Consistent Query Answers in Inconsistent Databases

Marcelo Arenas
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
marenas@ing.puc.cl

Leopoldo Bertossi
Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de Computación
Casilla 306, Santiago 22, Chile
bertossi@ing.puc.cl

Jan Chomicki
Monmouth University
Department of Computer Science
West Long Branch, NJ 07764
chomicki@monmouth.edu

Abstract

In this paper we consider the problem of the logical characterization of the notion of consistent answer in a relational database that may violate given integrity constraints. This notion is captured in terms of the possible repaired versions of the database. A method for computing consistent answers is given and its soundness and completeness (for some classes of constraints and queries) proved. The method is based on an iterative procedure whose termination for several classes of constraints is proved as well.

1 Introduction

Integrity constraints capture an important normative aspect of every database application. However, it is often the case that their satisfaction cannot be guaranteed, allowing for the existence of inconsistent database instances. In that case, it is important to know which query answers are consistent with the integrity constraints and which are not. In this paper, we provide a logical characterization of consistent query answers in relational databases that may be inconsistent with the given integrity constraints. Intuitively, an answer to a query posed to a database that violates the integrity constraints will be consistent in a precise sense: It should be the same as the answer obtained from any minimally repaired version of the original database. We also provide a method for computing such answers and prove its properties. On the basis of a query Q, the method computes, using an iterative procedure, a new query $T_\omega(Q)$ whose evaluation in an arbitrary, consistent or inconsistent, database returns the set of consistent answers to the original query Q. We envision the application of our results in a number of areas:

Data warehousing. A data warehouse contains data coming from many different sources. Some of it typically does not satisfy the given integrity constraints. The usual approach is thus to *clean* the data by removing inconsistencies before the data is stored in the warehouse [6]. Our results make it possible to determine which data is already clean and proceed to safely remove unclean data. Moreover, a different scenario becomes possible, in which the inconsistencies are not removed but rather query answers are marked as "consistent" or "inconsistent". In this way, information loss due to data cleaning may be prevented.

Database integration. Often many different databases are integrated together to provide a single unified view for the users. Database integration is difficult since it requires the resolution of many different kinds of discrepancies of the integrated databases. One possible discrepancy is due to different sets of integrity constraints. Moreover, even if every integrated database *locally* satisfies the same integrity constraint, the constraint may be *globally* violated. For example, different databases may assign different addresses to the same student. Such conflicts may fail to be resolved at all and inconsistent data cannot be "cleaned" because of the autonomy of different databases. Therefore, it is important to be able to find out, given a set of local integrity constraints, which query answers returned from the integrated database are consistent with the constraints and which are not.

Active and reactive databases. A violation of integrity constraints may be acceptable under the provision that it will be repaired in the near future. For example, the stock level in a warehouse may be allowed to fall below the required minimum if the necessary replenishments have been ordered. During this temporary inconsistency, however, query answers should give an indication whether they are consistent with the constraints or not. This problem is particularly acute in active databases that allow such consistency lapses. The result of evaluating a trigger condition that is consistent with the integrity constraints should be treated differently from the one that isn't.

The following example presents the basic intuitions behind the notion of consistent query answer.

Example 1. Consider a database subject to the following IC:

$$\forall x(P(x) \supset Q(x)).$$

The instance

$$\{P(a), P(b), Q(a), Q(c)\}$$

violates this constraint. Now if the query asks for all x such that $P(x)$, only a is returned as an answer consistent with the integrity constraint.

The plan of this paper is as follows. In section 2 we introduce the basic notions of our approach, including those of *repair* and *consistent query answer*. In section 3 we show a method how to compute the query $T_\omega(Q)$ for a given first-order query Q. In subsequent sections, the properties of this method are analyzed: soundness in section 4, completeness in section 5, and termination in section 6. In section 7 we discuss related work. In section 8 we conclude and outline some of the prospects for future work in this area. The proofs are given in the appendix.

2 Basic Notions

In this paper we assume we have a fixed database schema and a fixed infinite database domain D. We also have a first order language based on this schema with names for the elements of D. We assume that elements of the domain with different names are different. The instances of the schema are finite structures for interpreting the first order language. As such they all share the given domain D, nevertheless, since relations are finite, every instance has a finite active domain which is a subset of D. As usual, we allow built-in predicates that have infinite extensions, identical for all database instances. There is also a set of integrity constraints IC, expressed in that language, which the database instances are expected to satisfy. We will assume that IC is consistent in the sense that there is a database instance that makes it true.

Definition 1. (*Consistency*) A database instance r is *consistent* if r satisfies IC in the standard model-theoretic sense, that is, $r \vDash IC$; r is inconsistent otherwise.

This paper addresses the issue of obtaining meaningful and useful query answers in *any*, consistent or inconsistent, database. It is well known how to obtain query answers in consistent databases. Therefore, the challenging part is how to deal with the inconsistent ones.

2.1 Repairs

Given a database instance r, we denote by $\Sigma(r)$ the set of formulas $\{P(\bar{a}) | r \vDash P(\bar{a})\}$, where the Ps are relation names and $\bar{a}$ is ground tuple.

Definition 2. (*Distance*) The *distance* $\Delta(r, r')$ between database instances r and r' is the symmetric difference:

$$\Delta(r, r') = (\Sigma(r) - \Sigma(r')) \cup (\Sigma(r') - \Sigma(r)).$$

Definition 3. For the instances r, r', r'', $r' \leq_r r''$ if $\Delta(r, r') \subseteq \Delta(r, r'')$, i.e., if the distance between r and r' is less than or equal to the distance between r and r''.

Notice that built-in predicates do not contribute to the Δs because they have fixed extensions, identical in every database instance.

Definition 4. (*Repair*) Given database instances r and r', we say that r' is a *repair* of r if $r' \vDash IC$ and r' is $\leq_r$*-minimal* in the class of database instances that satisfy the ICs.

Clearly, what constitutes a repair depends on the given set of integrity constraints. In the following we assume that this set is fixed.

Example 2. Let us consider a database schema with two unary relations P and Q and domain $D = \{a, b, c\}$. Assume that for an instance r, $\Sigma(r) = \{P(a), P(b), Q(a), Q(c)\}$, and let $IC = \{\forall x(P(x) \supset Q(x))\}$. Clearly, r does not satisfy IC because $r \vDash P(b) \wedge \neg Q(b)$.

In this case we have two possibles repairs for r. First, we can falsify $P(b)$, obtaining an instance r' with $\Sigma(r') = \{P(a), Q(a), Q(c)\}$. As a second alternative, we can make $Q(b)$ true, obtaining an instance r'' with $\Sigma(r'') = \{P(a), P(b), Q(a), Q(b), Q(c)\}$.

The definition of a repair satisfies certain desirable and expected properties. Firstly, a consistent database does not need to be repaired, because if r satisfies IC, then, by the minimality condition wrt the relation $\leq_r$, r is the only repair of itself (since $\Delta(r, r)$ is empty). Secondly, any database r can always be repaired because there is a database r' that satisfies IC, and $\Delta(r, r')$ is finite.

Example 3. (motivated by [19]) Consider the IC saying that C is the only supplier of items of class T_4:

$$\forall(x, y, z)(Supply(x, y, z) \wedge Class(z, T_4) \supset x = C). \quad (1)$$

The following database instance r_1 violates the IC:

Supply		
C	D_1	I_1
D	D_2	I_2

Class	
I_1	T_4
I_2	T_4

The only repairs of this database are

Supply		
C	D_1	I_1

Class	
I_1	T_4
I_2	T_4

and

Supply		
C	D_1	I_1
D	D_2	I_2

Class	
I_1	T_4

Example 4. (motivated by [19]) Consider the IC:

$$\forall(x,y)(Supply(x,y,I_1) \supset Supply(x,y,I_2)), \qquad (2)$$

saying that item I_2 is supplied whenever item I_1 is supplied; and the following inconsistent instance, r_2, of the database

Supply		
C	D_1	I_1
C	D_1	I_3

This instance has two repairs:

Supply		
C	D_1	I_1
C	D_1	I_2
C	D_1	I_3

and

Supply		
C	D_1	I_3

Example 5. Consider a student database. $Student(x,y,z)$ means that x is the student number, y is the student's name, and z is the student's address. The two following ICs state that the first argument is a key of the relation

$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset y = u),$$
$$\forall(x,y,z,u,v)(Student(x,y,z) \wedge Student(x,u,v) \supset z = v).$$

The inconsistent database instance r_3

Student		
S_1	N_1	D_1
S_1	N_2	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

has two repairs:

Student		
S_1	N_1	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

and

Student		
S_1	N_2	D_1

Course		
S_1	C_1	G_1
S_1	C_2	G_2

2.2 Consistent query answers

We assume all queries are in prefix disjunctive normal form.

Definition 5. A formula Q is a *query* if it has the following syntactical form:

$$\bar{Q}\bigvee_{i=1}^{s}(\bigwedge_{j=1}^{m_i} P_{i,j}(\bar{u}_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg Q_{i,j}(\bar{v}_{i,j}) \wedge \psi_i),$$

where $\bar{Q}$ is a sequence of quantifiers and every ψ_i contains only built-in predicates. If $\bar{Q}$ contains only universal quantifiers, then we say that Q is a *universal query*. If $\bar{Q}$ contains existential (and possibly universal) quantifiers, we say that Q is *non-universal* query.

Definition 6. (*Query answer*) A (ground) tuple $\bar{t}$ is an *answer* to a query $Q(\bar{x})$ in a database instance r if $r \models Q(\bar{t})$. A (ground) tuple $\bar{t}$ is an *answer* to a set of queries $\{Q_1, \ldots, Q_n\}$ if $r \models Q_1 \wedge \cdots \wedge Q_n$.

Definition 7. (*Consistent answer*) Given a set of integrity constraints, we say that a (ground) tuple $\bar{t}$ is a *consistent answer* to a query $Q(\bar{x})$ in a database instance r, and we write $r \models_c Q(\bar{t})$ (or $r \models_c Q(\bar{x})[\bar{t}]$), if for every repair r' of r, $r' \vDash Q(\bar{t})$. If Q is a sentence, then *true* (*false*) is a *consistent answer* to Q in r, and we write $r \models_c Q$ ($r \not\models_c Q$), if for every repair r' of r, $r' \vDash Q$ ($r' \nvDash Q$).

Example 6. (example 3 continued) The only consistent answer to the query $Class(z,T_4)$, posed to the database instance r_1, is I_1 because $r_1 \models_c Class(z,T_4)[I_1]$.

Example 7. (example 4 continued) The only consistent answer to the query $Supply(C,D_1,z)$, posed to the database instance r_2, is I_3 because $r_2 \models_c Supply(C,D_1,z)[I_3]$.

Example 8. (example 5 continued) By considering all the repairs of the database instance r_3, we obtain C_1 and C_2 as the consistent answers to the query $\exists z Course(S_1,y,z)$, posed to r_3. For the query $\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))$, we obtain no (consistent) answers.

3 The General Approach

We present here a method to compute consistent answers to queries. Given a query Q, the query $T_\omega(Q)$ is defined based on the notion of *residue* developed in the context of semantic query optimization (SQO) [5]. In the context of deductive databases, SQO is used to optimize the process of answering queries using the semantic knowledge about the domain that is contained in the ICs. In this case, the basic assumption is that the ICs are satisfied by the database. In our case, since we allow inconsistent databases, we do not assume the satisfaction of the ICs while answering queries. A first attempt to obtain consistent answers to a query $Q(\bar{x})$ may be to use *query modification*, i.e., ask the query $Q(\bar{x}) \wedge IC$. However, this does not work, as we obtain *false* as the answer if the DB is inconsistent. Instead, we iteratively modify the query Q using the residues. As a result, we obtain the query $T_\omega(Q)$ with the property that the set of all answers to $T_\omega(Q)$ is the same as as the set of consistent answers to Q. (As shown later, the property holds only for restricted classes of queries and constraints.)

3.1 Generating residues in relational DBs

We consider only universal constraints. We begin by transforming every integrity constraint to the standard format (*expansion* step).

Definition 8. An integrity constraint is in *standard format* if it has the form

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi),$$

where $\forall$ represents the universal closure of the formula, $\bar{x}_i$, $\bar{y}_i$ are tuples of variables and ψ is a formula that mentions only built–in predicates, in particular, equality.

Notice that in such an IC there are no constants in the P_i, Q_i; if they are needed they can be pushed into ψ.

Many usual ICs that appear in DBs can be transformed to the standard format, e.g. functional dependencies, set inclusion dependencies of the form $\forall\bar{x}(P(\bar{x}) \supset Q(\bar{x}))$, transitivity constraints of the form $\forall x,y,z(P(x,y) \wedge P(y,z) \supset P(x,z))$. The usual ICs that appear in SQO in deductive databases as rules [5] can be also accommodated in this format, including rules with disjunction and logical negation in their heads. An inclusion dependency of the form $\forall\bar{x}(P(\bar{x}) \supset \exists y\, Q(\bar{x},y))$ cannot be transformed to the standard format.

After the expansion of *IC*, rules associated with the database schema are generated. This could be seen as considering an instance of the database as an extensional database expanded with new rules, and so obtaining an associated deductive database where semantical query optimization can be used.

For each predicate, its negative and positive occurrences in the ICs (in standard format) will be treated separately with the purpose of generating corresponding residues and rules. First, a motivating example.

Example 9. Consider the IC $\forall x\,(\neg P(x) \vee Q(x))$. If $Q(x)$ is false, then $\neg P(x)$ must be true. Then, when asking about $\neg Q(x)$, we make sure that $\neg P(x)$ becomes true. That is, we generate the query $\neg Q(x) \wedge \neg P(x)$ where $\neg P(x)$ is the residue attached to the query.

For each IC in standard format

$$\forall(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi), \tag{3}$$

and each positive occurrence of a predicate $P_j(\bar{x}_j)$ in it, the following residue for $\neg P_j(\bar{x}_j)$ is generated

$$\bar{Q}(\bigvee_{i=1}^{j-1} P_i(\bar{x}_i) \vee \bigvee_{i=j+1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{n} \neg Q_i(\bar{y}_i) \vee \psi), \tag{4}$$

where $\bar{Q}$ is a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{x}_j$.

If $R_1, \ldots, R_r$ are all the residues for $\neg P_j$, then the following rule is generated:

$$\neg P_j(\bar{w}) \longmapsto \neg P_j(\bar{w})\{R_1(\bar{w}), \ldots, R_r(\bar{w})\},$$

where $\bar{w}$ are new variables. If there are no residues for $\neg P_j$, then the rule $\neg P_j(\bar{w}) \longmapsto \neg P_j(\bar{w})$ is generated.

For each negative occurrence of a predicate $Q_j(\bar{y}_j)$ in (3), the following residue for $Q_j(\bar{y}_j)$ is generated

$$\bar{Q}(\bigvee_{i=1}^{m} P_i(\bar{x}_i) \vee \bigvee_{i=1}^{j-1} \neg Q_i(\bar{y}_i) \vee \bigvee_{i=j+1}^{n} \neg Q_i(\bar{y}_i) \vee \psi),$$

where $\bar{Q}$ is a sequence of universal quantifiers over all the variables in the formula not appearing in $\bar{y}_j$.

If $R'_1, \ldots, R'_s$ are all the residues for $Q_j(\bar{y}_j)$, the following rule is generated:

$$Q_j(\bar{u}) \longmapsto Q_j(\bar{u})\{R'_1(\bar{u}), \ldots, R'_s(\bar{u})\}.$$

If there are no residues for $Q_j(\bar{y}_j)$, then the rule $Q_j(\bar{u}) \longmapsto Q_j(\bar{u})$ is generated. Notice that there is exactly one new rule for each positive predicate, and exactly one rule for each negative predicate.

If there are more than one positive (negative) occurrences of a predicate, say P, in an IC, then more then one residue is computed for $\neg P$. In some cases, e.g., for functional dependencies, the subsequent residues will be redundant. In other cases cases, e.g., for *transitivity constraints*, multiple residues are not redundant.

Example 10. If we have the following ICs in standard format

$$IC = \{\forall x(R(x) \vee \neg P(x) \vee \neg Q(x)), \forall x(P(x) \vee \neg Q(x))\},$$

the following rules are generated:

$$\begin{array}{rcl}
P(x) & \longmapsto & P(x)\{R(x) \vee \neg Q(x)\} \\
Q(x) & \longmapsto & Q(x)\{R(x) \vee \neg P(x), P(x)\} \\
R(x) & \longmapsto & R(x) \\
\neg P(x) & \longmapsto & \neg P(x)\{\neg Q(x)\} \\
\neg Q(x) & \longmapsto & \neg Q(x) \\
\neg R(x) & \longmapsto & \neg R(x)\{\neg P(x) \vee \neg Q(x)\}.
\end{array}$$

Notice that no rules are generated for built-in predicates, but such predicates may appear in the residues. They have fixed extensions and thus cannot contribute to the violation of an IC or be modified to make an IC true. For example, if we have the IC $\forall x,y,z(\neg P(x,y) \vee \neg P(x,z) \vee y = z)$, and the database satisfies $P(1,2), P(1,3)$, the IC cannot be made true by making $2 = 3$.

Once the rules have been generated, it is possible to simplify the associated residues. In every new rule of the form $P(\bar{u}) \longmapsto P(\bar{u})\{R_1(\bar{u}),\ldots,R_r(\bar{u})\}$ the auxiliary quantifications introduced in the expansion step are eliminated (both the quantifier and the associated variable in the formula) from the residues by the process inverse to the one applied in the expansion. The same is done with rules of the form $\neg P \longmapsto \neg P\{\cdots\}$.

3.2 Computing $\mathrm{T}_\omega(Q)$

In order to determine consistent answers to queries in arbitrary databases, we will make use of a family of operators consisting of T_n, $n \geq 0$, and T_ω.

Definition 9. The application of an operator T_n to a query is defined inductively by means of the following rules

1. $\mathrm{T}_n(\Box) := \Box$, $\mathrm{T}_n(\neg\Box) := \neg\Box$, for every $n \geq 0$ ($\Box$ is the empty clause).

2. $T_0(\varphi) := \varphi$.

3. For each predicate $P(\bar{u})$, if there is a rule $P(\bar{u}) \longmapsto P(\bar{u})\{R_1(\bar{u}),\ldots,R_r(\bar{u})\}$, then
$$\mathrm{T}_{n+1}(P(\bar{u})) := P(\bar{u}) \wedge \bigwedge_{i=1}^{r} \mathrm{T}_n(R_i(\bar{u})).$$
If $P(\bar{u})$ does not have residues, then $\mathrm{T}_{n+1}(P(\bar{u})) := P(\bar{u})$.

4. For each negated predicate $\neg Q(\bar{v})$, if there is a rule $\neg Q(\bar{v}) \longmapsto \neg Q(\bar{v})\{R'_1(\bar{v}),\ldots,R'_s(\bar{v})\}$, then
$$\mathrm{T}_{n+1}(\neg Q(\bar{v})) := \neg Q(\bar{v}) \wedge \bigwedge_{i=1}^{s} \mathrm{T}_n(R'_i(\bar{v})).$$
If $\neg Q(\bar{v})$ does not have any residues, then $\mathrm{T}_{n+1}(\neg Q(\bar{u})) := \neg Q(\bar{u})$.

5. If φ is a formula in prenex disjunctive normal form, that is,
$$\varphi = \bar{Q} \bigvee_{i=1}^{s} \Big(\bigwedge_{j=1}^{m_i} P_{i,j}(\bar{u}_{i,j}) \wedge \bigwedge_{j=1}^{n_i} \neg Q_{i,j}(\bar{v}_{i,j}) \wedge \psi_i \Big),$$
where $\bar{Q}$ is a sequence of quantifiers and ψ_i is a formula that includes only built–in predicates, then for every $n \geq 0$:
$$\mathrm{T}_n(\varphi) := \bar{Q} \bigvee_{i=1}^{s} \Big(\bigwedge_{j=1}^{m_i} \mathrm{T}_n(P_{i,j}(\bar{u}_{i,j})) \wedge \bigwedge_{j=1}^{n_i} \mathrm{T}_n(\neg Q_{i,j}(\bar{v}_{i,j})) \wedge \psi_i \Big).$$

Definition 10. The application of operator T_ω on a query is defined as $\mathrm{T}_\omega(\varphi) = \bigcup_{n<\omega} \{\mathrm{T}_n(\varphi)\}$.

Example 11. (example 10 continued) For the query $\neg R(x)$ we have $\mathrm{T}_1(\neg R(x)) = \neg R(x) \wedge (\neg P(x) \vee \neg Q(x))$, $\mathrm{T}_2(\neg R(x)) = \neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))$ and finally $\mathrm{T}_3(\neg R(x)) = \mathrm{T}_2(\neg R(x))$. We have reached a fixed point and then
$$\mathrm{T}_\omega(\neg R(x)) = \{\neg R(x), \neg R(x) \wedge (\neg P(x) \vee \neg Q(x)), \neg R(x) \wedge ((\neg P(x) \wedge \neg Q(x)) \vee \neg Q(x))\}.$$

We show first that the operator T_ω conservatively extends standard query evaluation on consistent databases.

Proposition 1. Given a database instance r and a set of integrity constraints IC, such that $r \vDash IC$, then for every query $Q(\bar{x})$ and every natural number n: $r \vDash \forall \bar{x}(Q(\bar{x}) \equiv \mathrm{T}_n(Q(\bar{x})))$.

Corollary 1. Given a database instance r and a set of integrity constraints IC, such that $r \vDash IC$, then for every query $Q(\bar{x})$ and every tuple $\bar{t}$: $r \vDash Q(\bar{t})$ if and only if $r \vDash \mathrm{T}_\omega(Q(\bar{t}))$.

4 Soundness

Now we will show the relationship between consistent answers to a query Q in a database instance r (definition 7) and answers to the query $\mathrm{T}_\omega(Q)$ (definition 6). We show that $\mathrm{T}_\omega(Q)$ returns only consistent answers to Q.

Theorem 1. (*Soundness*) Let r be a database instance, IC a set of integrity constraints and $Q(\bar{x})$ a query (see definition 5) such that $r \vDash \mathrm{T}_\omega(Q(\bar{x}))[\bar{t}]$. If Q is universal or non-universal and domain independent[20], then $\bar{t}$ is a consistent answer to Q in r (in the sense of definition 7), that is, $r \models_c Q(\bar{t})$.

The second condition in the theorem excludes non-universal, but domain dependent queries like $\exists x \neg P(x)$.

Example 12. (example 6 continued) The IC (1) transformed into the standard format becomes
$$\forall(x,y,z,w)(\neg Supply(x,y,z) \vee \neg Class(z,w) \vee w \neq T_4 \vee x = C).$$

The following rule is generated:
$$Class(z,w) \longmapsto Class(z,w) \{\forall(x,y)(\neg Supply(x,y,z) \vee w \neq T_4 \vee x = C)\}.$$

Given the database instance r_1 that violates the IC as before, if we pose the query $Class(z,T_4)$, asking for the items of class T_4, directly to r_1, we obtain I_1 and I_2. Nevertheless, if we pose the query $\mathrm{T}_\omega(Class(z,T_4))$, that is
$$\{Class(z,T_4), Class(z,T_4) \wedge \forall(x,y)(\neg Supply(x,y,z) \vee x = C)\}$$

we obtain only I_1, eliminating I_2. I_1 is the only consistent answer.

Example 13. (example 8 continued) In the standard format, the ICs take the form

$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee \neg Student(x,u,v) \vee y = u),$$
$$\forall(x,y,z,u,v)(\neg Student(x,y,z) \vee \neg Student(x,u,v) \vee z = v).$$

The following rule is generated

$$Student(x,y,z) \longmapsto Student(x,y,z) \{\forall(u,v)(\neg Student(x,u,v) \vee y = u), \forall(u,v)(\neg Student(x,u,v) \vee z = v)\}.$$

Given the inconsistent database instance r_3, if we pose the query $\exists z Course(S_1,y,z)$, asking for the names of the courses of the student with number S_1, we obtain C_1 and C_2. If we pose the query

$$\mathrm{T}_\omega(\exists z Course(S_1,y,z)) = \{\exists z Course(S_1,y,z)\}$$

we obviously obtain the same answers which, in this case, are the consistent answers. Intuitively, in this case the T_ω operator helps us to establish that even when the name of the student with number S_1 is undetermined, it is still possible to obtain the list of courses in which he/she is registered. On the other hand, if we pose the query

$$\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y))$$

about the courses and grades for a student with name N_1, to r_3, we obtain (C_1,G_1) and (C_2,G_2). Nevertheless, if we ask

$$\mathrm{T}_\omega(\exists(u,v)(Student(u,N_1,v) \wedge Course(u,x,y)))$$

we obtain, in conjunction with the original query, the formula:

$$\exists(u,v)(Student(u,N_1,v) \wedge$$
$$\forall(y',z')(\neg Student(u,y',z') \vee y' = N_1) \wedge$$
$$\forall(y',z')(\neg Student(u,y',z') \vee z' = v) \wedge Course(u,x,y)),$$

from this we obtain the empty set of tuples. This answer is intuitively consistent, because the number of the student with name N_1 is uncertain, and in consequence it is not possible to find out in which courses he/she is registered. The set of answers obtained with the T_ω operator coincides with the set of consistent answers which is empty.

5 Completeness

5.1 Binary ICs

Definition 11. A *binary integrity constraint* (BIC) is a sentence of the form

$$\forall(l_1(\bar{x}_1) \vee l_2(\bar{x}_2) \vee \psi(\bar{x})),$$

where l_1 and l_2 are literals, and ψ is a formula that only contains built-in predicates.

Examples of BICs include: functional dependencies, symmetry constraints, set inclusions dependencies of the form $\forall\bar{x}(P(\bar{x}) \supset Q(\bar{x}))$.

Definition 12. Given a set of sentences Σ in the language of the database schema DB, and a sentence φ, we denote by $\Sigma \vDash_{DB} \varphi$ the fact that, for every instance r of the database, if $r \vDash \Sigma$, then $r \vDash \varphi$.

Theorem 2. (*Completeness for BICs*) Given a set *IC* of binary integrity constraints, if for every literal $l'(\bar{a})$, $IC \nvDash_{DB} l'(\bar{a})$, then the operator T_ω is complete, that is, for every ground literal $l(\bar{t})$, if $r \vDash_c l(\bar{t})$ then $r \vDash \mathrm{T}_\omega(l(\bar{t}))$.

The theorem says that every consistent answer to a query of the form $l(\bar{x})$ is captured by the T_ω operator. Actually, proposition 2 in the appendix and the completeness theorem can be easily extended to the case of queries that are conjunctions of literals. Notice that the finiteness $\mathrm{T}_\omega(l(\bar{x}))$ is not a part of the hypothesis in this theorem. The hypothesis of the theorem requires that the ICs are not enough to answer a literal query by themselves; they do not contain definite knowledge about the literals.

Example 14. We can see in the example 12 where BICs and queries which are conjunctions of literals appear, that the operator T_ω gave us all the consistent answers, as implied by the theorem.

Corollary 2. If *IC* is a set of functional dependencies (FDs)

$$IC = \{\forall(\neg P_1(\bar{x}_1,y_1) \vee \neg P_1(\bar{x}_1,z_1) \vee y_1 = z_1), \quad (5)$$
$$\ldots,$$
$$\forall(\neg P_n(\bar{x}_n,y_n) \vee \neg P_n(\bar{x}_n,z_n) \vee y_n = z_n)\},$$

then the operator T_ω is complete for consistent answers to queries that are conjunctions of literals.

Example 15. In example 13 we had FDs that are also BICs. Thus the operator T_ω found all the consistent answers, even for some queries that are not conjunctions of literals, showing that this is not a necessary condition.

Example 16. Here we will show that in general completeness is not obtained for queries that are not conjunctions of literals. Consider the IC: $\forall x,y,z(P(x,y) \wedge P(x,z) \supset y = z)$ and the inconsistent instance r with $\Sigma(r) = \{P(a,b),P(a,c)\}$. This database has two repairs: r' with $\Sigma(r') = \{P(a,b)\}$; and r'' with $\Sigma(r'') = \{P(a,c)\}$. We have that $r \models_c \exists x P(a,x)$, because the query is true in the two repairs.

Now, it is easy to see that $\mathrm{T}_\omega(\exists u P(a,u))$ is logically equivalent to $\exists u(P(a,u) \wedge \forall z(\neg P(a,z) \vee z = u))$. So, we have $r \not\models \mathrm{T}_\omega(\exists x P(a,x))$. Thus, the consistent answer *true* is not captured by the operator T_ω.

5.2 Other Constraints

The following theorem applies to arbitrary ICs and generalizes Theorem 2.

Theorem 3. (*Completeness*) Let *IC* be a set of integrity constraints, $l(\bar{x})$ a literal, and $\mathrm{T}_n(l(\bar{x}))$ of the form

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i,\bar{y}_i)(C_i(\bar{x},\bar{x}_i) \vee \psi_i(\bar{x},\bar{y}_i)).$$

If for every $n \geq 0$, there is $S \subseteq \{1,\ldots,m\}$ such that

1. for every $j \in S$ and every tuple $\bar{a}$: $IC \not\models_{DB} C_j(\bar{a})$, and
2. $\{\forall(\bar{x}_i,\bar{y}_i)(C_i(\bar{x},\bar{x}_i) \vee \psi_i(\bar{x},\bar{y}_i))|i \in S\}$ implies
$$\{\forall(\bar{x}_i,\bar{y}_i)(C_i(\bar{x},\bar{x}_i) \vee \psi_i(\bar{x},\bar{y}_i))|1 \leq i \leq m\}$$

then $r \vDash_c l(\bar{t})$ implies $r \vDash \mathrm{T}_\omega(l(\bar{t}))$.

This theorem can be extended to conjunctions of literals. Notice that the theorem requires a condition for every $n \in \mathbb{N}$. Its application is obviously simplified if we know that the iteration terminates. This is an issue to be analyzed in the next section.

6 Termination

Termination means that the operator T_ω returns a finite set of formulas. It is clearly important because then the set of consistent answers can be computed by evaluating a single, finite query. We distinguish between three different notions of termination.

Definition 13. Given a set of *IC*s and a query $Q(\bar{x})$, we say that $\mathrm{T}_\omega(Q(\bar{x}))$ is

1. *syntactically finite* if there is an an n such that $\mathrm{T}_n(Q(\bar{x}))$ and $\mathrm{T}_{n+1}(Q(\bar{x}))$ are syntactically the same.
2. *semantically finite* if there is an n such that for all $m \geq n$, $\forall\bar{x}(\mathrm{T}_n(Q(\bar{x}) \equiv \mathrm{T}_m(Q(\bar{x}))$ is valid.
3. *semantically finite in an instance* r, if there is an n such that for all $m \geq n$, $r \models \forall\bar{x}\ (\mathrm{T}_n(Q(\bar{x}) \equiv \mathrm{T}_m(Q(\bar{x}))$.

The number n in cases 2 and 3 is called a *point of finiteness*. It is clear that 1 implies 2 and 2 implies 3. In the full version we will show that all these implications are proper. In all these cases, evaluating $\mathrm{T}_\omega(Q(\bar{x})$ gives the same result as evaluating $\mathrm{T}_n(Q(\bar{x})$ for some n (in the instance r in case 3). If $\mathrm{T}_\omega(Q(\bar{x})$ is semantically finite, sound and complete, then the set of consistent answers to Q is *first-order definable*.

6.1 Syntactical finiteness

The notion of syntactical finiteness is important because then for some n and all $m > n$, $\mathrm{T}_m(Q(\bar{x}))$ will be exactly the same. In consequence, $\mathrm{T}_\omega(Q)$ will be a finite set of formulas. In addition, a point of finiteness n can be detected (if it exists) by syntactically comparing every two consecutive steps in the iteration. No simplification rules need to be considered, because the iterative procedure is fully deterministic.

Here we introduce a necessary and sufficient condition for syntactical finiteness.

Definition 14. A set of integrity constraints *IC* is *acyclic* if there exists a function f from predicate names plus negations of predicate names in the database to the natural numbers, that is, $f:\{p_1,\ldots,p_n,\neg p_1,\ldots,\neg p_n\} \longrightarrow \mathbb{N}$, such that for every integrity constraint $\forall(\bigvee_{i=1}^{k} l_i(\bar{x}_i) \vee \psi(\bar{x})) \in IC$ as in (3), and every i and j $(1 \leq i,j \leq k)$, if $i \neq j$, then $f(\neg l_i) > f(l_j)$. (Here $\neg l_i$ is the literal complementary to l_i.)

Example 17. The set of ICs

$$IC = \{\forall x(\neg P(x) \vee \neg Q(x) \vee S(x)),\\ \forall(x,y)(\neg Q(x) \vee \neg S(y) \vee T(x,y))\}.$$

is acyclic, because the function f defined by

$$f(P)=2 \quad f(Q)=2 \quad f(\neg P)=0 \quad f(\neg Q)=0$$
$$f(S)=1 \quad f(T)=0 \quad f(\neg S)=1 \quad f(\neg T)=2,$$

satisfies the condition of definition 14.

Example 18. The set of ICs

$$IC = \{\forall x(\neg P(x) \vee \neg Q(x) \vee S(x)),\\ \forall(x,y)(Q(x) \vee \neg S(y) \vee T(x,y))\}.$$

is not acyclic, because for any function f that we may attempt to use to satisfy the condition in definition 14, from the first integrity constraint we obtain $f(Q) > f(S)$, and from the second, we would obtain $f(S) > f(Q)$; a contradiction.

Theorem 4. A set of integrity constraints *IC* is acyclic iff for every literal name l in the database schema, $\mathrm{T}_\omega(l(\bar{x}))$ is syntactically finite.

The theorem can be extended to any class of queries satisfying Definition 5.

Example 19. The set of integrity constraints in example 18 is not acyclic. In that case $\mathrm{T}_\omega(Q(x))$ is infinite.

Example 20. The ICs in example 17 are acyclic. There we have

$$
\begin{aligned}
&\mathrm{T}_{\omega}(P(u)) = \\
&\quad \{P(u), \\
&\quad\ P(u) \wedge (\neg Q(u) \vee S(u)), \\
&\quad\ P(u) \wedge (\neg Q(u) \vee S(u) \wedge \forall v(\neg Q(v) \vee T(v,u)))\}
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{T}_{\omega}(Q(u)) = \\
&\quad \{Q(u), \\
&\quad\ Q(u) \wedge (\neg P(u) \vee S(u)) \wedge \forall v(\neg S(v) \vee T(u,v)), \\
&\quad\ Q(u) \wedge (\neg P(u) \vee S(u) \wedge \forall w(\neg Q(w) \vee T(w,u))) \wedge \\
&\qquad\qquad \forall v(\neg S(v) \wedge (\neg P(v) \vee \neg Q(v)) \vee T(u,v))\}
\end{aligned}
$$

$$\mathrm{T}_{\omega}(S(u)) = \{S(u), S(u) \wedge \forall(\neg Q(v) \vee T(v,u))\}$$

$$\mathrm{T}_{\omega}(T(u,v)) = \{T(u,v)\}$$

$$\mathrm{T}_{\omega}(\neg P(u)) = \{\neg P(u)\}$$

$$\mathrm{T}_{\omega}(\neg Q(u)) = \{\neg Q(u)\}$$

$$\mathrm{T}_{\omega}(\neg S(u)) = \{\neg S(u), \neg S(u) \wedge (\neg P(u) \vee \neg Q(u))\}$$

$$
\begin{aligned}
&\mathrm{T}_{\omega}(\neg T(u,v)) = \\
&\quad \{\neg T(u,v), \\
&\quad\ \neg T(u,v) \wedge (\neg Q(u) \vee \neg S(v)), \\
&\quad\ \neg T(u,v) \wedge (\neg Q(u) \vee \neg S(v) \wedge (\neg P(v) \vee \neg Q(v)))\}.
\end{aligned}
$$

Corollary 3. For functional dependencies and a query $Q(\bar{x})$, $\mathrm{T}_{\omega}(Q(\bar{x}))$ is always syntactically finite.

6.2 Semantical finiteness

Definition 15. A constraint C in clausal form is *uniform* if for every literal $l(\bar{x})$ in it, the set of variables in $l(\bar{x})$ is the same as the set of variables in $C - l(\bar{x})$. A set of constraints is uniform if all the constraints in it are uniform.

Examples of uniform constraints include set inclusion dependencies of the form $\forall\bar{x}(P(\bar{x}) \supset Q(\bar{x}))$, e.g., Example 4.

Theorem 5. If a set of integrity constraints *IC* is uniform, then for every literal name l in the database schema, $\mathrm{T}_{\omega}(l(\bar{x}))$ is semantically finite. Furthermore, a point of finiteness n can be bounded from above by a function of the number of variables in the query, and the number of predicates (and their arities) in the query and *IC*.

Theorem 6. Let l be a literal name. If for some n,

$$\forall\bar{x}(\mathrm{T}_n(l(\bar{x})) \supset T_{n+1}(l(\bar{x})))$$

is valid, then for all $m \geq n$,

$$\forall\bar{x}(\mathrm{T}_n(l(\bar{x})) \equiv \mathrm{T}_m(l(\bar{x})))$$

is valid.

According to Theorem 6, we can detect a point of finiteness by comparing every two consecutive steps wrt logical implication. Although this is undecidable in general, we might try to apply semidecision procedures, for example, automated theorem proving. We have successfully made use of OTTER [17] in some cases that involve sets of constraints that are neither acyclic nor uniform. Examples include multivalued dependencies, and functional dependencies together with set inclusion dependencies. For multivalued dependencies, Theorem 6 together with Theorem 3 gives completeness of $\mathrm{T}_{\omega}(l(\bar{x}))$ where $l(\bar{x})$ is a negative literal. The criterion from Theorem 6 is also applicable to uniform constraints by providing potentially faster termination detection than the proof of Theorem 5.

6.3 Instance based semantical finiteness

Theorem 7. If $Q(\bar{x})$ is a domain independent query, then for every database instance r there is an n, such that for all $m \geq n$, $r \models \forall\bar{x}(T_n(Q(\bar{x})) \equiv T_m(Q(\bar{x})))$.

Notice that this theorem does not include the case of negative literals, as in the case of theorem 5.

7 Related work

Bry [4] was, to our knowledge, the first author to consider the notion of consistent query answer in inconsistent databases. He defined consistent query answers based on provability in minimal logic, without giving, however, a proof procedure or any other computational mechanism for obtaining such answers. He didn't address the issues of of semantics, soundness or completeness.

It has been widely recognized that in database integration the integrated data may be inconsistent with the integrity constraints. A typical (theoretical) solution is to augment the data model to represent disjunctive information. The following example explains the need for a solution of this kind.

Example 21. Consider the functional dependency

$$\forall(x,y,z)(P(x,y) \wedge P(x,z) \supset y = z.$$

If the integrated database contains both $P(a,b)$ and $P(a,c)$, then the functional dependency is violated. Each of $P(a,b)$ and $P(a,c)$ may be coming from a different database that satisfies the dependency. Thus, both facts are replaced by their disjunction $P(a,b) \vee P(a,c)$ in the integrated database. Now the functional dependency is no longer violated.

To solve this kind of problems [1] introduced the notion of *flexible relation*, a non-1NF relation that contains tuples with sets of non-key values (with such a set standing for *one* of its elements). This approach is limited to primary key functional dependencies and was subsequently generalized to other key functional dependencies [9]. In the same context, [3, 12] proposed to use disjunctive Datalog and [16] tables with OR-objects. [1] introduced flexible relational algebra to query flexible relations, and [9] - flexible relational calculus (whose subset can be translated to flexible relational algebra). The remaining papers did not discuss query language issues, relying on the existing approaches to query disjunctive Datalog or tables with OR-objects. There are several important differences between the above approaches and ours. First, they rely on the construction of a single (disjunctive) instance and the deletion of conflicting tuples. In our approach, the underlying databases are incorporated into the integrated one *in toto*, without any changes. There is no need for introducing disjunctive information. It would be interesting to compare the scope and the computational requirements of both approaches. For instance, one should note that the single-instance approach is not incremental: Any changes in the underlying databases require the recomputation of the entire instance. Second, our approach seems to be unique, in the context of database integration, in considering tuple insertions as possible repairs for integrity violations. Therefore, in some cases consistent query answers may be different from query answers obtained from the corresponding single instance.

Example 22. Consider the integrity constraint $p \supset q$ and a fact p. The instance consisting of p alone does not satisfy the integrity constraint. The common solution for removing this violation is to delete p. However, in our approach inserting q is also a possible repair. This has consequences for the inferences about $\neg p$ and $\neg q$. Our approach returns *false* in both cases, as p (resp. q) is true in a possible repair. Other approaches return *true* (under CWA) or *undefined* (under OWA).

Our work has connections with research done on belief revision [10]. In our case, we have an implicit notion of revision that is determined by the set of repairs of the database, and corresponds to revising the database (or a suitable categorical theory describing it) by the set of integrity constraints. Thus, querying the inconsistent database expecting only correct answers corresponds to querying the revised theory without restrictions.

It is easy to see that our notion of repair of a relational database is a particular case of the local semantics introduced in [8], restricted to revision performed starting from a single model (the database). From this we obtain that our revision operator satisfies the postulates (R1) – (R5),(R7), (R8) in [13]. For each given database r, the relation $\leq_r$ introduced in definition 3 provides the partial order between models that determines the (models of the) revised database as described in [13]. [8] concentrates on the computation of the models of the revised theory, i.e. the repairs in our case, whereas we do not compute the repairs, but keep querying the original, non-revised database and pose a modified query. Therefore, we can view our methodology as a way of representing and querying simultaneously all the repairs of the database by means of a new query. Nevertheless, our motivation and starting point is quite different from belief revision. We attempt to take direct advantage of the semantic information contained in the integrity constraints in order to answer queries, rather than revising the database. Revising the database means repairing all the inconsistencies in it, instead we are interested in the information related to particular queries. For instance, a query referring only to the consistent portion of the database can be answered without repairing the database.

Reasoning in the presence of inconsistency has been an important research problem in the area of knowledge representation. The goal is to design logical formalisms that limit what can be inferred from an inconsistent set of formulas. One does not want to infer all formulas (as required by the classical two-valued logic). Also, one prefers not to infer a formula together with its negation. The formalisms satisfying the above properties, e.g., [15], are usually propositional. Moreover, they do not distinguish between integrity constraints and database facts. Thus, if the data in the database violates an integrity constraint, the constraint itself can no longer be inferred (which is not acceptable in the database context).

Example 23. Assume the integrity constraint is $\neg(p \wedge q)$ and the database contains the facts p and q. In the approach of [15], $p \vee q$ can be inferred (minimal change is captured correctly) but p, q and $\neg(p \wedge q)$ can no longer be inferred (they are all involved in an inconsistency).
Because of the above-mentioned limitations, such methods are not directly applicable to the problem of computing consistent query answers.

Deontic logic [18, 14], a modal logic with operators capturing permission and obligation, has been used for the specification of integrity constraints. [14] used the obligation operator O to distinguish integrity constraints that *have to hold always* from database facts that just *happen to hold*. [18] used deontic operators to describe policies whose violations can then be caught and handled. The issues of possible repairs of constraint violations, their minimality and consistent query answers are not addressed.

Gertz [11] described techniques and algorithms for computing repairs of constraint violations. The issue of query answering in the presence of an inconsistency is not addressed in his work.

8 Conclusions and Further Work

This paper represents a first step in the development of a new research area dealing with the theory and applications of consistent query answers in arbitrary, consistent or inconsistent, databases.

The theoretical results presented here are preliminary. We have proved a general soundness result but the results about completeness and termination are still partial. Also, one needs to look beyond purely universal constraints to include general inclusion dependencies. In a forthcoming paper we will also describe our methodology for using automated theorem proving, in particular, OTTER, for proving termination.

It appears that in order to obtain completeness for disjunctive and existentially quantified queries one needs to move beyond the T_ω operator on queries. Also, the upper bounds on the size of T_ω and the lower bounds on the complexity of computing consistent answers for different classes of queries and constraints need to be studied.

There is an interesting connection to modal logic. Consider the definition 7. We could write $r \models \Box Q(\bar{t})$, meaning that $Q(\bar{t})$ is true in all repairs of r, the database instances that are "accessible" from r. This is even more evident from example 16, where, in essence, it is shown that $\Box\exists x Q(\bar{x})$ is not logically equivalent to $\exists x \Box Q(\bar{x})$, which is what usually happens in modal logic.

Acknowledgments

This research has been partially supported by FONDECYT Grants (1971304 & 1980945) and NSF Grant (IRI-9632870). Part of this research was done when the second author was on sabbatical at the Technical University of Berlin (CIS Group) with the financial support from DAAD and DIPUC.

References

[1] S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *IEEE International Conference on Data Engineering*, 1995.

[2] M. Arenas, L. Bertossi, and M. Kifer. APC and Querying Inconsistent Databases. In preparation.

[3] C. Baral, S. Kraus, J. Minker, and V.S. Subrahmanian. Combining Knowledge Bases Consisting of First-Order Theories. *Computational Intelligence*, 8:45–71, 1992.

[4] F. Bry. Query Answering in Information Systems with Integrity Constraints. In *IFIP WG 11.5 Working Conference on Integrity and Control in Information Systems*. Chapman &Hall, 1997.

[5] U.S. Chakravarthy, J. Grant, and J. Minker. Logic-Based Approach to Semantic Query Optimization. *ACM Transactions on Database Systems*, 15(2):162–207, 1990.

[6] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26, March 1997.

[7] J. Chomicki and G. Saake, editors. *Logics for Databases and Information Systems*. Kluwer Academic Publishers, Boston, 1998.

[8] T. Chou and M. Winslett. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12:157–208, 1994.

[9] Phan Minh Dung. Integrating Data from Possibly Inconsistent Databases. In *International Conference on Cooperative Information Systems*, Brussels, Belgium, 1996.

[10] P. Gaerdenfors and H. Rott. Belief Revision. In D. M. Gabbay, J. Hogger, C, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 4, pages 35–132. Oxford University Press, 1995.

[11] M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*. PhD thesis, Universität Hannover, 1996.

[12] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity Constraints: Semantics and Applications. In Chomicki and Saake [7], chapter 9.

[13] H. Katsuno and A. Mendelzon. Propositional Knowledge Base Revision and Minimal Change. *Artificial Intelligence*, 52:263–294, 1991.

[14] K.L. Kwast. A Deontic Approach to Database Integrity. *Annals of Mathematics and Artificial Intelligence*, 9:205–238, 1993.

[15] J. Lin. A Semantics for Reasoning Consistently in the Presence of Inconsistency. *Artificial Intelligence*, 86(1-2):75–95, 1996.

[16] J. Lin and A. O. Mendelzon. Merging Databases under Constraints. *International Journal of Cooperative Information Systems*, 7(1):55–76, 1996.

[17] W.W. McCune. *OTTER 3.0 Reference Manual and Guide*. Argonne National Laboratory, Technical Report ANL-94/6, 1994.

[18] J.-J. Meyer, R. Wieringa, and F. Dignum. The Role of Deontic Logic in the Specification of Information Systems. In Chomicki and Saake [7], chapter 4.

[19] Jean-Marie Nicolas. Logic for Improving Integrity Checking in Relational Data Bases. *Acta Informatica*, 18:227–253, 1982.

[20] J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. I.* Computer Science Press, 1988.

Appendix: Proofs of Results

Some technical lemmas are stated without proof. Full proofs can be found in the file `proofspods99.ps` in `http://dcc.ing.puc.cl/~bertossi/`.

Lemma 1. If $r \vDash \mathrm{T}_\omega(l(\bar{a}))$, where $l(\bar{a})$ is a ground literal, then for every repair r' of r, it holds $r' \vDash l(\bar{a})$.

Lemma 2. If $r \vDash \mathrm{T}_\omega(\bigwedge_{i=1}^{n} l_i(\bar{a}_i))$, where $l_i(\bar{a}_i)$ is a ground literal, then for every repair r' of r, it holds $r' \vDash \bigwedge_{i=1}^{n} l_i(\bar{a}_i)$.

Lemma 3. If $r \vDash \mathrm{T}_\omega(\bigvee_{i=1}^{n} C_i(\bar{a}_i))$, with $C_i(\bar{a}_i)$ a conjunction of literals, then for every repair r' of r, $r' \vDash \bigvee_{i=1}^{n} C_i(\bar{a}_i)$.

Lemma 4. Let $Q(\bar{x})$ a universal query. If $r \vDash \mathrm{T}_\omega(Q(\bar{t}))$, for a ground tuple $\bar{t}$, then for every repair r' of r, $r' \vDash Q(\bar{t})$.

Lemma 5. Let $Q(\bar{x})$ a domain independent query. If $r \vDash \mathrm{T}_\omega(Q(\bar{t}))$, for a ground tuple $\bar{t}$, then for every repair r' of r, $r' \vDash Q(\bar{t})$.

Proof of Theorem 1: Lemmas 4 and 5.

Proposition 2. Given a set IC of integrity constraints, a ground clause $\bigvee_{i=1}^{m} l_i(\bar{t}_i)$, if $IC \nvDash_{DB} \bigvee_{i=1}^{m} l_i(\bar{t}_i)$ and, for every repair r' of r, $r' \vDash \bigvee_{i=1}^{m} l_i(\bar{t}_i)$, then $r \vDash \bigvee_{i=1}^{m} l_i(\bar{t}_i)$.

Proof of Proposition 2: Assume that $r \vDash \neg \bigvee_{i=1}^{m} l_i(\bar{t}_i)$. By hypothesis $IC \nvDash_{DB} \bigvee_{i=1}^{m} l_i(\bar{t}_i)$, thus there exists an instance of the database r' such that $r' \vDash IC \cup \{\neg \bigvee_{i=1}^{m} l_i(\bar{t}_i)\}$. Let us consider the set of database instances

$$R = \{r^* | r^* \vDash IC \text{ and } \Delta(r, r^*) \subseteq \Delta(r, r')\}.$$

We know that $\Delta(r, r')$ is finite, therefore there exists $r_0 \in R$ such that $\Delta(r, r_0)$ is minimal. Then, r_0 is a repair of r.

For every $1 \leq i \leq m$, if $l_i(\bar{t}_i)$ is $p(\bar{t})$ or $\neg p(\bar{t})$, then $p(\bar{t}) \notin \Delta(r, r')$. Using this fact we conclude that $p(\bar{t}) \notin \Delta(r, r_0)$, Therefore, $r \vDash \bigvee_{i=1}^{m} l_i(\bar{t}_i)$ if and only if $r_0 \vDash \bigvee_{i=1}^{m} l_i(\bar{t}_i)$. But we assumed that $r \vDash \neg \bigvee_{i=1}^{m} l_i(\bar{t}_i)$, then $r_0 \vDash \neg \bigvee_{i=1}^{m} l_i(\bar{t}_i)$; a contradiction.

Proof of Theorem 2: From theorem 3.

Proof of Corollary 2: In this case it holds:

1. For every tuple $\bar{a}$, $IC \nvDash_{DB} P_i(\bar{a})$, because the empty database instance (which has only empty base relations) satisfies IC, but not $P(\bar{a})$.

2. For every tuple $\bar{a}$, $IC \nvDash_{DB} \neg P_i(\bar{a})$, since the database instance $r^i_{\bar{a}}$, where the relation P_i contains only the tuple $\bar{a}$ and the other relations are empty, satisfies IC, but not $\neg P_i(\bar{a})$.

Proof of Theorem 3: Suppose that $r \vDash_c l(\bar{t})$. Let r' a repair of r, we have that $r' \vDash l(\bar{t})$. By proposition 1 we have that $r' \vDash T_n(l(\bar{t}))$, that is

$$r' \vDash l(\bar{t}) \wedge \bigwedge_{i=1}^{m} \forall(\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_{i,j}) \vee \psi_i(\bar{t}, \bar{x}_i)), \tag{6}$$

We want to prove that for every i and for every sequence of ground tuples $a_i, a_{i,1}, \ldots, a_{i,m_i}$:

$$r \vDash \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_{i,j}) \vee \psi_i(\bar{t}, \bar{a}_i), \tag{7}$$

To do this, first we are going to prove that for every $i \in S$ and for every sequence of ground tuples $a_i, a_{i,1}, \ldots, a_{i,m_i}$:

$$r \vDash \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_{i,j}) \vee \psi_i(\bar{t}, \bar{a}_i), \tag{8}$$

This is immediately obtained when $r \vDash \psi_i(\bar{t}, \bar{a}_i)$. Assume that $r \vDash \neg\psi_i(\bar{t}, \bar{a}_i)$. We know that ψ_i only mentions built-in predicates, thus for every repair r' of r we have that $r' \vDash \neg\psi_i(\bar{t}, \bar{a}_i)$. Therefore, by (6) we conclude that for every repair r' of r:

$$r' \vDash \bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{a}_{i,j}) \vee \psi_i(\bar{t}, \bar{a}_i),$$

By proposition 2 we conclude (8). Thus we have that

$$r \vDash l(\bar{t}) \wedge \bigwedge_{i \in S} \forall(\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_{i,j}) \vee \psi_i(\bar{t}, \bar{x}_i)),$$

but by the second condition in the hypothesis of the theorem we conclude that:

$$r \vDash l(\bar{t}) \wedge \bigwedge_{i=1}^{m} \forall(\bigvee_{j=1}^{m_i} l_{i,j}(\bar{t}, \bar{x}_{i,j}) \vee \psi_i(\bar{t}, \bar{x}_i)).$$

Proof of Theorem 4: ($\Longrightarrow$) Suppose that IC is acyclic, then there exists f as in the definition 14. We are going to prove by induction on k that for every literal name l, if $f(l) = k$, then $T_{k+1}(l(\bar{x})) = T_{k+2}(l(\bar{x}))$

(I) If $k = 0$. We know that that for every literal name l', $f(l') \geq 0$. Therefore, every integrity constraint containing $\neg l$ is of the form $\forall(\neg l(\bar{x}) \vee \psi(\bar{y}))$, where ψ only mentions built-in predicates. This is because if there were any other literal l' in the integrity constraint, we would have $f(l') < f(l) = 0$. Then $T_1(l(\bar{x})) = T_2(l(\bar{x}))$.

(II) Suppose that the property is true for every $m < k$. We know that $T_{k+2}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \bar{Q}_i(\bigvee_{j=1}^{m_i} T_{k+1}(l_{i,j}(\bar{x}_{i,j})) \vee \psi_i(\bar{x}_i)),$$

where $\bar{Q}_i$ is a sequence of quantifiers over all the variables $\bar{x}_{i,1}, \ldots, \bar{x}_{i,m_i}, \bar{x}_i$ not appearing in $\bar{x}$, and $T_{k+1}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \bar{Q}_i(\bigvee_{j=1}^{m_i} T_k(l_{i,j}(\bar{x}_{i,j})) \vee \psi_i(\bar{x}_i)).$$

By definition of f, we know that for every literal name $l_{i,j}$ in the previous formulas, $f(l_{i,j}) < k$. Then by induction hypothesis $T_k(l(\bar{x}_{i,j})) = T_{k+1}(l_{i,j}(\bar{x}_{i,j}))$ (since if $T_m(l'(\bar{x})) = T_{m+1}(l'(\bar{x}))$, then for every $n \geq m$, $T_n(l'(\bar{x})) = T_{n+1}(l'(\bar{x}))$).
($\Longleftarrow$) Suppose that for every literal name l, $\mathrm{T}_\omega(l(\bar{x}))$ is finite. The for every literal name l there exists a first natural number k such that $\mathrm{T}_k(l(\bar{x})) = \mathrm{T}_{k+1}(l(\bar{x}))$. Let us define a function f, from the literal names into the natural number, by $f(l) = k$ (k as before). We can show that this is a well defined function that behaves as in definition 14: since if $\forall(\bigvee_{i=1}^{m} l_i(\bar{x}_i) \vee \psi(\bar{y})) \in IC$, then for every $1 \leq s \leq m$, $\mathrm{T}_{f(\neg l_s)}(\neg l_s(\bar{x}_s))$ is of the form

$$\neg l_s(\bar{x}_s) \wedge \bar{Q}(\bigvee_{i=1}^{s-1} \mathrm{T}_{f(\neg l_s)-1}(l_i(\bar{x}_i)) \vee \bigvee_{i=s+1}^{m} \mathrm{T}_{f(\neg l_s)-1}(l_i(\bar{x}_i)) \vee \psi(\bar{y})) \wedge \theta(\bar{x}_s), \quad (9)$$

where $\bar{Q}$ is a sequence of quantifiers over all the variables $\bar{x}_1, \ldots, \bar{x}_m, \bar{y}$, not appearing in $\bar{x}_s$, and $\mathrm{T}_{f(\neg l_s)+1}(\neg l_s(\bar{x}_s))$ is of the form

$$\neg l_s(\bar{x}_s) \wedge \bar{Q}(\bigvee_{i=1}^{s-1} \mathrm{T}_{f(\neg l_s)}(l_i(\bar{x}_i)) \vee \bigvee_{i=s+1}^{m} \mathrm{T}_{f(\neg l_s)}(l_i(\bar{x}_i)) \vee \psi(\bar{y})) \wedge \theta'(\bar{x}_s). \quad (10)$$

By definition of f, $T_{f(\neg l_s)}(\neg l_s(\bar{x}_s)) = T_{f(\neg l_s)+1}(\neg l_s(\bar{x}_s))$. Then, by the form of (9) and (10), we conclude that for every $i \neq s$, $T_{f(\neg l_s)-1}(l_i(\bar{x}_i)) = T_{\neg f(l_s)}(l_i(\bar{x}_i))$, and then, again by definition of f, $f(l_i) < f(\neg l_s)$.

Proof of Corollary 3: The following stratification function from literals to $\mathbb{N}$ can be defined: $f(\neg P_i) = 0$ and $f(P_j) = 1$, where P_i, P_j are relation names.

Proof of Theorem 5: For uniform constraints the residues do not contain quantifiers. Therefore $\mathrm{T}_n(l(\bar{x}))$ for every $n \geq 0$ is quantifier-free and contains only the variables that occur in $\bar{x}$. There are only finitely many inequivalent formulas with this property, and thus $\mathrm{T}_\omega(l(\bar{x}))$ is finite.

Lemma 6. If $\mathrm{T}_n(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i, \bar{y}_i)(C_i(\bar{x}, \bar{x}_i) \vee \psi_i(\bar{x}, \bar{y}_i)),$$

then $\mathrm{T}_{n+1}(l(\bar{x}))$ is of the form:

$$l(\bar{x}) \wedge \bigwedge_{i=1}^{m} \forall(\bar{x}_i, \bar{y}_i)(\mathrm{T}_1(C_i(\bar{x}, \bar{x}_i)) \vee \psi_i(\bar{x}, \bar{y}_i)),$$

Lemma 7. If for a ground tuple $\bar{a}$, $\mathrm{T}_n(l(\bar{a})) \vDash \forall(\bigvee_{j=1}^{k} l'_j(\bar{a}, \bar{z}_j))$, then $\mathrm{T}_{n+1}(l(\bar{a})) \vDash \forall(\bigvee_{j=1}^{k} \mathrm{T}_1(l'_j(\bar{a}, \bar{z}_j)))$.

Proof of Theorem 6: Suppose that for a natural number n, $\forall\bar{x}(\mathrm{T}_n(l(\bar{x})) \supset \mathrm{T}_{n+1}(l(\bar{x})))$ is a valid sentence. We are going to prove that for every $m \geq n$, $\forall\bar{x}(\mathrm{T}_m(l(\bar{x})) \supset \mathrm{T}_{m+1}(l(\bar{x})))$ is a valid sentence, by induction on m.
(I) If $m = n$, by hypothesis.
(II) Suppose that $\forall\bar{x}(\mathrm{T}_m(l(\bar{x})) \supset \mathrm{T}_{m+1}(l(\bar{x})))$ is a valid sentence. For every clause $\bigvee_{j=1}^{k} l'_j(\bar{x}, \bar{z}_j) \vee \psi(\bar{x}, \bar{z})$ in $\mathrm{T}_{m+1}(l(\bar{x}))$ and for every ground tuple $\bar{a}$ we have that

$$\mathrm{T}_m(l(\bar{a})) \vDash \forall(\bigvee_{j=1}^{k} l'_j(\bar{a}, \bar{z}_j) \vee \psi(\bar{a}, \bar{z})).$$

By lemma 7 and considering that ψ only mentions built-in predicates we have that $\mathrm{T}_{m+1}(l(\bar{a})) \vDash \forall(\bigvee_{j=1}^{k} \mathrm{T}_1(l'_j(\bar{a}, \bar{z}_j)) \vee \psi(\bar{a}, \bar{z}))$, and from this and lemma 6 we can conclude that $\forall\bar{x}(\mathrm{T}_{m+1}(l(\bar{x})) \supset \mathrm{T}_{m+2}(l(\bar{x})))$ is a valid sentence.

Proof of Theorem 7: Let $Q(\bar{x})$ be a domain independent query and r a database instance. Define $A_n = \{\bar{t} \mid r \vDash \mathrm{T}_n(Q(\bar{t}))\}$. We know that for every n: $A_{n+1} \subseteq A_n$, therefore $A = \{A_i \mid i < \omega\}$ is a family of subsets of A_0. But A_0 is finite because $Q(\bar{x})$ is a domain independent query. Thus, there exists a minimal element A_m in A. For this element, it holds that for every $k \geq m$: $A_m = A_k$, since $A_k \subseteq A_m$.