ON DEDUCTIVE DATABASES WITH INCOMPLETE INFORMATION

QINZHENG KONG AND GRAHAM CHEN

***Abstract*—In order to extend the ability to handle incomplete information in a definite deductive database, a Horn clause based system representing incomplete information as incomplete constants is proposed. By using the notion of incomplete constants the deductive database system handles incomplete information in the form of sets of possible values, thereby giving more information than null values. The resulting system extends Horn logic to express a restricted form of indefiniteness. Although a deductive database with this kind of incomplete information is, in fact, a subset of an indefinite deductive database system, it represents indefiniteness in terms of value incompleteness and therefore it can make use of the existing Horn logic computation rules. The inference rules for such a system are presented, its model theory discussed and an indefinite model theory proposed. The indefinite model theory is consistent with minimal model theory and extends its expressive power.**

Categories and Subject Descriptors*—H.1.1 [Models and Principles]: Systems and Information Theory—*information theory*; H.2.4 [Database Management]: Systems—*query processing*; I.2.3 [Artificial Intelligence]: Deduction and Theorem Proving—*logic programming

***General Terms*—Design, Management, Theory**

***Additional Keywords and Phrases*—Deductive databases, incomplete information, Horn clause, query evaluation, Prolog**

***Source of Publication*—ACM Transactions on Information Systems, Vol. 13, No. 3, July 1995, pp 354-369.**

1 INTRODUCTION

Different information systems model the real world information and knowledge using different abstract frameworks. A deductive database is one of such information system that models the information by using a subset of the first order logic and provides the storage, manipulation and retrieval of the information by using the deductive capabilities offered by the first order logic system. However, most real world information and knowledge are not presented as the precise values, rather they are often presented as the missing, incomplete, uncertain values. Whereas such incomplete information existed in all types of information systems and has been studied extensively [3,5,9,10,16,18,24], this paper concentrates on the problems existing in deductive database systems.

A deductive database is a database containing both explicit and implicit facts. It provides the facilities to define implicit data in terms of general rules and the deductive ability to reason with the database [1,2,11,19,20]. First order logic can be used as the theoretical foundation for deductive databases [12,13].

Usually, a deductive database takes one of the following two formats as the general format of its rules. One is the full clausal form which is equivalent to first order predicate calculus, the other is a restricted form, known as the Horn clause. In a Horn clause system each clause is a definite clause which contains only one positive literal. A database which consists entirely of definite clauses is referred to as a definite deductive database, while a database containing non-Horn clauses is referred to as an indefinite deductive database.

The general non-Horn system is much more powerful than the definite deductive system. However, one of the difficulties that the deductive database community has encountered is the efficient implementation of systems which employ the full first order logic format. It seems that this difficulty cannot be overcome without applying restrictions to the language ability. This is due to the nature of the complexity problem of first order logic theorem proving in general. Hence one crucial decision to be taken in designing a deductive database system is the choice of format in which the clauses are to be stored. On the one hand full clausal form is a powerful representation but, as agreed by many researchers, the general case of non-Horn clause is undesirable since it adds substantially to the complexity of deductive operations in a database [5,20,23]. On the other hand, the Horn clause representation is more desirable since the satisfiability in a Horn clause system is decidable and query evaluation process can be constructed efficiently. However, a simple Horn clause system has no power to handle indefinite information which commonly occurs in a database system.

This leads to the question of whether there is any alternative form between full clausal form and Horn clausal form which can be accepted as the basic format for general rules in a deductive system. Alternatively,

is there some other way to represent a degree of basic indefinite information in a deductive database while adding only a little to the overall system complexity?

One approach would be to adopt the way of handling incomplete information in relational databases. Lipski proposed a mathematical model of an incomplete information system [9,10]. If the concept of incomplete information in relational databases can be introduced into a deductive database, the problem of a special indefiniteness can be confined to the problem of value incompleteness. In other words, instead of introducing the general form of disjunction into a deductive database to handle the indefinite problem, a new data type, called *an incomplete constant*, can be introduced into a Horn clause system to handle a degree of indefiniteness. For example, the piece of information that Tom's age is either 20 or 21 although it is not known which can be represented as a disjunction:

$$age(tom, 20) \vee age(tom, 21)$$

This can also be viewed as saying that the value of Tom's age is either 20 or 21. By using a *set* notation, the value *"either 20 or 21"* can be represented as an incomplete constant of the form $\{20, 21\}$. The above disjunctive clause can be represented as a Horn clause with an incomplete constant as:

$$age(tom, \{20, 21\})$$

Such an incomplete constant can occur in both positive and negative literals just as a complete constant can. A clause containing an incomplete constant can be used to represent disjunctive information in terms of data, that is, each disjunction in a disjunctive clause has the same predicate name and arity. The restriction imposed on the format of the clause is necessary since it is expected that this will reduce the computational complexity inherent in the satisfiability problem of first order logic.

2 THE INTERPRETATION OF INCOMPLETE INFORMATION

The introduction of incomplete information into deductive databases leads to many new problems which may not arise in the case of the relational model. The first is the problem of interpretation. Suppose a ground clause containing incomplete constant has the form:

$$p(\{a,b\})$$

which stands for either $p(a)$ or $p(b)$ or both hold in the database. It can be expressed as:

$$\exists_{U \in \{a,b\}} p(U)$$

which is equivalent to

$$p(a) \vee p(b)$$

However, in defining deductive rules, the concept of variables is used. A basic problem occurs with a clause containing both variables and incomplete constants. For example, a clause of the form $p(X, \{a,b\}) \leftarrow q(X)$ may have two possible interpretations. One is to put the universal quantifier in front of the existential quantifier, that is, the clause is interpreted as:

$$\forall_X \exists_{A \in \{a,b\}} p(X,A) \leftarrow q(X)$$

which is equivalent to

$$p(X,a) \vee p(X,b) \leftarrow q(X)$$

The other is to put the quantifiers in the reverse order, so that the clause is interpreted as:

$$\exists_{A \in \{a,b\}} \forall_X p(X,A) \leftarrow q(X)$$

which is equivalent to

$$(\forall_X p(X, a) \leftarrow q(X)) \vee (\forall_X p(X, b) \leftarrow q(X)) \Leftrightarrow (p(X, a) \leftarrow q(X)) \vee (p(Y, b) \leftarrow q(Y))$$

here, the concept of variable re-name is introduced to keep the clause under the full clausal form. When the number of incomplete constants and the number of variables in a clause increases, the number of different interpretations increases dramatically. These differences are caused by the different orders of the quantifiers.

Since it is not the intention to handle full first order logic which would be too expensive, but instead to investigate a practical way to handle some type of incompleteness in the Horn clause framework, two special cases of incomplete information are particularly interesting. One arises from the assumption that if a clause contains both incomplete data and variables it has the interpretation that all universal quantifiers precede the existential quantifiers. For example, a clause with the form

$$p(X, \{a,b\}, Y, \{c,d\}) \leftarrow q(X, Y)$$

is interpreted as:

$$\forall_X \forall_Y \exists_{A \in \{a,b\}} \exists_{B \in \{c,d\}} p(X,A,Y,B) \leftarrow q(X,Y)$$

This is referred to as *AE* notation.

The other arises from the assumption that for any number of incomplete constants and variables occurring in a clause, all existential quantifiers precede universal quantifiers. For example, a clause which has the form

$$p(X,\{a,b\},Y,\{c,d\}) \leftarrow q(X,Y)$$

is interpreted as:

$$\exists_{A \in \{a,b\}} \exists_{B \in \{c,d\}} \forall_X \forall_Y p(X,A,Y,B) \leftarrow q(X,Y)$$

This is referred to as *EA* notation.

By comparing these two notations [6,8], it is realised that both the *AE* notion and the *EA* notion of incomplete information introduce some indefinite features into a definite database though neither can cover the whole task of handling incomplete information. The differences between these two notions are:

1. A clause under the *AE* notation can be directly rewritten as a simple disjunction, while under the *EA* notation the concept of variable re-naming is introduced to keep the system under the scope of clausal form. This may increase the complexity and cost of the system.
2. The *AE* interpretation provides a simple way for the database user to represent incomplete values, while the *EA* interpretation provides a way to express a property, which is not known exactly, for a system.

The following examples illustrate the differences of the two interpretations.

Example 1: Consider a clause:

$$blood\_type(Child, \{a,o\}) \leftarrow father\_blood\_type(Child, a) \wedge mother\_blood\_type(Child, o)$$

that specifies that if the father has blood type *A* and the mother has blood type *O* then the blood type of the child might be either *A* or *O*. This rule has the *AE* interpretation, i.e.:

$$\forall_{Child} \exists_{B \in \{a, o\}} blood\_type(Child, B) \leftarrow father\_blood\_type(Child, a) \wedge mother\_blood\_type(Child, o)$$

Example 2: Consider a clause:

$$teaches(\{tom, peter\}, Student) \leftarrow register(Student, c101)$$

that specifies that either Tom or Peter teaches all students registered with the class c101. This rule has the *EA* interpretation, i.e.:

$$\exists_{Teacher \in \{tom, peter\}} \forall_{Student} teaches(Teacher, Student) \leftarrow register(Student, c101)$$

In most database applications, the requirement of the user for incompleteness is mainly at the attribute value level rather than at the system level. That is, users are mainly interested in how to represent a value of a particular attribute when the exact value is not known, rather than in how to represent a number of alternative system properties, and the *AE* notation provides a simple way to represent an incomplete attribute value, it is the better choice for the type of the incomplete information to be handled.

In the following sections, when incomplete information is referred to it is assumed to conform to the *AE* interpretation, that is, a clause with the form

$$p(X, \{a,b\}) \leftarrow q(X, \{c,d\})$$

has the interpretation

$$\forall_X \exists_{A \in \{a,b\}} p(X,A) \leftarrow \exists_{B \in \{c,d\}} q(X,B)$$

which is equivalent to

$$p(X,a) \vee p(X,b) \leftarrow q(X,c) \vee q(X,d)$$

3 INFERENCE MECHANISM

As pointed out by Lipski [9,10], a query applied to a database containing incomplete information might have many different interpretations depending on the certainty associated with it. But there are two bounds on the external interpretation of a query in an incomplete database system, namely the lower bound and the upper bound of the query.

Lipski's notion of the boundary can be extended to queries based on clauses in a deductive database containing incomplete information. If the lower bound of a query is required then all answers which can be concluded from the database are included. Such a query is referred to as a *definite query* and the prefix *def* is used to denote such a query. If the upper bound of a query is required, then all those potential answers with the possibilities that they might be true are included as part of the answer set. Such a query is referred to as a *possible query* and the prefix *pos* is used to denote a possible query.

3.1 THE PROOF THEORY OF INCOMPLETE DEDUCTIVE DATABASE

The satisfiability of a Horn system is decidable and there exist reasonably effective systems such as Prolog to construct a query evaluation system. Since incomplete information introduces indefiniteness into definite databases, the effective method of query evaluation for definite databases cannot be applied to indefinite databases directly. However, as shown by Loveland [14], a non-Horn system can be split into a set of Horn clause systems. The question of the satisfiability of the non-Horn system can be transformed into that of the satisfiability of the corresponding set of Horn clause systems.

A clause can be proved to be true in such a non-Horn system if and only if it can be proved to be true in each of the corresponding Horn subsystems. The proof of a clause in a Horn subsystem can be tested by using the proof by refutation procedure.

Although the idea of splitting a system into subsystems and checking the satisfiability in each is not efficient, it can still be adopted as the theoretical foundation on which the inference rules of the database containing incomplete information can be derived.

In general, a clause in the database which contains incomplete information has the following format:

$$p(\vec{A}) \vee \neg q_1(\vec{B}_1) \vee \ldots \vee \neg q_m(\vec{B}_m)$$

which can also be written in implication form as:

$$p(\vec{A}) \leftarrow q_1(\vec{B}_1) \wedge \ldots \wedge q_m(\vec{B}_m)$$

in which each argument of p and q_i ($1 \le i \le m$) can be a variable, a complete constant or an incomplete constant having the form $\{a_1, \ldots, a_n\}$. Indefiniteness due to incomplete information is introduced by allowing occurrences of incomplete constants as arguments of the positive literal p. According to the *AE* interpretation, a positive literal in a clause containing incomplete information can be transformed into a disjunction of literals without incomplete information in the same clause. The transformed clause can be split into a set of Horn clauses. Thus the evaluation of a query in an incomplete database can be transformed into the evaluations of the same query in a set of complete sub-databases.

Since a deductive rule containing variables can be represented as a set of rules by instantiating each variable with each value of the Herbrand universe of the system, and since the database concerned is function free, the closure of its ground instance clause set is finite. For a given database, an equivalent ground database can be constructed. Hence, in what follows the system is assumed to have only ground clauses. The splitting algorithm can be given as follows:

1. If a positive literal in a ground clause C contains an incomplete constant α with n possible values, then C is split into n sub-clauses where each sub-clause is a duplicate of C except the argument α is replaced by a distinct value of set α. This procedure can be repeated until there are no incomplete constants in any positive literal.
2. A system *DB* containing an incomplete clause C (the positive literal in the clause containing incomplete constants) can be split into a set of subsystems. Each subsystem contains only one sub-clause of C and the rest of the database. This procedure can be repeated until there is no incomplete clause in any subsystem.
3. Each subsystem is then a definite system, that is, a Horn clause system.

A general query evaluation strategy can be given as follows: Let $S(DB,Q,\theta_0,\theta)$ denote a status which specifies that a query Q is to be evaluated in the database *DB* with θ_0 as the initial substitution and θ as the result substitution. A substitution is a set having the form

$$\theta = \{t_1/v_1, t_2/v_2, \ldots, t_m/v_m\}$$

in which each t_i is a term, each v_i a variable and $v_i \neq v_j$ if $i \neq j$. The initial substitution for the original query is always the empty set.

If the query is an empty clause (denoted as $\square$) then the whole evaluation terminates, and the result substitution θ is the final substitution which constitutes an answer to the query.

The inference rules for a complete database are specified as follows:

Rule 1: If the query is an empty clause $\square$ then the evaluation terminates and the result substitution represents the answer to the original query.

$S(DB,\square,\theta,\theta)$ *final substitution* = θ

Rule 2: If the query Q is a conjunction of literals $Q_1 \wedge Q_2$ then, if the evaluation of Q_1 succeeds yielding result substitution θ_1, and if the evaluation of Q_2 with substitution θ_1 succeeds, then the evaluation of the query Q succeeds. The result substitution of Q_2 is the final substitution.

$S(DB,Q_1 \wedge Q_2, \theta_0, \theta) \Leftrightarrow S(DB,Q_1, \theta_0, \theta_1)$ *and* $S(DB, Q_2, \theta_1, \theta)$ *final substitution* = θ

Rule 3: If the query Q is a disjunction of literals $Q_1 \vee Q_2$ then, if the evaluation of either Q_1 *or* Q_2 with the initial substitution θ_0 succeeds then the evaluation of the query Q succeeds and its result substitution is the final substitution of query Q.

$S(DB,Q_1 \vee Q_2, \theta_0, \theta) \Leftrightarrow S(DB,Q_1, \theta_0, \theta)$ *or* $S(DB,Q_2, \theta_0, \theta)$ *final substitution* = θ

Rule 4: If the query is a single literal Q then, if either of the following two conditions is satisfied:

- there exists a ground fact p such that the query with the initial substitution $Q\theta_0$ can unify with p yielding a result substitution θ;
- there exists a deductive rule having the form $p \rightarrow R$ such that the query with the initial substitution $Q\theta_0$ can unify with the conclusion p yielding a substitution θ_1 as a result and the evaluation of the condition R with θ_1 as the initial substitution succeeds yielding θ as the result substitution

then the evaluation of Q succeeds and θ is the final substitution.

$S(DB,Q,\theta_0, \theta) \Leftrightarrow p \in DB \wedge (p = (Q\ \theta_0)\ \theta)$ *or* $p \leftarrow R \in DB \wedge (p\ \theta_1 = (Q\ \theta_0)\ \theta_1)$ *and* $S(DB,R, \theta_1, \theta)$

final substitution = θ

In an incomplete database rules 1, 2 and 3 can be used directly. A query containing incomplete information can be transformed into a disjunctive query without incomplete information, then rule 3 is applied to evaluate the query. If incomplete information is involved in some database clauses then the following two rules must be added to complement this rule.

Rule 5: If the database has the form $DB\ \&\ (p_1 \vee p_2)$ in which $p_1 \vee p_2$ is a disjunctive clause then the evaluation of a definite query Q is transformed into a set of evaluations in $DB\ \&\ p_1$ and $DB\ \&\ p_2$. If both the evaluations succeed then the whole evaluation succeeds.

$S(DB\ \&\ (p_1 \vee p_2),Q, \theta_0, \theta) \Leftrightarrow S(DB\ \&\ p_1,Q,\theta_0,\theta_1)$ *and* $S(DB\ \&\ p_2,Q,\theta_0,\theta_2)$

final substitution $\theta = \theta_1+\theta_2$

The combination of the result substitution in each subsystem (denoted as $\theta_1 + \theta_2$) constitutes the final substitution for the query. The combination function can be specified as:

$\exists_i\ t/v_i \in \theta_1 \wedge s/v_i \in \theta_2 \wedge t \neq s \rightarrow \{t,s\}/v_i \in \theta$

$\exists_i\ t/v_i \in \theta_1 \wedge t/v_i \in \theta_2 \rightarrow t/v_i \in \theta$

$\exists_i\ t/v_i \in \theta_1 \wedge s/v_i \notin \theta_2 \rightarrow t/v_i \in \theta$

$\exists_i\ t/v_i \in \theta_2 \wedge s/v_i \notin \theta_1 \rightarrow t/v_i \in \theta$

in which $\{t, s\}$ represents an incomplete constant.

Rule 6: If the database has the form $DB\ \&\ (p_1 \vee p_2)$ in which $p_1 \vee p_2$ is a disjunctive clause then the evaluation of a possible query Q is transformed into a set of evaluations in $DB\ \&\ p_1$ and $DB\ \&\ p_2$. If any evaluation in the subsystems succeeds then the whole evaluation succeeds. The result substitution of the successful evaluation constitutes the final substitution of the query.

$S(DB\ \&\ (p_1 \vee p_2), Q, \theta_0, \theta) \Leftrightarrow S(DB\ \&\ p_1, Q, \theta_0, \theta)$ *or* $S(DB\ \&\ p_2, Q, \theta_0, \theta)$ *final substitution* = θ

The rules 1, 2, 3, 4 and 5 form the definite query evaluation system which follows the normal process of logical deduction. Such a deduction is referred to as a definite deduction. Rules 1, 2, 3, 4 and 6 form the possible query evaluation system. Such a deduction is referred to as a possible deduction.

3.2 ANSWER DEFINITION

When a query Q is applied to a complete database DB, each answer is a substitution θ such that

$DB \models Q\theta$

in which $\models$ is a deduction which follows the inference rules given in the previous section. If either the query or the database contains incomplete information then the answer may be a combination of alternative substitutions of the form:

$\theta_1 + \theta_2 + \ldots + \theta_n$

such that

$DB \models Q\theta_1 \vee Q\theta_2 \vee \ldots \vee Q\theta_n$

Consider the answer to a query, Q, applied to a database, DB. Let θ be a ground substitution of the variables, $Ans(def(Q))$ be an answer set of definite query $def(Q)$, $Ans(pos(Q))$ be an answer set of pos query $pos(Q)$. Let $p_1 \vee p_2$ represent a disjunctive clause (obtained by rewriting a clause containing incomplete data) in the database DB and let Δ represent the remaining clauses in the database, that is, $DB = \Delta \,\&\, (p_1 \vee p_2)$. Let DB_1 denote $\Delta \,\&\, p_1$ and DB_2 denote $\Delta \,\&\, p_2$. The answer to a query can be given as follows.

$Ans(def(Q)) = \{\theta \mid DB \models Q\,\theta\} \cup \{\theta_1 + \theta_2 \mid DB_1 \models Q\theta_1 \text{ and } DB_2 \models Q\theta_2\}$

$Ans(def(Q_1 \wedge Q_2)) = \{\theta \mid DB \models (Q_1 \wedge Q_2)\,\theta\} \cup \{\theta_1 + \theta_2 \mid DB_1 \models (Q_1 \wedge Q_2)\,\theta_1 \text{ and } DB_2 \models (Q_1 \wedge Q_2)\theta_2\}$

$Ans(def(Q_1 \vee Q_2)) = \{\theta \mid DB \models Q_1\,\theta \text{ or } DB \models Q_2\theta\} \cup \{\theta_1 + \theta_2 \mid DB_1 \models Q_1\,\theta_1 \text{ and } DB_2 \models Q_2\theta_2\}$

$Ans(pos(Q)) = \{\theta \mid DB_1 \models Q\theta \text{ or } DB_2 \models Q\theta\}$

$Ans(pos(Q_1 \wedge Q_2)) = \{\theta \mid DB_1 \models (Q_1 \wedge Q_2)\theta \text{ or } DB_2 \models (Q_1 \wedge Q_2)\theta\}$

$Ans(pos(Q_1 \vee Q_2)) = \{\theta \mid DB_1 \models Q_1\theta \text{ or } DB_1 \models Q_2\theta \text{ or } DB_2 \models Q_1\theta \text{ or } DB_2 \models Q_2\theta\}$

If a database contains no incomplete information then it has

$Ans(Q) = Ans(def(Q)) = Ans(pos(Q))$.

4 THE INDEFINITE CLOSED WORLD ASSUMPTION

The Closed World Assumption [17] provides a simple and natural way of dealing with negation in relational and definite deductive databases. However, if one permits indefinite information to be stored, as in the case of indefinite deductive databases, the Closed World Assumption is no longer adequate. For this reason the Generalised Closed World Assumption [15] and the Extended Generalised Closed World Assumption [23] have been proposed. However, these approaches have one serious shortcoming which relates to the interpretation of disjunction. In both cases under certain circumstances the logical operation **or** ($\vee$) is interpreted as an exclusive **or** instead of as an inclusive **or**. For example, consider a database with the single clause:

$p(a) \vee p(b)$

which states that either $p(a)$ or $p(b)$ or both are true. This database has two minimal models, $M_1 = \{p(a)\}$ and $M_2 = \{p(b)\}$. Under the GCWA, since neither $p(a)$ nor $p(b)$ can be proved to be true, neither of them can be assumed to be false. Hence both $p(a)$ and $p(b)$ are regarded as unknown.

Suppose at some later stage, new information $p(a)$ is introduced into the system. By adding this into the database the latter now contains the two clauses:

$p(a) \vee p(b)$

$p(a)$

This does not introduce any new information about $p(b)$, and hence the satisfiability of $p(b)$ should not be affected. However, under the minimal model theory the new database will have only one minimal model, $M = \{p(a)\}$, and according to the GCWA, $p(a)$ can be proved to be true in the database and $p(b)$ can now be assumed to be false. This side effect of the GCWA is a serious problem in terms of the correctness of the conclusion.

For this reason a modified minimal model theory, I-Model theory, is proposed to cater for the case of indefinite deductive database systems. Under the I-Model theory, the logic operator **or** is interpreted as an inclusive **or**.

4.1 DEFINITION OF I-MODEL

In order to generate a finite set of answers for user queries, a database is usually assumed to be function free. The variables occurring in each deductive rule can be instantiated with each value of the Herbrand universe of the system and the closure of its ground instance clause set is finite if domains are finite. For a given database, an equivalent database containing only ground clauses can be constructed. Hence in the following discussions a database system can be assumed to have only ground clauses.

Definition Definite sub-clause: If a ground clause C contains n ($n \geq 1$) positive literals then C can be split into n sub-clauses, each of which contains only one distinct positive literal of C and all the negative literals in C. Such a sub-clause will be referred to as a definite sub-clause of C.

A constructive definition of an I-Model is as follows:

Definition I-Model: An I-Model of a ground database can be constructed by performing the following steps:

1. A database *DB* containing an indefinite clause *C* (a clause containing more than one positive literal) can be split into a set of sub-databases each of which contains only one definite sub-clause of *C* and the rest of the database. This procedure can be repeated until no indefinite clause remains in any sub-databases. Each of the sub-databases is a Horn clause system.
2. Each Horn clause system has a unique minimal model. Each distinct minimal model of the sub-databases constitutes an I-Model of the entire database.

Let *DB* denote the database system, MM_i represent one of the minimal models of *DB*, and IM_i represent an I-Model of *DB*.

To demonstrate the definition, consider the following example:

$p(a,a) \vee p(b,a) \leftarrow q(a)$

$p(a,b) \vee p(b,b) \leftarrow q(b)$

$p(a,a)$

$q(a)$

This database has only one minimal model, $MM_1 = \{p(a,a),q(a)\}$. According to the I-Model definition presented above, it can be seen that the database has two I-Models, $IM_1 = \{p(a,a),q(a)\}$ and $IM_2 = \{p(a,a),p(b,a),q(a)\}$. The only minimal model of the database, MM_1, is equivalent to IM_1.

4.2 I-MODEL INTERPRETATION

An I-Model interpretation is given as follows:

1. A ground positive literal (an atomic formula) is true if and only if it occurs in every I-Model. It can be assumed to be false if and only if it does not occur in any I-Model.
2. A ground negative literal (the negation of an atomic formula) is true if and only if its complement (the positive literal) does not occur in any I-Model. It can be assumed to be false if and only if its complement occurs in every I-Model.
3. A conjunctive formula is true if every element (literal) of the conjunction is true.
4. A disjunctive formula is true if any element (literal) of the disjunction is true.

This interpretation is referred to as the Indefinite Closed World Assumption (ICWA).

Consider again the last example in which the database has the two I-Models, $IM_1 = \{p(a,a),q(a)\}$ and $IM_2 = \{p(a,a),p(b,a),q(a)\}$. Under the ICWA, $p(a,a)$ and $q(a)$ can be proved to be true because each occurs in both IM_1 and IM_2; $p(a,b)$ and $p(b,b)$ can be assumed to be false since they do not belong to any I-Model; $p(b,a)$ occurs in one I-Model (IM_2) but not the other (IM_1) and hence its truth value is unknown. Compare this with the GCWA approach in which the system has only one minimal model $MM_1 = \{p(a,a),q(a)\}$, literals $p(a,a)$ and $q(a)$ can be proved true while all others, that is, $p(a,b)$, $p(b,b)$ and $p(b,a)$, can be assumed to be false.

4.3 MAIN PROPERTIES OF THE I-MODEL INTERPRETATION

The I-Model interpretation has the following important properties:

Property 1: An atomic formula is true in a consistent database under the I-Model interpretation if and only if it is true under the minimal model interpretation.

Property 2: If an atomic formula can be assumed false under the I-Model interpretation then it can also be assumed false under the minimal model interpretation.

Property 3: If an atomic formula can be proved neither true nor false under the minimal model interpretation, then it can be proved neither true nor false under the I-Model interpretation.

Let $T(IM)$, $F(IM)$ and $U(IM)$ denote the sets of atomic formulae which can be proved true, false and unknown respectively under the I-Model interpretation. Similarly, let $T(MM)$, $F(MM)$ and $U(MM)$ denote the sets of atomic formulae which can be proved true, false and unknown respectively under the minimal model interpretation. The properties of the I-Model interpretation given above can be summarised as:

$T(IM) = T(MM)$

$F(IM) \subseteq F(MM)$

$U(IM) \supseteq U(MM)$

This indicates that since both interpretations give the same set of atomic formulae which are true in a consistent database, the I-Model interpretation is therefore consistent with the minimal model interpretation and it only extends the minimal model interpretation by allowing the representation of inclusive **or**.

The following property demonstrates the inclusive nature of the I-Model in respect of the operator **or**.

Property 4: Let *DB* be a consistent database and *p*, *q* be positive literals. If

$DB \models p \vee q$,

$DB \models p$, *and*

$DB \not\models q$

and there exists a clause set $\{S\}$ in *DB* such that

$DB \setminus \{S\} \models p \vee q$ *and*

$DB \setminus \{S\} \not\models p$

in which $DB \setminus \{S\}$ is a database excluding the clause set $\{S\}$ from *DB*, then one of the I-Models of *DB* is a proper superset of a minimal model of *DB*.

The ICWA can also be applied to a system containing incomplete information of the form $p(\{a,b\})$ which is interpreted as $p(a) \vee p(b)$ [21,22] since such a system is only a special case of a general indefinite database.

Since the I-Model theory is consistent with the minimal model theory, it can be used to prove that the prove theory given in the previous section is sound and complete.

Property 5: A substitution θ is a definite answer to an atomic query *Q* if and only if *Q*θ is true in every I-Model of the database system. That is,

$\theta \in Ans(def(Q)) \Leftrightarrow \forall_i\, Q\theta \in IM_i$

A substitution θ is a definite answer to a complex query *Q* if and only if *Q*θ is true in every I-Model of the database system.

Property 6: A substitution θ is a possible answer to an atomic query *Q* if and only if there exists an I-Model such that *Q*θ belongs to this I-Model. That is,

$\theta \in Ans(pos(Q)) \Leftrightarrow \exists_i\, Q\theta \in IM_i$

A substitution θ is a possible answer to a complex query *Q* if there exists an I-Model such that *Q*θ is true in the I-Model.

5 IMPLEMENTATION

The proof theory and model theory of the query evaluation system of such a database were presented by using the method of splitting a non-Horn database into a group of Horn databases. However, splitting an indefinite database into several definite databases and then evaluating a query on these sub-databases is inefficient [8].

To counteract this shortcoming, a new data type, the incomplete constants, has been introduced. By using incomplete constants the problem of handling this special indefiniteness can be confined to that of handling incompleteness of values. A clause with a restricted form of disjunction can be expressed as a clause containing only positive literals with incomplete values. For example, the clause

$p(a) \vee p(b) \vee \neg q(c)$

can be expressed as:

$p(\{a,b\}) \vee \neg q(c)$

in which $p(\{a,b\})$ is the only one positive literal and $\{a,b\}$ is the incomplete constant. Thus it is possible to extend the query evaluation mechanism of definite database to cater for databases containing incomplete information.

Two main components of the proof by refutation procedures are unification and resolvent generation. The definition of substitution is extended to be able to handle the incomplete constants. Two inference rules are introduced. One is defined for definite query evaluation. That is, a clause can be deduced if and only if it is a logic consequence of the database. The other is defined for possible query evaluation. That is, answers generated from such a system are possibly true in the database.

Two implementation strategies are proposed for such a database system with incomplete information. They are referred to as the tuple oriented approach and the set oriented approach. The first approach evaluates a

single solution each time and construct the answer set for the query through backtracking. The second approach evaluates all answers for each subquery and computes the answer set of the query using a modified join operator.

Both strategies are implemented by using Prolog. Statistics shown that, since the tuple oriented approach uses the *fail and backtracking* technique, it causes repeated evaluation, hence is less efficient. The set oriented approach, although requires lager space, is much more efficient. Apart from the efficiency, the set oriented approach can handle recursive definition effectively, while the tuple oriented approach may lead to a non-terminating loop.

6 CONCLUSION

By introducing the incomplete constants, a certain type of indefiniteness can be represented, while the Horn clause form is still used as the basic format of a general rule in the system. Hence, the inference mechanism adopted for a Horn clause system can be used in a system employing the incomplete constants.

An I-Model theory is proposed which is used to as a mechanism to derive the negative information and it can also be used to prove the soundness and the completeness of the inference rules of the system [8].

Two implementation strategies are proposed for such a deductive database system. The set oriented approach is more effective when handling recursive definitions. The prototype implementations of both approaches are given in Prolog and discussed fully in [7,8].

REFERENCES

[1] L. Cholvy and R. Demolombe, "Querying a rule base", *Proc. of First International Conference on Expert Database Systems*, South Carolina, pp365-371, April, 1986.

[2] H. Gallaire, "Logic data bases vs deductive data bases", *Proc. of logic programming workshop'83*, pp608-622, (1983).

[3] J. Grant, "Null values in a relational data base", *Information Processing Letters*, vol(5), pp156-157, (1977).

[4] T. Imielinski and W. Lipski, "Incomplete information in relational databases", *Journal of ACM*, vol(31), 4, pp761-791, Oct. 1984.

[5] T. Imielinski, "Query processing in deductive databases with incomplete information", pp268-280, Oct. 1984.

[6] Q. Kong, "On representing incomplete information and null values", Technical report 89/6, Dept. of Computer Science, Heriot-Watt University, October 1988.

[7] Q. Kong and M.H. Williams, "Evaluating different strategies for handling incomplete information in a logic database", Expanding the horizons Workshop, Imperial College, T. Dodd, R. Owens and S. Torrance (eds.), Intellect, Oxford, England, (1991).

[8] Q. Kong, "Incomplete information in a deductive database", PhD Thesis, Dept. of Computer Science, Heriot-Watt University, Edinburgh, Sept. 1989.

[9] W. Lipski, "Informational systems with incomplete information", *Proc. 3rd Int. Symp. on Automata, Languages and Programming*, S. Michaelson and R. Milner (eds.), Edinburgh University Press, Edinburgh, pp120-130, (1976).

[10] W. Lipski, "On semantic issues connected with incomplete information databases", *ACM Trans. on Database Systems*, vol(4), 3, pp262-296, Sept. 1979.

[11] J.W. Lloyd, "An introduction to deductive database systems", *Australian Computer Journal*, vol(15), 2, pp52-57, May 1983.

[12] J.W. Lloyd and R.W. Topor, "A basis for deductive database systems", *Journal of Logic Programming*, vol(2), 2, pp93-109, (1985).

[13] J.W. Lloyd and R.W. Topor, "A basis for deductive database systems II", *Journal of Logic Program*, vol(3), 1, pp55-67, (1986).

[14] D.W. Loveland, "Automated theorm proving", North Holland Publishing Co., New York, (1978).

[15] J. Minker, "On indefinite databases and the closed world assumption", *Lecture Notes in Computer Science*, Springer Verlag, vol(138), (1982).

[16] J. Morrissey, "Imprecise information and uncertainty in information systems", *ACM Trans. on*

Information Systems, vol(8), 2, April 1990. pp158-180,
[17] R. Reiter, "On closed world data bases", *Logic and data bases*, H. Gallaire and J. Minker, Plenum Press, New York, pp55-76, (1978).
[18] R. Reiter, "A sound and sometimes complete query evaluation algorithm for relational database with null values", *Journal of ACM*, vol(33), 2, pp349-370, April 1986.
[19] C.B. Schwind, "Embedding deductive capabilities in relational database systems", *International Journal of Computer and Information Sciences*, vol(13), 5, pp327-338, (1984).
[20] M.Y. Vardi, "Querying Logical Databases", *Proc. 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp57-65, March 1985.
[21] M.H. Williams and Q. Kong, "Incomplete information in a deductive database", *Data and Knowledge Engineering*, vol(3), pp197-220, (1988).
[22] M.H. Williams Q. Kong and G. Chen, "Handling incomplete information in a logic database", *UK IT 88 Conference Publication*, pp224-227, (1988).
[23] A. Yahya and L.J. Henschen, "Deduction in non-Horn databases", *Journal of Automated Reasoning*, vol(1), pp141-160, (1985).
[24] C. Zaniolo, "Database relations with null values", *Proc. ACM SIGACT-SIGMOD Symposium on principles of Database Systems*, New York, pp27-33, (1982).