models generated from S-models. A P-model is a semi-logical program model of user requests, and is transformable to the connected RDBMS language. S-models are user's views of the connected databases. The users may be any level of the end users, application programmers, system programmers, and database administrators.

Concerning with the data access of an existing relational database through a logical system, several approaches will be considered. One extreme of them is to centralize distributed data files by converting them to the unified data format and then to manage them only within the logical system. We think this approach is promising for the user of a personal small database. But this approach suffers from several difficulties when both of public and very large databases are utilized. Even if the requested data files could be converted, we cannot make effective use of most of the functions of their management systems. It is undesirable for us; even so from the general standpoint of effective use of existing softwares. On the other hand, there exists another approach which makes effective use of the existing RDBMS functions including query evaluations. It is a system by system connection, by which we can cover the disadvantages of the former. We would like to adopt this approach in this paper.

For the purpose, it must be first considered what operations and/or data should be responsible to each of the systems. S-models of the connected RDB described in the next section are solely responsible to the users of KAUS, and are written in KAUS knowledge representation language. They are used for KAUS to produce the semi-logical program models called P-models of database operations. The transformation of P-models to the programs in the data sublanguage (DDL/DML) supported by the connected RDBMSs may be done in either KAUS or RDBMS application programs. But we do not deal with the real compilation problem in the paper. Instead, we restrict our discussions to (1) knowledge representation for RDBMS connection, (2) P-model specification together with examples and (3) the method of generation of P-models, where we also give some illustrative examples.

2. Knowledge Representation in S-model

The figure 1 shows the overview of the connection between KAUS and RDBMSs. The translators in the figure are interface modules which translate P-models to the DDL/DML supported by the connected RDBMSs. As noted in the introduction, the details of these modules are not described hereafter. Only some examples of intuitive transformations of P-models to the host RDBMS language will be shown in the next section.

Aside from the translators, the main problems arising in the connection are that

(1). what knowledge should be described in the S-model,
(2). in what forms the S-model is represented,
(3). the specification of P-models, and
(4). how the inference engine derives a P-model from the S-model.

This section is concerned with (1) and (2).

Knowledge described in the S-model consist of the semantical data model of databases, user's views of the relations and derivation rules which define virtual or derived relations among base relations in the databases. Constraint rules are also included in the S-model. In the ANSI/SPARC terminology, the S-model contains schemas in the conceptual and external level of the connected databases.

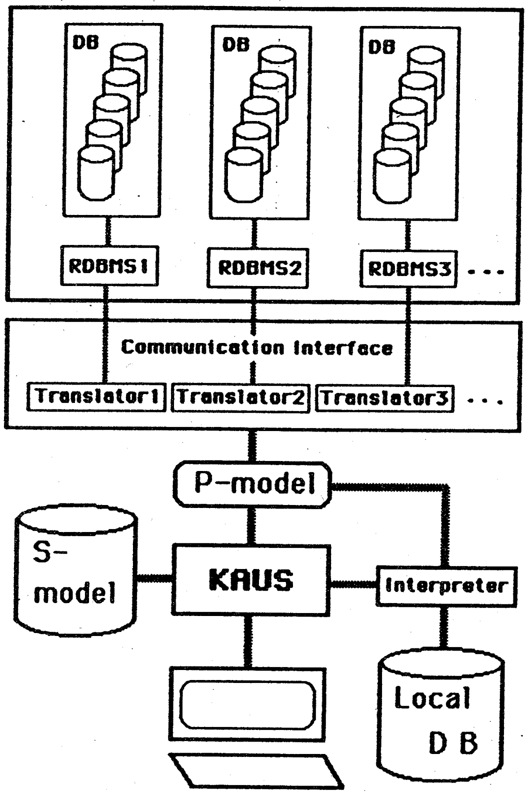


Fig.1. An overview of the connection

The S-model is written in KAUS knowledge representation language. Hereafter, we call our knowledge representation language KRL/KAUS. With relation to database applications, KRL/KAUS possesses the following three characteristics:

(a). It provides us with the data abstraction facilities such as generalization, aggregation and association of objects used at conceptual data modelling [2]. The abstraction hierarchies are specified by KRL/KAUS commands.
(b). All variables in well-formed formulas (wffs) in KRL/KAUS are typed variables, so that occur check is done automatically.
(c). Types can be also variables. This enables us to formulate hyperrelations as is the case in the DEDUCE-II [8], but also hierarchically aggregated data objects in relations.

Figure 2 shows an example of abstraction related to 'person'. This is represented by KRL/KAUS commands as follows.

```
!sk_e *person employee;                                  (2.1)
!sk_e *2employee rel_emp;                                (2.2)
!def_struct     employee = [ ps_name   : string,
                             emp_sal   : integer,
                          emp_mng   : string,
                          dept      : string
                        ];                              (2.3)
```

The first line (2.1) says that 'an employee is a person'. In other words, it means that 'employee' is generalized to 'person'. The second line (2.2) says that 'a rel_emp is a set of subsets of employees'. The succeeding last few lines (2.3) define the aggregation hiearachies of 'employee'. In the command line !def_struct, 'ps_name:string' denotes that the domain of the attribute ps_name is a string. Then, the rel_emp can be interpreted that it is a set of employee relations.

As seen in the above example, KRL/KAUS applies the set theory to represent abstraction hierarchies of attributes, domains and relations in the database definition. For example, a set of the all subsets of a given set, which is called a power set, is applied to define generalization and association hierarchies of objects. Then, the command (2.1), in which '*person' denotes the power set of persons, is interpreted that 'employee is an element of the power set of persons'. It is noted here that the word 'employee' written in the command line denotes the name of 'the set of employees'. Also, *2employee denotes the power set of the power set of employees. In database terminology, it is said that each element of a power set is an association of base elements which constructs the power set. (The more details of the syntax and semantics of KRL/KAUS are omitted here because it is not the purpose of this paper.).

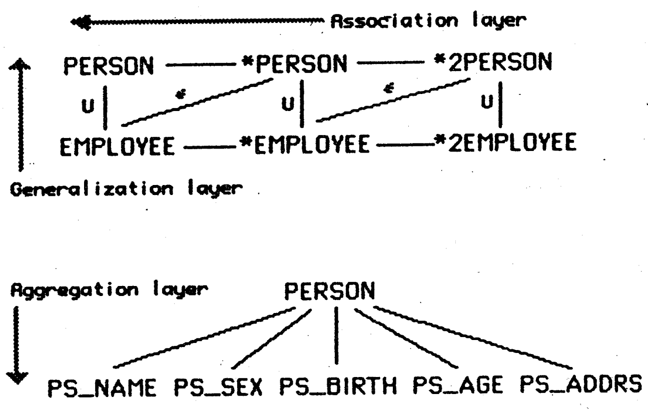


Fig.2. An example of abstraction

The role of abstraction of objects in the S-model is mainly concerned with type checking of objects at the inference stage. Another role of the S-model is to represent the user's views or constraint rules for database operations. They are written in logical formulas defined in KRL/KAUS. Together with abstraction of objects, they complete the S-model for data definition and data manipulation of objects. Let us consider, for example, the next relations totally defined in the connected RDBMS.

```
    EMP(NAME,SAL,MNG,DEPT)                              (2.4a)
    SALES(DEPT,ITEM,VOL)                                (2.4b)
    LOC(DEPT,FLOOR)                                     (2.4c)
```

Relation EMP describes the fact that employees have a salary, work for a manager who is also an employee and work in a department. Relation SALES says that a department sells items in certain volumes. Relation LOC says that a department is on a certain floor. If a user's view on relation EMP is that it is a relation which says that a certain employee works in a certain department, it is written as follows.

```
    [AX/person][AY/department]
    (| (work X Y) ~'(get_db #emp #name X #dept Y)).          (2.5)
```

The formula denotates the following meaning: To show that a person X works in a department Y, get the relation EMP and find a tuple where EMP.NAME is X and EMP.DEPT is Y. If it is found, it is known that X works in Y. The variables X and Y are appreciated here that they play a role of domain variables in relation EMP. Another KRL/KAUS expression which is semantically equivalent to the above formula is an expression with a tuple variable:

```
   [AX/#emp](work X:#name X:#dept).                           (2.6)
```

In the formula (2.6), the variable X plays a role of the tuple variable of relation EMP. Comparing with the first expression, it is more compact, but rather implicit or skolemized with respect to the arguments of the predicate work. Another point is that the first expression with domain variables is more rule like expression and the second with a tuple variable is more fact like expression. Due to our method of generating a P-model from the S-model, we adopt to express views by formulas which use only domain variables. Considering with a criteria for providing a natural language interface, the rule based expression seems to be more suitable and flexible for the purpose.

The above example is of less deductive use, because the relation work is nothing than a projection on NAME and DEPT of the base relation EMP. The next example which defines a relation co-worker is of more deductive use.

```
   [AX,Y/person][AZ/department]
   (| (coWorker X Y) ~(work X Z) ~(work Y Z) ~'(ne X Y)).   (2.7)
```

This says that if X and Y work in the same department Z, they are co-workers with each other.

On the other hand, aggregate operations may appear frequently to evaluate queries. For example, consider that we want to get the average salary of employees in a certain department. Then, S-model representation becomes as follows.

```
   [ADept/department][AX/integer]
   [AEmp/person][ASals/*integer][ESal/Sals]
   (| (avg_salary Dept X)
          ~(work Emp Dept)
          ~(earn Emp Sal)
          ~'(average Sals X)
   ).                                                         (2.8)
```

We can observe here that the variable Sals is a set type each of which elements is an integer Sal satisfying (earn Emp Sal). This suggests that a grouping operation would be necessary at the database manipulation.

The last example of a view expression is related to a recursive relation. Assume that we think that a manager who manages a manager of a certain employee is also a manager of the employee. This is written as follows.

```
    [AX,Y/person]
    (| (manager X Y)  ~'(get_db #emp #name Y #mng X)).       (2.9a)
    [AX,Y,Z/person]
    (| (manager X Z)  ~(manager X Y)  ~(manager Y Z)).       (2.9b)
```

In the conventional database systems, such a recursive relation would be ruled out from views because it might fall into an infinite loop in the query evaluator unless it is treated well with considerable attention. Recently, the recursion problem in database fields is being investigated by several authors[3,6]. Using connection graphs is one approach to solve the problem [3,6]. But we take a different approach, where we embed the meta-predicate concerning with recursion in the given view description. Then, the (2.9b) is corrected to

```
    [AX,Y,Z/person]
    (| (manager X Z)
         ~'(recursion manager)
          ~(manager X Y) ~(manager Y Z)).                   (2.9c)
```

This is read that if the predicate manager is treated as a recursive predicate, and if X is a manager of Y and Y is a manager of Z, then X is a manager of Z. Such a meta-level information will be utilized for generating P-models from the S-model. The subject is to be discussed in the section 4.

In addition to view modelling, constraint modelling is also a significant part of the S-model construction. As is well known, the valid database state is maintained by imposing integrity constraints on the database update. Type constraints and data dependencies constraints are among them. Similar to view modelling, these constraint rules can be described by the wffs in KRL/KAUS. As described earlier, type constraints are self-contained in the quantification part of wffs in KRL/KAUS. The body of the followed definite clause describes conditions that must be obeyed before or after the database modification is performed. For example, assume that relation FATHER(PS1,PS2), where PS1 is the father of PS2, is defined in the database. We constrain relation FATHER to obey the following conditions.

(C1). The sex of PS1 is male.
(C2). The PS2's father is unique.
(C3). Relation FATHER is asymmetric.
(C4). Relation FATHER is not cyclic.

Then, these constraints would be described as follows.

```
     [AX/male][AY/person]
     (| (validated #rel_father X Y)
           ~(cond c2 X Y)
           ~(cond c3 X Y)
           ~(cond c4 X Y)
     ).                                             (2.10)
```

This rule can be addressed to be evaluated by the With statement in a P-model (see the next section). The other constraint rules as well, for example, that of access right can be described in the similar fashion.

3. P-model specification

The S-model described in the previous section has exclusively concerned with view expressions in the semantical level of the connected databases. In terms of deductive databases, it defines the proper axioms (non-logical axioms) of the connected databases. P-models described here are derived from the given S-model. They present rather operational aspects of the user's requests. Therefore, a P-model is said to be a semi-logical program model of a user's request. The P-model constructs are designed so that it satisfies the following requirements.

(a). Expressiveness for vast classes of user requests.
(b). Transformabilityto various target DSL and embedded DSL.
(c). Readability.

With respect to (a), P-model should be able to express requests for negated facts, computations of aggregates, transitive closures, data definitions, database update, and so forth, as well as ordinary requests.

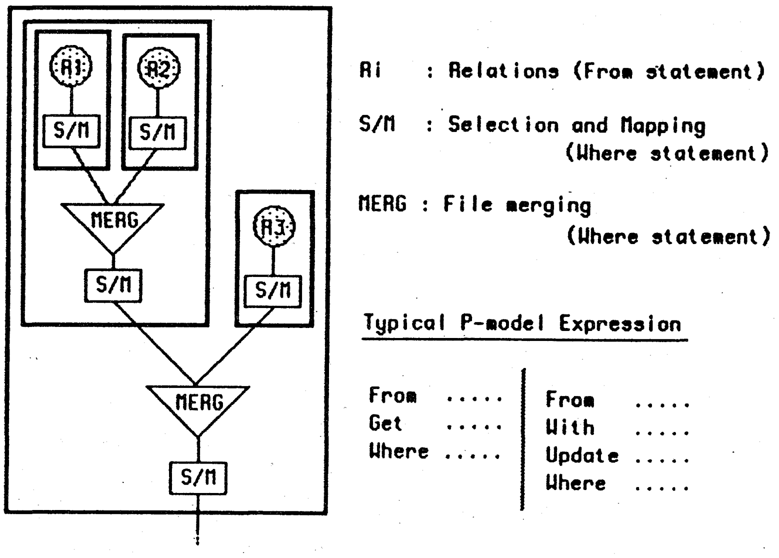

R1
R2
R3
S/M
MERG
Ri : Relations (From statement)
S/M : Selection and Mapping (Where statement)
MERG : File merging (Where statement)
Typical P-model Expression
From .....
Get .....
Where .....
From .....
With .....
Update .....
Where .....

Fig.3. A partial graphical representation of a P-model

Considering (b), the existing relational data sublanguages are classified into two types; ones based on relational algebra and the others based on relational calculus. Relational calculus is also subdivided into tuple calculus and domain calculus. SQL in System-R and QUEL in INGRES are typically tuple calculus oriented; QBE in

Query-by-Example is typically domain calculus oriented. Our P-model is then required to be transformable to each of the types. We note here that the P-model expressions seem to be a hybrid of the above types, that is, that possesses both of algebraic and relational calculus features.

Figure 3 shows partial graphical representation of P-models described hereon. According to the figure, a P-model description is a semi-logical program model of a user's request, which describes his or her goal with a goal command preceded by a From statement and followed by a Where statement. The set of goal commands consist of Get, Insert, Delete, Update, Define and Create. They denote the classification of offered queries. A From statement describes the used relations to attain the goal. The Where statement describes conditions (restrictions) to attain the goal. As the Where statement takes a leading part in P-model constructs, we give the details of it here. The general form of the Where statement is as follows.

```
   Where   [Q1V1/D1][Q2V2/D2]...[QnVn/Dn]
           (C  R1[S1/M1]<V11,...,V1i>,
               R2[S2/M2]<V21,...,V2j>)[S3/M3]<V31,...,V3k>    (3.1)
```

Where, [QiVi/Di] denotes that the variable Vi is quantified by Qi (Qi=A or E) and its domain is Di. C denotes the file merge type, which is either one of &(AND-merge),|(OR-merge), and *(RECURSIVE-merge). The R1 and R2 are relational file names to be merged. [S1/M1] and [S2/M2] denote selection and/or arithmetic calculation related only to relation R1 and R2 respectively. [S3/M3] denotes selection and/or arithmetic calculation to be done in the C-merged relation. <...> denotes a projecting list; for example, <V11,...,V1i> denotes a projecting list onto which R1[S1/M1] is projected, and <V31,...,V3k> is a projecting list for the C-merged relation (C R1... , R2...) under restriction [S3/M3]. It is remarked here that all of the components of the projecting lists must be declared in the quantification part, and that because a merged file is also a relation, (C ....) constructs a nested form.

In the sequel, we give some examples for the illustrative purpose. Rather the strict notation of P-model constructs will be seen in the appendix.

<u>Example 3.1.</u> The following is the P-model expression of the query to find the items sold by at least two departments on the second floor.

```
        From   sales(dept,item), loc(dept,floor)
        Get    item
        Where  [EX/item][EYY/*dept][AY/YY]
               (& sales[]<Y,X>,
                  loc[floor='2']<Y>)[count(YY)>=2]<X>          (3.2)
```

The quantification [EX/item][EYY/*dept][AY/YY] is interpreted in the generative mode; that means that, for a given item X, a subset YY is generated from the set of departments under the condition that each element of YY satisfies the succeeding formula (& sales[]<Y,X>, loc[floor='2']<Y>). Then, the items X associated with count(YY)>=2 are selected. The followings are the semantically equivalent (in the sense of denotation) SQL, QUEL and QBE expressions which would be obtained from the above P-model.

```
(SQL)    SELECT   ITEM
         FROM     SALES
         WHERE    DEPT  IN
                  (SELECT DEPT FROM LOC WHERE FLOOR='2')
         GROUP BY ITEM
         HAVING   COUNT(DEPT) >= 2                        (3.3)

(QUEL)   RANGE OF S IS SALES
         RANGE OF L IS LOC
         RETRIEVE (S.ITEM)
         WHERE COUNT(S.DEPT BY S.ITEM WHERE S.DEPT=L.DEPT
                     AND L.FLOOR='2') >= 2                (3.4)

(QBE)    LOC(DEPT:dpt, FLOOR:'2')
         SALES(DEPT:cnt.all.dpt>=2, ITEM:p)               (3.5)
```

Example 3.2. The example is related to obtaining the transitive closure of given relations. In the section 2, we have discussed about the recursive relation, where we have considered that a manager who manages a manager of a certain employee is also as a manager of the employee. If we inquire that who are all managers of Anderson with this view, its P-model becomes as follows. (see details in the next section).

```
         From    emp(name,mng)
         Get     mng
         Where   [EXX/*mng][EY/XX][EZ/XX]
                 (* emp[name='Anderson']<Y>,
                    emp[name=Y]<Z>)[]<XX>                 (3.6)
```

Explaining shortly, the Where statement declares that the required managers are obtained by applying the usual transitive definition R such that xRy ^ yRz --> xRz; in the above case, it is read that if x is a manager of y and y is a manager of z, then x is a manager of z. Then, given z='Anderson', his managers are obtained to the set of all x and y each of which satisfies the transitive definition. The extended file merging connective * in the Where statement specifies the recursive join between relation EMP and itself.

Unfortunately, most of the existing data sublanguages do not allow transitive view definitions of relations. So that, it may be unable to obtain the direct transformation of the transitive P-model to the target data sublanguage. But in the alternative, we will be able to compile it to the host language. As for the transitive closure, many authors have proposed solutions of such a compilation problem, though they might not be general solutions [3,4,6].

<u>Example 3.3.</u> A P-model is also expressive of the request for updating database. For example, the following is a P-model which means that Anderson's salary is increased by the amount of 10000.

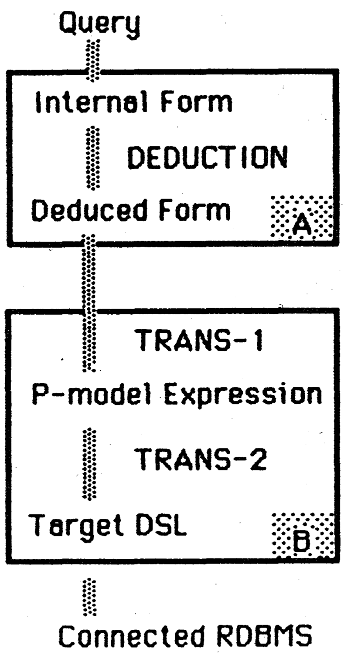
```
From      emp(name,sal)
With      [EX/sal]emp[name='Anderson]<X>
Update    sal
where     [EY/sal]
          emp[name='Anderson', Y=X+10000]<>              (3.7)
```

Usually, a RDBMS performs constraint checking before the actual update is done. Constraint checking is very important to maintain the database to be consistent. We provide constraint rules in the S-model, and they can be applied to consistency checking of the database. Associated with this, the P-model constructs may include a With statement before the modification statement, that specifies constraint rules applied to the succeeding statements. The With statement in (3.7) says that Anderson's current salary should be used before updating.

4. Generation of P-model from S-model

We have described, in the section 2, that the S-model represents hierarchical abstraction of domains and attributes of data objects, user's views for applications and constraint rules for database update. They are general and domain specific knowledge concerning with the connected databases, and they constitute the knowledge base of KAUS. They are used for derivation of P-models of user's requests under the control of the inference engine in KAUS.

Figure 4 shows the total steps by which user's requests are sent to the connected RDBMS. In the figure, deduction and transformation-1 are processes related to P-model generation. The deduction process is a proof procedure, which is similar to that in the Robinson's resolution principle [9]. First, a query in KRL/KAUS is transformed to the internal query tree, and then the deduction tree is generated from the query tree by the deductive proof procedure. After the termination of deduction, all the leaf nodes of the deduction tree has been replaced by the special nodes called procedural type atoms (PTAs for short). A PTA is a primitive evaluated by the special procedure associated with it. For



<u>Fig.4.</u> Query transformation

example, '(add Z X Y) is a PTA, whose truth value is determined by evaluating the relation that X plus Y is equal to Z. The predicate get_db presented in the section 2 is also a PTA. After all, the deduction tree thus obtained can be interpreted as a program model of the given query. Then, a P-model expression is a translation of the deduction tree and is semantically equivalent to it.

Before explaining the deduction and transformation process, we must supplement here that, the quantification part of wffs in KRL/KAUS usually represent not only internal data type constraints of variables but also their semantical data type constraints. For example, [AX/person] constrains X to be a person. In comparison with variables, the real instances of data objects in databases are usually represented according to the internal schema, and they are internally either of strings, integers, and floats. For our purpose of database connections, these syntactical type constraints are bridged over the semantical type constraints in somewhat tricky way in KRL/KAUS. For example, by the command '!sk_e *person string;', instances of 'person' are declared that they are represented in strings.

In the following, we illustrate the deduction process and transformation process by some examples. In the examples, notations of wffs in KRL/KAUS are slightly modefied for the purpose of comprehension. Queries in natural language and their corresponding KRL/KAUS expressions are also illustrated. Let us begin with a simple example.

<u>Example 4.1.</u> Assume that a view (2.5) is given in the S-model and a query, 'in what department Anderson works?', is supposed:

```
  View    : [AX/person][AY/department]                            (4.1a)
            (work X Y) <-- '(get_db #emp #name X #dept Y)         (4.1b)

  Query   : In what department Anderson works?
            [EX?/department]                                      (4.2a)
            (work "Anderson" X)                                   (4.2b)
```

First, by the unification algorithm peculiarly tailored for KAUS, corresponding arguments of the predicate work of (4.1b) and (4.2b) are examined with type constraints (4.1a) and (4.2a). As no violation occurs in this case, the arguments and their types are unified respectively, and (4.2a) and (4.2b) are then replaced by

```
            [EX?/department]                                      (4.3a)
            '(get_db #emp #name "Anderson" #dept X)               (4.3b)
```

Any more deduction process does not need to proceed in this state because the leaf nodes of the deduction tree represented by (4. 3a) and (4.3b) consist of only one PTA. Next, the transformation process begins as follows: From (4.3a), it is recognized that the value of X is addressed to output. From (4.3b), it is also recognized that X is found in the column #dept in relation #emp and that #name is another column data related to X. Then, 'From emp(name,dept)' and 'Get dept' statements are obtained. Again from (4.3b), it is known that #name is restricted to "Anderson". This fact contributes to construct the selection expression [name='Anderson'] in the Where statement. As the domain of X is complied to be expressed with the real attribute name in the relation, the quantification part of the Where statement becomes [EX/dept]. The notion of projection in the Where statement is <X> because the value X is required to output. Consequently, the generated P-model is

```
  P-model: From emp(name,dept)
           Get dept
           Where [EX/dept]emp[name='Anderson']<X>            (4.4)
```

Example 4.2. Assume the following views sell and totalSal saying that the total salary Y in the department X is obtained from relation EMP by summing up N to Y, and a query in (4.7).

```
  View: [AX/department][AY/item]
        (sell X Y) <-- '(get_db #sales #dept X #item Y)      (4.5)
        [AX/department][AIntSet/*salary][EN/IntSet]
        [AY/integer]
        (totalSal X Y) <-- '(get_db #emp #sal N #dept X) ^
                           '(sum IntSet Y)                   (4.6)
```

Query: Among all departments with total salary greater than 10000, find the departments which sell dresses:

```
         [EXdepts/*department][AX?/Xdepts][EY/integer]
         (totalSal X Y) ^ Y>10000 ^ (sell X "DRESS")         (4.7)
```

For obtaining the P-model expression, we first reduce (totalSal X Y) in (4.7) to the premise of (4.6), and (sell X "DRESS") in (4.7) to the premise of (4.5) in the deduction process. Then, the process is terminated with the following result:

```
        [EX?/department][EIntSet/*salary][AN/IntSet][EY/integer]
        '(get_db #emp #sal N #dept X) ^
        '(sum IntSet Y) ^  Y>10000 ^
        '(get_db #sales #dept X #item "DRESS")               (4.8)
```

It should be noted here that the order of quantifications of variables in (4.8) is significant. They are obtained from the unification rule provided by KAUS, and contribute to keep the correct meaning of the original query. The next is to transform (4.8) to the P-model expression: The Get statement and From statement concerning with (4.8) are easily obtained in the same way described in the example 4.1. Then, we go ahead to create the Where statement from (4.8). We first remember that all PTAs are AND-connected. Next, we examine the arguments of the first PTA get_db in (4.8). As it is known that they are not locally restricted by the other PTAs, emp[ ]<N,X> is obtained. Then, the next predicate get_db is searched for in (4.8). As we found it, we transform it to sales[item='DRESS']. The remaining PTAs '(sum

IntSet Y) and Y>10000 are understood that they do not contribute to local restrictions on both get_db PTAs but contribute to the restriction on their merged relation. They are simplified to sum(IntSet)>10000. Consequently, we obtain the following P-model expression of (4.7).

```
   From  emp(sal,dept), sales(dept,item)
   Get   dept
   Where [EX/dept][EIntSet/*sal][AN/IntSet]
         (& emp[]<N,X>,
            sales[item='DRESS']<X>)[sum(IntSet)>10000]<X>  (4.9)
```

Example 4.3. Generating the P-model of a query (4.11) asked for a negated fact is exhibited. Assume the same views sell in the example 4.2 and a view floorOf in (4.10).

```
View :  [AX/department][AY/location]
        (floorOf X Y) <-- '(get_db #loc #dept X #floor Y)  (4.10)
```

Query: Find the departments and their items sold by no department on the second floor.

```
        [EX?/department][EY?/item][AZ/department]
        (sell X Y) ^ ~(sell Z Y) ^ (floorOf Z 2)          (4.11)
```

The deduction process on this query is complicated for the reason that there exist three literals to be resolved and among them, (sell Z Y) is negated. We note here that the negation as failure in CWA [10] is applied during the whole process of generation of the P-model. Then, (4.11) is deduced to the following wff.

```
           [EX?/department][EY?/item][AZ/department]
           '(get_db #sales #dept X #item Y) ^
          ~'(get_db #sales #dept Z #item Y) ^
           '(get_db #loc #dept Z #floor 2)                (4.12)
```

In the transformation process, we are complied to select a couple of relations from (4.12) which is claimed to be merged in the first. We can find that the couple is the relation SALES in the second PTA and relation LOC in the third PTA, because the fifth argument of the third PTA of get_db is constant and its third one is common to the second PTA of get_db. The priority of selecting a couple of relations to be merged relates to the optimization of query evaluations. In consequence, we obtain the P-model of the query (4.12) as follows.

```
    From  sales(dept,item), loc(dept,floor)
    Get   dept,item
    where [EX/dept][EY/item][AZ/dept]
          (& (& ~sales[]<Z,Y>,
                 loc[floor='2']<Z>)[]<Y>,
             sales[]<X,Y>)[]<X,Y>                        (4.13)
```

Example 4.4. The recursive P-model is generated in this example. Assume a couple of views (2.9a) and (2.9c) related to the transitivity of managers, where we have thought that a manager who manages a manager of a certain employee is also a manager of the employee. Then, with this view, the next query can be supposed.

```
    Query: Find all managers of Anderson.
           [EXX/*person][AX?/XX](manager  X "Anderson")     (4.14)
```

As noted in the previous section, a view expression of a recursive relation is complied to declare explicitly with a meta-level predicate that the view is recursively defined. In (2. 9c), '(recursion manager) is a meta-level predicate that says that the view is recursive, and should be evaluated by the special functional module (demon) in the deductive program so that the deductive process does not fall into an infinite loop. Then, the query (4.14) is reduced by the deductive process to the following deduced form.

```
        [EXX/*person][AX?/XX]
        '(get_db #emp #name "Anderson" #mng X) v
        { [EXX/*person][AX?/XX][EY/person]
          '(get_db #emp #name Y #mng X) ^
          '(get_db #emp #name "Anderson" #mng Y) }*         (4.15)
```

We note here that (4.15) is equivalent to a regular expression b*a, where b*a = a U ba U bba U bbba U ..., and a = '(get_db #emp #name "Anderson" #mng Y), b = '(get_db #emp #name Y #mng X), bb = '(get_db #emp #name Y #mng X) ^ '(get_db #emp #name X #mng XX), bbb= '(get_db #emp #name Y #mng X) ^ '(get_db #emp #name X #mng XX) ^ '(get_db #emp #name XX #mng XXX), and so on. Then, the P-model expression (3.6) of the query (4.14) is obtained from (4. 15), and the required managers are to be obtained in the set XX, where each elements satisfies (4.15), namely, (3.6).

<u>Example 4.5.</u> We show that a P-model describing an update operation is also obtained from a user's request and S-model expressions in the knowledge base. Suppose the following query:

```
        Query: Increase Anderson's salary by the amount of 10000.
               (increase salary "Anderson" 10000)           (4.16)
```

To obtain the P-model, we need the following S-model expression.

```
   [AXattr/attr][AOfS/object][AByN/number][AXval/number]
   (increase Xattr OfS ByN)
      <-- (cond Xattr OfS Xval) ^ '(update Xval added ByN) (4.17)
```

This S-model expression is very general, and it would be available for any queries asked for such updating that 'the value(Xval) of an attribute(Xattr) of an object(OfS) is increased by the amount of n(ByN)'. In (4.17), the PTA update is a meta-level predicate, and (4.17) is read that, to increase the value of Xattr of the object OfS by ByN, add ByN to Xval under the condition (Xattr OfS Xval) is currently true. Then, the P-model (3.7) of (4.16) is obtained from

```
[EXval/number]
'(get_db #emp #name "Anderson" #sal Xval) ^
'(update Xval added 10000)                               (4.18)
```

5. Concluding Remarks

We have discussed about the model based connection to conventional RDBMSs so as to make it possibble that they are put into deductive use via logical inference system KAUS. Knowledge representation in the S-model and P-model expressions for user requests provide us with the general facilities concerning with flexible view expressions, integrity checking, and vast classes of data manipulations including transitive closure operations.

It is also expected that the S-model representation would be suitable for the bridge between natural language input and deductive access to the conventional databases. One more point is that views and integrity rules in engineering database systems as well as those in social economy could be expressible in KRL/KAUS and then put into deductive use. We have not exhibited how it can be done and several other details for the lack of spaces.

APPENDIX. SYNTAX OF P-MODEL EXPRESSION

```
P-model-expr     ::=  dm-expr | dd-expr
dm-expr          ::=  from-stmt dm-goal-stmt where-stmt
                    | from-stmt with-stmt dm-goal-stmt where-stmt
dd-expr          ::=  Create relation-name ( active-col-list )
                    | Define relation-name ( active-col-list )
From-stmt        ::=  From R-list
with-stmt        ::=  With cond-expr
dm-goal-stmt     ::=  dm-goal-name arg-list
where-stmt       ::=  Where cond-expr
R-list           ::=  relation-name ( active-col-list )
                    | relation-name ( active-col-list ) , R-list
relation-name    ::=  defined-name-in-the-connected-RDBMS
active-col-list  ::=  column-name
                    | column-name , active-col-list
column-name      ::=  defined-name-in-the-connected-RDBMS
cond-expr        ::=  quantification body-expr
quantification   ::=  [ quantifier var-name / domain ]
                    | [quantifier var-name/ domain ]
                      quantification
quantifier       ::=  A | E
var-name         ::=  string-beginning-with-upper-case-letter
domain           ::=  column-name | powered-col-name | var-name
powered-col-name::=   *n column-name  ( n: integer > 0 or omitted)
arg-list         ::=  column-name | function
                    | column-name , arg-list | function , arg-list
function         ::=  function-name ( active-col-list )
body-expr        ::=  ( merge-type file , file ) [ restriction ]
                                                 < proj-list   >
```

```
merge-type      ::=   & | | | *
file            ::=   relation-name [ restriction ] < proj-list >
                    | body-expr
restriction     ::=   boolean-expr | arithmetic-expr
                    | boolean-expr , restriction
                    | arithmetic-expr , restriction
proj-list       ::=   var-name | var-name , proj-list
dm-goal-name    ::=  Get | Insert | Update | Delete
```

REFERENCES

1. GALLAIRE,H., MINKER,J., and NICOLAS,J., Logic and DAtabases: A Deductive Approach, Computing Survey, Vol.16,No.2,1984.
2. BRODIE,M.L., MYLOPOULOS,J., and SCHMIDT,J.W. (ed.), On Conceptual Modelling, (Springer-Verlag), 1984.
3. HENSCHEN,L.J., and NAQVI,S.A. On compiling queries in recursive first-order databases. J.ACM 31, 1984, 47-85.
4. ULLMAN,J.D. Implementation of Logical Query Languages for Databases. Trans. ACM 10,3. 1985, 289-321.
5. SCHMIDT,J.W., and BRODIE,M.L. (eds.) Relational Database Systems. Springer-Verlag, 1983.
6. CHANG,C.L. On Evaluation of Queries Containing Derived Relations In a Relational Data Base. In Advances in Data Base Theory,vol.1. Gallaire,H. andMinker,J., and Nicolas, J.M. (Eds.). Plenum Press, New York, 1981, 235-260.
7. CHANG,C.L., and SLAGLE,J.R. Using Rewriting Rules for Connection Graphs to Prove Theorems. Artificial Intelligence 12, North-Holland, 1979, 159-180.
8. CHANG,C.L., A Hyperrelational Model of Data Bases, IBM Research , 1975.
9. CHANG,C.L., and LEE,R.C.T., Symbolic Logic and Mechanical Theorem Proving, (Academic Press), 1973.
10. REITER, R., Deductive Question-Answering on Relational Data Bases, (in Gallaire,H., et.al, Logic and Database, Prenum Press), 1978.
11. OHSUGA, S., A New Method of Model Description --- Use of Knowledge Base and Inference, Proc. of IFIP W.G.5.2. Working Conf. on 'CAD system Framework', (North-Holland, 1983).
12. OHSUGA, S., Knowledge Based Man-Machine Systems, Automatica, vol.19, No.6 (1983), pp.685-691.
13. OHSUGA, S., Predicate Logic Involving Data Structure as a Knowledge Representation Language, Proc. 1983 8th IJCAI., (1983), pp.391-394.
14. OHSUGA, S., and YAMAUCHI H., Multi-Layer Logic - A Predicate Logic Including Data Structure as Knowledge Representation Language, NEW GENERATION COMPUTING 3, Ohm-sha Ltd., 1985.
15. YAMAUCHI, H., and OHSUGA, S., KAUS as a tool for Model Building and Evaluation, Proc. 5th International Workshop on Expert Systems and Their Applications, Agence de l'Informatique, Avignon, France, 1985.
16. YAMAUCHI, H., Processing of Syntax and Semantics of Natural Language by Predicate Logic, COLING 80, The International Conference on Computational Linguistics, 1980.