INTEGRITY CONSTRAINTS IN LOGIC DATABASES

PATRIZIA ASIRELLI,* MICHELE DE SANTIS,** AND MAURIZIO MARTELLI

We consider logic databases as logic programs and suggest how to deal with the problem of integrity constraint checking. Two methods for integrity constraint handling are presented. The first one is based on a metalevel consistency proof and is particularly suitable for an existing database which has to be checked for some integrity constraints. The second method is based on a transformation of the logic program which represents the database into a logic program which satisfies the given integrity constraints. This method is specifically suggested for databases that have to be built specifying, separately, which are the deductive rules and the facts and which are the integrity constraints on a specific relation. Different tools providing for the two mechanisms are proposed for a flexible logic database management system.

1. INTRODUCTION

Many authors have pointed out how convenient and easy it is to handle database (DB) problems using logic. As a matter of fact, logic is used not only to represent relational databases, but also to formulate queries (i.e. it is used as a query language), to express views, and to express integrity constraints (i.e. conditions which must be satisfied by the database).

An idea now widely accepted in the world of logic databases is that a database can be represented by a logic program. In this framework, assertions (i.e. ground unit clauses) are considered the extensional component of the DB, whereas rules (i.e. nonunit program clauses), constitute the intensional component of the DB and thus represent the general laws of the DB [9, 10, 13, 14, 18–20].

The problem of integrity constraints in a DB is becoming increasingly important [7, 8, 11, 12, 17, 20], and it is obviously relevant for a logic DB.

*IEI, Institute of CNR, Pisa, Italy.
**Sand M, Pisa, Italy.
In this paper we propose that a logic DB (extensional and intensional components) is given as a logic program (it is a definite deductive database as described in [10]). We suggest how integrity checking could be carried out and how consistency with integrity constraints can be maintained in a DB where rules and integrity constraints are kept distinct. As we shall see, integrity constraints are described by a set of formulas which might express the "only if" conditions of relations.

A very special case of integrity constraints is represented by the necessity of constraining variables to range over a specific domain. Many proposals have dealt with this problem, but we believe that the solution is to use a logic language with types or, more generally, a many-sorted logic. We will not stress this point any further in this paper; for a detailed discussion see [1, 17].

Our research work in Pisa is oriented towards the definition of a language that integrates functional and logic programming [2, 3] and that enables the programmer to benefit from all of the more important features that have been proved necessary in a "good" programming language (types, modules, etc.). Moreover, a good programming language needs a good programming environment to be useful. This means that all facilities needed to develop, manage, test, and verify programs have to be available, and also tools have to be available to help the user in the development of specific applications.

This paper is an effort in this last direction, since we believe that the declarative part of an integrated language (i.e. Horn clause language) is particularly suitable to describe and to manage databases, expert systems, or—more generally—knowledge. In this respect the problem of handling integrity constraints is an important one in defining tools and methodologies for databases.

This paper describes, in Section 2, some general concepts concerning logic DBs. Section 3 and 4 are devoted to present two different methods for integrity constraint handling. The first one suggests how to check that a given DB obeys a set of integrity constraints. The second one deals with the problem of building a DB specifying separately facts, rules, and integrity constraints. Remarks about the applicability of the above methods are also given.

In the rest of the paper we will abbreviate integrity constraints as IC, and the extensional component of the DB (i.e. facts) as EDB.

2. LOGIC AND DATA BASES

In this section we will briefly point out some considerations that have been relevant to the understanding of the relationship between Horn clause logic and databases and that are related to the aims of this paper.

The first point to mention is the relation between logic DB and the existing standard relational DBs. Many authors [13, 19] believe that retrieval of data from the EDB, can be performed using existing and efficient DBs or specific DB machines. This fact must be taken into account in any proposed methodology for logic DB handling.

The second, and more relevant, aspect is the approach used for representing a perceived world in a logic DB. As described in [19], there are three main approaches:

The first approach considers the perceived world as a first order theory; in this framework general laws are considered as deductive rules.
The second approach considers the set of elementary information as an interpretation of a first order theory; general laws are interpreted as integrity constraints.

The third approach, which we find to be the most interesting, is an integration of the previous two approaches. In this case some general laws are considered as deductive rules and others as ICs. Some criteria to decide how a rule should be interpreted can be found in [19].

Another important aspect is the form of definitions in a logic DB. As Kowalski shows [14, 15], definite clauses (i.e. assertions and rules) express the “if” half of the complete definition of a relation. The “only if” half can be expressed, at a metalevel, by stating that $A \leftarrow B$ expresses the only condition under which $A$ can be proved. With this interpretation, definite clauses have the intended meaning of expressing the complete ($\leftrightarrow$) definition of the relation $A$. The “if” halves (i.e. definite clauses) are sufficient to derive all positive facts about relations, thus allowing the use of standard resolution methods for querying the DB, assuming the query has only positive atoms. The full “if and only if” definition of relations is nevertheless needed for proving properties of programs and for negation (closed world assumption [22] and negation as failure rule [5]), or for answering queries involving universal quantifiers (see Kowalski [15]).

Moreover, considering the DB as a first order theory, Kowalski suggests that given a definite clause defining a relation ($\leftrightarrow$), on adding clauses which are exceptions to the general law, we have to modify the general law accordingly. This approach can lead to a methodology which allows a user to define all the aspects of a relation.

Let us give an example:

(a) $\text{grandparent}(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y)$

can be interpreted as a deductive rule stating that to derive that $X$ is the grandparent of $Y$, one must find a $Z$ which is a parent of $Y$ and such that $X$ is a parent of $Z$.

On the other hand, we might know that $a$ is the grandparent of $b$, although we do not know who the parent of $b$ is such that $a$ is his/her parent. Thus we would like to state

$\text{grandparent}(a, b) \leftarrow$.

But following Kowalski’s approach, to maintain the complete definition of the DB ($\leftarrow$ interpreted as $\leftrightarrow$, for all clauses), we would have to modify (a) as follows:

(a') $\text{grandparent}(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y), X \neq a, Y \neq b$.

Let us remark that in this approach ICs appear as part of the definitions, while, as suggested in [19] (third approach above), it is often desirable to have separate rules which express ICs, i.e. specific properties that the DB is supposed to have.

As we have already point out, a logic DB consists of a set of known facts (EDB) and a set of general laws (intensional component). General laws can be interpreted as deductive rules or as ICs.

When a general law is interpreted as a deductive rule (first approach above), there is no interest in proving that the EDB is also a model of the first order theory composed of the EDB and the deductive rules. In fact it probably will not be a model, or there will not be a great advantage in having deductive rules. Indeed, if the
EDB is always a model of such a theory, that means that the obtained DB is redundant: Deductive rules generate facts which are already in the EDB.

On the other hand, when general laws are interpreted as ICs, the EDB must be proved to be a model of a theory whose proper axioms are the ICs themselves (second approach above).

According to the third approach, in considering the problem of consistency, Kowalski [14] suggests that one should define a predicate \textsc{inconsistent} which can be used to prove the consistency of a DB when necessary. The definition of such a predicate will consist of the collection of all properties that must be checked. This means that it is necessary to prove that the explicit information derivable from a DB is not a model of the predicate \textsc{inconsistent}, while in general the problem of ICs is considered nothing more than proving program properties [15].

Moreover, since the integrity checking can be very heavy, the proof of inconsistency can be done periodically. Other researchers suggest that integrity checking must be carried out every time the DB state changes and thus suggest methods to simplify the structure of the IC [12, 21].

Recently, Lloyd and Topor [17] have proposed to use first order formulas as clause bodies, thus introducing \textit{extended programs} and \textit{goals}. In this view the form of an IC can be any first order formula that will be transformed into a suitable set of Horn clauses.

In general, when considering the rules as "if and only if" formulas, the real constraints are indeed the formulas which express the "only if" halves. For example, the EDB

(b) \[ \text{grandparent}(a, b) \leftarrow \]
\[ \text{parent}(c, d) \leftarrow \]
\[ \text{parent}(d, e) \leftarrow \]

must be proved to be a model of the following integrity constraint:

\[ \text{grandparent}(X, Y) \rightarrow \text{parent}(X, Z), \text{parent}(Z, Y). \]

Note that a general law of the form \( A \rightarrow B_1, \ldots, B_n \) (expressing ICs) is not a Horn clause, but a formula that expresses an "only if" condition. Thus its interpretation is

\[ \neg A \lor (B_1 \land \cdots \land B_n). \]

With the IC (a") we have that (b) is not a model for (a") + (b) because "\text{grandparent}(a, b) \leftarrow" falsifies (a"'); in fact, it does not satisfy the IC.

We think that expressing IC with the "only if" form of clauses is the correct approach to expressing that specific ground instances of a DB must obey some constraint. An example of the kind of constraints we consider is the following:

\[ \forall X \forall Y(\text{age}(X, Y) \rightarrow (Y < 150)), \]

which expresses the general knowledge "the age of any person is less than 150". However, expressing ICs with "only if" halves of clauses makes it impossible to use definite clauses and thus to see the EDB together with ICs as a logic program.

More generally, and differently from [20], our view of ICs is that they are outside the theory, which consists only of EDB and deductive rules, plus negation as failure, if a closed world is assumed [10].
3. THE CONSISTENCY PROOF METHOD

We consider that a logic DB is given as a set of definite Horn clauses (EDB and deductive rules) plus a set of integrity constraints (IC). This view corresponds to considering the ICs as outside the theory [10].

In general, ICs can be any kind of formulas, as in [20]. Let us first restrict them to be of the "only if" form.

Given an existing logic DB, the problem of proving its consistency, with respect to a set (IC) of formulas, could be solved by proving that each formula in IC is true in the minimal model of the logic DB.

Let us first assume that the DB consists just of the EDB. Given a set of ICs, i.e. a set of formulas such as

\[ A^* \rightarrow B_1, \ldots, B_n, \]

then, for each ground instance of \( A^* \) in the DB, we must prove that \((B_1 \land \cdots \land B_n)\theta_i\) holds, where \( \theta_i \) is the substitution which unifies \( A^* \) with the \( i \)th clause defining \( A^* \) in the DB. Then, \((B_1 \land \cdots \land B_n)\theta_i\) can be proved by SLD refutation in the DB.

When deductive rules are also given, the above proof is not so immediate. In fact we have to prove that all facts which can be derived in the logic DB satisfy the ICs. This means that, given the logic program \( P \), we have to prove that its minimal model is also a model for IC, or else that all formulas in IC are true in the minimal model \( M_P \) of \( P \). Thus we need to prove, for each relation \( A_k \) and for each formula in IC, that

\[ (f) \quad A_k^i \rightarrow B_1^i, \ldots, B_n^i \]

is true in \( M_P \).

In general, we can prove \((f)\) by refutation, assuming \( A_k^i \) and then proving all the \( B_j^i \) by refutation or by induction [4,23]. This is to prove that IC is a theorem for \( P \). This is a stronger property than the one required for IC (see also p. 38 in [10]), i.e., we are saying that IC is true in all models of \( P \).

Another approach that can be adopted for proving IC is a metalevel proof by using the logic program \( P \). That is, let IC = \{IC_1, IC_2, \ldots, IC_n\} be the set of formulas expressing IC; then for all IC, \( A_k^i \rightarrow B_1^i, \ldots, B_n^i \), find all the answers to \( A_k^i \) in \( P \), via SLD resolution. Then a set of bindings \( \mathcal{S} \) will be obtained. For each \( \tau_i \in \mathcal{S} \) prove, by SLD resolution in \( P \), that \( (B_1^i, \ldots, B_n^i)\tau_i \) is true.

Then IC can be forgotten until a new checking is needed, say because the DB has changed. Note that if no deductive rule is given and if the above algorithm succeeds for a given IC, then IC can be easily proved to be true in the minimal model \( M_{EDB} \). On the other hand, when deductive rules are given a problem may arise if the substitution \( \tau_i \) contains bindings such as \( X/Y \) (i.e. variable/variable or unbound variables), or in general, bindings such as \( X/t \) where \( t \) is a term containing unbound variables. In this case, the set of values (solutions for \( A_k^i \)) is the Herbrand universe or a subset of it. For example, let \( P \) be

\[
\begin{align*}
p(Y) & \leftarrow q(a) \\
q(a) & \leftarrow \\
q(b) & \leftarrow \\
r(a) & \leftarrow \\
IC : p(X) & \rightarrow r(X)
\end{align*}
\]
Then $M_p$ is \{p(a), p(b), q(a), q(b), r(a)\},

$\tau = \{X/Y\}$,

and " $\leftarrow r(Y)$ " succeeds for \{Y/a\}.

Obviously, in this case, IC is not true in $M_p$. In general, we should find all possible answers to " $\leftarrow r(Y)$ " and then see whether or not the obtained set of bindings for $Y$ is exactly the Herbrand universe (HU) or, in general, the subset of the HU corresponding to $\tau$.

One might observe that the algorithm, as it is, would anyway be a useful tool to prove properties of existing relational (nonlogic) databases. In fact, there are many proposals to use logic languages for querying existing DBs; then our algorithm would allow one to prove DB properties using the same querying mechanism.

On the other hand, when for all clauses in a DB all the variables occurring in the conclusion occur in the condition part as well (i.e. they are range-restricted clauses \[10, 21\]), SLD resolution never produces answers such as \{X/Y\}. Thus, in such a case, we obtain an efficient algorithm. Moreover, this limitation does not restrict significantly the kind of databases that can be implemented.

Let us observe that this method, although it is here applied for IC formulas such as:

(i) $A \rightarrow B_1, \ldots, B_n$,

could be easily extended to formulas such as:

(ii) $A_1 \land \cdots \land A_m \rightarrow B_1, \ldots, B_n$

(iii) \[A_1 \land \cdots \land A_m \rightarrow B_1, \ldots, B_n\]

(iv) $A_1 \land \cdots \land A_m \rightarrow$

Note that either formulas (ii)–(iv), as well as formula (i), must be range-restricted (in which case all variables are intended to be universally quantified), or else local variables (i.e. variables occurring only on the right hand side) are intended to be existentially quantified while all others are universally quantified. Moreover, note that ICs of the form (ii) are suitable to express functional and other kinds of database dependencies [7].

To conclude, let us note that this approach, in the context of combining object language and metalanguage, can be viewed as the operational counterpart of the more general approach described by Kowalski [15].

4. THE MODIFIED PROGRAM METHOD

4.1. The Construction of the Modified Program

Another method to handle ICs is to find a logic program whose minimal model is exactly the subset of $M_p$ which satisfies IC.

Let us introduce some notation. We assume that all formulas in IC are such that the left hand side has only distinct variables. Thus, we assume that all ICs are of the form

\[(g) \quad p(X_1, \ldots, X_n) \rightarrow \Psi(X_1, \ldots, X_n)\]
IC formulas with terms different from variables on the left hand side can also be transformed to the above form. In fact, consider a generic IC as follows:

\[(\forall Y_1, \ldots, Y_m)(p(t_1, \ldots, t_n) \rightarrow \exists W_1, \ldots, W_d(\forall(Y_1, \ldots, Y_m, W_1, \ldots, W_d)))\]

where \(t_1, \ldots, t_n\) are terms in which variables \(Y_1, \ldots, Y_m\) may occur and \(\forall(Y_1, \ldots, Y_m, W_1, \ldots, W_d)\) is a conjunction of atoms \(A_1, \ldots, A_s\), where variables \(W_1, \ldots, W_d\) and \(Y_1, \ldots, Y_m\) may occur. Then we can define for the given IC a predicate \(\Psi\) such as

\[
\Psi(X_1, \ldots, X_n) \leftarrow \text{not-unify}([X_1, \ldots, X_n], [t_1, \ldots, t_n]),
\]

\[
\Psi(t_1, \ldots, t_n) \leftarrow \forall(Y_1, \ldots, Y_m, W_1, \ldots, W_d),
\]

where \(\text{not-unify}\) succeeds if the two lists \([X_1, \ldots, X_n]\) and \([t_1, \ldots, t_n]\) do not unify. It can be proved that the minimal model of a program \(P\) satisfies (h) if and only if the minimal model of the program \(P\) modified by adding the definition of \(\Psi\) satisfies (g).

Let us now describe how the modified program can be obtained:

1. Let \(P\) be a set of clauses and IC a set of integrity constraints, both on predicates \(p_1, \ldots, p_m\). Then, for each predicate \(p_k\), consider the following set:

\[
C_k = \{S^k_{i, j}\}
\]

which is built as follows:

(a) Let \(i \in \text{IC}|p_k\), i.e., \(i\) ranges over the integrity constraints on \(p_k\) of the form

\[
p_k(X) \rightarrow \Psi_k(X)
\]

where \(X\) stands for any sequence of variables.

(b) Let \(j \in P|p_k\), i.e., \(j\) ranges over the set of clauses defining the relation \(p_k\) of the form

\[
p_k(t^k_j) \leftarrow \Phi_k^j
\]

where \(t^k_j\) stands for any sequence of terms and where \(\Phi_k^j\) either is a conjunction of atoms or is empty.

(c) \(S^k_{i, j}\) is \(\Psi_k(t^k_j)\).

2. The corresponding modified logic program \(\text{mod}(P)\) is given by a set of clauses

\[
p_k^j(t^k_j) \leftarrow \Phi_k^j, C_k^j,
\]

where

\[
C_k^j = \bigwedge_{i \in \text{IC}|p_k} S^k_{i, j}
\]

and \(\bigwedge\) is an \(\land\) of the elements of the subset of \(C_k\), relative to clause \(j\).

As an example consider the following program:

\[
a(3) \leftarrow v(Z)
a(X) \leftarrow h(X)
\]
and the set of IC

\[ a(3) \rightarrow c(Y) \]
\[ a(f(U, V)) \rightarrow b(U), c(V) \]
\[ a(Z) \rightarrow b(Y). \]

By adding to the above program the following definitions:

\[ \Psi_1(X) \leftarrow \text{not-unify}([X],[3]) \]
\[ \Psi_2(X) \leftarrow c(Y) \]
\[ \Psi_2(f(U, V)) \leftarrow \text{not-unify}([f(U, V)]) \]
\[ \Psi_3(X) \leftarrow \text{not-unify}([X],[Z]) \]
\[ \Psi_3(3) \leftarrow b(Y) \]

we can rewrite the IC as

\[ a(X) \rightarrow \Psi_1(X) \]
\[ a(X) \rightarrow \Psi_2(X) \]
\[ a(X) \rightarrow \Psi_3(X). \]

Note that this is equivalent to the IC

\[ a(X) \rightarrow \Psi_1(X), \Psi_2(X), \Psi_3(X). \]

Thus the final program, i.e. the modified version of the given program, is

\[ a(3) \leftarrow v(Z), \Psi_1(3), \Psi_2(3), \Psi_3(3) \]
\[ a(X) \leftarrow h(X), \Psi_1(X), \Psi_2(X), \Psi_3(X) \]

plus all above definitions of \( \Psi \).

As another example let us apply the algorithm to the DB (b) with the IC (a") of Section 2. The transformed program results in

\[ \Psi_1(U, V) \leftarrow \text{not-unify}([U, V],[X, Y]) \]
\[ \Psi_1(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y) \]
\[ \text{grandparent}(a, b) \leftarrow \Psi_1(a, b) \]
\[ \text{parent}(c, d) \leftarrow \Psi_1(a, b) \]
\[ \text{parent}(d, e) \leftarrow . \]

Note that if the terms \( t_i \) in the conclusion parts of clauses are all distinct variables, as in this case, a possible optimization could lead to

\[ \Psi_1(X, Y) \leftarrow \text{parent}(X, Z), \text{parent}(Z, Y). \]

4.2. Formal Motivations

This subsection will give some hints of the formal proof (under some conditions) of the equivalence between the minimal model of the modified program and the subset of the minimal model of the original program which satisfies the ICs.

Let us suppose to have a program \( P \) and a set \( \text{IC} = \{ \text{IC}_i \} \). The modified program \( \text{mod}_{\text{IC}}(P) \) can be obtained from \( P \) through a sequence of programs \( P_i = \text{mod}_{\text{IC}_i}(P_{i-1}) \) (where \( P_0 = P \)) after considering the integrity constraint \( \text{IC}_i \). The last \( \text{IC}_i \) will produce \( \text{mod}_{\text{IC}}(P) \).
It is easy to prove that this construction is equivalent to the construction described in Section 4.1. In the following we will suppose we have just one IC and we will write $\text{mod}(P)$ instead of $\text{mod}_{\text{IC}}(P)$.

The minimal model of $P$ can be thought of as partitioned into two sets: NS, the set of ground atoms which do not satisfy the IC, and $S$, the set of atoms satisfying the IC. Now we want to find out whether

$$M_{\text{mod}(P)} \equiv S.$$ 

Unfortunately, the above is not true for every kind of database. As an example, let $P$ be the following program:

$$p(a) \leftarrow q(X)$$
$$q(b) \leftarrow .$$

Let IC be

$$4(x) \rightarrow \neg v(y).$$

Then

$$M_P: \{p(a), q(b)\}$$
$$S: \{p(a), q(b)\},$$

while $\text{mod}(P)$ is

$$p(a) \leftarrow q(X)$$
$$q(b) \leftarrow p(Y)$$

where $M_{\text{mod}(P)} = \emptyset$.

In fact, in the general case it can be proved \([6]\) that:

**Theorem 1.**

$$M_{\text{mod}(P)} \subseteq S.$$ 

The proof of the theorem follows from

$$M_{\text{mod}(P)} \subseteq M_P$$ 

and from proving that a ground atom $A \in M_{\text{mod}(P)}$ for the construction of $\text{mod}(P)$ must satisfy the IC, i.e. $A \in S$.

A condition under which $M_{\text{mod}(P)} = S$ is that $P$ is a hierarchical DB \([5]\). But this condition is stronger than necessary. In fact, it is sufficient to constrain the form of IC formulas. That is, we can restrict the class of ICs to those formulas which do not induce in $\text{mod}(P)$ any further recursion (besides the recursion already in $P$).

This last fact can be formalized in the following way: Given a program $P$, we build the graph of the program by letting predicate symbols denote nodes, while an arc from a node $p$ to a node $q$ denotes that there is a clause in the program where $p$ is in the conclusion and $q$ is in the condition part. Thus, for example,

$$p(X) \leftarrow q(X), r(Y)$$
$$r(X) \leftarrow q(X)$$
FIGURE 1.

\[
\begin{align*}
    r(X) & \leftarrow p(X) \\
    q(X) & \leftarrow q(X) \\
    q(a) & \leftarrow \\
    r(b) & \leftarrow
\end{align*}
\]

has the graph shown in Figure 1.

**Definition.** Given a program \( P \) and its corresponding graph \( G(P) \), a predicate \( p1 \) in \( P \) depends on the predicate \( p2 \) iff there exists a path in \( G(P) \) from node \( p1 \) to node \( p2 \).

In the example in Figure 1, \( p \) depends on \( p, q, \) and \( r \); \( q \) depends on \( q \); and \( r \) depends on \( p, q, \) and \( r \).

A first property of \( M_{\text{mod}(P)} \) can be stated as follows:

**Lemma 1.** Let the IC be

\[
p(X_1, \ldots, X_n) \rightarrow \Psi(X_1, \ldots, X_n).
\]

If \( q(a_1, \ldots, a_m) \in M_P \) and \( q \) does not depend on \( p \), then \( q(a_1, \ldots, a_m) \in M_{\text{mod}(P)} \).

This lemma, proved in [6], shows that a ground atom of a predicate \( q \) that does not depend on \( p \) is not affected by the modification of the program.

We are now ready to give:

**Theorem 2.** If the IC is of the form

\[
p(X_1, \ldots, X_n) \rightarrow \Psi(X_1, \ldots, X_n)
\]

and does not depend on \( p \), then \( M_{\text{mod}(P)} = S \).

This proof follows from Theorem 1 and from proving that \( S \subseteq M_{\text{mod}(P)} \). This last proof is based on induction on the length of the SLD resolution (see [16]) of a generic atom belonging to the difference of the two models \( (M_P - M_{\text{mod}(P)}) \) and using Lemma 1. In fact, we prove that if \( A \in M_P - M_{\text{mod}(P)} \) and \( i \) is the maximum length of its SLD-resolution in \( P \), then \( A \) does not satisfy the IC, i.e., \( A \in \text{NS} \).
5. CONCLUSION

In this paper we have considered logic databases as logic programs, and tackled the problem of expressing and checking ICs.

Our proposal has been integrated into a Logic Data Base Management System called DBLOG [6].

In this framework (as we have seen), a DB is given by:

1. a set of known (ground) facts, EDB;
2. a set of deductive rules;
3. a set of integrity constraints, IC.

We suggest using an ad hoc environment to build a DB by specifying separately all of the above information.

Two methods have been presented which can be used for different purposes in the system.

The first method, i.e. the consistency proof, can be used for an existing DB that must be checked for some given property. The advantages of such a method is the ability to carry on that proof by using the same querying mechanism. Problems arise if the DB is not hierarchical.

The second method, i.e. the modified program, can be used for proving different properties of the DB stated separately. The modified program can be an internal form that generates correct answers to queries of the DB. Correctness is related to the derivability from the DB and satisfiability of the ICs. We have proved that the obtained modified program is equivalent to the original program constrained by the ICs if IC has the property of not introducing any further recursion.

Notably, in the modified program method, no condition is required on the form of either EDB or deductive rules, and the method is easily implementable [6]. On the other hand, IC formulas cannot have a more general form as in the case of the consistency proof method. Moreover, we should notice that there are cases where the modified program could make the search for a query very costly, so that thus the proof method would be more suitable. Our system for DB handling provides tools which permit both methods to be applied to different kind of integrity constraint formulas.

We are grateful to Jack Minker for useful comments and advice on a first draft. We also thank U. Montanari and J. A. Robinson for reading the paper and advising us. Our thanks go also to G. Levi, R. Barbuti, and M. Bellia for many useful discussions.

REFERENCES


