Applications of Annotated Predicate Calculus to Querying Inconsistent Databases

Marcelo Arenas, Leopoldo Bertossi, and Michael Kifer

[1] P. Universidad Catolica de Chile, Depto. Ciencia de Computacion
Casilla 306, Santiago 22, Chile
{marenas,bertossi}@ing.puc.cl
[2] Department of Computer Science, University at Stony Brook
Stony Brook, NY 11794, USA
kifer@cs.sunysb.edu

Abstract. We consider the problem of specifying and computing consistent answers to queries against databases that do not satisfy given integrity constraints. This is done by simultaneously embedding the database and the integrity constraints, which are mutually inconsistent in classical logic, into a theory in annotated predicate calculus — a logic that allows non trivial reasoning in the presence of inconsistency. In this way, several goals are achieved: (a) A logical specification of the class of all minimal "repairs" of the original database, and the ability to reason about them; (b) The ability to distinguish between consistent and inconsistent information in the database; and (c) The development of computational mechanisms for retrieving consistent query answers, *i.e.*, answers that are not affected by the violation of the integrity constraints.

1 Introduction

Databases that violate stated integrity constraints is an (unfortunate) fact of life for many corporations. They arise due to poor data entry control, due to merges of previously separate databases, due to the incorporation of legacy data, and so on. We call such databases "inconsistent."

Even though the information stored in such a database might be logically inconsistent (and, thus, strictly speaking, *any* tuple should be viewed as a correct query answer), this has not been a deterrent to the use of such databases in practice, because application programmers have been inventing ingenious techniques for salvaging "good" information. Of course, in such situations, what is good information and what is not is in the eyes of beholder, and each concrete case currently requires a custom solution. This situation can be compared to the times before the advent of relational databases, when every database query required a custom solution.

Thus, the problem is: what is the definition of "good information" in an inconsistent database and, once this is settled, what is the meaning of a query in this case. Several proposals to address these problems — both semantically and computationally — are known (*e.g.*, [1]), and we are not going to propose

J. Lloyd et al. (Eds.): CL 2000, LNAI 1861, pp. 926–941, 2000.

yet another definition for consistent query answers. Instead, we introduce a new *semantic framework*, based on Annotated Predicate Calculus [9], that leads to a different computational solution and provides a basis for a systematic study of the problem.

Ultimately, our framework leads to the query semantics proposed in [1]. According to [1], a tuple $\bar{t}$ is an answer to the query $Q(\bar{x})$ in a possibly inconsistent database instance r, if $Q(\bar{t})$ holds true in all the "repairs" of the original database, that is in all the databases that satisfy the given constraints and can be obtained from r by means of a "minimal" set of changes (where minimality is measured in terms of a smallest symmetric set difference).

In [1], an algorithm is proposed whereby the original query is modified using the set of integrity constraints (that are violated by the database). The modified query is then posed against the original database (with the integrity constraints ignored). In this way, the explicit integrity checking and computation of all database repairs is avoided.

In this paper, we take a more direct approach. First, since the database is inconsistent with the constraints, it seems natural to embed it into a logic that is better suited for dealing with inconsistency than classical logic. In this paper we use *Annotated Predicate Calculus* (abbr. APC) introduced in [9]. APC is a form of "paraconsistent logic," *i.e.*, logic where inconsistent information does not unravel logical inference and where causes of inconsistency can be reasoned about. APC generalizes a number of earlier proposals [12,11,3] and its various partial generalizations have also been studied in different contexts (*e.g.*, [10]).

The gist of our approach is to embed an inconsistent database theory in APC and then use APC to define database repairs and query answers. This helps understand the results of [1], leads to a more straightforward complexity analysis, and provides a more general algorithm that covers classes of queries not included in [1]. Furthermore, by varying the semi-lattice underlying the host APC theory, it is possible to control how exactly inconsistency is resolved in the original database.

Section 2 formalizes the problem of querying inconsistent databases. Section 3 reviews the basic definitions of Annotated Predicate Calculus, and Section 4 applies this calculus to our problem. In Section 5, we provide a syntactic characterization for database repairs and discuss the associated computational process. Section 6 studies the problem of query evaluation in inconsistent databases and Section 7 concludes the paper.

2 Preliminaries

We assume we have a fixed database schema $P = \{p_1, \ldots, p_n\}$, where p_1, ..., p_n are predicates corresponding to the database relations; a fixed, possibly infinite database domain $D = \{c_1, c_2, ...\}$; and a fixed set of built-in predicates $B = \{e_1, \ldots, e_m\}$. Each predicate has *arity*, *i.e.*, the number of arguments it takes. An integrity constraint is a closed first-order formula in the language defined by

the above components. We also assume a first order language $\mathcal{L} = D \cup P \cup B$ that is based on this schema.

Definition 1. *(Databases and Constraints) A* database instance **DB** *is a finite collection of facts, i.e., of statements of the form $p(c_1, ..., c_n)$, where p is a predicate in P and $c_1, ..., c_n$ are constants in D.*

An integrity constraint *is a clause of the form*

$$p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{S}_1) \vee \cdots \vee \neg q_m(\bar{S}_m)$$

where each p_i $(1 \leq i \leq n)$ and q_j $(q \leq j \leq m)$ is a predicate in $P \cup B$ and $\bar{T}_1, ..., \bar{T}_n, \bar{S}_1, ..., \bar{S}_m$ are tuples (of appropriate arities) of constants or variables. As usual, we assume that all variables in a clause are universally quantified, so the quantifiers are omitted.

Throughout this paper we assume that both the database instance **DB** and the set of integrity constraints **IC** are consistent when considered in isolation. However, together $\mathbf{DB} \cup \mathbf{IC}$ might not be consistent.

Definition 2. *(Sentence Satisfaction) We use $\models_{DB}$ to denote the usual notion of formula satisfaction in a database. The subscript DB is used to distinguish this relation from other types of implication used in this paper. In other words,*

- $\mathbf{DB} \models_{DB} p(\bar{c})$*, where $p \in P$, iff $p(\bar{c}) \in \mathbf{DB}$;*
- $\mathbf{DB} \models_{DB} q(\bar{c})$*, where $q \in B$, iff $q(\bar{c})$ is true;*
- $\mathbf{DB} \models_{DB} \neg\varphi$ *iff it is not true that* $\mathbf{DB} \models_{DB} \varphi$*;*
- $\mathbf{DB} \models_{DB} \phi \wedge \psi$ *iff* $\mathbf{DB} \models_{DB} \phi$ *and* $\mathbf{DB} \models_{DB} \psi$*;*
- $\mathbf{DB} \models_{DB} (\forall X)\phi(X)$ *iff for all $d \in D$,* $\mathbf{DB} \models_{DB} \phi(d)$*;*

and so on. Notice that the domain is fixed, and it is involved in the above definition.

Definition 3. *(IC Satisfaction) A database instance* **DB** *satisfies a set of integrity constraints* **IC** *iff for every $\varphi \in \mathbf{IC}$,* $\mathbf{DB} \models_{DB} \varphi$*.*

If **DB** *does not satisfy* **IC***, we say that* **DB** *is* inconsistent *with* **IC***. Additionally, we say that a set of integrity constraints is* consistent *if there exists a database instance that satisfies it.*

Next we recall the relevant definitions from [1].

Given two database instances $\mathbf{DB}_1$ and $\mathbf{DB}_2$, the *distance* $\Delta(\mathbf{DB}_1, \mathbf{DB}_2)$ between them is their symmetric difference: $\Delta(\mathbf{DB}_1, \mathbf{DB}_2) = (\mathbf{DB}_1 - \mathbf{DB}_2) \cup (\mathbf{DB}_2 - \mathbf{DB}_1)$. This leads to the following partial order:

$$\mathbf{DB}_1 \leq_{\mathbf{DB}} \mathbf{DB}_2 \text{ iff } \Delta(\mathbf{DB}, \mathbf{DB}_1) \subseteq \Delta(\mathbf{DB}, \mathbf{DB}_2).$$

That is, $\leq_{\mathbf{DB}}$ determines the "closeness" to **DB**. The notion of closeness forms the basis for the concept of a repair of an inconsistent database.

Definition 4. *(Repair) Given database instances* **DB** *and* $\mathbf{DB}'$*, we say that* $\mathbf{DB}'$ *is a* repair *of* **DB** *with respect to a set of integrity constraints* **IC** *iff* $\mathbf{DB}'$ *satisfies* **IC** *and* $\mathbf{DB}'$ *is* $\leq_{\mathbf{DB}}$*-minimal in the class of database instances that satisfy* **IC**.

Clearly if **DB** is consistent with **IC**, then **DB** is its own repair. Concepts similar to database repair were proposed in the context of database maintenance and belief revision [7,4].

Example 1. (Repairing a Database) Consider a database schema with two unary relations p and q and domain $D = \{a, b, c, \dots\}$. Let $\mathbf{DB} = \{p(a), p(b), q(a), q(c)\}$ be a database instance over the domain D and let $\mathbf{IC} = \{\neg p(x) \vee q(x)\}$ be a set of constraints. This database does not satisfy **IC** because $\neg p(b) \vee q(b)$ is false.

Two repairs are possible. First, we can make $p(b)$ false, obtaining $\mathbf{DB}' = \{p(a), q(a), q(c)\}$. Alternatively, we can make $q(b)$ true, obtaining $\mathbf{DB}'' = \{p(a), p(b), q(a), q(b), q(c)\}$.

Definition 5. *(Consistent Answers) Let* **DB** *be a database instance,* **IC** *be set of integrity constraints and* $Q(\bar{x})$ *be a query. We say that a tuple of constants* $\bar{t}$ *is a* consistent answer *to the query, denoted* $\mathbf{DB} \models_c Q(\bar{t})$*, if for every repair* $\mathbf{DB}'$ *of* **DB**, $\mathbf{DB}' \models_{DB} Q(\bar{t})$.

If Q *is a closed formula, then true (respectively, false) is a* consistent answer *to* Q*, denoted* $\mathbf{DB} \models_c Q$*, if* $\mathbf{DB}' \models_{DB} Q$ *(respectively,* $\mathbf{DB}' \not\models_{DB} Q$*) for every repair* $\mathbf{DB}'$ *of* **DB**.

3 Annotated Predicate Calculus

Annotated predicate calculus (abbr. APC) [9] is a generalization of annotated logic programs introduced by Blair and Subrahmanian [3]. It was introduced in order to study the problem of "causes of inconsistency" in classical logical theories, which is closely related to the problem of consistent query answers being addressed in our present work. This section briefly surveys the basics of APC used in this paper.

The syntax and the semantics of APC is based on classical logic, except that the classical atomic formulas are annotated with values drawn from a *belief semilattice* (abbr. *BSL*) — an upper semilattice[1] with the following properties:

(i) *BSL* contains at least the following four distinguished elements: **t** (true), **f** (false), $\top$ (contradiction), and $\bot$ (unknown);
(ii) For every $\mathbf{s} \in BSL$, $\bot \leq \mathbf{s} \leq \top$ ($\leq$ is the semilattice ordering);
(iii) $\mathtt{lub}(\mathbf{t}, \mathbf{f}) = \top$, where `lub` denotes the least upper bound.

As usual in the lattice theory, `lub` imposes a partial order on *BSL*: $a \leq b$ iff $b = \mathtt{lub}(a, b)$ and $a < b$ iff $a \leq b$ and a is different from b. Two typical examples of *BSL* (which happen to be complete lattices) are shown in Figure 1. In both

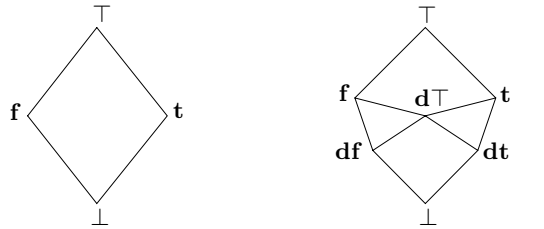

Fig. 1. Typical Belief Semilattices

of them, the lattice elements are ordered upwards. The specific *BSL* used in this paper is introduced later, in Figure 2.

Thus, the only syntactic difference between APC and classical predicate logic is that the atomic formulas of APC are constructed from the classical atomic formulas by attaching annotation suffixes. For instance, if $\mathbf{s}$, $\mathbf{t}$, $\top$ are elements of the belief semilattice, then $p(X) : \mathbf{s}$, $q : \top$, and $r(X, Y, Z) : \mathbf{t}$ all are atomic formulas in APC.

We define only the Herbrand semantics of APC (this is all we need here), and we also assume that the language is free of function symbols (because we are dealing with relational databases in this paper). We thus assume that the *Herbrand universe* is D, the set of all domain constants, and the *Herbrand base*, *HB*, is the set of all ground (*i.e.*, variable-free) atomic formulas of APC.

A *Herbrand interpretation* is any downward-closed subset of *HB*, where a set $I \subseteq HB$ is said to be *downward-closed* iff $p : \mathbf{s} \in I$ implies that $p : \mathbf{s}' \in I$ for every $\mathbf{s}' \in BSL$ such that $\mathbf{s}' \leq \mathbf{s}$. Formula satisfaction can then be defined as follows, where ν is a variable assignment that gives a value in D to every variable:

- $I \models_v p : \mathbf{s}$, where $\mathbf{s} \in BSL$ and p is a classical atomic formula, if and only if $p : \mathbf{s} \in I$.
- $I \models_v \phi \wedge \psi$ if and only if $I \models_v \phi$ and $I \models_v \psi$;
- $I \models_v \neg\psi$ if and only if not $I \models_v \psi$;
- $I \models_v (\forall X)\psi(X)$ if and only if $I \models_u \psi$, for *every* u that may differ from v only in its X-value.

It is thus easy to see that the definition of $\models$ looks very much classical. The only difference (which happens to have significant implications) is the syntax of atomic formulas and the requirement that Herbrand interpretations must be downward-closed. The implication $a \longleftarrow b$ is also defined classically, as $a \vee \neg b$.

It turns out that whether or not APC has a complete proof theory depends on which semilattice is used. It is shown in [9] that for a very large and natural class of semilattices (which includes all finite semilattices), APC has a sound and complete proof theory.

[1] That is, the least upper bound, $\texttt{lub}(a, b)$, is defined for every pair of elements $a, b \in BSL$.

The reason why APC is useful in analyzing inconsistent logical theories is because classical theories can be embedded in APC in various ways. The most useful types of embeddings are those where theories that are inconsistent in classical logic become consistent in APC. It then becomes possible to reason about the embedded theories and gain insight into the original inconsistent theory.

The two embeddings defined in [9] are called *epistemic* and *ontological*. Under the epistemic embedding, a (classically inconsistent) set of formulas such as $\mathbf{S} = \{p(1),\ \neg p(1),\ q(2)\}$ is embedded in APC as $\mathbf{S}^e = \{p(1) : \mathbf{t},\ p(1) : \mathbf{f},\ q(2) : \mathbf{t}\}$ and under the ontological embedding it is embedded as $\mathbf{S}^o = \{p(1) : \mathbf{t},\ \neg p(1) : \mathbf{t},\ q(2) : \mathbf{t}\}$.[2] In the second case, the embedded theory is still inconsistent in APC, but in the first case it does have a model: the downward closure of $\{p(1) : \top,\ q(2) : \mathbf{t}\}$. In this model, $p(1)$ is annotated with $\top$, which signifies that its truth value is "inconsistent." In contrast, the truth value of $q(2)$ is $\mathbf{t}$. More precisely, while both $q(2)$ and $\neg q(2)$ follow from $\mathbf{S}$ in classical logic, because $\mathbf{S}$ is inconsistent, only $q(2) : \mathbf{t}$ (but not $q(2) : \mathbf{f}$!) is implied by $\mathbf{S}^e$. Thus, $q(2)$ can be seen as a consistent answer to the query $?- q(X)$ with respect to the inconsistent database $\mathbf{S}$.

In [9], epistemic embedding has been shown to be a suitable tool for analyzing inconsistent classical theories. However, this embedding does not adequately capture the inherent lack of symmetry present in our setting, where inconsistency arises due to the incompatibility of two distinct sets of formulas (the database and the constraints) and only one of these sets (the database) is allowed to change to restore consistency. To deal with this problem, we develop a new type of embedding into APC. It uses a 10-valued lattice depicted in Figure 2, and is akin to the epistemic embedding of [9], but it also has certain features of the ontological embedding.

The above simple examples illustrate one important property of APC: a set of formulas, $\mathbf{S}$, might be *ontologically consistent* in the sense that it might have a model, but it might be *epistemically inconsistent* (abbr. *e-inconsistent*) in the sense that $\mathbf{S} \models p : \top$ for some p, *i.e.*, $\mathbf{S}$ contains at least one inconsistent fact. Moreover, $\mathbf{S}$ can be e-consistent (*i.e.*, it might not imply $p : \top$ for any p), but each of its models in APC might contain an inconsistent fact nonetheless (this fact must then be different in each model, if $\mathbf{S}$ is e-consistent).

It was demonstrated in [9] that ordering models of APC theories according to the amount of inconsistency they contain can be useful for studying the problem of recovering from inconsistency. To illustrate this order, consider $\mathbf{S} = \{p : \mathbf{t},\ p : \mathbf{f} \vee q : \mathbf{t},\ p : \mathbf{f} \vee q : \mathbf{f}\}$ and some of its models:

$\mathcal{M}_1$, where $p : \top$ and $q : \top$ are true;
$\mathcal{M}_2$, where $p : \top$ and $q : \bot$ are true;
$\mathcal{M}_3$, where $p : \mathbf{t}$ and $q : \top$ are true.

Among these models, both $\mathcal{M}_2$ and $\mathcal{M}_3$ contain strictly less inconsistent information than $\mathcal{M}_1$ does. In addition, $\mathcal{M}_2$ and $\mathcal{M}_3$ contain incomparable amounts

[2] $\neg p : \mathbf{v}$ is to be always read as $\neg(p : \mathbf{v})$.

of information, and they are both "minimal" with respect to the amount of inconsistent information that they have. This leads to the following definition.

Definition 6. *(E-Consistency Order) Given $\Delta \subseteq BSL$, a semantic structure I_1 is more (or equally)* e-consistent *than I_2 with respect to Δ (denoted $I_2 \leq_\Delta I_1$) if and only if for every atom $p(t_1, \ldots, t_k)$ and $\lambda \in \Delta$, whenever $I_1 \models p(t_1, \ldots, t_k) : \lambda$ then also $I_2 \models p(t_1, \ldots, t_k) : \lambda$.*

I is most e-consistent *in a class of semantic structures with respect to Δ, if no semantic structure in this class is strictly more e-consistent with respect to Δ than I (i.e., for every J in the class, $I \leq_\Delta J$ implies $J \leq_\Delta I$).*

4 Embedding Databases in APC

One way to find reliable answers to a query over an inconsistent database is to find an algorithm that implements the definition of consistent answers. While this approach has been successfully used in [1], it is desirable to see it as part of a bigger picture, because consistent query answers were defined at the meta-level, without an independent logical justification. A more general framework might (and does, as we shall see) help study the problem both semantically and algorithmically.

Our new approach is to embed inconsistent databases into APC and study the ways to eliminate inconsistency there. A similar problem was considered in [9] and we are going to adapt some key ideas from that work. In particular, we will define an embedding, $\mathcal{T}$, such that the repairs of the original database are precisely the models (in the APC sense) of the embedded database. This embedding is described below.

First, we define a special 10-valued lattice, $\mathcal{L}^{\mathbf{db}}$, which defines the truth values appropriate for our problem. The lattice is shown in Figure 2. The values $\bot$, $\top$, $\mathbf{t}$ and $\mathbf{f}$ signify undefinedness, inconsistency, truth, and falsehood, as usual. The other six truth values are explained below.

Informally, values $\mathbf{t_c}$ and $\mathbf{f_c}$ signify the truth values as they should be for the purpose of constraint satisfaction. The values $\mathbf{t_d}$ and $\mathbf{f_d}$ are the truth values as they should be according to the database **DB**. Finally, $\mathbf{t_a}$ and $\mathbf{f_a}$ are the *advisory* truth values. Advisory truth values are intended as keepers of the information that helps resolve conflicts between constraints and the database.

Notice that $\mathtt{lub}(\mathbf{f_d}, \mathbf{t_c})$ is $\mathbf{t_a}$ and $\mathtt{lub}(\mathbf{t_d}, \mathbf{f_c})$ is $\mathbf{f_a}$. This means that in case of a conflict between the constraints and the database the advise is to change the truth value of the corresponding fact to the one prescribed by the constraints. Intuitively, the facts that are assigned the advisory truth values are the ones that are to be removed or added to the database in order to satisfy the constraints. The gist of our approach is in finding an embedding of **DB** and **IC** into APC to take advantage of the above truth values.

Embedding the ICs. Given a set of integrity constraints **IC**, we define a new theory, $\mathcal{T}(\mathbf{IC})$, which contains three kinds of formulas:

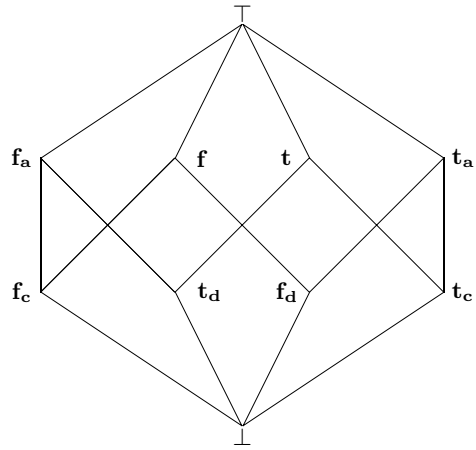

Fig. 2. The lattice $\mathcal{L}^{\mathbf{db}}$ with *constraints values*, *database values* and *advisory values*.

1. For every constraint in **IC**:

$$p_1(\bar{T}_1) \quad \vee \quad \cdots \quad \vee \quad p_n(\bar{T}_n) \quad \vee \quad \neg q_1(\bar{S}_1) \quad \vee \quad \cdots \quad \vee \quad \neg q_m(\bar{S}_m),$$

   $\mathcal{T}(\mathbf{IC})$ has the following formula:

$$p_1(\bar{T}_1) : \mathbf{t_c} \ \vee \cdots \vee \ p_n(\bar{T}_n) : \mathbf{t_c} \ \vee \ q_1(\bar{S}_1) : \mathbf{f_c} \ \vee \cdots \vee \ q_m(\bar{S}_m) : \mathbf{f_c}.$$

   In other words, positive literals are embedded using the "constraint-true" truth value, $\mathbf{t_c}$, and negative literals are embedded using the "constraint-false" truth value $\mathbf{f_c}$.
2. For every predicate symbol $p \in P$, the following formulas are in $\mathcal{T}(\mathbf{IC})$:

$$p(\bar{x}) : \mathbf{t_c} \ \vee \ p(\bar{x}) : \mathbf{f_c}, \ \neg \ p(\bar{x}) : \mathbf{t_c} \ \vee \ \neg \ p(\bar{x}) : \mathbf{f_c}.$$

   Intuitively, this says that every embedded literal must be either constraint-true or constraint-false (and not both).

Embedding Database Facts. $\mathcal{T}(\mathbf{DB})$, the embedding of the database facts into APC is defined as follows:

1. For every fact $p(\bar{a})$, where $p \in P$: if $p(\bar{a}) \in \mathbf{DB}$, then $p(\bar{a}) : \mathbf{t_d} \in \mathcal{T}(\mathbf{DB})$; if $p(\bar{a}) \notin \mathbf{DB}$, then $p(\bar{a}) : \mathbf{f_d} \in \mathcal{T}(\mathbf{DB})$.

Embedding Built-In Predicates. $\mathcal{T}(\mathcal{B})$, the result of embedding of the built-in predicates into APC is defined as follows:

1. For every built-in fact $p(\bar{a})$, where $p \in B$, the fact $p(\bar{a}) : \mathbf{t}$ is in $\mathcal{T}(\mathcal{B})$ iff $p(\bar{a})$ is true. Otherwise, if $p(\bar{a})$ is false then $p(\bar{a}) : \mathbf{f} \ \in \mathcal{T}(\mathcal{B})$.

2. $\neg\ p(\bar{x}) : \top\ \in \mathcal{T}(\mathcal{B})$, for every built-in $p \in B$.

The former rule simply says that built-in facts (like 1=1) that are true in classical sense must have the truth value **t** and the false built-in facts (*e.g.*, 2=3) must have the truth value **f**. The second rule states that built-in facts cannot be both true and false. This ensures that theories for built-in predicates are embedded in 2-valued fashion: every built-in fact in $\mathcal{T}(\mathcal{B})$ is annotated with either **t** or **f**, but not both.

Example 2. (Embedding, I) Consider the database $\mathbf{DB} = \{p(a), p(b), q(a)\}$ over the domain $D = \{a, b\}$ and let **IC** be $\{\neg p(x) \vee q(x)\}$. Then

$$\mathcal{T}(\mathbf{DB}) = \{p(a) : \mathbf{t_d},\ p(b) : \mathbf{t_d},\ q(a) : \mathbf{t_d},\ q(b) : \mathbf{f_d}\}$$

and $\mathcal{T}(\mathbf{IC})$ consists of:

$$\begin{array}{l} p(x) : \mathbf{f_c}\ \vee\ q(x) : \mathbf{t_c}, \\ p(x) : \mathbf{t_c}\ \vee\ p(x) : \mathbf{f_c},\ \ \neg\ p(x) : \mathbf{t_c}\ \vee\ \neg\ p(x) : \mathbf{f_c}, \\ q(x) : \mathbf{t_c}\ \vee\ q(x) : \mathbf{f_c},\ \ \neg\ q(x) : \mathbf{t_c}\ \vee\ \neg\ q(x) : \mathbf{f_c} \end{array}$$

Example 3. (Embedding, II) Let $\mathbf{DB} = \{p(a,a), p(a,b), p(b,a)\}$, $D = \{a, b\}$, and let **IC** be $\{\neg p(x,y) \vee \neg p(x,z) \vee y = z\}$. It is easy to see that this constraint represents the functional dependency $p.1 \rightarrow p.2$. Since this constraint involves the built-in "=", the rules for embedding the built-ins apply.

In this case, $\mathcal{T}(\mathbf{DB}) = \{p(a,a) : \mathbf{t_d},\ \ p(a,b) : \mathbf{t_d},\ \ p(b,a) : \mathbf{t_d},\ \ p(b,b) : \mathbf{f_d}\}$ and $\mathcal{T}(\mathbf{IC})$ is:

$$\begin{array}{l} p(x,y) : \mathbf{f_c}\ \vee\ p(x,z) : \mathbf{f_c}\ \vee\ y = z : \mathbf{t_c}, \\ p(x,y) : \mathbf{t_c}\ \vee\ p(x,y) : \mathbf{f_c},\ \neg\ p(x,y) : \mathbf{t_c}\ \vee\ \neg\ p(x,y) : \mathbf{f_c}. \end{array}$$

The embedded theory $\mathcal{T}(\mathcal{B})$ for the built-in predicate "=" is: $(a = a) : \mathbf{t},\ \ (b = b) : \mathbf{t},\ \ (a = b) : \mathbf{f},\ \ (b = a) : \mathbf{f},\ \ \neg\ (x = y) : \top$. □

Finally, we define $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ as $\mathcal{T}(\mathbf{DB}) \cup \mathcal{T}(\mathbf{IC}) \cup \mathcal{T}(\mathcal{B})$. We can now state the following properties that confirm our intuition about the intended meanings of the truth values in $\mathcal{L}^{\mathbf{db}}$.

Lemma 1. *If $\mathcal{M}$ is a model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, then for every predicate $p \in P$ and a fact $p(\bar{a})$, the following is true:*

1. $\mathcal{M} \models \neg\ p(\bar{a}) : \top$.
2. $\mathcal{M} \models p(\bar{a}) : \mathbf{t}\ \vee\ p(\bar{a}) : \mathbf{f}\ \vee\ p(\bar{a}) : \mathbf{t_a} \vee p(\bar{a}) : \mathbf{f_a}$. □

The first part of the lemma says that even if the initial database **DB** is inconsistent with constraints **IC**, every model of our embedded theory is *epistemically consistent* in the sense of [9], *i.e.*, no fact of the form $p(\bar{a}) : \top$ is true in any such model.[3] The second part says that any fact is either true, or false, or it has

[3] Note that an APC theory *can* entail $p(\bar{a}) : \top$ and be consistent in the sense that it can have a model. However, such a model must contain $p(\bar{a}) : \top$, which makes it epistemically inconsistent.

an advisory value of true or false. This indicates that database repairs can be constructed out of these embeddings by converting the advisory truth values to the corresponding values **t** and **f**. This idea is explored next.

Given a pair of database instances $\mathbf{DB}_1$ and $\mathbf{DB}_2$ over the same domain, we construct the Herbrand structure $\mathcal{M}(\mathbf{DB}_1, \mathbf{DB}_2) = \langle D, I_P, I_B \rangle$, where D is the domain of the database and I_P, I_B are the interpretations for the predicates and the built-ins, respectively. I_P is defined as follows:

$$I_P(p(\bar{a})) = \begin{cases} \mathbf{t} & p(\bar{a}) \in \mathbf{DB}_1,\ p(\bar{a}) \in \mathbf{DB}_2 \\ \mathbf{f} & p(\bar{a}) \notin \mathbf{DB}_1,\ p(\bar{a}) \notin \mathbf{DB}_2 \\ \mathbf{f_a} & p(\bar{a}) \in \mathbf{DB}_1,\ p(\bar{a}) \notin \mathbf{DB}_2 \\ \mathbf{t_a} & p(\bar{a}) \notin \mathbf{DB}_1,\ p(\bar{a}) \in \mathbf{DB}_2 \end{cases} \tag{1}$$

The interpretation I_B is defined as expected: if q is a built-in, then $I_P(q(\bar{a})) = \mathbf{t}$ iff $q(\bar{a})$ is true in classical logic, and $I_P(q(\bar{a})) = \mathbf{f}$ iff $q(\bar{a})$ is false.

Notice that $\mathcal{M}(\mathbf{DB}_1, \mathbf{DB}_2)$ is not symmetric. The intent is to use these structures as the basis for construction of database repairs. In fact, when $\mathbf{DB}_1$ is inconsistent and $\mathbf{DB}_2$ is a repair, I_P shows how the advisory truth values are to be changed to obtain a repair.

Lemma 2. *Given two database instances* **DB** *and* $\mathbf{DB}'$*, if* $\mathbf{DB}' \models_{DB} \mathbf{IC}$*, then* $\mathcal{M}(\mathbf{DB}, \mathbf{DB}') \models \mathcal{T}(\mathbf{DB}, \mathbf{IC})$. □

The implication of this lemma is that whenever **IC** is consistent, then the theory $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ is also consistent in APC. Since in this paper we are always dealing with consistent sets of integrity constraints, we conclude that $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ is always a consistent APC theory.

We will now show how to generate repairs out of the models of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. Given a model $\mathcal{M}$ of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, we define $\mathbf{DB}_{\mathcal{M}}$ as:

$$\{p(\bar{a}) \mid p \in P \text{ and } \mathcal{M} \models\ p(\bar{a}) : \mathbf{t} \vee p(\bar{a}) : \mathbf{t_a}\}. \tag{2}$$

Note that $\mathbf{DB}_{\mathcal{M}}$ can be an infinite set of facts (but finite when $\mathcal{M}$ corresponds to a database instance).

Lemma 3. *If* $\mathcal{M}$ *is a model of* $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ *such that* $\mathbf{DB}_{\mathcal{M}}$ *is finite, then* $\mathbf{DB}_{\mathcal{M}} \models_{DB} \mathbf{IC}$.

Proposition 1. *Let* $\mathcal{M}$ *be a model of* $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$*. If* $\mathcal{M}$ *is most e-consistent with respect to* $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ *(see Definition 6) among the models of* $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ *and* $\mathbf{DB}_{\mathcal{M}}$ *is finite, then* $\mathbf{DB}_{\mathcal{M}}$ *is a repair of* **DB** *with respect to* **IC**.

Proposition 2. *If* $\mathbf{DB}'$ *is a repair of* **DB** *with respect to the set of integrity constraints* **IC***, then* $\mathcal{M}(\mathbf{DB}, \mathbf{DB}')$ *is most e-consistent with respect to* $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ *among the models of* $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$.

Example 4. (Repairs as Most e-Consistent Models) Consider a database instance $\mathbf{DB} = \{p(a)\}$ over the domain $D = \{a\}$ and a set of integrity constraints $\mathbf{IC} = \{\neg p(x) \vee q(x),\ \neg q(x) \vee r(x)\}$. In this case $\mathcal{T}(\mathbf{DB}) = \{p(a) : \mathbf{t_d},\ q(a) : \mathbf{f_d},\ r(a) : \mathbf{f_d}\}$, and $\mathcal{T}(\mathbf{IC})$ is

$$\begin{array}{ll} p(x) : \mathbf{f_c} \vee q(x) : \mathbf{t_c}, & q(x) : \mathbf{f_c} \vee r(x) : \mathbf{t_c}, \\ p(x) : \mathbf{t_c} \vee p(x) : \mathbf{f_c}, & \neg p(x) : \mathbf{t_c} \vee \neg p(x) : \mathbf{f_c}, \\ q(x) : \mathbf{t_c} \vee q(x) : \mathbf{f_c}, & \neg q(x) : \mathbf{t_c} \vee \neg q(x) : \mathbf{f_c}, \\ r(x) : \mathbf{t_c} \vee r(x) : \mathbf{f_c}, & \neg r(x) : \mathbf{t_c} \vee \neg r(x) : \mathbf{f_c} \end{array}$$

This theory has four models, depicted in the following table:

	$p(a)$	$q(a)$	$r(a)$
$\mathcal{M}_1$	$\mathbf{t}$	$\mathbf{t_a}$	$\mathbf{t_a}$
$\mathcal{M}_2$	$\mathbf{f_a}$	$\mathbf{f}$	$\mathbf{f}$
$\mathcal{M}_3$	$\mathbf{f_a}$	$\mathbf{f}$	$\mathbf{t_a}$
$\mathcal{M}_4$	$\mathbf{f_a}$	$\mathbf{t_a}$	$\mathbf{t_a}$

It is easy to verify that $\mathcal{M}_1$ and $\mathcal{M}_2$ are the most e-consistent models with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ among the models in the table and the database instance $\mathbf{DB}_{\mathcal{M}_1} = \{p(a), q(a), r(a)\}$ and $\mathbf{DB}_{\mathcal{M}_2} = \emptyset$ are exactly the repairs of **DB** with respect to **IC**.

Example 5. (Example 3 Continued) The embedding of the database described in Example 3 has nine models listed in the following table. The table omits the built-in "=", since it has the same interpretation in all models.

	$p(a,a)$	$p(a,b)$	$p(b,a)$	$p(b,b)$
$\mathcal{M}_1$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_2$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_3$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t_a}$
$\mathcal{M}_4$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_5$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_6$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f_a}$	$\mathbf{t_a}$
$\mathcal{M}_7$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t}$	$\mathbf{f}$
$\mathcal{M}_8$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f}$
$\mathcal{M}_9$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{f_a}$	$\mathbf{t_a}$

It is easy to see that $\mathcal{M}_1$ and $\mathcal{M}_4$ are the most e-consistent models with respect to $\Delta = \{\mathbf{t_a}, \mathbf{f_a}, \top\}$ among the models in the table, and the database instances $\mathbf{DB}_{\mathcal{M}_1} = \{p(a,a),\ \ p(b,a)\}$, and $\mathbf{DB}_{\mathcal{M}_4} = \{p(a,b),\ \ p(b,a)\}$ are exactly the repairs of **DB** with respect to **IC**.

5 Repairing Inconsistent Databases

To construct all possible repairs of a database, **DB**, that is inconsistent with the integrity constraints **IC**, we need to find the set of all ground clauses of the form

$$\mathbf{p}_1 : ?_\mathbf{a} \vee \cdots \vee \mathbf{p}_n : ?_\mathbf{a}, \tag{3}$$

that are implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$, where each $?_{\mathbf{a}}$ is either $\mathbf{t_a}$ or $\mathbf{f_a}$. Such clauses are called *a-clauses*, for advisory clauses.[4]

A-clauses are important because one of the disjuncts of such a clause must be true in each model of $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. Suppose that, say, $\mathbf{p} : ?_{\mathbf{a}}$ is true in some model I. This means that the truth value of $\mathbf{p}$ with respect to the database is exactly the opposite of what is required in order for I to satisfy the constraints. This observation can be used to construct a repair of the database by reversing the truth value of $\mathbf{p}$ with respect to the database. We explore this idea next.

Constructing Database Repairs. Let $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ be the set of all *minimal a-clauses* that are implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$. "Minimal" here means that no disjunct can be removed from any clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ and still have the clause implied by $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$.

In general, this can be an infinite set, but in most practical cases this set is finite. Conditions for finiteness of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ are given in Section 5.1. If $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ is finite, it can be represented as the following set of clauses:

$$\begin{array}{rcl} C_1 & = & \mathbf{p}_{1,1} : \mathbf{a}_{1,1} \vee \cdots \vee \mathbf{p}_{1,n_1} : \mathbf{a}_{1,n_1} \\ & \ddots & \ddots \qquad\qquad \ddots \\ C_k & = & \mathbf{p}_{k,1} : \mathbf{a}_{k,1} \vee \cdots \vee \mathbf{p}_{k,n_k} : \mathbf{a}_{k,n_k} \end{array}$$

Here, the $\mathbf{p}_{i,j} : \mathbf{a}_{i,j}$ are ground positive literals and their annotations, $\mathbf{a}_{i,j}$, are always of the form $\mathbf{t_a}$ or $\mathbf{f_a}$.

It can be shown that all *a-clauses* can be generated using the APC resolution inference rule [9] between $\mathcal{T}(\mathbf{IC})$, $\mathcal{T}(\mathbf{DB})$, and $\mathcal{T}(\mathcal{B})$. It can be also shown that all *a*-clauses generated in this way are ground and do not contain built-in predicates.

Given $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ as above, a *repair signature* is a set of APC literals that contains at least one literal from each clause C_i and is minimal in the sense that no proper subset has a literal from each C_i. In other words, a repair signature is a minimal hitting set of the family of clauses $C_1, \ldots, C_k$ [6].

Notice that if the clauses C_i do not share literals, then each repair signature contains exactly k literals and every literal appearing in a clause C_i belongs to some repair signature.

It follows from the construction of repairs in (2) and from Propositions 1 and 2 that there is a one to one correspondence between repair signatures and repairs of the original database instance $\mathbf{DB}$. Given a repair signature `Repair`, a repair $\mathbf{DB}'$ can be obtained from $\mathbf{DB}$ by removing the tuples $\mathbf{p}(\bar{t})$, if $\mathbf{p}(\bar{t}) : \mathbf{f_a} \in$ `Repair`, and inserting the tuples $\mathbf{p}(\bar{t})$, if $\mathbf{p}(\bar{t}) : \mathbf{t_a} \in$ `Repair`. It can be shown that it is not possible for any fact, $\mathbf{p}$, to occur in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ with two different annotations. Therefore, it is not possible that the same fact will be inserted and then removed (or vice versa) while constructing a repair as described here.

[4] Here, bold face symbols, *e.g.*, $\mathbf{p}$, denote classical ground atomic formulas.

5.1 Finiteness of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$

We now examine the issue of finiteness of the set $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$.

Definition 7. *(Range-Restricted Constraints) An integrity constraint,* $p_1(\bar{T}_1) \vee \cdots \vee p_n(\bar{T}_n) \vee \neg q_1(\bar{T}'_1) \vee \cdots \vee \neg q_m(\bar{T}'_m)$*, is* range-restricted *if and only if every variable in* $\bar{T}_i$ $(1 \leq i \leq n)$ *also occurs in some* $\bar{T}'_j$ $(1 \leq j \leq m)$*. Both* p_i *and* q_j *can be built-in predicates.*

A set **IC** *of constraints is* range-restricted *if so is every constraint in* **IC**.

Lemma 4. *Let* **IC** *be a set of range-restricted constraints over a database* **DB**. *Then every a-clause implied by* $\mathcal{T}(\mathbf{DB}, \mathbf{IC})$ *(i.e., every clause of the form (3)) mentions only the constants in the active domain of* **DB**.[5]

Corollary 1. *If* **IC** *is range-restricted, then* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ *is finite.*

6 Queries to Inconsistent Databases

In general, the number of all repair signatures can be exponential in the size of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$, so using this theory directly is not likely to produce a good query engine. In fact, for the propositional case, [5] shows that the problem of deciding whether a formula holds in all models produced by Winslett's theory of updates [4] is Π_2^P*-complete*. Since, as mentioned before, our repairs are essentially Winslett's updated models, the same result applies to our case.

However, there are cases when complexity is manageable. It is easy to see that if k is the number of clauses in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ and n_1, ..., n_k are the numbers of disjuncts in C_1, ..., C_k, respectively, then the number of repair signatures is $O(n_1 \times \ldots \times n_k)$. Therefore, two factors affect the number of repairs:

1. The number of clauses in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$;
2. The number of disjuncts in each clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$.

So, we should look into those types of constraints where either k is bound or all but a bound number of n_i's equal 1.

Other cases when query answering is feasible arise when the set of *a-clauses* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ is precomputed. Precomputing this set might be practical for read-only databases. In other cases, $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ might be easy to compute because of the special form of constraints (and in this case, the size of $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ turns out to be P-bounded). For instance, suppose **IC** consists of range-restricted formulas and is closed under the resolution inference rule (e.g. if **IC** is a set of functional dependencies). In this case, *a*-clauses can be generated by converting each constraint into a query that finds all tuples that violate the constraint. For instance, the constraint $p(\bar{x}) \supset q(\bar{x})$ can be converted into the query $p(\bar{x}) \wedge \neg q(\bar{x})$ (which is the denial form of this constraint). If the tuple $\bar{a}$ is an answer, then one *a-clause* is $p(\bar{a}) : \mathbf{f_a} \vee q(\bar{a}) : \mathbf{t_a}$.

[5] The active domain consists of the constants in D that appear in some database table.

Answering Ground Conjunctive Queries. To consistently answer a ground conjunctive query of the form $\mathbf{p}_1 \wedge \ldots \wedge \mathbf{p}_k \wedge \neg\mathbf{q}_1 \wedge \ldots \wedge \neg\mathbf{q}_m$, we need to check the following:

For each $\mathbf{p}_i$**:** if $\mathbf{p}_i \in \mathbf{DB}$ and $\mathbf{p}_i : \mathbf{f_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$; or if $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ has a clause of the form $\mathbf{p}_i : \mathbf{t_a}$.

For each $\mathbf{q}_j$**:** if $\mathbf{q}_j \notin \mathbf{DB}$ and $\mathbf{q}_j : \mathbf{t_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$; or if $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ has a clause of the form $\mathbf{q}_j : \mathbf{f_a}$.

If all of the above holds, *true* is a consistent answer to the query. Otherwise, the answer is *not true*, meaning that there is at least one repair where our conjunctive query is false. (Note that this is not the same as answering *false* in definition 5).

Non-ground Conjunctive Queries. Let $\mathbf{DB}$ have the relations $p_1, \ldots, p_n$. We construct a new database, $\mathbf{DB}^{O,U}$, with relations $p_1^O, \ldots, p_n^O, p_1^U, \ldots, p_n^U$ (where O and U stand for "original" and "unknown", resp.), as follows:

p_i^O *consists of*: all the tuples such that $p_i(\bar{t}) \in \mathbf{DB}$ and $p_i(\bar{t}) : \mathbf{f_a}$ is not mentioned in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ plus the tuples $\bar{t}$ such that $p(\bar{t}) : \mathbf{t_a}$ is a clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$.

p_j^U *consists of*: all the tuples $\bar{t}$ such that $p_j(\bar{t}) : \mathbf{t_a}$ or $p_j(\bar{t}) : \mathbf{f_a}$ appear in a clause in $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ that has more than one disjunct.

To answer an open conjunctive query, for example, $p(x) \wedge \neg q(x)$, we pose the query $p^O(x) \wedge \neg q^O(x) \wedge \neg q^U(x)$ to $\mathbf{DB}^{O,U}$. This can be done in polynomial time in the database size plus the size of the set of *a-clauses*.

Ground Disjunctive Queries. Sound and complete query evaluation techniques for various types of queries and constraints are developed in [1]. Our present framework extends the results in [1] to include disjunctive queries. We concentrate on ground disjunctive queries of the form

$$\mathbf{p}_1 \vee \cdots \vee \mathbf{p}_k \vee \neg\mathbf{q}_1 \cdots \neg\mathbf{q}_r. \tag{4}$$

First, for each p_i we evaluate the query $\mathbf{p}_i^O$ and for each q_j we evaluate the query $\neg\mathbf{q}_j^O \wedge \neg\mathbf{q}_j^U$ against the database $\mathbf{DB}^{O,U}$. If at least one *true* answer is obtained, the answer to (4) is *true*. Otherwise, if all these queries return *false*, we evaluate the queries of the form $\neg\mathbf{p}_i^O \wedge \neg\mathbf{p}_i^U$ and $\mathbf{q}_j^O$ against $\mathbf{DB}^{O,U}$. For each answer *true*, the corresponding literal is eliminated from (4). Let $\mathbf{p}_{i_1} \vee \cdots \vee \mathbf{p}_{i_s} \vee \neg\mathbf{q}_{j_1} \cdots \neg\mathbf{q}_{j_t}$ be the resulting query. If this query is empty, then the answer to the original query is *false*, *i.e.*, the original query is false in every repair. If the resulting query is not empty, we must check if there is a minimal hitting set for $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$, that contains $\{\neg\mathbf{p}_{i_1}, \ldots, \neg\mathbf{p}_{i_s}, \mathbf{q}_{j_1}, \ldots, \mathbf{q}_{j_t}\}$. If such a hitting set exists, the answer to the original query is *maybe*, meaning that there is at least one repair where the answer is *false*. Otherwise, the answer to the query is *true*.

Therefore, the problem of answering disjunctive queries for a given $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ is equivalent to the problem of deciding whether a given set can be extended to a minimal hitting set of the family. Since this is an *NP-complete* problem, we have the following result.

Proposition 3. *Suppose that* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$ *has been precomputed. Then the problem of deciding whether true is a consistent answer to a disjunctive ground query is NP-complete with respect to the size of* $\mathbf{DB}$ *plus* $\mathcal{T}^{\mathbf{a}}(\mathbf{DB}, \mathbf{IC})$.

7 Conclusions

We presented a new semantic framework, based on Annotated Predicate Calculus [9], for studying the problem of query answering in databases that are inconsistent with integrity constraints. This was done by embedding both the database instance and the integrity constraints into a single theory written in an APC with an appropriate truth values lattice. In this way, we obtain a general logical specification of database repairs and consistent query answers.

With this new framework, we are able to provide a better analysis of the computational complexity of query answering in such environments and to develop a more general query answering mechanism than what was known previously [1]. We also identified certain classes of queries and constraints that have lower complexity, and we are looking into better query evaluation algorithms for these classes.

The development of the specific mechanisms for consistent query answering in the presence of universal ICs, and the extension of our methodology to constraints that contain existential quantifiers (*e.g.*, referential integrity constraints) is left for future work.

Acknowledgements

We would like to thank the anonymous referees for their valuable comments. Work supported by Fondecyt Grants # 1980945, # 1000593; and ECOS/CONICYT Grant C97E05.

References

1. M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM Symposium on Principles of Database Systems (ACM PODS'99, Philadelphia)*, pages 68–79, 1999.
2. N. Belnap. A Useful Four-Valued Logic. In M. Dunn and G. Epstein, editors, *Modern Uses of Multi-Valued Logic*, pages 8–37. Reidel Publ. Co., 1977.
3. H.A. Blair and V.S. Subrahmanian. Paraconsistent Logic Programming. *Theoretical Computer Science*, 68:135–154, 1989.
4. T. Chou and M. Winslett. A Model-Based Belief Revision System. *J. Automated Reasoning*, 12:157–208, 1994.
5. T. Eiter and G. Gottlob. On the Complexity of Propositional Knowledge Base Revision, Updates, and Counterfactuals. *J. Artificial Intelligence*, 57(2-3):227–270, 1992.
6. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.
7. M. Gertz. *Diagnosis and Repair of Constraint Violations in Database Systems*. PhD thesis, Universität Hannover, 1996.

8. M.L. Ginsberg. Multivalued Logics: A Uniform Approach to Reasoning in Artificial Intelligence. *Computational Intelligence*, 4:265–316, 1988.
9. M. Kifer and E.L. Lozinskii. A Logic for Reasoning with Inconsistency. *Journal of Automated Reasoning*, 9(2):179–215, November 1992.
10. M. Kifer and V.S. Subrahmanian. Theory of Generalized Annotated Logic Programming and its Applications. *Journal of Logic Programming*, 12(4):335–368, April 1992.
11. V.S. Subrahmanian. On the Semantics of Quantitative Logic Programs. In *IEEE Symposium on Logic Programming*, pages 173–182, 1987.
12. M.H. van Emden. Quantitative Deduction and its Fixpoint Theory. *Journal of Logic Programming*, 1(4):37–53, 1986.