Integrity Constraints Checking In Deductive Databases

Antoni Olivé
Universitat Politècnica de Catalunya
Facultat d'Informàtica
Pau Gargallo 5
E 08028 Barcelona - Catalonia

Abstract

We propose a new method for integrity checking in deductive databases. The method augments a database with a set of transition and internal events rules,which explicitly define the insertions and deletions induced by a database update. Standard SLDNF resolution can then be used to check satisfaction of integrity constraints. The method has the full power of the methods developed so far, and its implementation in Prolog does not require any meta-interpreter. A second main advantage is that it deals with both static and dynamic integrity constraints, providing a simple and uniform approach in which both classes of integrity constraints can be defined and efficiently enforced.

1. Introduction

Deductive databases generalize relational databases by including not only facts and integrity constraints, but also deductive rules. Using these rules, new facts may be derived from facts explicitly stored. Deductive database systems include a query processing system that provide the users with an uniform interface, in which they can write queries requesting stored and/or derived facts.

An integrity constraint is a condition that a database is required to satisfy at any time. In a deductive database, integrity constraints may refer to stored and derived facts and, thus, their evaluation may involve the deductive rules that define the derived facts.

The simplest solution to integrity checking would be to evaluate each constraint whenever the database is updated. However, it is usually too costly and highly redundant, since it does not take advantage of the fact that the database satisfies the constraints prior to the update. To avoid such redundancy, all practical methods assume that the database is consistent prior to the update and, then, given a particular update, they derive simplified conditions of the constraints such that, if the database satisfies the simplified conditions, it is guaranteed that the database will be consistent after the update.

There exists a large cumulative effort in the field of integrity checking. The methods that have been developed so far differ in the kind of databases considered (for example, relational or deductive), in the kind of integrity constraints they allow and enforce, in the kind of updates they consider (for example only single insertions or deletions of facts, or complex updates) and, of course, in the particular approach taken by each method.

The first methods were developed in the context of relational databases [26,3,22,12,21]. Nicolas and Yazdanian [23] studied for the first time the problems involved in integrity checking for deductive databases. Given that an integrity constraint may refer to derived facts, it may well happen that an update needs to activate some deductive rules. This poses new problems that were surveyed by the authors. It is also noteworthy their distinction between state (or static) and transition integrity constraints. The former are to be satisfied in any state of the database, while the latter constraint the possible evolution from one state to another.

Lloyd and Topor [18,19,20] developed the first practical method for integrity checking in deductive databases. They assume that the database (including integrity constraints) before and after any updates is range-restricted. Updates considered are insertions and deletions of clauses (both facts and deductive rules) and sets of such insertions and deletions.

Kowalski, Sadri and Soper [15,25] proposed a Consistency method for checking integrity constraints in deductive databases. The method also assumes that the database is range-restricted, and allows single and multiple insertions or deletions of database clauses. It

uses a proof procedure that allows reasoning forwards from the updates, which has the effect of focussing attention only on the parts of the database and the constraints that are affected by the updates. The proof procedure extends SLDNF by allowing forward reasoning as well as backward reasoning, incorporating additional inference rules for reasoning about implicit deletions caused by changes to the database, and incorporating a generalised resolution step.

Bry, Decker and Manthey [4,5] developed a method that separates constraints enforcement into two phases: generation and evaluation. In the generation phase, which is done at compile time, update constraints are generated for each update, independently from any access to the facts of the database. In the evaluation phase, when a transaction occurs, update constraints are evaluated, and the transaction is rejected if they are not satisfied.

Das and Williams [9] have recently proposed a path finding method for integrity checking in deductive databases. The method has many points in common with the Consistency method, but it differs in the way in which computes induced updates. The method can be implemented efficiently in Prolog by taking full advantage of Prolog's computation mechanism, but it requires the use of a meta-interpreter.

We propose here a new method for integrity checking in deductive databases, which we call the Internal Events method. The method is a particular application to deductive databases of an approach that we developed [24] for the design of information systems from deductive conceptual models. The method augments a database with a set of rules, called transition and internal events rules, which explicitly define the insertions and deletions induced by a database update. Standard SLDNF resolution can then be used to check satisfaction of integrity constraints. The method has the full power of the methods developed so far, and its implementation in a Prolog system does not require any meta-interpreter. A second main advantage is that it deals with both static and transition integrity constraints, providing a uniform approach in which both classes of integrity constraints can be defined and enforced.

The paper is organized as follows. The next section defines basic concepts of deductive databases, and presents an example (taken from [15]) that will be used throughout the paper. Section 3 defines the key concept of internal event, and presents a method for deriving transition and internal events rules. Section 4 discusses the application of these rules for integrity constraint checking when the database is updated. Updates considered are single insertion or deletion of database clauses (including integrity constraints) or sets of them. Section 5 extends the method to transition integrity constraints. In section 6 we compare our method with some of the methods presented above. Finally, in section 7 we present our conclusions. We assume the reader is familiar with logic programming.

2. Deductive databases

A deductive database D consists of three finite sets: a set F of facts, a set R of deductive rules, and a set I of integrity constraints. A relational database is a deductive database without deductive rules. A fact is a ground atom. The set of facts is called the extensional database, and the set of deductive rules is called the intensional database.

We assume that database predicates are either base or derived. A base predicate appears only in the extensional database and (eventually) in the body of deductive rules. A derived predicate appears only in the intensional database. Every database can be defined in this form [2,7].

Our database example contains four base predicates:

Cit(x)	x is a citizen
Emp(x)	x is an employee
Ra(x)	x is a registered alien
Cr(x)	x has criminal record

and a derived predicate:

Rr(x)	x has the right of residence

2.1 Deductive rules

A deductive rule is a formula of the form:

$A \leftarrow L_1 \wedge ... \wedge L_n$ with $n \geq 1$

where A is an atom denoting the conclusion, and the $L_1, ..., L_n$ are literals representing conditions. Each L_i is either an atom or a negated atom. Any variables in A, $L_1, ..., L_n$ are assumed to be universally quantified over the whole formula. The terms in the conclusion must be distinct variables, and the terms in the conditions must be variables or constants.

As usual [5,9,25], we require that the database before and after any updates is allowed, that is any variable that occurs in a deductive rule has an occurrence in a positive condition of the rule. This ensures that all negative conditions can be fully instantiated before they are evaluated by the "negation as failure" rule.

In the example, we assume the database has the following deductive rules, defining predicate Rr:

DR.1 $Rr(x) \leftarrow Ra(x) \wedge \neg Cr(x)$

DR.2 $Rr(x) \leftarrow Cit(x)$

2.2 Integrity constraints

An integrity constraint is a closed first-order formula that the database is required to satisfy. We deal with constraints that have the form of a denial:

$\leftarrow L_1 \wedge ... \wedge L_n$ with $n \geq 1$

where the L_i are literals and variables are assumed to be universally quantified over the whole formula. More general constraints can be transformed into this form as described in [17]. For the sake of uniformity, we (as in [9,13]) associate to each integrity constraint an inconsistency predicate Icn, with or without terms, and thus it has the same form as the deductive rules. We call them integrity rules.

The above are called static integrity constraints, since they must be satisified in any state of the database. Constraints involving two or more database states, which are called transition integrity constraints, will be considered in section 5.

In the example, the database contains the constraint:

$Ic1(x) \leftarrow Emp(x) \wedge \neg Rr(x)$

stating that employees must have the right of residence.

3. Transition and internal events rules

We now define the concept of internal events, a key concept in our method. We also explain how to derive the transition and internal events rules for a given database. These rules depend only on the deductive rules and integrity constraints of the database. They are independent from the base facts stored in the database, and from any particular update. In a later section, we will discuss the use of these rules for integrity constraints enforcement.

3.1 Internal events

Let D be a database, U an update and D' the updated database. We say that U induces a transition from D (the current state) to D' (the new, updated state). We assume for the moment that U consists of an unspecified set of base facts to be inserted and/ or deleted.

Due to the deductive rules, U may induce other updates on some derived predicates. Let P be a derived predicate in D, and let P' denote the same predicate evaluated in D'. Assuming that a fact P(K) holds in D, where K is a vector of constants, two cases are possible:

a.1. P'(K) also holds in D' (both P(K) and P'(K) are true).

a.2. P'(K) does not hold in D' (P(K) is true but P'(K) is false).

and assuming that P'(K) holds in D', two cases are also possible:

b.1 P(K) also holds in D (both P(K) and P'(K) are true).

b.2. P(K) does not hold in D (P'(K) is true but P(K) is false).

In case a.2 we say that a deletion internal event occurs in the transition, and we denote it by $\delta P(K)$. In case b.2 we say that occurs an insertion internal event, and denote it by $\iota P(K)$. Thus, for example, $\iota Rr(Mary)$ denotes an insertion internal event corresponding to predicate Rr: Rr(Mary) is true after the update and was false before.

Formally, we associate to each base, derived or inconsistency predicate P an insertion internal events predicate ιP and a deletion internal events predicate δP, defined as:

(1) $\forall x(\iota P(x) \leftrightarrow P'(x) \wedge \neg P(x))$

(2) $\forall x(\delta P(x) \leftrightarrow P(x) \wedge \neg P'(x))$

where x is a vector of variables. From the above, we have:

(3) $\forall x(P'(x) \leftrightarrow (P(x) \wedge \neg \delta P(x)) \vee \iota P(x))$

(4) $\forall x(\neg P'(x) \leftrightarrow (\neg P(x) \wedge \neg \iota P(x)) \vee \delta P(x))$

If P is a base predicate, then ιP facts and δP facts represent insertions and deletions of base facts, respectively. Therefore, we assume from now on that U consists of an unspecified set of insertion and/or deletion internal events of base predicates. A concrete example could be U = {ιCit(John), δCr(Alan)}, consisting of an insertion and a deletion. Note that by (1) and (2) above, we require:

(5) $\forall x(\iota P(x) \rightarrow \neg P(x))$ and

(6) $\forall x(\delta P(x) \rightarrow P(x))$

also to hold for base predicates.

If P is an inconsistency predicate, then ιP facts that occur during the transition will correspond to violations of its integrity constraint. Thus if a given transition induces, for example, a fact ιIc1(Alan) this will mean that such transition leads to a violation of integrity constraint Ic1 for Alan. Note that, for inconsistency predicates, δP facts cannot happen in any transition, since we assume that the database is consistent before the update and, thus, P(x) is always false.

3.2 Transition rules

Let us take a derived or inconsistency predicate P of the database. The definition of P consists of the rules in the database having P in the conclusion. Assume that there are m ($m \geq 1$) such rules. For notation's sake, we rename the conclusions of the m rules by $P_1, ..., P_m$, change the implication by an equivalence and add the set of clauses:

$P \leftarrow P_i$ $\quad i = 1...m$

In the example, rules DR.1 and DR.2, which define predicate Rr, will be rewritten as:

DR.1' $Rr_1(x) \leftrightarrow Ra(x) \wedge \neg Cr(x)$
DR.2' $Rr_2(x) \leftrightarrow Cit(x)$
DR.1.1 $Rr(x) \leftarrow Rr_1(x)$
DR.1.2 $Rr(x) \leftarrow Rr_2(x)$

Given a rule $P_i(x) \leftrightarrow L_1 \wedge ... \wedge L_n$, we will denote by $U(P_i)$ the conjunction of the literals in the body having a vector of variables y such that $x \supseteq y$, and denote by $E(P_i)$ the conjunction of the literals in the body having some variable which is not in x. Thus, in the above rules, we will have $U(Rr_1) = Ra(x) \wedge \neg Cr(x)$, $E(Rr_1) = \emptyset$, $U(Rr_2) = Cit(x)$ and $E(Rr_2) = \emptyset$.

Consider now one of the rules $P_i(x) \leftrightarrow L_1 \wedge ... \wedge L_n$. When the rule is to be evaluated in the updated state its form is $P'_i(x) \leftrightarrow L'_1 \wedge ... \wedge L'_n$. Now if we replace each literal in the body by its equivalent definition in terms of the current state (before update) and the internal events, we get a new rule, called a *transition rule*, which defines predicate P'_i (new state) in terms of current state predicates and internal events.

More precisely, if L'_j is a positive literal $Q'_j(x_j)$ we apply (3) and replace it by:

$$(Q_j(x_j) \wedge \neg \delta Q_j(x_j)) \vee \iota Q_j(x_j)$$

and if L'_j is a negative literal $\neg Q'_j(x_j)$ we apply (4) and replace it by:

$$(\neg Q_j(x_j) \wedge \neg \iota Q_j(x_j)) \vee \delta Q_j(x_j)$$

It will be easier to refer to the resulting expressions if we denote by:

$$\begin{aligned} O(L'_j) &= (Q_j(x_j) \wedge \neg \delta Q_j(x_j)) && \text{if } L'_j = Q'_j(x_j) \\ &= (\neg Q_j(x_j) \wedge \neg \iota Q_j(x_j)) && \text{if } L'_j = \neg Q'_j(x_j) \\ N(L'_j) &= \iota Q_j(x_j) && \text{if } L'_j = Q'_j(x_j) \\ &= \delta Q_j(x_j) && \text{if } L'_j = \neg Q'_j(x_j) \end{aligned}$$

With this notation we then have:

(7) $$P'_i(x) \leftrightarrow \bigwedge_{r=1}^{n} [O(L'_r) \vee N(L'_r)]$$

After distributing $\wedge$ over $\vee$, we get an equivalent set of transition rules, each of them with the general form:

(8) $$P'_{i,j}(x) \leftrightarrow \bigwedge_{r=1}^{n} [O(L'_r) \mid N(L'_r)] \qquad \text{for } j = 1,...,2^n$$

where n is the number of literals in the P_i rule, and

(9) $$P'_i(x) \leftarrow P'_{i,j}(x) \qquad j = 1,..., 2^n$$

In the above set of rules (8) it will be useful to assume that the rule corresponding to j = 1 is:

(10) $$P'_{i,1}(x) \leftrightarrow O(L'_1) \wedge ... \wedge O(L'_n)$$

The transition rules corresponding to the database example are:

T.1 $Rr'_{1,1}(x) \leftarrow Ra(x) \wedge \neg \delta Ra(x) \wedge \neg Cr(x) \wedge \neg \iota Cr(x)$
T.2 $Rr'_{1,2}(x) \leftarrow Ra(x) \wedge \neg \delta Ra(x) \wedge \delta Cr(x)$
T.3 $Rr'_{1,3}(x) \leftarrow \iota Ra(x) \wedge \neg Cr(x) \wedge \neg \iota Cr(x)$
T.4 $Rr'_{1,4}(x) \leftarrow \iota Ra(x) \wedge \delta Cr(x)$
T.5 $Rr'_{2,1}(x) \leftarrow Cit(x) \wedge \neg \delta Cit(x)$
T.6 $Rr'_{2,2}(x) \leftarrow \iota Cit(x)$
T.7 $Ic1'_{1,1}(x) \leftarrow Emp(x) \wedge \neg \delta Emp(x) \wedge \neg Rr(x) \wedge \neg \iota Rr(x)$
T.8 $Ic1'_{1,2}(x) \leftarrow Emp(x) \wedge \neg \delta Emp(x) \wedge \delta Rr(x)$
T.9 $Ic1'_{1,3}(x) \leftarrow \iota Emp(x) \wedge \neg Rr(x) \wedge \neg \iota Rr(x)$
T.10 $Ic1'_{1,4}(x) \leftarrow \iota Emp(x) \wedge \delta Rr(x)$

Each of the above rules has a clear intuitive meaning. Thus, for example, T.1 states that x has the right of residence in the new state ($Rr'_{1,1}(x)$), if x was a registered alien in the old state ($Ra(x)$), and this fact has not been deleted in the transition ($\neg \delta Ra(x)$), and x did not have a criminal record in the old state ($\neg Cr(x)$), and a criminal record for x has not been inserted during the transition ($\neg \iota Cr(x)$). Recall that, by (9) and DR 1.1, $Rr'(x) \leftarrow Rr'_{1,1}(x)$.

3.3 Insertion internal events rules

Let P be a derived predicate. Insertion internal events for P were defined in (1) as:

$$\forall x(\iota P(x) \leftrightarrow P'(x) \wedge \neg P(x))$$

If there are m rules for predicate P, then $P'(x) \leftrightarrow P'_1(x) \vee ... \vee P'_m(x)$, and replacing $P'(x)$ we obtain the set of rules: $\iota P(x) \leftarrow P'_i(x) \wedge \neg P(x)$ with $i = 1 ... m$ and replacing again $P'_i(x)$ by its equivalent definition given in (9) we get:

(11) $$\iota P(x) \leftarrow P'_{i,j}(x) \wedge \neg P(x) \qquad \text{for } i = 1 ... m \text{ and } j = 1 ... 2^n$$

Rules (11) above are called *insertion internal events rules* of predicate P. They allow us to deduce which ιP facts (induced insertions) happen in a transition. If P is an inconsistency predicate, ιP facts correspond to violations of an integrity constraint, and the rules (11) define how we can derive them in terms of the current state and other internal events.

We can remove some of these rules and, in some cases, we can simplify them. Thus, it is easy to prove that for any i, the rule corresponding to j = 1 cannot produce ιP facts, since in this case $P'_{i,1}(x) \rightarrow P(x)$. We can then reduce the set (11) to:

(11') $$\iota P(x) \leftarrow P'_{i,j}(x) \wedge \neg P(x) \qquad \text{for } i = 1 ... m \text{ and } j = 2 ... 2^n$$

If P is an inconsistency predicate we can remove in (11') the literal $\neg P(x)$ since we assume that $P(x)$ is false, for all x, in the current state.

In [24a]we prove two interesting simplifications, that can be applied to rules in (11') for which the transition rule corresponding to $P'_{i,j}(x)$ has some literal $N(L'_h)$ in $U(P'_{i,j})$, with $U(P_i) \neq \emptyset$. In this case, we can remove predicate P_i from P and the rule becomes:

$$(12)\quad \iota P(x) \leftarrow P'_{i,j}(x) \wedge \neg P_1(x) \wedge \ldots \wedge \neg P_{i-1}(x) \wedge \neg P_{i+1}(x) \wedge \ldots \wedge \neg P_m(x)$$

In the example, the insertion internal events rules are:

I1...3 $\iota Rr(x) \leftarrow Rr'_{1,j}(x) \wedge \neg Rr_2(x) \quad j = 2 \ldots 4$

I4 $\iota Rr(x) \leftarrow Rr'_{2,2}(x) \wedge \neg Rr_1(x)$

I5...7 $\iota Ic1(x) \leftarrow Ic1'_{1,j}(x) \quad j = 2 \ldots 4$

The intuitive meaning of these rules is also clear. Take, for example, I.3. It states that the right of residence of x is inserted during a transition ($\iota Rr(x)$) if x has this right in the new state by rule $Rr'_{1,4}$ (T.4) and x did not have this right in the old state by rule Rr_2 (DR .2'). Note that rule T.4 states that x has the right of residence in the new state if x is inserted as a registered alien during the transition ($\iota Ra(x)$) and his criminal record is deleted ($\delta Cr(x)$).

3.4 Deletion internal events rules

Let P be a derived predicate. Deletion internal events were defined in (2) as:

$\forall x(\delta P(x) \leftrightarrow P(x) \wedge \neg P'(x))$

If there are m rules for predicate P, we then have:

$\delta P(x) \leftarrow P_i(x) \wedge \neg P'(x) \quad \text{for } i = 1 \ldots m$

and replacing $P'(x)$ by its equivalent definition $P'(x) \leftrightarrow P'_1(x) \vee \ldots \vee P'_m(x)$ we obtain:

$$(16)\quad \delta P(x) \leftarrow P_i(x) \wedge \neg P'_1(x) \wedge \ldots \wedge \neg P'_i(x) \wedge \ldots \wedge \neg P'_m(x) \quad \text{for } i = 1 \ldots m$$

Replacing now, in each of these rules, $P'_i(x)$ by its equivalent definition given in (9) and after a number of simple transformations described in [24a], we get the set of rules:

For i = 1 ... m

If $U(P_i) = L_1 \wedge \ldots \wedge L_q$

For j = 1...q

$$(17)\quad \delta P(x) \leftarrow L_1 \wedge \ldots \wedge L_{j-1} \wedge [\delta Q_j(x_j) | \iota Q_j(x_j)] \wedge L_{j+1} \wedge \ldots \wedge L_q \wedge E(P_i) \wedge \neg P'_1(x) \wedge \ldots \wedge \neg P'_{i-1}(x) \wedge \neg P'_{i+1}(x) \wedge \ldots \wedge \neg P'_m(x)$$

where the first option is taken if $L_j = Q_j(x_j)$ is positive and the second if negative

If $E(P_i) = L_{q+1} \wedge \ldots \wedge L_n$

For j = q+1...n

$$(18)\quad \delta P(x) \leftarrow U(P_i) \wedge L_{q+1} \wedge \ldots \wedge L_{j-1} \wedge [\delta Q_j(x_j) | \iota Q_j(x_j)] \wedge L_{j+1} \wedge \ldots \wedge L_n \wedge \neg E(P'_{i,1}) \wedge \ldots \wedge \neg E(P'_{i,2}n) \wedge \neg P'_1(x) \wedge \ldots \wedge \neg P'_{i-1}(x) \wedge \neg P'_{i+1}(x) \wedge \ldots \wedge \neg P'_m(x)$$

where the first option is taken if $L_j = Q_j(x_j)$ is positive and the second if negative

Rules (17) and (18) are called *deletion internal events rules* of predicate P. They allow us to deduce which δP facts (induced deletions) happen in a transition. In the example, these rules are:

D.1 $\delta Rr(x) \leftarrow \delta Ra(x) \wedge \neg Cr(x) \wedge \neg Rr'_2(x)$

D.2 $\delta Rr(x) \leftarrow Ra(x) \wedge \iota Cr(x) \wedge \neg Rr'_2(x)$

D.3 $\delta Rr(x) \leftarrow \delta Cit(x) \wedge \neg Rr'_1(x)$

Note again the meaning of these rules. For example, D.1 defines that the right of residence of x is deleted during a transition ($\delta Rr(x)$) if x is deleted as a registered alien ($\delta Ra(x)$), and x did not have a criminal record in the old state ($\neg Cr(x)$), and x has not that right in the new state according to Rr'_2.

3.5 The augmented database

Let D be a database. We call *augmented database*, or A(D), the database consisting of D, its transition rules and its internal events rules. In the next section we will discuss the important role of A(D) in our method for integrity constraints enforcement. But before closing this section, we want to comment on the properties of A(D) with respect to those of D. In particular, given that our method is based on SLDNF-resolution, we are interested in the syntactic properties related to the completeness of SLDNF-resolution.

SLDNF-resolution is incomplete, in general, but there are large classes of logic programs (databases) and goals for which it is complete (or have been conjectured complete). Thus, Clark [8] proved completeness for hierarchical and allowed databases. Cavedon and Lloyd [6] showed completeness for databases and goals which are allowed, strict and stratified, and Kunen [16] for databases and goals which are allowed, strict and call-consistent. More recently, Decker and Cavedon [11] have proved completeness for databases and goals which are call-consistent, even and recursively covered (a generalization of allowed).

It is easy to show (see [24a]) the following relationships between the properties of D and those of A(D):

a) If D is hierarchical (resp., call-consistent) then A(D) is also hierarchical (resp., call-consistent)

b) If D is stratified then A(D) is call-consistent.

c) If $D \cup \{\leftarrow Icn(x)\}$ is strict (resp., even) then $A(D) \cup \{\leftarrow \iota Icn(x)\}$ is also strict (resp., even). (the meaning of the goals will become clear in the next section).

d) If D is allowed then A(D) is also allowed.

Therefore, if the properties of D are such that SLDNF-resolution is complete, according to the above mentioned completeness results, then it will also be complete for A(D).

4. Integrity constraints enforcement

The transition and internal events rules described above can be used directly to verify that a transaction does not produce inconsistencies.

Let D be a database, A(D) the augmented database, and T a transaction consisting of a set of base internal events. If T leads to an inconsistency then some of the ιIcj facts will be true in the transition. Using the SLDNF proof procedure, T violates integrity constraint Icj if the goal $\leftarrow \iota Icj(x)$ succeeds from input set $A(D) \cup T$. If every branch of the SLDNF-search space for $A(D) \cup T \cup \{\leftarrow \iota Icj(x)\}$ is a failure branch, then T does not violate Icj.

Thus, our method for integrity constraints enforcement is entirely based on the use of standard SLDNF resolution. We take as input set $A(D) \cup T$ and, for each integrity constraint Icj, $\{\leftarrow \iota Icj(x)\}$ as goal. Transaction T leads to an inconsistent database state if there is a refutation for some of the above goals. Otherwise, T can be accepted, and the database is then updated. Note that in our method the database is updated after verification of constraint satisfaction.

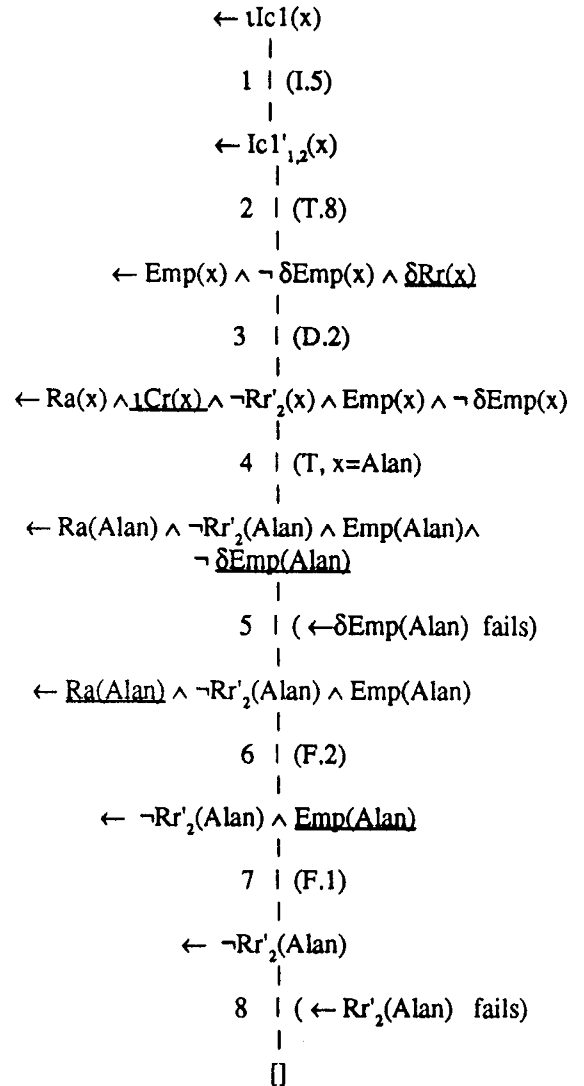
In what follows we illustrate the method and comment some implementation details with an example.

4.1 Insertion or deletion of base facts

Assume that a transaction T consists either of an insertion or deletion of a single base fact, and let us denote it by $T = \{\iota Q(\mathbf{K})\}$ or $T = \{\delta Q(\mathbf{K})\}$, respectively, with Q being a base predicate and **K** a vector of constants. We say that $\iota Q(\mathbf{K})$ or $\delta Q(\mathbf{K})$ are the input internal events given or produced by the update. Recall that by (4) and (5) we require verifying that the database does not contain (if insertion), or contains (if deletion) $Q(\mathbf{K})$ prior to the inconsistency analysis. The following example shows the application of our method.

Example 4.1.a

Assume the database has the following facts:
F.1. Emp(Alan)
F.2. Ra(Alan)
and let the transaction be the insertion of Cr(Alan), which we denote by:
T. ιCr(Alan)
The following refutation shows that T violates Ic1, with x = Alan (the subsidiary trees showing the failure of goals $\leftarrow \delta Emp(Alan)$ and $\leftarrow Rr'_2(Alan)$ are not included).



Some obvious simplifications in the transition and internal events rules may reduce the search space. For example, we may combine rules I.5 and T.8 into a single rule:

$\iota Ic1(x) \leftarrow Emp(x) \wedge \neg \delta Emp(x) \wedge \delta Rr(x)$
thus reducing in one step the derivations.

Note that in the third step we have selected literal $\delta Rr(x)$ instead of $Emp(x)$ or $\neg \delta Emp(x)$. Given that in most real databases the number of facts is likely to be much greater than the number of internal events produced in a transition, it seems convenient to use the strategy of selecting first the internal events (once fully instantiated if they are negative).

4.2 Other updates

Our method can also be applied with complex updates, such as

- Insertion or deletion of qualified base facts.
- Insertion or deletion of deductive rules.
- Insertion or deletion of integrity constraints.
- Transaction with multiple updates

In [24a] we explain the details of the method in these cases, and show its application with examples.

5. Transition integrity constraints

Up to now, we have only dealt with static integrity constraints. Our method, however, can also handle transition integrity constraints. A transition integrity constraint is a closed first-order formula that database transitions are required to satisfy. We only consider here transitions between two successive states.

Transition integrity constraints in deductive databases were formalized for the first time by Nicolas and Yazdanian [23]. Their approach consists in extending the database with new relations (predicates), called "action relations". Three action relations are associated with each relation R in the database, one for each of the three operations: update, deletion and insertion. They are denoted UPD-R, DEL-R and ENT-R, respectively.

If R is a n-ary relation, then UPD-R is a 2n-ary relation. The meaning of a tuple:

$$<a_1,\dots,a_n,a'_1,\dots,a'_n> \in \text{UPD-R}$$

is that the tuple $<a_1,\dots,a_n> \in R$ is being updated in a transition and the new value is $<a'_1,\dots,a'_n>$. The use of the UPD-R action relation in the formalization of transition constraints can be illustrated by means of the following example (taken from [23]).

TIC: Let FS(p,s) be a relation giving the family status s of a person p and let VALT(s,s1) be a relation giving the valid transitions in family status (such as VALT(single,married)). The constraint that changes is family status must be valid can be defined:

$\forall p,s,p',s'\ (\text{UPD-PAS}(p,s,p',s') \land p = p' \land s \neq s' \rightarrow \text{VALT}(s,s'))$

Once the transition constraints have been formalized in this way, they can be enforced using any method applicable to static constraints, provided the method computes adequately the induced updates.

In our method, both input and induced updates of a predicate P are represented explicitly, by means of the ιP and δP predicates. This suggests that, for us, a transition constraint is a condition $\leftarrow L_1 \land \dots \land L_n$ (with $n \geq 1$) like a static constraint, but where literals can be not only database predicates, but also internal events predicates. Database predicates in the condition have to be evaluated in the current state of the database, and not in the state after the update. This implies that we don't have to transform transition constraints, as we had to for static constraints, and that the associated transition constraint predicate will have the nature of an internal event. Therefore, in our method a transition constraint Ticn is a rule with the following general form:

$\iota \text{Ticn} \leftarrow L_1 \land \dots \land L_n$ with $n \geq 1$

We give below the definition of the above transition constraint example in our formalization.

(TIC) $\iota \text{Tic} \leftarrow \delta FS(p,s) \land \iota FS(p,s1) \land \neg VALT(s,s1)$

Enforcement of a transition constraint is not different from the static case. A transaction T will violate a transition integrity constraint Ticj if the goal $\leftarrow \iota \text{Ticj}(x)$ succeeds from input set $A(D) \cup T$. If every branch of the SLDNF-search space for $A(D) \cup T \cup \{\leftarrow \iota \text{Ticj}(x)\}$ is a failure branch, then T does not violate Ticj. The following example shows the application of the method in this case.

Example 5.a

Assume the database contains the following facts:

F.1 Cit(Alan)

F.2 Emp(Alan)

and the transition constraint (The right of residence of an employee can not be deleted):

TIC.1 $\iota \text{Tic1}(x) \leftarrow Emp(x) \land \delta Rr(x)$

Let the transaction be the deletion of Cit(Alan), that is:

T. δCit(Alan)

The following refutation shows that the transaction violates the constraint, with x = Alan.

$\leftarrow \iota \text{Tic1}(x)$

| (TIC.1)

$\leftarrow Emp(x) \land \underline{\delta Rr(x)}$

| (D.3)

$\leftarrow \underline{\delta Cit(x)} \land \neg Rr'_1(x) \land Emp(x)$

| (T, x = Alan)

$\leftarrow \neg Rr'_1(Alan) \land \underline{Emp(Alan)}$

| (F.2, x = Alan)

$\leftarrow \neg Rr'_1(Alan)$

| ($\leftarrow \neg Rr'_1(Alan)$ fails)

[]

6. Comparison with other methods

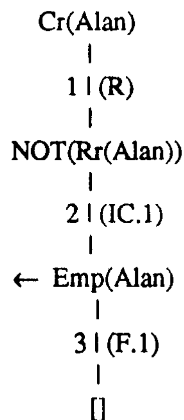
In this section we compare in detail our approach to integrity checking in deductive databases with the approaches taken by some of the methods mentioned in the Introduction. All of these methods deal with the same class of databases, static integrity constraints an updates as ours. However, they don't handle transition integrity constraints.

6.1 The Consistency method [15,25]

The Consistency method has inspired our strategy in integrity constraints enforcement (Section 4). The distinction we make between the database and its transition and internal events rules allow us to use the standard SLDNF procedure. Without this distinction, the Consistency method needs to extend SLDNF by:
- Allowing forward reasoning as well as backward reasoning
- Incorporating additional inference rules for reasoning about deletions caused by changes to the database, and
- Incorporating a generalised resolution step, which is needed for reasoning forward from negation as failure.

The significance of this difference can be made apparent by means of a simple example. We use the database example given in Section 2, and the facts and transaction given in example 4.1.a.

The following search space illustrates the violation of IC.1 in the updated database.



The first step of the above refutation uses an inference rule (R) that allows reasoning as follows:

(R) **because** in the updated database
Cr(Alan) holds and we have
Rr(x) ← Ra(x) and NOT(Cr(x))
and we have no other way of proving Rr(Alan) and
Rr(Alan) was provable in the database
then Rr is deleted

Thus NOT(Rr(Alan)) is provable in the updated database.

The second step is an "extended" resolution of clauses:
<u>NOT(Rr(Alan))</u> and
← Empx) and <u>NOT(Rr(x))</u>
on the underlined literals. The resolvent is ← Emp(Alan).
The last step is a standard SLDNF step.

Comparing this solution with ours (given in example 4.1.a) we note several differences. The first, minor, difference is in the top clause used in both methods. In our method the top clause is the goal {← ιIc1(x)}, while in the Consistency method the top clause is the update {Cr(Alan)}. The Consistency method could also choose the constraint as top clause, but then the search space would be somewhat larger, because it would not take advantage of the assumption that the constraint was satisfied before the update [25]. In our method we don't take ← Ic1(x) as top clause, but ← ιIc1(x), which means that we only search for violations of Ic1 produced in the transition.

The major difference we see lies in the inference rule (R) required in the Consistency method for reasoning about implicit deletions. Its implementation in Prolog systems requires a meta-interpreter, which may cause a significant overhead. In our method we achieve the same reasoning capabilities, while still remaining in the SLDNF framework. It is interesting to observe how we infer that Rr(Alan) has been implicitly deleted. Rule D.2 (used in step 3) states that δRr(Alan) if Ra(Alan) and ιCr(Alan) and ¬Rr'$_2$(Alan). Given that ιCr(Alan) is the input internal event, step 4 reduces the goal to Ra(Alan) and ¬Rr'$_2$(Alan). Ra(Alan) is fact F.2 in the database and, thus, step 6 reduces again the goal to ¬Rr'$_2$(Alan), which succeeds in step 8, because Rr'$_2$(Alan) fails (Cit(Alan) and δCit(Alan) are not provable).

Finally, we note that our search space includes a step 5 to check that Emp(Alan) has not been deleted in the transition. There is not a similar step in the Consistency method. The reason for this lies in the fact that we have a single search space for each transaction, independent of the updates induced in it, while in the Consistency method there is a search space for each update in the transaction. In this particular case, if the transaction were:

Insert Cr(Alan)
Delete Emp(Alan)

no violation of the constraint would occur. This makes our search space somewhat larger for simple transactions. However, if we know the transaction types in advance we can do some preparatory work at compile time, which can restrict significantly the search space.

6.2 The method of Bry, Decker and Manthey [4,5]

This method distinguishes clearly two phases in integrity checking:
- a generation phase, in which "update constraints" are derived for every possible update.
- an evaluation phase, in which the update constraints are evaluated.

Update constraints are derived at compile time, involving only the rules and not the facts of the database. There is a set of update constraints for every update. Let U be a ground single-fact update, positive (insertion) or negative (deletion). Due to the deductive rules, U may induce other updates. However, in the generation phase the method only computes the potential updates induced by U. A literal L is a potential update induced by U iff L directly depends on U (through a deductive rule) or on a literal that is a potential update induced by U. If, for example, the database contains the deductive rules:

R.1 $R(x) \leftarrow P(x,y) \wedge Q(y)$

R.2 $S(x) \leftarrow R(x) \wedge \neg T(x)$

then the potential updates induced by P(A,B) are R(A) and S(A), and the potential updates induced by Q(A) are R(x) and S(x). Every induced update is an instance of a potential update, depending on the facts stored in the database, but there may be potential updates no instance of which is an induced update.

Now, let L be an update or an induced update, and C be a constraint that may be violated by L. Bry et al. apply Nicolas' method [22] to derive simplified instances s(C) of C, such that if s(C) are satisfied in the updated database so will be C. If, for example, C1 is $\forall x\ [\neg R(x) \vee V(x)]$ and the update is R(A) then s(C1) is V(A). But if C2 is $\exists x\ [S(x)]$ and the update is $\neg S(A)$ then s(C2) = C2, and no simplification is possible.

The general form of an update constraint for a literal L is the universal closure of the formula:

$$\text{delta}(U,L) \rightarrow \text{new}(U,s(C))$$

where:
- U denotes an update
- C is a constraint relevant to L
- s(C) is a simplified instance of C
- delta is a meta-predicate such that delta(U,L) holds if L is satisfied in the updated database, but not before.
- new is a meta-predicate such that new(U,F) holds if F holds in the updated database.

Given an update U, the updated database will satisfy all integrity constraints iff they are satisfied prior to the update, and every update constraint for U or for a potential update induced by U is satisfied after the update. These update constraints are evaluated when an update occurs. Implementation of the predicates delta and new in Prolog can be made easily by means of meta-interpreters.

We see three noteworthy differences beteween this method and the method proposed here. First, as said before, we do not require a meta-interpreter. The "delta" predicate corresponds to our internal events rules, and we use SLDNF resolution to evaluate the "new" predicate. Second, this method may do some redundant work in the evaluation of update constraints. Thus, in rules R.1 and R.2 above, with the update Q(A), if both R(x) and S(x) are relevant for some constraints, then an update constraint will evaluate delta(Q(A),R(x)), computing all induced updates of predicate R, and another update constraint will evaluate delta(Q(A),S(x)), computing all induced updates of predicate S, which in fact requires re-computing the former induced updates. In our method, we define $\iota S(x)$ in terms of $\iota R(x)$, thus avoiding this redundancy.

Finally, the advantage of the method of Bry et al. lies in its distinction beteween the two phases. It would be interesting to explore this distinction in our method. A possible idea could be to generate, at compile time, a reduced set of transition and internal events rules for a given update. The concept of potential induced update could be used to restrict the search space.

7. Conclusions

In this paper, we have presented a method for checking integrity constraints in deductive databases. The method deals with range-restricted databases and with integrity constraints in denial form. Updates considered are insertion or deletion of a base fact, qualified base facts, deductive rules and integrity constraints. Transaction with multiple updates are also allowed.

The method augments the database with a set of transition and internal events rules. These rules play an important role in integrity constraints enforcement, since they define explicitly the induced updates caused by a database update. We can then use the standard SLDNF procedure to check consistency and, in this way, the method can be implemented directly in Prolog. However other methods could be used as well. Some optimization techniques [7] can be easily incorporated into our method.

The main cost implied by our method is the space required to store the transition and internal events rules. The cost may only be important in databases with a large number of deductive rules and integrity constraints. In most practical databases, however, this number is small as compared to the number of base facts stored and, thus, the cost should not be significant.

The gain is the time saved in checking integrity constraints satisfaction, since our method does not require a meta-interpreter. We find here again the classical tradeoff between space and time.

On the other hand, our method also handles transition integrity constraints, which extend the range of constraints a database may be subjected to. These constraints have received little attention in the past, probably because efficient methods for their enforcement were not available. We believe that our method provides a simple and elegant approach to the enforcement of transition integrity constraints, which is as efficient as for static constraints.

Acknowledgment

I wish to thank D. Costal, J.A. Pastor, C. Quer, M.R. Sancho, T. Sales, J. Sistac, E. Teniente and T. Urpí for their comments and suggestions on an earlier draft of this paper. This work has been supported by the CICYT PRONTIC program, project TIC 680.

References

1. Apt,K.R.; Blair, H.A.; Walker,A. "Towards a theory of declarative knowledge". In Minker,J. (Ed.) "Foundations of deductive databases and logic programming", Morgan Kaufmann Pub., 1988, pp. 89-148.
2. Bancilhon,F.; Ramakrishnan,R. "An amateur's introduction to recursive query processing strategies". Proc. ACM SIGMOD Int. Conf. on Management of data. Washington D.C., May 1986, pp. 16-52.
3. Bernstein,P.A.;Blaustein,B.;Clarke,E. "Fast maintenance of integrity assertions using redundant aggregate data". Proc. of the 6th. Intl. Conf. on Very Large Data Bases, October 1980, pp. 126-136
4. Bry,F.;Decker,H. "Préserver l'intégrité d'une base de données deductive: une methode et son implementation". In Compte-rendu des 4èmes Journées Base de Données Avancées, Mai 1988, Bénodet, France (in french).
5. Bry,F.;Decker,H.;Manthey,R. "A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases". Proc. of Extending Database Technology, Venice (1988), pp. 488-505.
6. Cavedon,L.;Lloyd,J.W. "A completeness theorem for SLDNF-Resolution". TR 87/9, Dept. of Comp. Sc., University of Melbourne, Australia, 1987.
7. Chakravarthy,U.S.;Grant,J.;Minker,J. "Foundations of semantic query optimization for deductive databases". In Minker,J. (Ed.) "Foundations of deductive databases and logic programming", Morgan Kaufmann Pub., 1988, pp. 243-273.
8. Clark,K.L. "Negation as failure". In Gallaire, H.; Minker,J. (Eds.), "Logic and data bases", Plenum Press, New York, pp. 293-322.
9. Das,S.;Williams,H. "A path finding method for constraint checking in deductive databases". Data & Knowledge Engineering, 1989, No.4, pp. 223-244.
10. Decker,H. "Integrity enforcement on deductive databases", In Kerschberg, L.(Ed.) Proc. of the first Int. Conf. on Expert Database Systems, Charleston, South Caroliona (April 1986), pp. 271-285.
11. Decker,H.;Cavedon,L. "Generalizing syntactic properties which ensure the completeness of SLDNF-resolution". ECRC IR-KB-52, Nov. 1989.
12. Henschen,L.; McCune,W.; Naqvi,S. "Compiling constraint-checking programs from first-order formulas", In Gallaire,H.; Minker,J. Nicolas,J-M. (Eds.),"Advances in Database theory", Vol.2, 1984, pp. 145-170.
13. Kowalski,R. "Logic for data description". In Gallaire,H.;Minker,J. (Eds.) "Logic and Data Bases", Plenum Press, New York, 1978, pp. 77-103.
14. Kowalski,R. "Logic for problem solving". North-Holland, 1979.
15. Kowalski,R.; Sadri,F.; Soper,P. "Integrity checking in deductive databases". Proc. of the 13th. VLDB Conference, Brighton 1987, pp. 61-69.
16. Kunen,K. "Signed data dependencies in logic programs". TR 719, Comp. Sc. Dept., Univ. Wisconsin, 1987.
17. Lloyd,J.W.;Topor,R.W. "Making Prolog more expressive". J. Logic Programming, 1984, No.3, pp. 225-240.
18. Lloyd,J.W.; Topor, R.W. "A basis for deductive database systems". J. Logic Programming, 1985, No. 2, pp. 93-109.
19. Lloyd,J.W.;Topor,R.W. "A basis for deductive database systems II". J. Logic Programming, 1986, No. 1, pp. 55-67.
20. Lloyd,J.W.;Sonenberg,E.A.;Topor,R.W. "Integrity constraint checking in stratified databases". J. Logic Programming, 1987, No.4, pp. 331-343.
21. McCune,W.;Henschen,L. "Maintaining state constraints in relational datbases: A proof theoretic basis", JAM, Vol. 36, No. 1, January 1989, pp. 46-68.

22. Nicolas,J.M. "Logic for improving integrity checking in relational data bases". Acta Informatica 1982, 18,3, pp. 227-253.
23. Nicolas,J.M.;Yazdanian,K. "Integrity checking in deductive data bases". In Gallaire, H.; Minker,J. (Eds.), "Logic and data bases", Plenum Press, 1978, New York, pp. 325-344.
24. Olivé,A. "On the design and implementation of information systems from deductive conceptual models". Proc. of the 15th. VLDB, Amsterdam, 1989, pp. 3-11.
24a. Olivé, A. "The internal events method for integrity checking in deductive databases". Research report, Univ. Pol. Catalunya, Dept. LSI, March 1990.
25. Sadri,F.; Kowalski,R. "A theorem-proving approach to database integrity". In Minker,J. (Ed.) "Foundations of deductive databases and logic programming", Morgan Kaufmann Pub., 1988, pp. 313-362.
26. Stonebraker,M. "Implementation of integrity constraints and views by query modification". ACM SIGMOD Int. Conf. on Management of data, 1975, pp. 65-78