Integrity checking in deductive databases : an exposition

Eindhoven University of Technology

Department of Mathematics and Computing Science

Integrity Checking in Deductive Databases; An Exposition

by

R.R. Seljée

92/16

Computing Science Note 92/16
Eindhoven, July 1992

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

editors: prof.dr.M.Rem
prof.dr.K.M.van Hee.

INTEGRITY CHECKING IN DEDUCTIVE DATABASES; AN EXPOSITION.

R.R. Seljée

Co-operation Centre
Tilburg and Einhoven Universities

ABSTRACT.

Relational systems are extended in several ways. We consider extensions with rules and integrity constraints to so-called deductive databases. There are several problems in handling these rules and constraints. One problem is to check, in an efficient way, the integrity of a deductive database when an insertion or deletion is made. There exist several integrity maintenance methods. All these methods avoid a full check of the integrity constraints in order to keep them efficient. Therefore, the database is supposed to be consistent before a transaction is made. We take a look at one of the most efficient methods proposed by Bry, Decker and Manthey. They determine from the updates, rules and constraints and without accessing the fact base the so called update constraints which have to be checked in the updated database. These constraints are instantiations of the constraints that are relevant to the update. Although this method is more efficient than other ones, it is still susceptible of improvement. Our goal in this article was to show by some well-chosen examples that in some regards the proposed method is still very inefficient. By these examples it becomes clear where the inefficiencies come from; they were not mentioned in the article of Bry, Manthey and Decker.

§0 INTRODUCTION:RELATIONAL DATABASES, RULES, INTEGRITY CONSTRAINTS, DEDUCTIVE DATABASES, INTEGRITY ENFORCEMENT

Most commercially available database systems are based on the relational database model. In this relational model facts are stored in tables. Tables contain explicit data. A relational database management system (RDBMS) takes care of fast retrieval of information from a database. Besides explicit information tables also contain implicit information. In conventional relational systems the derivation of implicit from explicit information is left to the user. There is a growing need to delegate such derivations to the system. We can do this by extending a relational database with deductive rules. In that case, we speak of a deductive database.
On the basis of the next example, consisting of a PARENT-table and a GRANDFATHER-table, we shall illustrate the extensions mentioned above of a conventional RDBMS.[1)]

PARENT-table

NAME	AGE	FATHER	MOTHER
ALEX	15	CARL	JANE
JANE	?	PETER	LUCY
JACOB	38	PETER	LUCY
CARL	41	LEON	ANNE
LEON	?	JACK	?

GRANDFATHER-table

NAME	GRANDFATHER
ALEX	LEON
ALEX	PETER
LEON	JACK

Besides storing facts, there is a great need to store general knowledge about the universe of discourse in the database. Both rules and integrity constraints represent such knowledge.

RULES

Rules are added to make implicit knowledge explicit. As we see in the PARENT-table, JANE and JACOB are siblings, because their FATHER and MOTHER are PETER and LUCY for both of them. By adding a rule like "People who have the same parents are siblings", we represent the knowledge of the sibling

1) In this example the tables are not physically present in the database. Normally, for reasons of maintenance of the database the date of birth of an individual is restored. In order to keep the problem clear we store the age of all individuals in our tables. So, in fact our tables are views.

relation. Now, a deductive database management system must take care of answering questions about siblings using this rule and the PARENT-table.
Rules are also used to make explicit knowledge implicit. Sometimes this means a considerable saving of space, as will be pointed out in the next example. Instead of one big GRANDFATHER-table in the database, we could simply state one rule: "the FATHER of a MOTHER or FATHER of an individual, is the GRANDFATHER of that individual". With this rule and the already existing PARENT-table we could generate the GRANDFATHERs of individuals from the PARENT-table.

INTEGRITY CONSTRAINTS

Integrity constraints do not generate facts but check if the database is consistent. Under no circumstances do we want the database to give erroneous or contradictory information. Database transactions may affect the integrity of the database, as will be shown by the following example. In the PARENT-table we see that JANE is the MOTHER of ALEX. The age of JANE is unknown. Suppose the age of JANE is added to the database. If we want to insert "8 years" for her age, then the conventional systems would allow that without objection. However, there is some inconsistency with respect to our knowledge of the universe of discourse. JANE is mother and may never be 8 years old. So, the addition of a certain age can lead for several reasons to a database which does not agree with our knowledge of the universe of discourse. The next scheme will show several of these reasons.

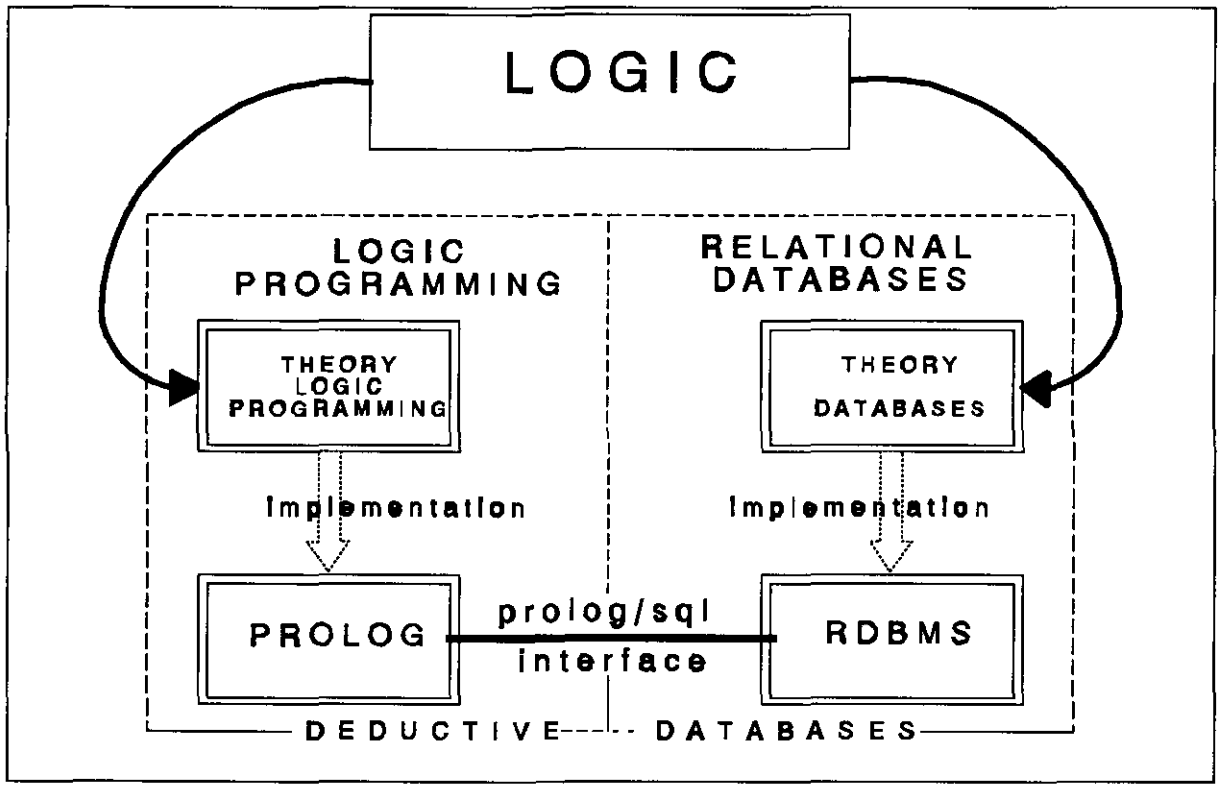
The inserted age of Jane:	Reason why we reject this age:
24 years	The age difference between her and her son is 9 years.
80 years	Her son was born when she was 65 years old.
146 years	Human beings will never be this old.

Integrity constraints which prohibit undesirable situations like the ones above are:

"The age of a mother is at least 14 years",
"Women don't have babies after their 50^{th} birthday",
"The difference between the age of a parent and his/her child is at least 14 years",
"The maximum age of a human being is 120 years".

DEDUCTIVE DATABASES

As we stated earlier, deductive databases are an extension of conventional databases, because besides facts also rules are stored. Some problems have to be solved, because conventional RDBMSs have no facilities to handle rules. However, the logic programming language PROLOG does have this capacity. An interface between PROLOG and a database query language enables us to combine rules with a RDBMS. Now one should search for efficient methods in order to be able to use these rules within RDBMSs.



Our coupling will be achieved by choosing for the following strategy:

1) Instead of extending a logic programming system with database management facilities, we want to extend an (existing) relational database management system with logic programming facilities. A lot of time and money has already been invested in developing advanced, efficient and safe relational database management systems. It is not feasible and above all not necessary to do this work all over again.
2) We start from the most current implementations, which result from the theory of logic programming and the theory of databases when we are implementing new concepts. We choose PROLOG to be our logic programming language and SQL to be our query language. SQL is the most current query language used in relational systems.

To describe deductive databases the theory of logic programming is coupled to the database theory. The scheme above shows that logic plays a dominant role in this. Logic is used not only as the basis for the theory of logic programming, but also to describe databases formally. The coupling between the theory of logic programming and the database theory will be brought about through logic. Important work in this area has been performed by Gallaire, Minker and Nicolas. ([GAL78], [GAL84], [MIN88], [MKR88]).
They conceive a database as a first order theory from which the answers to the queries and the integrity constraints have to be deduced by means of an appropriate proof procedure. This theory consists of a logical translation of facts and rules which are stored in the database.

INTEGRITY ENFORCEMENT

With large database systems it is not feasible for the user of the system to check all their integrity constraints. That is why we try to lay down this responsibility with the system.
To handle integrity constraints efficiently, we assume that the database obeys the integrity constraints before a transaction is carried out. Notice that this is a simplification. After a transaction the database will then be put to the test of the integrity constraints. If they are not fulfilled, then an analysis should be given of which facts and/or rules from the database are responsible for violating them.
The "simplification theorems for checking the integrity" occurring in the literature take the line mentioned above. These theorems give conditions by which we can conclude whether or not a database is consistent with the integrity constraints after a transaction. Lloyd, Sonenberg and Topor ([LLO87]), Lloyd and Topor ([LLO86], [LLO85]) formulate and prove these theorems for deductive databases; they are a generalization of comparable simplification theorems for relational databases which were proved by Nicolas ([NIC82]).

There are serious problems of finding efficient ways to check the integrity conditions in deductive databases. Because rules generate facts, a transaction may cause a considerable change in the state of the database and therefore may influence the integrity a great deal. It will be clear that one must search for efficient methods to decide which actions have to be taken in order to maintain the integrity.
In this connection, useful articles are those of Decker ([DEC86]) and Sadri and Kowalski ([SAD88]), who give sound and complete proof procedures to check the integrity of deductive databases. These procedures are extensions of resolution, which is used by PROLOG, in order to be able to reason not only about facts but also about rules.

Lloyd, Sonenberg, Topor, Nicolas, Decker, Sadri and Kowalski basically follow several principles. If a transaction has caused a violation of the integrity conditions, then the system will look for a possibility to cancel one or more deletions, changes and insertions. If this is not possible, the whole transaction will be cancelled. Only new facts are to be blamed for violating the integrity constraints. This means that in order to check the consistency of the database with respect to the integrity constraints we don't have to consider the whole database and its integrity constraints, but only the facts occurring in the transaction and the related integrity rules.

Extensions of the considerations mentioned above are thinkable. For example

- how can we allow rules in transactions; In their integrity proof Sadri and Kowalski ([SAD88]) also take rules into account. For instance, we may want to add the rule "when x is PARENT of y and y is PARENT of z, then x is GRANDPARENT of z" to the PARENT-table.
- how can we involve facts, which were present in the database before the transaction, in restoring the integrity? It may be possible that these facts are not correct, and therefore they should play a role in restoring the integrity. Notice that this doesn't mean that the database before a transaction is inconsistent with its specified integrity constraints. For example, the previously given PARENT-table is consistent with the specified integrity constraints, even if 15 wrongly has been inserted as the age of ALEX. Suppose we would like to insert JANET with ALEX as her FATHER. Then there will be an inconsistency with the integrity constraint which says that a parent must be at least 16 years old. The cause of the inconsistency is not the new appearence of JANET, but the already existing occurrence of the age of ALEX in the PARENT-table.

- what happens if we add new integrity constraints to a transaction? Until now we assumed the database to satisfy integrity constraints which were previously specified. Integrity constraints played a rather static role.

§1 INTEGRITY CONSTRAINTS

Integrity constraints ensure that the data in the database always make sense.
We distinguish several kinds of integrity constraints such as

- Column and Domain constraints,
- Entity and Referential integrity,
- User-defined integrity constraints.

For a better comprehension of these different and sometimes interrelated conceptions we give a short description of each sort of constraint. For more information about these and other relational concepts we refer to [BRO89], [COD90], [DAT90], [MAI83], [ULL88].

1.1 COLUMN & DOMAIN CONSTRAINTS

A domain in relational terms is the set of all unique values permitted to appear in one or more specified columns.
To start with one can specify a simple property, common to all values of a certain basic data type. For instance, the values of the column "Day_of_week" consist of at most eleven characters, and the values of the column "Account_number" is an integer number from 10000 to 99999. This is what we call a *column constraint*. Further limitation of these values is specified in the domain definition. We call this limitation of values the *domain constraint*. Suppose in our "Day_of_week" column only the following values can occur:

Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday,
Sunday.

We can give this list a name, say "Day_names". Then "Day_names" is the domain from which "Day of_week" can pick the values.
The domain concept plays a significant role in the relational model for the following reasons.

- A RDBMS contains an area in which all information about a column is stored. In such a column declaration area several items occur. A domain specification is one of these items. Instead of giving the whole domain specification in each column declaration area, a reference to this specification, which is declared in another area of the management system, the domain declaration area, is given. This means a reduction of redundancy, which exists if the same domain specification

should be declared for several columns. For instance, instead of declaring a list of all days of a week in the column declaration area every time a column contains values of this list, we state in this area that a column contains values of type "Day_names". In the domain declaration area we specify which values correspond with the type "Day_names".

- For each operation (e.g. join) acting on two tables a check must be made to see if the operation has any meaning. Consider, for instance, the join of two columns A and B of which the value is an integer of eight digits, but A contains account numbers and B fiscal numbers. In this case joining these two columns should not be allowed. To solve this problem we can specify two different domains, one of account numbers and one of fiscal numbers. Before applying an operation it must be checked if the related columns contain the same type of information. In other words, if they pick values of a common domain. This check can take place at compile time.
- As the example of account numbers and fiscal numbers shows, the domain concept plays a major role in distinguishing the semantics of the facts stored in the database and therefore allows all sorts of integrity maintenance.

A domain is used as a kind of derived data type. That is why it is also called an extended data type. (See [COD90]).

1.2 ENTITY & REFERENTIAL INTEGRITY

Before explaining the concepts of entity and referential integrity it is necessary to define some other concepts. We do this by the following example, illustrated by table I. Suppose, we have a table with the results of exams of all students of a high school in all subjects they have selected at the beginning of the school-year. If a student has not selected a certain subject, he will never do the corresponding exam. So, in the table related columns may contain values marked as *inapplicable* ("_"). If the student has not yet done an exam which he or she must do in the near future then the corresponding value is *applicable* but marked as *unknown* ("?"). In all other cases the values are applicable and *known*.

STUDENT_NAME	EN	FR	GR	MA	CH	PH	EC	HI
Nick James	7	?	_	?	5	?	6	_
Jane Dickson	8	9	8	?	6	_	8	_
Peter McMann	6	_	5	?	_	?	8	6

Table I.

An inapplicable value in a column means that a regular value in this column can never occur. On the other hand, applicable values could be unknown at a certain time but are supposed to have a value.

The names of the columns of a table are called attributes. The values of a row in a table are often called a tuple of the table. We represent tuples as follows. Let $\{A_1,A_2,...,A_n\}$ be the collection of attributes of a table. A tuple t with values $t_1,t_2,...,t_n$ for the attributes $A_1,A_2,...,A_n$, respectively is denoted as:

$$t = < A_1;t_1 , A_2;t_2 ,..., A_n;t_n >$$

or, if it is clear from the context which attributes belong to which values:

$$t = < t_1,t_2 ,..., t_n >.$$

Let t be a tuple and $\{B_1,B_2,...,B_m\} \subseteq \{A_1,A_2,...,A_n\}$. By $t|\{B_1,B_2,...,B_m\}$ we mean the restriction of t to the attributes $B_1,B_2,...,B_m$:

$$t|\{B_1,B_2,...,B_m\} = < B_1;t_1 , B_2;t_2 ,..., B_m;t_m >$$

or shorter, when the context makes it clear:

$$t|\{B_1,B_2,...,B_m\} = < t_1,t_2 ,..., t_m >.$$

One or more attributes of a table can form a primary key or a foreign key. A *key* is a collection of attributes of a table of which the values provide for a unique reference to the tuples of the table. So, $\{B_1,B_2,...,B_m\}$ is a key of a table T, if for each pair of tuples s and t for which $s|\{B_1,B_2,...,B_m\} = t|\{B_1,B_2,...,B_m\}$ it holds that s = t.
Note that trivially all attributes of a table form a key.

A *primary key*[2] is a key, for which elimination of an attribute means that it is no longer a key. So, the number of attributes of a primary key must be minimal. Consider, for instance, table II. The values of the attributes STUDENT_NR and NAME provide for a unique identification of values in the row, for if we know someone's name and student_nr we can say which tuple we deal with.
So, {STUDENT_NR,NAME} is a key for table II. However, {STUDENT_NR,NAME} is not a primary key, because {STUDENT_NR} is also a key, a primary key.

STUDENT_NR	NAME	MENTOR	CLASS
790123	Nick James	Miss. F. White	6B
801234	Jane Dickson	D. P. Allison	4C
781533	Nick James	Miss. F. White	6B
781635	Peter McMann	Miss. F. White	6B

Table II.

2) In the literature, the concepts "primary key" and "foreign key" do not have well-established definitions. (See discussions in [COD90] en [DAT90]). We took the definitions of Date ([DAT90]).

Note that in cases of two objects, also called entities, with the same values, the primary key provides for an unique identification of the objects. In our example, we have two different people with the same name "Nick James" in class 6B. They are distinguished by the values of the primary key consisting of the attribute "STUDENT_NR".

A *foreign key*[2] is the collection of attributes of a primary key used in another table in order to provide for a reference to entities identified by the primary key. A foreign key value provides for an unique link to the values stored in the table of the related primary key by the corresponding primary key value. So, for instance, the first column of table III is a foreign key for this table, connected to the primary key of table II.

STUDENT_NR	EN	FR	GR	MA	CH	PH	EC	HI
790123	7	?	_	?	5	?	6	_
781344	8	9	8	?	6	_	8	_
781533	6	_	5	?	_	?	8	6

Table III.

Note that Table III without table II has no meaning. For, students, like Nick James of class 6B with mentor Miss. F. White, are doing exams and not numbers, like 781533. So, foreign keys are used for a link to entities to which the values in the rest of the table refer.

Now, the concepts of entity and referential integrity for a database can be explained. When no column of a primary key contains a missing (i.e. inapplicable or unknown) value and no column of an foreign key an inapplicable value, then the database is said to fulfill the *entity integrity*[3].
So, primary key values must always be known and foreign key values must be applicable but could be unknown. Informally, entities must be distinguishable from each other by the primary key value, and a foreign table (i.e. a table with a foreign key) must always contain references to entities, though they can be unknown.

Interrelated with entity integrity is the concept of referential integrity. One speaks of *referential integrity*[3] if for every known foreign key value there exists an identical primary key value from the same domain. So, spoken informally a reference must be possible.

Tables IV & V illustrate the concepts of entity and referential integrity. The database consisting of these two tables satisfies the entity integrity property, because every primary key value (i.e. value of student nr) is known in table IV and all foreign key values in table V are applicable. However, both tables do not satisfy the referential integrity property, because 781344 in table V refers to no primary key value in table IV.

3) In the literature, the concepts "entity integrity" and "referential integrity" do not have well-established definitions. (See discussions in [COD90] en [DAT90]). We took the definitions of these concepts of Date ([DAT90]).

STUDENT_NR	NAME	MENTOR	CLASS
790123	Nick James	Miss. F. White	6B
801234	Jane Dickson	D. P. Allison	4C
781533	Nick James	Miss. F. White	6B
781635	Peter McMann	?	5C

Table IV.

STUDENT_NR	EN	FR	GR	MA	CH	PH	EC	HI
790123	7	?	_	?	5	?	6	_
781344	8	9	8	?	6	_	8	_
?	6	_	5	?	_	?	8	6

Table V.

1.3 USER-DEFINED INTEGRITY CONSTRAINTS

This type of integrity constraint is a central issue in this paper. In this case, a user can specify constraints on the data stored in the database. We have given some examples in section 0. Although our main concern in this paper is how the database management system has to preserve consistency with respect to these constraints, the responsibilities of the system should go far beyond that. For instance, the user must also have the possibility to specify the actions to be taken if some integrity constraint is violated. For example, if a supply shrinks and a certain level has been reached, then an order must go out.

The intention is that the integrity maintenance mentioned above is the responsibility of the database management system.

§2 AN INTEGRITY CHECKING METHOD FOR DATABASES

From a logical point of view, a relational or deductive database can be looked at in two different ways. (See [GAL84]).
First we have *the model theoretic view*. Here, the facts of the relational database are regarded at as an interpretation (or model) of the set of logical formulas corresponding to the integrity constraints.
The logical formulas are evaluated under the intended interpretation. If they are true, the database can be seen as a *model* for the formulas (i.e. an interpretation in which all formulas are satisfied). A database is called *consistent* with respect to its specified integrity constraints iff it forms a model for the logical formulas corresponding to the integrity constraints.

Second we have *the proof theoretic view* in which the database is a first order theory from which integrity constraints (and queries also) may be deducible. If this is the case we call the relational database consistent with respect to the specified integrity constraints. Now, suppose a database transaction, which can be any collection of updates, additions and/or deletions of facts, takes place. Then the consistent database can change into an inconsistent one. Proof theoretically, the integrity of the database is preserved if all the old integrity constraints are still deducible from the new database.

In this article, we describe the concepts from the model theoretic view. Then, the constraints are evaluated under the intended interpretation, which follows from all the facts in the database. A naive evaluation of the constraints will involve all these facts. Not only is this rather cumbersome but also unnecessary provided that we have a consistent database before the transaction. It is obvious that inserting a fact only affects those integrity constraints that say something about this fact. When we insert a fact for which no constraint is applicable the database will also be consistent after this transaction, because after all the database was supposed to be consistent with its integrity constraints before the transaction. Evaluation of only the *relevant* integrity constraints turns out to be sufficient.

Deductive databases contain rules. By the generating capabilities of rules an inserted fact can lead to a great amount of induced updates. These induced updates can in their turn also affect the integrity of the database. So, in deductive databases integrity maintenance is more complicated than in relational databases.

2.1 AN INFORMAL INTRODUCTION

Integrity maintenance methods are intended to guarantee that all integrity constraints remain satisfied after an update, provided they were satisfied before.
A method proposed by Bry, Decker and Manthey [BDM88] is described here. Their approach is different from others, because they compute *relevant* constraint instances which must be checked in order to prove the database to be consistent without accessing the fact base. This means that optimalization is possible before evaluating these constraint instances.

As an introduction we will give an example for the relational case.
Consider a company with several departments. Every department has one manager, which gives lead to one or more employees. Employees can be employed in several departments. We state the following integrity constraint:

- **an employee who may work at several departments can temporarily be replaced by another person only if there is a manager of both employees who did not mark this other person as busy.**

Suppose employee Patrick has to be replaced because he is ill. Suppose a database is given in which we want to insert that Patrick will be replaced by John. It is not necessary to check again all integrity constraints together with all facts in the database. But it is sufficient to restrict ourselves to the constraints that are relevant to this update. The following situations will be distinguished.

- Suppose it is already known that Patrick is replaced by John. This means that before the transaction this fact was already present in the database. So, the database does not change and neither does the integrity, because the database is supposed to be consistent before any transaction. So, in this situation no integrity check has to be made. Even stronger, facts already known to a database may not be inserted for a second time for they are no updates of the database. With large databases one has no idea of what the stored facts are, so a check must be made to skip "redundant" updates.
- Suppose this fact happens to be unknown to the database before the transaction; this means that it is not present in the database. Consequently, the fact is supposed to be false. (This assumption is called the closed world assumption).
  We can limit the amount of integrity checks in several ways.
  In the first place, we limit the collection of constraints to only relevant constraints. The inserted fact can influence the integrity by constraints in which something is said about replacing some person X by some person Y. Notice that it does affect the integrity by the constraint mentioned above.
  In the second place, given a relevant constraint, we can limit the collection of people which are represented in the database. It is not neccessary to check these constraints for all people. For instance, with our constraint it is only necessary to look for managers of Patrick and John who did not mark the substitute as busy, instead of checking this constraint for all people who have a manager.
  By these restrictions we derive a collection of so called simplified instances of the integrity constraints which have to be checked in order to prove the consistency of the new database.

In the case of an update in a deductive database, rules can cause a considerable amount of induced updates.
However, in this case one can construct certain compound constraints, to be called update constraints. In order to prove that the new database is consistent with the specified integrity constraints, the only thing to do is to prove that the update constraints are consistent with the fact base. In fact the deductive case can be reduced to the relational case.

In the next section we formalize the concepts mentioned above.

2.2 FORMAL DESCRIPTION

In a deductive database we distinguish facts, rules and integrity constraints. These facts, rules and constraints will now be expressed in a logical language, the first-order predicate language. The symbols used in this language are (1) parentheses, (2) variables and constants, (3) predicate symbols, (4) logical connectives like $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\leftarrow$ (implication), and (5) quantifiers $\forall$ (for all) and $\exists$ (there exists). Throughout the paper we use uppercase letters to represent variables and lowercase letters or words to represent constants and predicate names. In our databases terms are supposed to be function-free, i.e. a *term* is a constant or a variable. We call $p(t_1,t_2,...,t_n)$ an *atomic formula*, or shortly an *atom*, if p is a predicate symbol of arity n and $t_1,t_2,...,t_n$ are terms. An atomic formula or its negation is called a *literal*. We also call atoms and negated atoms *positive* and *negative literals* respectively. The expressions allowed in this language, the well-formed formulas (wffs), are defined recursively as follows.

i) Any literal is a wff.

If w_1 and w_2 are wffs, then

ii) $(\neg w_1)$, $(w_1 \vee w_2)$, $(w_1 \wedge w_2)$ and $(w_1 \leftarrow w_2)$ are wffs, and

iii) $\forall X[w_1(X)]$ and $\exists X[w_2(X)]$ are wffs if X is free in w_1 and w_2 respectively.
(In $\forall X[w_1(X)]$ (resp. $\exists X[w_2(X)]$) we call $w_1(X)$ (resp. $w_2(X)$) the *scope* of $\forall X$ (resp. $\exists X$). A variable X is *bound by* $\forall X$ (resp. $\exists X$) if it occurs in the scope of $\forall X$ (resp. $\exists X$). A variable X in an expression is called *bound* if it is bound by $\forall X$ (resp. $\exists X$) or its occurrence is in $\forall X$ (resp. $\exists X$). A variable is called *free* if it is not bound.)

iv) The only wffs are those given by i), ii) and iii).

A formula is called *closed*, if it contains no free variables. Otherwise, it is called *open*. A formula in which no variables occur is called a *ground* formula. Facts, rules and constraints are defined as follows.

facts: - are ground atoms.

rules: - are clausal form expressions $B \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$, where B is a positive literal and $A_1, A_2, \ldots, A_n$ are literals, positive or negative.
B and $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ are called the *head* and the *body*, respectively, of the rule $B \leftarrow A_1 \wedge A_2 \wedge \ldots \wedge A_n$.
Moreover, a variable which occurs in B or in a negative literal of the body of the rule must also occur in a positive literal of the body of the rule. In other words, the rules are *range-restricted*. Suppose a variable in a negative literal A' of a rule R does not occur in a positive literal of the body of R. The instances of the head of the rule will depend of the database domain as a whole. Application of such rules would require a complete domain search, which is in most cases extremely inefficient.

Constraints: - are closed first-order formulas which are also function-free.
- Moreover, we consider only a subclass of these formulas, namely the restricted quantified formulas. These formulas have one of the following forms:
  1) true or false,
  2) $\exists X_1 \ldots \exists X_n [A_1 \wedge \ldots \wedge A_m \wedge Q]$,
  3) $\forall X_1 \ldots \forall X_n [(\neg A_1) \vee \ldots \vee (\neg A_m) \vee Q]$
     or, equivalently, $\forall X_1 \ldots \forall X_n [A_1 \wedge \ldots \wedge A_m \rightarrow Q]$,

  where $m,n \geq 0$, each A_i is an atom, $i=1,2,\ldots,m$, each variable X_i occurs in one or more A_j, $j=1,2,\ldots,m$, and Q is an expression of the form descibed in 1),2) or 3) in which variables $X_1,\ldots,X_n$ are free.

For instance,

$\exists X \exists Y \exists Z [p(X) \wedge r(X,Z) \wedge r(Y,X) \wedge (\forall V \forall W [\neg p(V) \vee \neg q(V,W)])]$

is a constraint and

$\forall X \forall Y [\neg p(X,X) \vee \neg q(Y,Y,X) \vee (\exists Z [\neg r(Z)])]$

is not.

Constraints are supposed to be restricted quantified. Restricted quantified formulas are domain independent, i.e. the truth value of such a formula does not depend on any domain element other than those occurring in the relations $A_1 \wedge \ldots \wedge A_m$ that are explicitly mentioned in the formula.

Throughout this paper we assume that updates are represented by literals. By U we represent an update to a database D. U is called an insertion (resp. deletion) if it is a positive (resp. negative) literal. Let U(D) denote the updated database.

DEFINITION 1:

An atom A is *explicit* in a database D if A appears in the fact base of the database. An atom A is *implicit* in D if it is not explicit but it is derivable from the rules and the facts of D.

If U is an atom explicit in D, then the updated database U(D) is identical with D.
If U is an atom not explicit in D, then U(D) is D augmented with U to the fact base.

If U is a negative literal ¬A and A is explicit in D, then the updated database U(D) is D without A in the fact base.
If U is a negative literal ¬A and A is not explicit in D, then U(D) is identical with D.

DEFINITION 2:

An update U to a consistent database D is *permitted* if U(D) is also consistent.

DEFINITION 3:

A constraint C is *relevant to an update* U iff the complement of U is unifiable with a literal in C. (Resolution is the supposed inference mechanism).

To keep integrity constraint checking methods efficient, we want the constraints to be domain independent, in order to avoid validation of formulas for which the whole database domain is needed. A formula like for example ∃X¬books(X,dahl), which represents a query such as "Give all books Dahl did not write", would require consideration of all domain elements in the database. So, by requiring the constraints to be restricted quantified, we do not have to consider all the domain elements. For example, consider the constraint

C: ∃X [p(X) ∧ ¬q(X)],

and a database D represented by

D := {p(a),p(b),q(b),q(c),q(d),r(a),r(e)}.

Here, p(X) limits the values which X can take from its domain elements. In this example, the only domain elements, which have to be considered are a and b, in order to validate the constraint. Further ¬q(X) is fully instantiated before its validation.
Now, consider a consistent database D with respect to the constraint C.

Case 1] An update U=p(a) does not have an influence upon C, because there was already a ground instance of p, say p(b), in the database D by which C was satisfied. So, the update does not change the ground instance of C, $p(b) \wedge \neg q(b)$, because p is positive in C.

Case 2] An update U=q(a) may have an influence upon C. It is thinkable that "a" was the only constant satisfying the constraint C in D. So, $p(a) \wedge \neg q(a)$ is satisfied in D. Now, by inserting q(a) as a fact the only satisfying instance of C is lost in U(D).

Case 3] A deletion $U=\neg p(a)$ may also have influence upon C. Suppose we have a database before the deletion where "a" was the only constant satisfying C. After the deletion C is violated, because the only satisfying instance of C is now lost in U(D).

Case 4] A deletion $U=\neg q(a)$ does not have an influence upon C, because q is negative in C. For, there was already a ground instance of p in the database D by which C was satisfied. Suppose this instance is X=b. We can distinguish two cases.
Case one in which $a \neq b$. Nothing happens to $p(b) \wedge \neg q(b)$ by the transaction and so C remains satisfied.
Case two in which a=b. Apparently $p(a) \wedge \neg q(a)$ was satisfied before the transaction. The deletion U does not change that.

In order to determine whether an updated database is consistent all integrity constraints could be checked. But this is in most cases an endless task. It turns out that we can restrict ourselves to simplified instances of only those constraints that are relevant to an update (see Proposition 1). So, the amount of constraints to be checked can often be reduced considerably.

DEFINITION 4:

Let $C=C(L_1,L_2,...,L_n)$ be an integrity constraint relevant to update U, where $L_1,L_2,...,L_n$ are all the literals (positive or negative) occurring in C. So, there is a literal L_j which is unifiable with the complement of U. Let σ be a most general unifier of L_j and $\neg U$. To obtain a *simplified instance* $C\tau$ of C we partially instantiate C by a restriction τ of σ to only those universally quantified variables which are not governed by an existentially quantified variable in C. We simplify $C\tau$ by removing all complements of U occurring in $C\tau$ and the (universal) quantifiers grounded by τ. The remaining formula, $C\tau$, is called the simplified instance of C with respect to U and L_j. τ is called the *defining substitution* of the simplified instance.

Take the employee example in the previous subsection to illustrate Definition 4. The integrity constraint can be described as follows:

-an employee X, who may work at several departments, can temporarily be replaced by another person Y only if there is a manager Z of X and of Y, who did not mark Y as busy.

and in a more formal format:

$$C : \forall X \forall Y\ [\neg rpl(X,Y) \vee \exists Z\ [mngr(X,Z) \wedge mngr(Y,Z) \wedge \neg st(Y,Z,busy)]]$$

Note that C is restricted quantified. A more readible and equivalent form of C is

$$C' : \forall X \forall Y\ [rpl(X,Y) \rightarrow \exists Z\ [mngr(X,Z) \wedge mngr(Y,Z) \wedge \neg st(Y,Z,busy)]].$$

Suppose we have the update:

U : ¬mngr(john,george).

C is relevant to U. A simplified instance of C with respect to U is obtained as follows:
First note that U is unifiable with the negation of two literals of C, L_1: mngr(X,Z) and L_2: mngr(Y,Z). This implies that there are two simplified instances of the same constraint with respect to the one update U. Take the first one, L_1. The most general unifier of ¬U and L_1 is σ={X/john,Z/george}. The universally quantified variables in C that are not governed by an existentially quantified variable are X and Y. Let τ be the restriction of σ to {X,Y}; so, τ={X/john}. Then the first simplified instance of C associated with the update U is C_1:

C_1: $\forall Y\ [\neg rpl(john,Y) \vee \exists Z\ [mngr(john,Z) \wedge mngr(Y,Z) \wedge \neg st(Y,Z,busy)]]$.

Intuitively, for the constraint C the update ¬mngr(john,george) means that now it has to be checked that for every person Y who replaces john there must be another manager than george who did not mark Y busy.

The other simplified instance C_2, when taking mngr(Y,Z) as the unifiable literal, is:

C_2: $\forall X\ [\neg rpl(X,john) \vee \exists Z\ [mngr(X,Z) \wedge mngr(john,Z) \wedge \neg st(john,Z,busy)]]$.

In this case john replaces someone.
Note: Although transactions on a database are supposed to fail if they cause an inconsistent database, this is not always desirable. In some cases we may prefer to make the database consistent again. In our example it is a fact that george is no longer the manager of john. We may want to enforce the update when C is violated. We can restore the integrity by:

- searching for each person x, who has to replace john, another manager z of john who gives permission to x to replace john. This implicates an addition of one or more facts to the database, namely ¬st(x,z,busy)-facts.
- deletion of all facts in which john is replaced by someone and george was the only manager who had the right and actually gave his permission to this replacement. So, john will not be replaced by these people.
- by even deleting the whole integrity constraint, if for some reason that rule in the company is canceled.

Before getting to proposition 1 we give a LEMMA first, which can be found in [NIC79].

LEMMA:

Let F be a formula which satisfies the range restricted property. Given any interpretation I of F and any interpretation I^* obtained from I only by adding a new element to its set of elements then:

F is true in I iff F is true in I^*.

This LEMMA means that if we have, for instance, a database consisting of the facts {R(a),R(b),P(a),P(b),P(c)} and one specified constraint C: $\forall X[\neg R(X) \vee P(X)]$, an element like for instance d has no influence on the truth of C, because C is range resticted and d is no element of the extensions of relations R and P. Here, the set of elements in the interpretation corresponding to D is {a,b,c}.

PROPOSITION 1:

Let D be a relational database. Let U be an update.
Suppose all constraints are satisfied in D.
Then they are satisfied in U(D) iff every simplified instance of a constraint relevant to U is satisfied in U(D).

Before we prove this proposition, we illustrate it by some examples.

EXAMPLE

Let U:=p(b) be an update to a relational database D.
Suppose we have one existentially quantified constraint

C: $\exists X\ [\neg p(X) \wedge \forall Z[q(X,Z)]]$

which is satisfied in D. C is relevant to U. Because X is an existentially quantified variable in C the defining substitution of the simplified instance is $\tau=\{\}$. So, the simplified instance of C is C itself. Generally, this is the case for all existentially quantified constraints.

EXAMPLE

Let U:=p(b) be an update to a relational database D.
Suppose we have one constraint

C: $\forall X \forall Y\ [\neg p(X) \vee \neg r(Y) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,Y)]]$

which is satisfied in some database D. C is relevant to U. Because of the existence of just one negated literal with predicate symbol p of C, we get only one simplified instance , C_S, with respect to U:

C_S: $\forall Y\ [\neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$.

A] Let D consist of the following facts

p(a)	r(b)	q(a,b)	q(c,a)	q(d,b)	q(e,d)	q(f,b)	q(g,b)
p(d)		q(a,c)					
p(e)							

then constraint C is satisfied in D. The updated database U(D):={p(b)} $\cup$ D violates C and C_S respectively, because there is no fact q(b,_) in the updated database. (_ stands for some element in the domain of the database).

B] Let D consist of the following facts

p(a)	r(b)	q(a,b)	q(b,a)	q(c,a)	q(d,b)	q(e,d)	q(f,b)	q(g,b)
p(d)		q(a,c)	q(b,c)					
p(e)			q(b,d)					
			q(b,f)					
			q(b,g)					

then constraint C is satisfied. The updated datadase U(D):={p(b)} ∪ D satisfies C and C_s respectively, because now there are facts, q(b,_), in the database. (_ stands for some element of the database).

We can say that the simplified instance C_s gives an indication of the kind of facts which are important for proving the constraint to be satisfied in U(D).

Proof (proposition 1):
In this proof we suppose that the update is an insertion. The case in which the update is a deletion is treated similarly.

⇒: Suppose all constraints are satisfied in D and U(D). So, D and U(D) can be seen as interpretations in which the constraints are true. (Therefore, we speak of constraints which are true in D and U(D) respectively). Now, let C be a constraint which is relevant to U. A simplified instance of C can be represented by Cτ, where τ is some substitution of variables which are universally quantified and not governed by any existentially quantified one. Since C is true in U(D), clearly Cτ is true in U(D) too. Consequently, every simplified instance of a constraint relevant to U is satisfied in U(D).

⇐: Suppose all constraints are satisfied in D and suppose every simplified instance C_s of a constraint C relevant to U is satisfied in U(D). We consider two cases:

Case 1: C is not relevant to U.
Then the update does not influence C. So, C is satisfied in U(D), because U does not play a role in satisfying C and C is satisfied in D.

Case 2: C is relevant to U.
C is satisfied in D means that the facts of D imply an interpretation in which C is true. In case $C=C_s$ for some simplified instance C_s, when the defining substitution is empty, there is nothing left to be proved. Note that this is the case if C is an existentially quantified constraint, i.e. a formula of type 2) in the formal definition of constraints. So, in the case of $C \neq C_s$ for all such C_s which we consider now, we can assume that C is universally quantified. In order to prove that U(D) is a "model" for C, it suffices now to prove that Cτ is satisfied in U(D) for every substitution τ of the universally quantified variables which are not governed by an existentially quantified one.
Let ξ(D) (resp. ξ(U(D))) the collection of elements of D (resp. U(D)). Let ξ(U) represent the set of all elements of U.
So, let $\tau=\{X_1/t_1,X_2/t_2,...,X_n/t_n\}$, where $t_1,t_2,...,t_n \in \xi(U(D))$ and where $X_1,X_2,...,X_n$ are all the universally quantified variables which are not governed by an existentially quantified one in C. There are two possibilities:
The first one is that the update only contains elements already present in D. So, ξ(U(D))=ξ(D). The second one is that the update U introduces one ore more new elements to the database. So, ξ(U(D))-ξ(D)≠∅.
In the first situation we distinguish two cases:

Case A: τ corresponds to an instantiation of some C_S.
Because C_S is satisfied in U(D), $C\tau$ is satisfied in U(D).
Case B: τ does not correspond to an instantiation of some C_S.
Because for each i=1,2,...,n, $t_i \in \xi(U(D))$ and $\xi(U(D))=\xi(D)$, $C\tau$ is satisfied in U(D). For the satisfiability of $C\tau$ only depends on facts other than the update, which were present in D and still are present in the updated database.
In the second situation we distinguish three cases:
Case A: τ corresponds to an instantiation of some C_S.
Because C_S is satisfied in U(D), $C\tau$ is satisfied in U(D).
Case B: τ does not correspond to an instantiation of some C_S and $t_1,t_2,...,t_n \in \xi(D)$.
This case corresponds to case B in the first situation. Because of one or more new elements in the database, imported by the update, we now need also the lemma to prove the satisfiability of $C\tau$ in U(D).
Case C: τ does not correspond to an instantiation of some C_S and $t_i \in \xi(U(D))-\xi(D)$ for some i.
Let C: $\forall X_1 \forall X_2 ... \forall X_n [\neg A_1 \vee \neg A_2 \vee ... \neg A_m \vee Q]$. Now, there is a literal $\neg A_j\tau$ which contains element t_i from the set $\xi(U(D))-\xi(D)$, for some $j \in \{1,2,...,m\}$ and which is not unifiable with the negation of U. For if it was unifiable, τ would correspond to an instantiation of some C_S. Because $A_j\tau$ is not unifiable with U and U(D) only consists of the facts in D and the update U, it can only be satisfied by facts in D. However, these facts do not contain the newly introduced element. So, $A_j\tau$ is falsified in U(D). So, $\neg A_j\tau$ is satisfied in U(D) and therefore also $C\tau$ is.

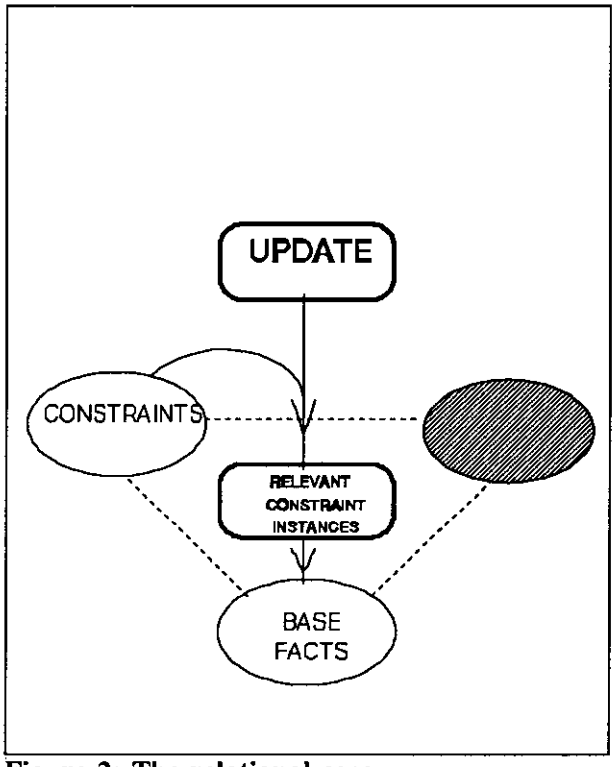


Figure 2: The relational case

By this we proved that for every substitution of the universally quantified variables which are not governed by an existentially quantified one all constraints are satisfied in U(D). And so we have proved the proposition.
The proposition above is illustrated by figure 2.

WARNING:
Bry, Decker and Manthey have written down proposition 1 in the following different form:
All constraints are satisfied in U(D) iff they are satisfied in D and every simplified instance of a constraint relevant to U is satisfied in U(D).

So, their proposition is not the same proposition as which has been proved by Nicolas [NIC79], although they say it is.
Their proposition even is not true. Suppose we have a database D with fact:

$P(a_1,a_3)$

and constraint:

C: $\forall X\ [\neg P(a_1,X) \vee Q(a_2,X)]$

This database is not consistent, because constraint C is violated.
Now by inserting $Q(a_2,a_3)$ we could create a new database which is consistent. So, all constraints are satisfied in U(D), but they are not in D.

The approach in proposition 1 is extendible to deductive databases in two ways, a naive one and the better approach. First, we give a formal description of the naive approach.

2.3 THE DEDUCTIVE CASE: A naive approach

When we are dealing with rules, an update may induce several other implicit changes to the database. These induced updates must also satisfy the integrity constraints. The semantics of integrity constraints are defined according to a canonical interpretation, which consists of true atoms corresponding to the facts which are in the database or derivable from the database by its rules. A unique canonical interpretation can be determined by resticting the rules to be stratified in the sense of [APT87].
Let us describe the concept of induced update more formally:

DEFINITION 5:

Let $C \leftarrow A \wedge A_2 \wedge ... \wedge A_n$ be a deductive rule R. Let U be an update and U(D) the updated database of D.
Let L be a positive (resp. $\neg L$ a negative) literal which is unifiable with A_i (resp. the complement of A_i) in R, for some i. Let τ be the most general unifier of L and A_i. Let $C'=(C\tau)\sigma$, where σ is a substitution by which

$$(A_1 \wedge ... \wedge A_{i-1} \wedge A_{i+1} \wedge ... \wedge A_n)\tau\sigma$$

is true in U(D). C' evaluates to false in D (resp. $\neg C'$ evaluates to true in D). Because rules are range restricted and every variable of C therefore appears also in one or more $A_1,A_2,...,A_n$, C' is ground. C' (resp. $\neg C'$) is now called *directly induced by* L over U(D).
A literal is *induced by* L over U(D) iff

i) it is directly induced by L over U(D), or
ii) it is directly induced by a literal induced by L over U(D).

Now, every literal induced by update U over U(D) is called *an update induced by U*.

Let us give an example, which is based on CASE 2 of EXAMPLE A.

EXAMPLE C
Suppose we have a deductive database D consisting of the facts in EXAMPLE B and with one (range restricted) rule

R: $q(Y,X) \leftarrow q(X,Y) \wedge p(X) \wedge \neg p(Y)$

Note that all the facts which can be derived from R and the facts appear in the fact base of D. So, the fact base of D can be seen as an interpretation.
Let U:=p(b) be an update. This update is unifiable with the literal p(X) of R. Now $\tau=\{X/b\}$ is the most general unifier of U and p(X). We instantiate the body of the rule by applying τ, which becomes:

$$q(b,Y) \wedge p(b) \wedge \neg p(Y) \qquad (*)$$

We are searching for a substitution for which this is true in the updated database. Because the update is ground and the rule is range restricted we always find the instantiated unifiable literal (in this case p(b)) to be equal to the update and therefore true in U(D). So, instead of (*) we can search for a substitution for which

$$q(b,Y) \wedge \neg p(Y)$$

evaluates to true in U(D).
For instance, $\sigma=\{Y/c\}$ is such a substitution. In this case, we find q(b,c) and $\neg$p(c) to be true in the interpretation corresponding to the updated database. The head of the rule q(Y,X) is instantiated by τ and σ. The resulting fact, i.e. q(c,b), is called an (directly) induced update by p(b) only if such a fact was not already present in D which is obviously true. Note that $\sigma_1=\{Y/f\}$ and $\sigma_2=\{Y/g\}$ and respectively are substitutions for which the formula is true in U(D), but the resulting facts q(f,b) and q(g,b) respectively are no induced updates, because they were already present in D.
Note that $\neg$q(b,a), and $\neg$q(b,d) are also directly induced updates, for the complement of p(b) is also unifiable with $\neg$p(Y) in R and $q(X,b) \wedge p(X)$ is true in U(D) for $\sigma=\{X/a\}$ and $\sigma'=\{X/d\}$ respectively and q(a,b), p(a) and q(d,b), p(d) respectively are present in U(D).
So, the collection of directly induced updates is

$$U_i = \{q(c,b), \neg q(b,a), \neg q(b,d)\}.$$

Note that this is also the collection of induced updates by U, because from U_i, the fact base and {p(b)} and rule R there are no other facts derivable.

As another more practical example, suppose we have a database containing the deductive rule R:

$$R: mngr(Y,Z) \leftarrow mngr(X,Z) \wedge coll(X,Y),$$

and the facts

"mngr(john,carl)", "coll(john,george)" and "coll(john,frank)",

which states that "a collegue Y of a person X with manager Z has the same manager Z". Let U = $\neg$mngr(john,carl) be an update. The update is unifiable with the complement of "mngr(X,Z)" in R. (Remember that with this update carl is not a manager of john anymore and so with rule R in the updated database U(D) all collegues of john no longer have carl as their manager also). Now, the most general unifier τ of these two literals is $\tau=\{X/john,Z/carl\}$. The body of the rule without the unifiable literal is partially instantiated by τ to

$$B: coll(john,Y).$$

The corresponding instantiation of the head of R by τ is

H: mngr(Y,carl).

For every substitution σ for which B is true in the updated database we consider the corresponding instantiation of H. Then σ_1={Y/george} and σ_2={Y/frank} are two substitutions which make B true. The negation of the corresponding substitutions of H, $\neg H\sigma_1$ and $\neg H\sigma_2$ must evaluate to true in U(D). (or, intuitively, george and frank, collegues of john, do not have carl as their manager anymore). Now, these ground literals "¬ mngr(george,carl)" and "¬ mngr(frank,carl)" respectively, are called updates directly induced by U.
Note that we only need to compute those induced updates to which some constraint is relevant. The other ones do not influence the integrity of the database U(D) when D is consistent.

In deductive databases all instances of constraints relevant to an update induced by a given update must be checked in order to prove the consistency of the database:

PROPOSITION 2:
Suppose all constraints are satisfied in D.
Then they are satisfied in U(D) iff
every simplified instance of a constraint
- relevant to U, or
- relevant to an update induced by U

is satisfied in U(D).

Proof: The property follows from proposition 1 by reduction to the relational case. Consider the canonical interpretation of D as a relational database. A canonical interpretation consists of true atoms corresponding to the facts which are in the database or derivable from the database by its rules. A unique canonical interpretation can be determined by resticting the rules to be stratified in the sense of [APT87]. Treat the induced updates as explicit updates to this database.

To illustrate this proposition, consider the following example:

EXAMPLE D
Let U:=p(b) be an update to the deductive database D of EXAMPLE C.
Let there be one specified constraint

C: $\forall X \forall Y\ [\neg p(X) \vee \neg r(Y) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,Y)]]$

which is satisfied in D. C is relevant to U. (See EXAMPLE B).
Induced updates are of the form q(_,b) (resp. ¬q(b,_)), which are derived from q(Y,b) (resp. ¬q(b,X)) by a substitution of Y (resp. X) for which $q(b,Y) \wedge \neg p(Y)$ (resp. $q(X,b) \wedge p(X)$) is true in the interpretation of U(D). According to EXAMPLE C, the collection of induced updates is

U_i = {q(c,b),¬q(b,a),¬q(b,d)}.

Now, we must select all the induced updates from U_i which are relevant for C. From the update itself and the induced updates for which this is the case, $\neg q(b,a)$ and $\neg q(b,d)$, five simplified instances are derived:

C_S : $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

C_{S1}: $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

C_{S2}: $\forall X\ [\ \neg p(X) \vee \neg r(a) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,a)]]$,

C_{S3}: $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$ and

C_{S4}: $\forall X\ [\ \neg p(X) \vee \neg r(d) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,d)]]$.

C_S is derived from the update U, C_{S1} and C_{S2} are derived from the induced update $\neg q(b,a)$ and C_{S3} together with C_{S4} are derived from the induced update $\neg q(b,d)$. (Note that C_{S1} and C_{S3} are identical. The method following from proposition 3 in the next section protect us from such redundancies.)
These simplified instances must be satisfied in U(D) in order to prove that C is satisfied in U(D). We can see that C_S, C_{S1}, C_{S2}, C_{S3} and C_{S4} are all satisfied in U(D).

Figure 3 visualizes proposition 2. The first step represented by the dotted arrows reproduces the determination of the induced updates by applying the update to the rules with the aid of the fact base. The second step represented by the continuous arrows reproduces:

- the determination of the simplified instances of the constraints relevant to at least one induced update, and
- the checking of the simplified instances of the constraints with the aid of the fact base.

Following proposition 2 has certain disadvantages: all induced updates are computed even those for which no constraint is relevant. For example, in case of a rule R:

R: $coll(X,Y) \leftarrow mngr(X,Z) \wedge mngr(Y,Z)$,

a constraint C:

C: $\forall X \forall Y\ [\neg st(X,Y,free) \vee \exists Z\ [coll(X,Z) \wedge state(Z,Y,busy)]]$,

(intuitively, if a person X is marked as free by manager Y, then there must be a collegue of X who is busy. So, not all employees of a manager are free. A department must be manned)

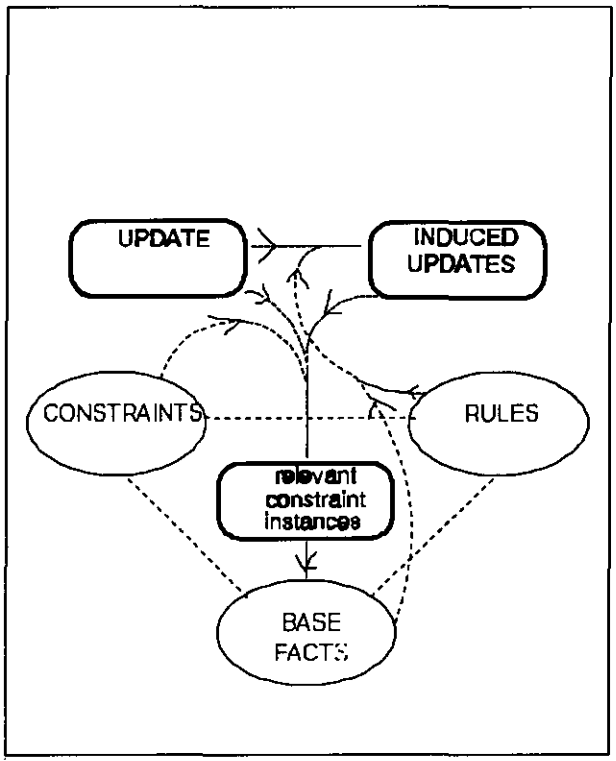


Figure 3: the naive deductive approach

and a lot of mngr(_,george) facts in the database, this could lead to an enormous overhead when a fact mngr(harold,george) is inserted. Suppose that harold does not have any collegues according to the database. Then, for every mngr(_,george)-fact there are two induced updates of this rule, because of the symmetry in the body of the rule. These induced updates do have one of the following forms:

- coll(harold,_), or
- coll(_,harold).

However, C is not relevant to all these induced updates, because these induced updates are not unifiable with the negation of any literal in C.

As one already saw in EXAMPLE D, generating all simplified instances before independently validating them could lead to redundancy. For the example above, the deletion of mngr(john,george) could lead because of rule R to several induced coll(john,_)-deletions and therefore lead to the generation of several identical simplified instances of a constraint. To illustate this, suppose database D contains rule R and the following facts:

coll(john,a)	mngr(a,george)	mngr(john,george),
coll(john,b)	mngr(b,george)	
coll(john,c)	mngr(c,george)	
coll(john,d)		

in which a,b,c and d are names of certain people. By deleting the fact mngr(john,george) the induced updates with respect to R are ¬coll(john,a), ¬coll(john,b) and ¬coll(john,c). (The rule states that two people are collegues when they have a same manager. So, if john no longer has george as a manager, then the rule implies that people who still have george as a manager can no longer be collegues of john).

Then, for all induced updates ¬coll(john,a), ¬coll(john,b) and ¬coll(john,c), which are all relevant with respect to C, there is generated one simplified instance:

C': $\forall Y\ [\neg st(john,Y,free) \vee \exists Z\ [coll(john,Z) \wedge st(Z,Y,busy)]]$

So, integrity maintenance in deductive databases according to proposition 2 has a number of drawbacks. For this reason Bry, Decker and Manthey propose another approach, which is based on generating (not necessarily ground) potential updates from the rules and the update without using the fact base. This approach does not have the drawbacks mentioned earlier.

In the next section this approach will be explained and formalized. Figure 4 is an illustration of this method.

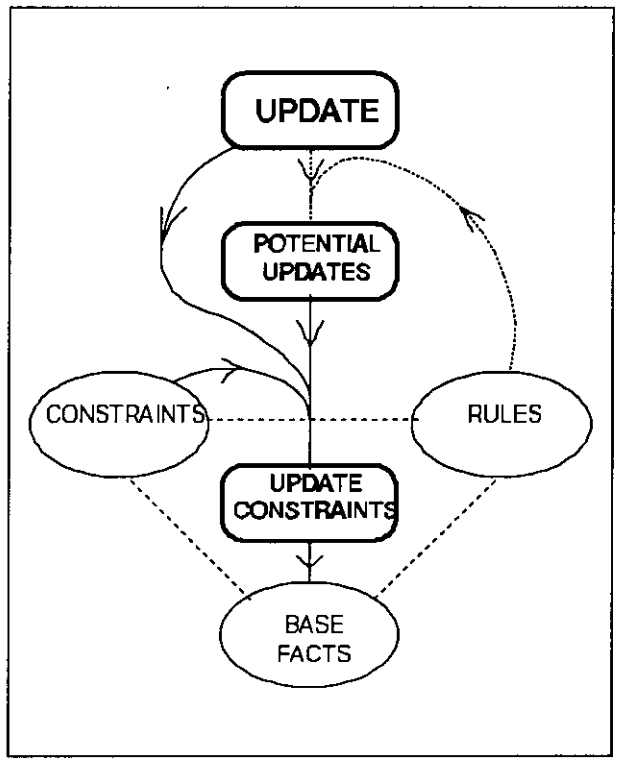


Figure 4: The proposed method

2.4 THE DEDUCTIVE CASE: The proposed method

Compared with the naive deductive approach the only step to be cancelled is the last evaluation phase from which the induced updates are determined. By cancelling this phase we get potential updates. In other words, without applying the σ in DEFINITION 5. The definition of potential updates is therefore a slight reformulation of definition 5:

DEFINITION 6:

Let $B \leftarrow A_1 \wedge A_2 \wedge ... \wedge A_n$ be a deductive rule R. Suppose L is a positive (resp. $\neg L$ a negative) literal which is unifiable with A_i (resp. the complement of A_i) in R, for some i. Let τ be the most general unifier of L and A_i. Let $B'=(B\tau)$. B' (resp. $\neg B'$) *directly depends* on L.
A literal *depends on* L iff
i) it directly depends on L, or
ii) it directly depends on a literal depending on L.

DEFINITION 7:

Let D be a deductive database and U an update.
Each literal which depends on update U with respect to some rule of D is called *a potential update induced by U*.

Potential updates which are not necessarily ground represent all the related updates in the previous approach. For instance, in our earlier mentioned example, the insertion of the mngr(harold,george)-fact causes two potential updates with respect to rule R:

$$coll(X,Y) \leftarrow mngr(X,Z) \wedge mngr(Y,Z),$$

namely coll(X,harold) and coll(harold,Y). All induced updates are instances of these potential updates.

Now, a more abstract example to illustrate the concept of potential updates is given.

EXAMPLE E
Let us return to our deductive database in example C.
Let U:=p(b) be the update. This update is unifiable with the literal p(X) of R. Now $\tau=\{X/b\}$ is the most general unifier of U and p(X). Now, instead of instantiating the body of the rule by applying τ and then deriving the induced updates by finding the substitutions which make the remaining formule true in U(D) like in the previous chapter, we partially instantiate the head of R by τ. In this example, the instantiated head is q(Y,b).
Now, we say that q(Y,b) directly depends on p(b), and so q(Y,b) is a potential update induced by p(b). Note that the other potential update, which directly depends on p(b), is $\neg q(b,X)$. It is derived by the subsitution $\tau'=\{Y/b\}$.
So, the collection of potential updates which directly depend on update U is

$$U_p := \{q(Y,b), \neg q(b,X)\}.$$

To determine the collection of potential updates induced by U, we must also find the collection of the literals which depend on the literals of U_p with respect to rule R. Note that this collection is $U_p' := \{q(b,Y),\neg q(X,b)\}$. From this collection no new literals are derived. So, the collection of all potential updates induced by U with respect to R is

$$\{q(Y,b),\neg q(b,X),q(b,Y),\neg q(X,b)\}.$$

Potential updates express the intuitive idea mentioned in the previous section that the induced updates look, for instance, like q(_,b). Now, we do not use the fact base to derive these induced updates, but postpone the call to this base by first constructing an update constraint with the help of potential updates. The update constraints must be satisfied in U(D) in order to prove the updated database consistent again. (See proposition 3)

Definitions 3 and 4 are now stated for potential updates instead of (ground) updates:

DEFINITION 8:

A constraint C is *relevant to a potential update* PU iff the complement of PU is unifiable with a literal in C.

DEFINITION 9:

Let $C=C(L_1,L_2,...,L_n)$ be an integrity constraint relevant to potential update PU, where $L_1,L_2,...,L_n$ are all the literals (positive or negative) in C. So, there is a literal L_j which is unifiable with the complement of PU. Let σ be the most general unifier of L_j and PU. To obtain a *simplified instance* of C we partially instantiate C by a restriction of σ. Let τ be this restriction to only those universally quantified variables which are not governed by an existentially quantified variable in C. When PU is ground, we simplify $C\tau$ by removing all complements of PU occurring in $C\tau$. The (universal) quantifiers grounded by τ are also removed. Now, the remaining formula, is called the simplified instance of C with respect to PU and L_j. τ is called the *defining substitution* of the simplified instance.

For every simplified instance of a constraint with respect to an update U (resp. a potential update PU) with defining substitution τ_U (resp. τ_{PU}) an update constraint C_U (resp. C_{PU}) is defined which has to be consistent with the updated database.

DEFINITION 10:

Let D be a deductive database, U an update and U(D) the updated database.

Let L be a literal which represents an update or a potential update. For every constraint C, an *update constraint for L with respect to* τ is defined as the universal closure in U(D) of the simplified instance of C relevant to L with defining substitution τ, i.e. $C\tau$.

In [BDM88] two meta-predicates, delta and new, are used in order to define update constraints. Instead of the universal closure of $C\tau$, this definition states that the update constraint is defined as the universal closure of $\neg delta(U,L\tau) \vee new(U,C\tau)$ (or equivalently, $delta(U,L\tau) \rightarrow new(U,C\tau)$), where the metapredicate

delta(U,Lτ) holds if Lτ is satisfied in U(D), but not in D and new(U,Cτ) holds if Cτ is satisfied in U(D). This import of meta-predicates in the object language leads to a mixture of two different concepts which is not supported by the theory. So, we take our own definition of update constraints to continue our exposition.

Note that only the update constraints imply a call to the database query-evaluator. Till then all the work can be done by PROLOG. (See Figure 5).

With definition 10 the analogue of proposition 2 is:

PROPOSITION 3:

Suppose that all constraints are satisfied in D.
Then they are satisfied in U(D)
iff every update constraint
- for U, or
- for a potential update induced by U

is satisfied in U(D).

Let us illustrate proposition 3 with an example.

EXAMPLE F

Let U:=p(b) be an update to the deductive database D of EXAMPLE C with one specified constraint

C: $\forall X \forall Y\ [\neg p(X) \vee \neg r(Y) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,Y)]]$

which is satisfied in D. C is relevant to U. According to EXAMPLE E, the potential updates are q(Y',b), ¬q(b,X'), q(b,Y") and ¬q(X",b). Note that ¬q(b,X') and ¬q(X",b) are the only potential updates to which C is relevant.

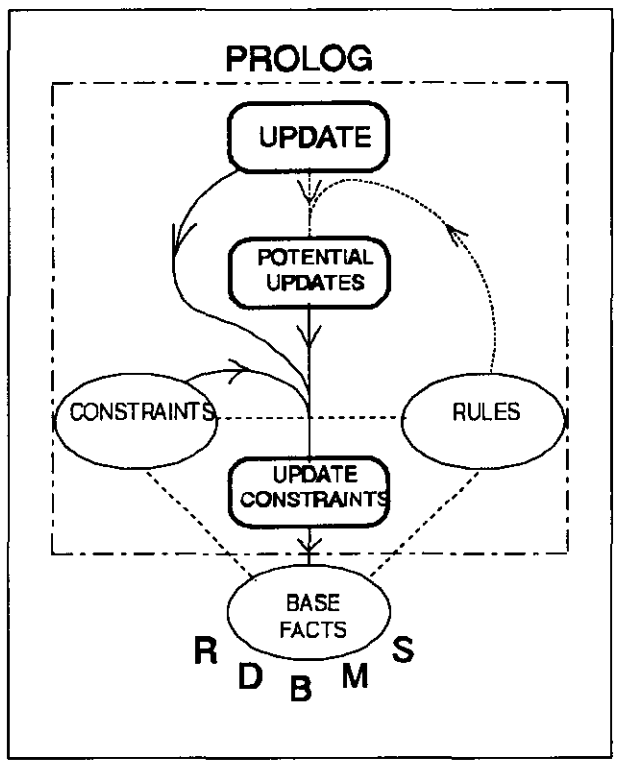


Figure 5: Coupling PROLOG to a DBMS

Because ¬q(b,X') can be unified to the complement of q(X,Z) and q(W,Y) respectively in C, we derive from this potential update two simplified instances of C:

C_1: $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

with defining substitution $\tau_1=\{X/b\}$ and

C_2: $\forall X\ [\neg p(X) \vee \neg r(X') \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,X')]]$,

with defining substitution $\tau_2=\{Y/X'\}$.
Analogously, we derive from $\neg q(X'',b)$ two simplified instances of C:

C_3: $\forall Y\ [\neg p(X'') \vee \neg r(Y) \vee \exists Z \exists W\ [q(X'',Z) \wedge q(W,Y)]]$,

with defining substitution $\tau_3=\{X/X''\}$ and

C_4: $\forall X\ [\neg p(X) \vee \neg r(b) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,b)]]$,

with defining substitution $\tau_4=\{Y/b\}$.
Besides these four instances, we have a simplified instance derived from U:

C^U: $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

with defining substitution $\tau_U=\{X/b\}$.
So, the update constraints for U with respect to τ_U, for $\neg q(b,X')$ and $\neg q(X'',b)$ with respect to τ_1, τ_2 and τ_3, τ_4 respectively are:

C_U : $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

C_{U1}: $\forall Y\ [\ \neg r(Y) \vee \exists Z \exists W\ [q(b,Z) \wedge q(W,Y)]]$,

C_{U2}: $\forall X' \forall X\ [\ \neg p(X) \vee \neg r(X') \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,X')]]$,

C_{U3}: $\forall X'' \forall Y\ [\ \neg p(X'') \vee \neg r(Y) \vee \exists Z \exists W\ [q(X'',Z) \wedge q(W,Y)]]$ and

C_{U4}: $\forall X\ [\ \neg p(X) \vee \neg r(b) \vee \exists Z \exists W\ [q(X,Z) \wedge q(W,b)]]$.

Note that the induced updates which are relevant to C in EXAMPLE D, $\neg q(b,a)$ and $\neg q(b,d)$, are instances of the potential update $\neg q(b,X')$. Further, C_{S1} and C_{S3} (C_{S2} and C_{S4}, respectively) are implied by C_{U1} (C_{U2} respectively). (The simplified instance of the constraint, C_S, derived from update U in EXAMPLE D and the update constraint for U, i.e. C_U, in this example are the same). Note that C_U and C_{U1} are the same. So, we have to check just one of these constraints. This argument is also applicable to C_{U2} and C_{U3}. Note also that C_{U4} does not have any instance which corresponds to a simplified instance of C with respect to an induced update. In other words, the generation of the potential update $\neg q(X'',b)$ and the related update constraint, C_{U4}, was redundant.
The most negative point is that C_{U2} and C_{U3} are equal to the original constraint C, so we have nothing gained here, because we want to get an instantiation of C which constrains the search space in the database.

In order to explain for what purpose Bry, Decker and Manthey introduce their meta-predicate "delta", let us consider a potential update. For this update we constructed some simplified constraints which have to be satisfied in the updated database. In fact we only have to check those instances of these simplified constraints which correspond to instances of the potential update corresponding to induced updates. So, with predicate "new" they express the simplified instance of potential updates which have to be evaluated in the updated database. And with meta-predicate "delta" they express that we only

want to evaluate the new-predicate (read simplified constraint) if we deal with an induced update related to the potential update, i.e. an instance of the potential update which are satisfied in the updated database but not in the database before the update.

For instance, suppose we have a constraint C: $\forall X \forall Y$ [$\neg w(X,Y) \vee s(X,Y)$], a potential update w(b,Y) and two induced updates corresponding to this potential update, w(b,a) and w(b,c), and let w(b,b) and w(b,d) all other w(b,_)-facts in the database, which are not affected by the update. Suppose further that s(_,_)-facts are not affected by the update. The simplified update C' with respect to the potential update is C': $\forall Y$ [$\neg w(b,Y) \vee s(b,Y)$]. Now C' has to be evaluated in the updated database.
But there exists some redundancy in such an evaluation, because C' must be instantiated by all unaffected w(b,_)-facts in the database. These instantiations correspond to instantiations of constraints which were already satisfied in the old database state. Because this part of the database has not changed the instantiations of the constraints will still be satisfied. So, to prevend such evaluations Bry, Decker and Manthey introduced the delta-predicate to check if we deal with an induced update related to the potential update before evaluating the simplified constraint. If we deal with an induced update the simplified constraint can further be instantiated by this induced update before evaluating it.
So, in our example delta(U,w(b,Y)) checks for which instantiation of Y we have an induced update. If τ is such an evaluation then C'τ will be evaluated in the updated database. So eventually, in this example [$\neg w(b,a) \vee s(b,a)$] and [$\neg w(b,c) \vee s(b,c)$] are evaluated.

Now, the main difference of the naive integrity checking approach and the proposed approach is that in the first case all possible induced updates are generated, even those which are not relevant to a constraint. In the latter case, the delta handles only induced updates which are instances of potential updates relevant to some constraint. So, here all generated induced updates are relevant to some constraint.

However, as we will now see the evaluation of the predicate delta is sometimes unnecessary.
In our example, we derived C_{U1}: $\forall Y$ [$\neg r(Y) \vee \exists Z \exists W$ [$q(b,Z) \wedge q(W,Y)$]] from potential update $\neg q(b,X')$. As noted, we have two induced updates $\neg q(b,a)$ and $\neg q(b,d)$ which are instances of the potential update $\neg q(b,X')$. Here, the simplified instances of C which correspond to the induced updates relevant to C with respect to the first occurrence of the predicate q are equal to C_{U1}. This is caused by the existential quantified variable Z. So, in this case a meta-predicate delta to check first if we deal with an induced update before checking the simplified constraint is not necessary, because it causes no instantiation of the simplified constraint.

2.5 EXTENSIONS OF THE PROPOSED METHOD

Besides fact updates, rule updates or even integrity constraint updates are also possible.

- New rules will generate new facts. We can consider these as induced facts. At this point we can go further with these facts as if these facts were a set of updates. To this set we can apply some integrity maintenance method. If some constraint is violated then the rule must be withdrawn.
- New constraints must be consistent with the other constraints. If that is the case, the constraint is accepted. However, the database can violate this constraint. No other constraint is violated by the facts and rules, because the database was supposed to be consistent before the transaction.

The only task to be done is to search for relevant facts which could violate the new constraint. When there are facts which violate this constraint, a choice must be made to decide which facts must be deleted to restore the integrity of the database.

§3 CONCLUSIONS

The proposed method for integrity maintenance is suitable in order to optimize the constraint check before actually consulting the fact base by the query-evaluator. It also seems possible to build a deductive database management system with PROLOG linked to a relational database management system which has an acceptable degree of efficiency as is shown empirically by Das and Williams [DAW89]. Further, it seems (see section 2.5) that this method can be easily extended to rule updates and constraint updates. But the proposed method has also some drawbacks. For instance, it may happen that some potential updates and their related update constraints are redundant. In a next article some possible solutions to the redundancies mentioned above will be proposed.

REFERENCES

[APT87] Apt, K.R., Blair, H. and Walker, A.
"Towards a theory of declarative knowledge",
In Minker, J., Proc. Workshop on Deductive Databases and Logic Programming, Aug., 1987.

[ASI85] Asirelli, Patricia, Michèle de Santis and Maurizio Martelli
"Integrity Constraints in Logic Databases",
J. Logic programming (3), pp. 221-232,
1985.

[BDM88] Bry, François, Hendrik Decker and Rainer Manthey,
"A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases",
Proceedings of the Conference Extending Data Base Technology,
14-18 March 1988, Venice, LCNS Springer Verlag.
also: ECRC Technical Report KB-16,
7 Nov. 1987.

[BRO89] Brock, E.O. de
"De Grondslagen van Semantische Databases",
Academic Press,
1989.

[COD90] Codd, T
"The Relational Model for Database Management",
version 2,
Addison-Wesley,
1990.

[DAT90] Date, C.J.
"Relational Database, writings 1985-1989",
Addison-Wesley,
1990.

[DAW89] Das, S.K. and M.H. Williams
"Integrity Checking Methods in Deductive Databases: A Comparative Evaluation",
In: M.H. Williams (ED.), Proceedings of the Seventh British National Conference on Databases,
Cambridge University Press,
1989.

[DEC86] Decker. H.
"Integrity Enforcement on Deductive Databases",
Proceedings Expert Database Systems, pp. 271-285,
Charleston, Sc.,
1986.

[GAL78] Gallaire, H. and J. Minker
"Logic and Databases",
Plenum Press,
New York 1978.

[GAL84] Gallaire, H., J. Minker and J.-M. Nicolas
"Logic and Databases, a deductive approach",
Computing Surveys 16:1, pp. 154-185,
1984.

[LLO85] Lloyd, J.W. and R.W. Topor
"A Basis for Deductive Database Systems",
J. Logic Programming 2, pp. 93-109.
1985.

[LLO86] Lloyd, J.W. and R.W. Topor
"A Basis for Deductive Database Systems II",
J. Logic Programming 3 (1), pp. 55-67.
1986.

[LLO87] Lloyd, J.W., E.A. Sonenberg and R.W. Topor
"Integrity Constraint Checking in Stratified Databases",
J. Logic Programming 4, pp. 331-343,
1987.

[MAI83] Maier, D.
"The Theory of Relational Databases",
Computer Science Press,
Rockville, Md.,
1983.

[MIN88] Minker, J.
"Foundations of Deductive Databases and Logic Programming",
Morgan Kaufmann,
Los Altos,
1988.

[MKR88] Minker, J.
"Perspectives in Deductive Databases",
J. Logic Programming 5, pp. 33-60,
1988.

[NIC79] Nicolas, J.M.
"A property of logical formulas corresponding to integrity constraints on data base relations.
Preprints of the Workshop on "Formal bases for data bases",
Toulouse, 1979.

[NIC82] Nicolas, J.M.
"Logic for Improving Integrity Checking in Relational Databases".
Acta Informatica 18 (3), pp. 227-253,
1982.

[NIY78] Nicolas, J.M. and K. Yazdanian
"Integrity Checking in Deductive Data Bases",
in "Logic and Databases",[GAL78], pp. 325-346,
1978.

[SAD88] Sadri, Fariba, and Robert Kowalski
"A Theorem-Proving Approach to Database Integrity",
in "Foundations of Deductive Databases and Logic Programming", J. Minker [MIN88],
Chapter 9.

[SCI86] Sciore, Edward and David S. Warren
"Towards an Integrated Database-PROLOG System",
pp. 293-306 uit "Expert Database Systems" van L. Kerschberg,
1986.

[SOP86] Soper, P.J.R.
"Integrity Checking in Deductive Databases",
Math. Sc. Thesis,
Department of Computing, Imperial College,
University of London, 1986.

[ULL88] Ullman Jeffrey D.
"Principles of Database and Knowledge Base Systems"
Volume 1,
Computer Science Press,
Stanford University, 1988.

In this series appeared:

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.

91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.
91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.