# Computing Approximate Query Answers over Inconsistent Knowledge Bases

**Sergio Greco, Cristian Molinaro, Irina Trubitsyna**
University of Calabria, Italy
{greco,cmolinaro,trubitsyna}@dimes.unical.it

## Abstract

Consistent query answering is a principled approach for querying inconsistent knowledge bases. It relies on the notion of a *repair*, that is, a maximal consistent subset of the facts in the knowledge base. One drawback of this approach is that entire facts are deleted to resolve inconsistency, even if they may still contain useful "reliable" information. To overcome this limitation, we propose a new notion of repair allowing values within facts to be updated for restoring consistency. This more fine-grained repair primitive allows us to preserve more information in the knowledge base. We also introduce the notion of a *universal repair*, which is a compact representation of all repairs. Then, we show that consistent query answering in our framework is intractable (coNP-complete). In light of this result, we develop a polynomial time approximation algorithm for computing a sound (but possibly incomplete) set of consistent query answers.

## 1 Introduction

Reasoning in the presence of inconsistent information is a problem that has attracted much interest in the last decades. Many inconsistency-tolerant semantics for query answering have been proposed, and most of them rely on the notions of *consistent query answer* and *repair*. A consistent answer to a query is a query answer that is entailed by every repair, where a repair is a "maximal" consistent subset of the facts of the knowledge base. Different maximality criteria have been investigated, but all the resulting notions of repair share the same drawback: a fact is either kept or deleted altogether, and deleting entire facts can cause loss of "reliable" information.

**Example 1.** Consider the knowledge base $(D, \Sigma)$ where $D$ contains the following facts:

works

| | | |
|---|---|---|
| john | cs | nyc |
| john | math | rome |
| mary | math | sydney |

and $\Sigma$ is an ontology consisting of the following equality-generating dependency (EGD) $\sigma$:

$$\mathsf{works}(\mathsf{E}_1, \mathsf{D}, \mathsf{C}_1) \wedge \mathsf{works}(\mathsf{E}_2, \mathsf{D}, \mathsf{C}_2) \rightarrow \mathsf{C}_1 = \mathsf{C}_2.$$

As an example, the fact works(john, cs, nyc) states that john is an employee working in the cs department located in nyc. The dependency $\sigma$ says that every department must be located in a single city. Clearly, the last two facts violate $\sigma$, so every repair would discard either of them.

If we pose a query asking for the employees' name, the only consistent answer is john. However, intuitively, we might consider reliable the information on mary being an employee, as the only uncertainty concerns the math department and its city—roughly speaking, the information in the first column of the works table can be considered "clean". Dropping entire facts causes loss of information.

To overcome the drawback illustrated above, we propose a notion of repair based on updating values within facts. Update-based repairing allows for rectifying errors in facts without deleting them altogether, thereby preserving consistent values.

**Example 2.** Consider again the knowledge base of Example 1. Using value updates as the primitive to restore consistency, and assuming that the only uncertain values are math's cities, we get the following two repairs:

| | | |
|---|---|---|
| john | cs | nyc |
| john | math | rome |
| mary | math | rome |

| | | |
|---|---|---|
| john | cs | nyc |
| john | math | sydney |
| mary | math | sydney |

If we ask again for the employees' name, both mary and john are consistent answers.

We consider knowledge bases where ontologies are expressed through (a particular class of) EGDs. Equality-generating dependencies are one of the two major types of data dependencies—the other major type consists of tuple-generating dependencies (TGDs)—and can model several kinds of constraints commonly arising in practice, such as functional dependencies and thus also key dependencies.

We show that consistent query answering in this setting is coNP-complete. Then, we show how to compute a "universal" repair, which compactly represents all repairs and can be computed in polynomial time. A universal repair is a valuable tool to compute approximate query answers. We propose a polynomial time approximation to compute a sound (but possibly incomplete) set of consistent query answers. The basic idea is illustrated in the following example.

**Example 3.** A universal repair for the two repairs of Example 2 is reported below:

| | | |
|---|---|---|
| john | cs | nyc |
| john | math | $\perp_1$ |
| mary | math | $\perp_1$ |

$\perp_1 = \text{rome} \vee \perp_1 = \text{sydney}$

where $\perp_1$ is a labeled null, and the "global" condition at the bottom restricts the admissible values for $\perp_1$, which are rome and sydney. We exploit such a representation for query answering, by combining the condition of the universal repair with provenance information during query evaluation. For instance, if we ask for the departments that are not located in nyc, we get math with condition $\perp_1 \neq \text{nyc}$, which combined with the global condition allows us to conclude that math is a consistent answer (as its city is either rome or sydney, and thus cannot be nyc for sure).

As shown in the following, leveraging universal repairs, our approximation schema yields strictly better results than well-known approximation approaches such as the *intersection of repairs* and the *intersection of closed repairs* semantics [Lembo *et al.*, 2010; Bienvenu *et al.*, 2014].

Below we provide a further example showing that even when knowledge bases are expressed by unary/binary facts only, our approach can preserve more information than the classic notion of repair.

**Example 4.** Consider the knowledge base $(D, \Sigma)$ where

$$D = \{\text{emp}(\text{john}, \text{sydney}), \text{stud}(\text{john}, \text{nyc})\} \text{ and}$$
$$\Sigma = \{\text{emp}(\text{N}, \text{C}) \wedge \text{stud}(\text{N}, \text{C}') \rightarrow \text{C} = \text{C}'\}.$$

Here emp(john, sydney) means that john is an employee living in sydney, and stud(john, nyc) means that john is a student living in nyc.

The EGD says that every person must be associated with the same city in emp and stud. Clearly, the knowledge base is inconsistent.

Consider now the query asking for the people who are both an employee and a student. Assuming that the only uncertain values are john's cities, we can still say that john is certainly an employee and a student, even if we are not sure where he lives. This information is preserved by our framework, as john is a consistent answer, but it is completely lost using the classic notion of repair, as there are no consistent answers. In fact, in our framework, there are two repairs obtained from $D$ by changing either sydney into nyc or nyc into sydney—in both of them john is an employee and a student. According to the classical notion of repair, there are two repairs obtained from $D$ by deleting either emp(john, sydney) or stud(john, nyc)—in the former john is a student but not an employee while in the latter he is an employee but not a student.

**Contributions.**
We consider the problem of querying possibly inconsistent knowledge bases in the presence of *acyclic* sets of EGDs. In this setting, we propose a new notion of repair based on a more fine-grained repair primitive, which updates values within facts rather than deleting them altogether, enabling us to preserve more information of the knowledge base. For such a repair strategy, we introduce the notion of a *universal repair*, which is a compact representation of all repairs, can be computed in polynomial time, and can be leveraged to compute consistent query answers.

We then develop a chase-like procedure to compute a universal repair, introducing a formalism that augments instances with provenance information. We show that consistent query answering in our framework is coNP-complete (data complexity). In light of this, we leverage universal repairs and provenance information to develop an approximation algorithm that provides a sound (but possibly incomplete) set of consistent query answers in polynomial time.

**Organization.**
Section 2 discusses related work. Preliminaries are reported in Section 3. Our inconsistency-tolerant semantics and its complexity are investigated in Section 4. A compact representation of all repairs and its computation are proposed in Section 5. Query evaluation is addressed in Section 6. Our approximation algorithm is introduced in Section 7. Conclusions and directions for future work are reported in Section 8.

## 2 Related Work

Reasoning in the presence of inconsistent information is a problem that has attracted a great deal of interest in the AI and database communities. Consistent query answering was first proposed in [Arenas *et al.*, 1999]. Query answering under various inconsistency-tolerant semantics for ontologies expressed in DL languages has been studied in [Lembo *et al.*, 2010; 2011; 2015; Bienvenu, 2011; 2012; Rosati, 2011; Bienvenu and Rosati, 2013], and in [Lukasiewicz *et al.*, 2012b; 2015; 2012a] for ontologies expressed by fragments of Datalog+/–. Several notions of maximality for a repair have been considered in [Bienvenu *et al.*, 2014]. Bienvenu and Rosati (2013) have proposed an approach for the approximation of consistent query answers from above and from below. Furfaro *et al.* (2007) proposed an approach based on three-valued logic to compute a sound but possibly incomplete set of consistent query answers.

Different from our proposal, all the approaches above adopt the most common notion of repair, where whole facts are removed. This can cause loss of information, as illustrated in the toy scenario of Example 1—in real-life scenarios, it might well be the case that facts have much more attributes and only a few of them are involved in inconsistencies, leading to significant loss of useful data.

There have also been different proposals adopting a notion of repair that allows values to be updated [Bohannon *et al.*, 2005; Bertossi *et al.*, 2008; Greco and Molinaro, 2008; 2012; Flesca *et al.*, 2010]. Our repair strategy behaves similar to the one of [Bohannon *et al.*, 2005; Greco and Molinaro, 2008; 2012] in that values on the right-hand side of functional dependencies (FDs) are updated. However, those works focus on FDs only. [Bertossi *et al.*, 2008] allow only numerical attributes to be updated, one primary key per relation is allowed, but keys are assumed to be satisfied by the original database: thus, no repairing is possible w.r.t. keys, while we allow it, and we allow much more general constraints. Our repair strategy can be seen as an instantiation of the value-based family of policies proposed in [Martinez *et al.*, 2014], even though the two approaches differ in how multiple dependencies are handled, and they focus on FDs only. Flesca *et al.* (2010) consider numerical databases and a different class of (aggregate) constraints.

The main difference between this paper and the aforementioned ones is that none of them has investigated the approximate computation of consistent query answers—notice that [Bertossi *et al.*, 2008; Bohannon *et al.*, 2005] have proposed approximation algorithms for computing a repair with minimum distance from the original database, and [Greco and Molinaro, 2008] consider approximate probabilistic query answers. Also, we introduce the notion of a *universal repair*, which compactly represents all repairs and can be used for exact/approximate query answering.

Approximation algorithms for computing sound but possibly incomplete sets of query answers in the presence of nulls have been proposed in [Guagliardo and Libkin, 2016; Libkin, 2015; 2016; Greco *et al.*, 2017; Fiorentino *et al.*, 2018], but no dependencies are considered therein, and thus the database is assumed to be consistent.

## 3 Preliminaries

We assume the existence of the following pairwise disjoint (countably infinite) sets: a set $\mathsf{Const}$ of *constants*, a set $\mathsf{Var}$ of *variables*, and a set $\mathsf{Null}$ of *labeled nulls*. Nulls are denoted by the symbol $\perp$ subscripted. A *term* is a constant, variable, or null. We also assume a set of *predicates*, disjoint from the aforementioned sets, with each predicate being associated with an *arity*, which is a non-negative integer.

An *atom* $A$ is of the form $p(t_1, \dots, t_n)$, where $p$ is an $n$-ary predicate and the $t_i$'s are terms. We write an atom also as $p(\mathbf{t})$, where $\mathbf{t}$ is a sequence of terms. An atom without variables is also called a *fact*. An *instance* is a finite set of facts. A *database* is an instance containing constants only.

A *homomorphism* is a mapping $h : \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null} \rightarrow \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null}$ that is the identity on $\mathsf{Const}$. Homomorphisms are also applied to atoms and set of atoms in the natural fashion, that is, $h(p(t_1, \dots, t_n)) = p(h(t_1), \dots, h(t_n))$, and $h(S) = \{h(A) \mid A \in S\}$ for any set $S$ of atoms. A *valuation* is a homomorphism $\nu$ whose image is $\mathsf{Const}$, that is, $\nu(t) \in \mathsf{Const}$ for every $t \in \mathsf{Const} \cup \mathsf{Var} \cup \mathsf{Null}$.

**Conditional instances.**
*Conditional instances* (also known as "conditional tables" [Imielinski and Lipski, 1984; Grahne, 1991]) are instances augmented with conditions restricting the set of admissible values for nulls. Let $\mathcal{E}$ be the set of all expressions, called *conditions*, that can be built using the standard logical connectives $\wedge$, $\vee$, $\neg$, $\Rightarrow$ and expressions of the form $t_i = t_j$, $\mathsf{true}$, and $\mathsf{false}$, where $t_i, t_j \in \mathsf{Const} \cup \mathsf{Null}$. We will also use $t_i \neq t_j$ as a shorthand for $\neg(t_i = t_j)$. We say that a valuation $\nu$ *satisfies* a condition $\phi$, denoted $\nu \models \phi$, if its assignment of constants to nulls makes $\phi$ true. Formally, a *conditional instance* (CI) is a pair $\langle I, \Phi \rangle$, where $I$ is an instance and $\Phi \in \mathcal{E}$. The semantics of $C = \langle I, \Phi \rangle$ is given by the set of its *possible worlds*, that is, the set of databases $pw(C) = \{\nu(I) \mid \nu \text{ is a valuation and } \nu \models \Phi\}$.

**Equality generating dependencies.**
An *equality generating dependency* (EGD) $\sigma$ is a first-order formula of the form $\forall \mathbf{x}\, \varphi(\mathbf{x}) \rightarrow x_i = x_j$, where $\varphi(\mathbf{x})$ is a conjunction of atoms (without labeled nulls) whose variables are exactly $\mathbf{x}$, and $x_i$ and $x_j$ are variables from $\mathbf{x}$. We call $\varphi(\mathbf{x})$ the *body* of $\sigma$, and call $x_i = x_j$ the *head* of $\sigma$. We will omit the universal quantification in front of dependencies and assume that all variables are universally quantified. With a slight abuse of notation, we sometimes treat a conjunction as the *set* of its atoms. An instance $I$ *satisfies* $\sigma$, denoted $I \models \sigma$, if whenever there exists a homomorphism $h$ s.t. $h(\varphi(\mathbf{x})) \subseteq I$, then $h(x_i) = h(x_j)$. A instance $I$ *satisfies* a set $\Sigma$ of EGDs, denoted $I \models \Sigma$, if $I \models \sigma$ for every $\sigma \in \Sigma$.

A *knowledge base* (KB) is a pair $(D, \Sigma)$, where $D$ is a database and $\Sigma$ is a finite set of EGDs. It is *consistent* if $D \models \Sigma$, otherwise it is *inconsistent*.

**Query language.**
The query language we consider is non-recursive safe Datalog with negation. This choice will ease presentation of our approximation algorithm.

A *(positive) rule* is of the form $p(\mathbf{x}) \leftarrow \phi(\mathbf{x}, \mathbf{y}), \gamma(\mathbf{z})$, where there are no nulls, $p(\mathbf{x})$ is an atom (whose variables are $\mathbf{x}$), $\phi(\mathbf{x}, \mathbf{y})$ is a conjunction of atoms (whose variables are $\mathbf{x}$ and $\mathbf{y}$), $\gamma(\mathbf{z})$ is a conjunction of expressions (whose variables are $\mathbf{z}$) of the form $t = t'$ or $t \neq t'$ with $t, t' \in \mathsf{Const} \cup \mathsf{Var}$, and $\mathbf{z} \subseteq \mathbf{x} \cup \mathbf{y}$. W.l.o.g. we assume that there cannot be multiple occurrences of the same variable in $\phi(\mathbf{x}, \mathbf{y})$—so equalities must be made explicit in $\gamma(\mathbf{z})$.

A *(negative) rule* is of the form $p(\mathbf{x}) \leftarrow p'(\mathbf{x}), \neg p''(\mathbf{x})$, where $p(\mathbf{x})$, $p'(\mathbf{x})$, and $p''(\mathbf{x})$ are atoms without nulls. W.l.o.g. we assume that there cannot be multiple occurrences of the same variable in $p'(\mathbf{x})$.

In the rules above, $p(\mathbf{x})$ is called the *head* of the rule, and $\phi(\mathbf{x}, \mathbf{y}), \gamma(\mathbf{z})$ (resp. $p'(\mathbf{x}), \neg p''(\mathbf{x})$) is called the *body*.

A *program* $P$ is a finite set of rules. The *predicate graph* of $P$ is a directed graph whose vertices are the predicates appearing in $P$, and there is a directed edge from a predicate $p'$ to a predicate $p$ if there is a rule in $P$ where $p'$ appears in the body and $p$ appears in the head. As we consider non-recursive Datalog, the predicates of $P$ can be ordered into a sequence $\langle p_1, \dots, p_n \rangle$ according to a topological sorting of the predicate graph. Given a predicate $p$ appearing in $P$, we use $P[p]$ to denote the set of all rules in $P$ having $p$ in the head.

The *immediate consequence operator* $T_P$ of a program $P$ is defined as follows. For every database $D$,

$$\begin{aligned} T_P(D) = D \cup \{\nu(p(\mathbf{x})) \mid\ & \text{there exist a valuation } \nu \text{ and} \\ & \text{a rule } p(\mathbf{x}) \leftarrow \phi(\mathbf{x}, \mathbf{y}), \gamma(\mathbf{z}) \text{ of } P \text{ s.t.} \\ & \nu(\phi(\mathbf{x}, \mathbf{y})) \subseteq D \text{ and } \nu \models \gamma(\mathbf{z})\} \cup \\ \{\nu(p(\mathbf{x})) \mid\ & \text{there exist a valuation } \nu \text{ and} \\ & \text{a rule } p(\mathbf{x}) \leftarrow p'(\mathbf{x}), \neg p''(\mathbf{x}) \text{ of } P \text{ s.t.} \\ & \nu(p'(\mathbf{x})) \in D \text{ and } \nu(p''(\mathbf{x})) \notin D\}. \end{aligned}$$

Given a database $D$ and a program $P$, we define:

$$\begin{aligned} & T_P^0(D) = D, \\ & T_P^i(D) = T_{P[p_i]}(T_P^{i-1}(D)), \quad \text{for } 1 \le i \le n, \\ & \mathsf{CN}(P, D) = T_P^n(D). \end{aligned}$$

A *query* $Q$ is a pair $(P, p)$, where $P$ is a program and $p$ is a predicate. The result of evaluating $Q$ on a database $D$ is $Q(D) = \{p(\mathbf{t}) \mid p(\mathbf{t}) \in \mathsf{CN}(P, D)\}$.

## 4 Repairing and Querying Inconsistent KBs

In this section, we present our notion of repair and analyze the computational complexity of two central problems: repair checking and consistent query answering.

We start by defining the EGDs we consider. Let $\Sigma$ be a set of EGDs. An *argument* of $\Sigma$ is an expression of the form $p[i]$, where $p$ is an $n$-ary predicate appearing in $\Sigma$ and $1 \le i \le n$.

**Definition 1** (Argument and dependency graphs)**.** *The* argument graph *of a set $\Sigma$ of EGDs is a directed graph $G_\Sigma = (V, E)$, where $V$ is the set of all arguments of $\Sigma$, and $E$ contains a directed edge from $p[i]$ to $q[j]$ labeled $\sigma$ iff there is an EGD $\sigma \in \Sigma$ such that:*

- *the body of $\sigma$ contains an atom $p(t_1, \dots, t_n)$ such that either $t_i$ is a constant or $t_i$ is a variable occurring more than once in the body of $\sigma$, and*
- *the body of $\sigma$ contains an atom $q(u_1, \dots, u_m)$ such that $u_j$ is a variable also appearing in the head of $\sigma$.*

*The* dependency graph *of $\Sigma$ is a directed graph $\Gamma_\Sigma = (\Sigma, \Omega)$, where $\Omega$ is the following set:*

$$\{(\sigma_1, \sigma_2) \mid G_\Sigma = (V, E) \wedge (p([i], q[j], \sigma_1), (q[j], r[k], \sigma_2) \in E\}.$$

*We say that $\Sigma$ is* acyclic *if its dependency graph is acyclic.*

In the following, we write $\sigma_i < \sigma_j$ if $(\sigma_i, \sigma_j) \in \Omega$ or there is $\sigma_k$ such that $\sigma_i < \sigma_k$ and $\sigma_k < \sigma_j$.

**Example 5.** Consider the following set of EGDs $\Sigma$:

$$\sigma_1\colon\ \mathsf{works}(\mathsf{X}, \mathsf{Y}_1, \mathsf{Z}_1) \wedge \mathsf{works}(\mathsf{X}, \mathsf{Y}_2, \mathsf{Z}_2) \to \mathsf{Y}_1 = \mathsf{Y}_2$$
$$\sigma_2\colon\ \mathsf{works}(\mathsf{X}_1, \mathsf{Y}, \mathsf{Z}_1) \wedge \mathsf{works}(\mathsf{X}_2, \mathsf{Y}, \mathsf{Z}_2) \to \mathsf{Z}_1 = \mathsf{Z}_2$$

$G_\Sigma$ has two edges: one from works[1] to works[2] labeled $\sigma_1$, and another one from works[2] to works[3] labeled $\sigma_2$. Then, $\Gamma_\Sigma$ has only the edge $(\sigma_1, \sigma_2)$. Thus, $\Sigma$ is acyclic.

In the rest of the paper we consider acyclic sets of EGDs—so, from now on, $\Sigma$ is understood to be acyclic. Even if the tools developed in the paper can be generalized to arbitrary EGDs, we focus on acyclic sets because for them we can easily compute a compact representation of all repairs in polynomial time. This is particularly important for our purpose of developing polynomial time approximation algorithms.

Before introducing our notion of repair, we provide some intuitions. As already mentioned, it is based on updating values within facts. Specifically, we adopt a chase-like procedure that acts as follows: whenever an EGD $\varphi(\mathbf{x}) \to x_i = x_j$ is not satisfied by a set of facts $\nu(\varphi(\mathbf{x}))$ (for some valuation $\nu$), and thus $\nu(x_i) \ne \nu(x_j)$, then either $\nu(x_i)$ replaces $\nu(x_j)$ or vice versa. A repair is obtained through an exhaustive application of this repair step guaranteeing that the set of changes is minimal. Note that there is a non-deterministic choice to be made when updating values, and this may lead to multiple repairs.

**Example 6.** Consider the database below and the set of EGDs of Example 5.

| | | |
|---|---|---|
| john | cs | rome |
| john | math | rome |
| mary | math | sydney |

By enforcing $\sigma_1$ into the first two facts, either cs or math can be chosen as john's department. If the latter is chosen, then the database $D'$ below is obtained.

Suppose now that $\sigma_2$ is enforced into the last two facts of $D'$. Then, either rome or sydney can be chosen as math's city. If the former is chosen, then the database $D''$ below is obtained.

$D' =$

| | | |
|---|---|---|
| john | math | rome |
| john | math | rome |
| mary | math | sydney |

$D'' =$

| | | |
|---|---|---|
| john | math | rome |
| john | math | rome |
| mary | math | rome |

No further dependency enforcement is applicable at this point and thus $D''$ is a repair.

Notice that it might be possible to restore consistency in other different ways. For instance, in the first step of the example above, one may modify the employee names. However, we do not consider this option because it is unclear which (different) values should be assigned (any constant in Const is a candidate value). For instance, john in the first fact might be replaced with rome, but this is somewhat arbitrary and indeed does not make much sense. In contrast, our repair strategy chooses candidate values that are in sense "justified" by the content of the database (e.g., in the example above, john works for either the cs or the math department). Moreover, when EGDs are key dependencies, the aforementioned way of restoring consistency may lead to the introduction of entities that are not meaningful. Indeed, our choice has been made by different approaches relying on value updates (e.g., [Bohannon *et al.*, 2005]).

The computation of repairs as informally described above is quite involved in that, when applying a repair step, we have to check that the current database has not been previously obtained (to avoid infinite repair steps) and check minimality of the changes performed. In the following, we show how to deal with such issues and compute repairs efficiently.

When repairing inconsistent databases, conditional instances can be used to keep track of which values must be equal and which constants can be assigned to nulls. For instance, in Example 6 above, after the last dependency enforcement, we have to make sure that the last column contains the same city (which can be either rome or sydney).

Below we introduce the technical definitions, starting with the notion of a *repair step*.

**Definition 2** (Repair step)**.** *Let $\langle I, \Phi\rangle$ be a CI, $\Sigma$ a set of EGDs, and $\sigma$ an EGD $\varphi(\mathbf{x}) \to x_i = x_j \in \Sigma$. Let $h$ be a homomorphism s.t. $h(\varphi(\mathbf{x})) \subseteq h(I)$, $h \vDash \Phi$, and $h(x_i) \ne h(x_j)$.*

*Moreover, let $\perp_m$ be a fresh null and $\langle I', \Phi'\rangle$ the CI obtained from $\langle I, \Phi\rangle$ as follows:*

1. *for each fact $p(u_1, ..., u_n)$ in $I$, if $\varphi(\mathbf{x})$ contains an atom $p(t_1, ..., t_n)$ s.t. $h(p(t_1, ..., t_n)) = h(p(u_1, ..., u_n))$, then for every $1 \le k \le n$, if $t_k \in \{x_i, x_j\}$,*
   - *replace $u_k$ in $p(u_1, ..., u_n)$ with $\perp_m$;*
   - *if $u_k \in$ Null, replace every occurrence of $u_k$ elsewhere with $\perp_m$;*
2. *Either $\Phi' = \Phi \wedge (\perp_m = h(x_i))$ or $\Phi' = \Phi \wedge (\perp_m = h(x_j))$.*

*We say that $\langle I, \Phi\rangle \xrightarrow{\sigma,h} \langle I', \Phi'\rangle$ is a* repair step.

Intuitively, a repair step enforces an EGD that is not satisfied by the knowledge base.

A *repair sequence* of a knowledge base $(D, \Sigma)$ is a (possibly empty) finite sequence of repair steps $C_i \xrightarrow{\sigma_i, h_i} C_{i+1}$ $(0 \leq i < m)$ such that $C_0 = \langle D, \mathsf{true}\rangle$ and $\sigma_i \in \Sigma$ for every $i$. We also say that the repair sequence is *from* $C_0$ *to* $C_m$. We call $C_m$ the *result* of the repair sequence. Also, we say that the repair sequence is *maximal* if there does not exist a repair step of the form $C_m \xrightarrow{\sigma_m, h_m} C_{m+1}$.

It is easy to see that for each repair step $\langle I, \Phi\rangle \xrightarrow{\sigma, h} \langle I', \Phi'\rangle$, if $h'$ is a homomorphism s.t. $h' \models \Phi'$, then it assigns constants to nulls as dictated by $\Phi'$ (see item 2 of Definition 2). Thus, all $h'$ satisfying $\Phi'$ yield the same database $h'(I')$. For a conditional instance $(I, \Phi)$ obtained by means of a repair sequence, we use the notation $\Phi(I)$ to denote the database derived from $I$ by iteratively replacing nulls with constants or other nulls as dictated by $\Phi$.

For ease of presentation, we assume that one can keep of a given fact $f$ in the database during the repair process despite the value changes. Thus, we use $D[f, i]$ to denote the $i$-th term of a fact $f$ in a database $D$. Given a repair sequence $S$ from $C_0 = (D, \mathsf{true})$ to $C_m = (I_m, \Phi_m)$, we define the set of changes made by $S$ as $update(S) = \{(f, i) \mid D[f, i] \neq \Phi_m(I_m)[f, i]\}$, where $f$ is a fact of the database.

**Example 7.** Consider the database of Example 6 and the set of EGDs of Example 5. By enforcing first $\sigma_1$ using the first two facts, then $\sigma_2$ using the last two facts, and finally $\sigma_2$ using the first two facts, we get, respectively, the CIs $C_1$, $C_2$ and $C_3$ reported below:

$C_1$

| | | |
|---|---|---|
| john | $\perp_1$ | rome |
| john | $\perp_1$ | rome |
| mary | math | sydney |

$\perp_1 = \mathsf{math}$

$C_2$

| | | |
|---|---|---|
| john | $\perp_1$ | rome |
| john | $\perp_1$ | $\perp_3$ |
| mary | math | $\perp_3$ |

$\perp_1 = \mathsf{math} \wedge$
$\perp_3 = \mathsf{sydney}$

$C_3$

| | | |
|---|---|---|
| john | $\perp_1$ | $\perp_4$ |
| john | $\perp_1$ | $\perp_4$ |
| mary | math | $\perp_4$ |

$\perp_1 = \mathsf{math} \wedge$
$\perp_3 = \mathsf{sydney} \wedge$
$\perp_4 = \mathsf{rome}$

No further repair steps are applicable at this point. Notice that $\perp_3 = \mathsf{sydney}$ can be deleted from the condition of $C_3$ because it does not play a role anymore. A repair is obtained from $C_3$ by replacing every occurrence of $\perp_1$ with math, and every occurrence of $\perp_4$ with rome.

**Definition 3** (Repair)**.** *A* repair *for a knowledge base* $K = \langle D, \Sigma\rangle$ *is a database* $\Phi(J)$ *such that there exists a maximal repair sequence* $S$ *of* $K$ *from* $\langle D, \mathsf{true}\rangle$ *to* $\langle J, \Phi\rangle$ *and* $update(S)$ *is minimal w.r.t.* $\subseteq$.

We use $repair(D, \Sigma)$ to denote the set of all repairs of a knowledge base $(D, \Sigma)$. The following simple proposition states that every repair is indeed consistent.

**Proposition 1.** *Let* $(D, \Sigma)$ *be a knowledge base. For every* $D' \in repair(D, \Sigma)$, $D' \models \Sigma$.

The consistent answers to a query are defined in the standard way as follows.

**Definition 4** (Consistent Query Answers)**.** *Let* $(D, \Sigma)$ *be a knowledge base and* $Q$ *a query. The* consistent answers *to* $Q$ *over* $(D, \Sigma)$ *are*

$$Q(D, \Sigma) = \bigcap\{Q(D') \mid D' \in repair(D, \Sigma)\}.$$

In the context of managing inconsistent knowledge bases, two fundamental problems are *repair checking* and *consistent query answering*, which are defined as follows. Let $(D, \Sigma)$ be a knowledge base, $D'$ a database, $Q$ a query, and $f$ a fact of constants. Then, the following two problems are defined:

- *repair checking*: decide whether $D' \in repair(D, \Sigma)$;
- *consistent query answering*: decide whether $f \in Q(D, \Sigma)$.

Below we show that the first problem is in PTIME and the second one is coNP-complete All the complexity results in this paper are in the data complexity, that is, the query and the EGDs are fixed.

**Theorem 2.** *Repair checking is in PTIME.*

*Proof. (Sketch)* Let $(D, \Sigma)$ be a knowledge base and $D'$ a database. As stated in Theorems 4 and 5 reported in the next section, it is possible to compute in polynomial time a CI $C = \langle I, \Phi\rangle$ such that $pw(C) = repair(D, \Sigma)$. After $C$ is computed, for each null $\perp_i$ in $I$, replace every occurrence of $\perp_i$ in $I$ and $\Phi$ with the corresponding constant in $D'$—if there are different constants corresponding to (different occurrences of) $\perp_i$, then answer 'no'. After that, if the updated instance $I$ is equal to $D'$ and the updated condition $\Phi$ is true, then answer 'yes', otherwise answer 'no'. Clearly, the entire process can be carried out in polynomial time. $\square$

**Theorem 3.** *Consistent query answering is coNP-complete.*

*Proof. (Sketch) Membership.* Let $(D, \Sigma)$ be a knowledge base, $Q$ a query, and $f$ a fact of constants. The complementary problem is in NP. In fact, we can guess a database $D'$ and verify that (i) $D' \in repair(D, \Sigma)$ and (ii) $f \notin Q(D')$. By the definition of repair, the size of $D'$ is linear in the size of $D$. By Theorem 2, condition (i) can be verified in polynomial time, and obviously condition (ii) too.

*Hardness.* We reduce 3DNF TAUTOLOGY to our problem. Let $\phi$ be a 3DNF formula $d_1 \vee \ldots \vee d_m$ over variables $X = \{x_1, \ldots, x_n\}$. For every literal $\ell$ in $\phi$, we define $var(\ell) = \mathsf{x_i}$ if $\ell = x_i$ or $\ell = \neg x_i$, $tval(\ell) = \mathsf{t}$ if $\ell = x_i$, and $tval(\ell) = \mathsf{f}$ if $\ell = \neg x_i$, where $\mathsf{t}$, $\mathsf{f}$, and the $\mathsf{x_i}$'s are constants.

We define $D_\phi$ as follows. For each disjunct $d_i = (\ell_1 \wedge \ell_2 \wedge \ell_3)$, $1 \leq i \leq m$, $D_\phi$ contains the following facts:

$$\mathsf{disj}(\mathsf{i}, var(\ell_1), \mathsf{t}, var(\ell_2), \mathsf{t}, var(\ell_3), \mathsf{t}),$$
$$\mathsf{disj}(\mathsf{i}, var(\ell_1), \mathsf{f}, var(\ell_2), \mathsf{f}, var(\ell_3), \mathsf{f}),$$
$$\mathsf{disjsat}(\mathsf{i}, var(\ell_1), tval(\ell_1), var(\ell_2), tval(\ell_2), var(\ell_3), tval(\ell_3)).$$

Then, $\Sigma$ contains the following EGDs:

$$\begin{aligned}
&\mathsf{disj}(\mathsf{I}, \mathsf{V}, \mathsf{A_1}, \mathsf{V_2}, \mathsf{A_2}, \mathsf{V_3}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V}, \mathsf{A'_1}, \mathsf{V'_2}, \mathsf{A'_2}, \mathsf{V'_3}, \mathsf{A'_3}) \rightarrow \mathsf{A_1} = \mathsf{A'_1} \\
&\mathsf{disj}(\mathsf{I}, \mathsf{V}, \mathsf{A_1}, \mathsf{V_2}, \mathsf{A_2}, \mathsf{V_3}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V'_1}, \mathsf{A'_1}, \mathsf{V}, \mathsf{A'_2}, \mathsf{V'_3}, \mathsf{A'_3}) \rightarrow \mathsf{A_1} = \mathsf{A'_2} \\
&\mathsf{disj}(\mathsf{I}, \mathsf{V}, \mathsf{A_1}, \mathsf{V_2}, \mathsf{A_2}, \mathsf{V_3}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V'_1}, \mathsf{A'_1}, \mathsf{V'_2}, \mathsf{A'_2}, \mathsf{V}, \mathsf{A'_3}) \rightarrow \mathsf{A_1} = \mathsf{A'_3} \\
&\mathsf{disj}(\mathsf{I}, \mathsf{V_1}, \mathsf{A_1}, \mathsf{V}, \mathsf{A_2}, \mathsf{V_3}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V'_1}, \mathsf{A'_1}, \mathsf{V}, \mathsf{A'_2}, \mathsf{V'_3}, \mathsf{A'_3}) \rightarrow \mathsf{A_2} = \mathsf{A'_2} \\
&\mathsf{disj}(\mathsf{I}, \mathsf{V_1}, \mathsf{A_1}, \mathsf{V}, \mathsf{A_2}, \mathsf{V_3}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V'_1}, \mathsf{A'_1}, \mathsf{V'_2}, \mathsf{A'_2}, \mathsf{V}, \mathsf{A'_3}) \rightarrow \mathsf{A_2} = \mathsf{A'_3} \\
&\mathsf{disj}(\mathsf{I}, \mathsf{V_1}, \mathsf{A_1}, \mathsf{V_2}, \mathsf{A_2}, \mathsf{V}, \mathsf{A_3}) \wedge \\
&\quad \mathsf{disj}(\mathsf{I'}, \mathsf{V'_1}, \mathsf{A'_1}, \mathsf{V'_2}, \mathsf{A'_2}, \mathsf{V}, \mathsf{A'_3}) \rightarrow \mathsf{A_3} = \mathsf{A'_3}
\end{aligned}$$

It can be easily verified that $\Sigma$ is acyclic.

Finally, $Q = (P, \mathsf{sat})$, where $P$ contains the following rule:

$$\begin{aligned}\mathsf{sat}() \leftarrow\ & \mathsf{disjsat}(\mathsf{I}, \mathsf{V}_1, \mathsf{A}_1, \mathsf{V}_2, \mathsf{A}_2, \mathsf{V}_3, \mathsf{A}_3),\\ & \mathsf{disj}(\mathsf{I}, \mathsf{V}_1, \mathsf{A}_1, \mathsf{V}_2, \mathsf{A}_2, \mathsf{V}_3, \mathsf{A}_3).\end{aligned}$$

It can be shown that $\phi$ is a tautology iff $\mathsf{sat}() \in Q(D_\phi, \Sigma)$. □

Indeed, coNP-hardness holds even for Boolean conjunctive queries. In the next section we introduce the notion of a *universal repair*, which is a compact representation of all repairs for a given knowledge base, and can be computed in polynomial time.

## 5 Universal Repair

In this section, we show how to compute a conditional instance $U$, called *universal repair*, that represents all repairs of a knowledge base, that is, the possible worlds of $U$ are exactly the repairs. As shown in the following, such a representation will prove to be a valuable tool for query answering.

Roughly speaking, we generalize the repair step introduced in the previous section in order to keep track of all possible choices, and when each choice should be applied.

Any conjunction of atoms $\varphi(\mathbf{x})$ in the body of an EGD can be rewritten as follows:

- every occurrence of a constant $c$ in $\varphi(\mathbf{x})$ is replaced with a fresh variable $z$ and $(z = c)$ is added to the conjunction;
- for every variable occurring $n > 1$ times in $\varphi(\mathbf{x})$, replace $n-1$ occurrences with $n-1$ fresh variables $x_2, ..., x_n$ and add $(x = x_2) \wedge ... \wedge (x = x_n)$ to the conjunction.

As an example, $r(x, y, x) \wedge s(x, a, a)$, where $a$ is a constant, can be rewritten into $r(x, y, x_2) \wedge s(x_3, z_a, z'_a) \wedge (x = x_2) \wedge (x = x_3) \wedge (z_a = a) \wedge (z'_a = a)$. In the following definition, we assume that the body of every EGD has been rewritten as above, and thus it is of the form $\varphi(\mathbf{x}) \wedge eq(\mathbf{x}')$, where $\varphi(\mathbf{x})$ is a conjunction of atoms (with no constants and no multiple occurrences of the same variable) and $eq(\mathbf{x}')$ is a conjunction of equalities over variables $\mathbf{x}' \subseteq \mathbf{x}$.

**Definition 5** (Universal repair step). *Let $C = \langle I, \Phi\rangle$ be a CI and $\sigma$ an EGD $\varphi(\mathbf{x}) \wedge eq(\mathbf{x}') \to x_i = x_j$. Let $h$ be a homomorphism s.t. $h(\varphi(\mathbf{x})) \subseteq I$ and for which there exists a valuation $\nu$ s.t. $\nu \vDash h(eq(\mathbf{x}'))$, $\nu \vDash \Phi$, and $\nu(h(x_i)) \neq \nu(h(x_j))$.*

*Moreover, let $\bot_\ell$ and $\bot_m$ be two fresh nulls, and $\langle I', \Phi'\rangle$ be the CI obtained from $\langle I, \Phi\rangle$ as follows:*

1. *for each fact $p(u_1, ..., u_n)$ in $I$, if $\varphi(\mathbf{x})$ contains an atom $p(t_1, ..., t_n)$ s.t. $\nu(h(p(t_1, ..., t_n))) = \nu(p(u_1, ..., u_n))$, then for every $1 \le k \le n$, if $t_k = x_i$ or $(t_k = x_i)$ appears in $eq(\mathbf{x}')$ (resp. $t_k = x_j$ or $(t_k = x_j)$ appears in $eq(\mathbf{x}')$), then replace $u_k$ in $p(u_1, ..., u_n)$ with $\bot_\ell$ (resp. $\bot_m$);*
2. *$\Phi' = \Phi\ \wedge \phi$, where $\phi$ is defined as follows:*

$$\begin{aligned}&(h(eq(\mathbf{x}')) \Rightarrow ((\bot_m = \bot_\ell) \wedge (\bot_\ell = h(x_i) \vee \bot_m = h(x_j)))) \wedge\\ &(\neg(h(eq(\mathbf{x}'))) \Rightarrow (\bot_\ell = h(x_i) \wedge \bot_m = h(x_j)));\end{aligned}$$

3. *for each $u_k$ of Item 1 above, if $u_k \in$ Null, replace every occurrence of $u_k$ with $\bot_\ell$ (resp. $\bot_m$) in both the current instance and condition $\Phi'$.*

*We say that $\langle I, \Phi\rangle \overset{\sigma,h}{\longmapsto} \langle I', \Phi'\rangle$ is a* universal repair step.

In the previous definition, $h$ is a homomorphism used to map $\varphi(\mathbf{x})$ into $I$, while $\nu$ is a valuation used to check if $h(\varphi(\mathbf{x}))$ may violate $\sigma$, that is, $\nu(h(\varphi(\mathbf{x})))$ are admissible facts (cf. $\nu \vDash \Phi$) that satisfy the join conditions in the body of $\sigma$ (cf. $\nu \vDash h(eq(\mathbf{x}'))$) but do not satisfy the head condition (cf. $\nu(h(x_i)) \neq \nu(h(x_j))$).

A *universal repair sequence* of a knowledge base $(D, \Sigma)$ is a (possibly empty) finite sequence of universal repair steps $C_i \overset{\sigma_i,h_i}{\longmapsto} C_{i+1}$ $(0 \le i < m)$ such that $C_0 = \langle D, \mathsf{true}\rangle$ and $\sigma_i \in \Sigma$ for every $i$. We call $C_m$ the *result* of the universal repair sequence. Also, we say that the repair sequence is *maximal* if there does not exist a universal repair step of the form $C_m \overset{\sigma_m,h_m}{\longmapsto} C_{m+1}$. A universal repair sequence is said to be *ordered* if for each $C_i \overset{\sigma_i,h_j}{\longrightarrow} C_{i+1}$ there is no $C_j \overset{\sigma_j,h_i}{\longrightarrow} C_{j+1}$ such that $j > i$ and $\sigma_j < \sigma_i$ (intuitively, EGDs in $\Sigma$ must be considered according to a topological sorting of $\Gamma_\Sigma$). If $C$ is the result of an ordered maximal universal repair sequence, then $C$ is called a *universal repair*.

**Example 8.** Consider the database of Example 6 and the set of EGDs of Example 5. After the application of $\sigma_1$ to the first two facts we get the following conditional instance:

| | | |
|---|---|---|
| john | $\bot_1$ | rome |
| john | $\bot_2$ | rome |
| mary | math | sydney |

$\Phi_1$

where $\Phi_1$ is as follows:

$$\begin{aligned}&((\mathsf{john} = \mathsf{john}) \Rightarrow ((\bot_1 = \bot_2) \wedge (\bot_1 = \mathsf{cs} \vee \bot_2 = \mathsf{math}))) \wedge\\ &((\mathsf{john} \neq \mathsf{john}) \Rightarrow (\bot_1 = \mathsf{cs} \wedge \bot_2 = \mathsf{math}))\end{aligned}$$

By applying $\sigma_2$ to the first and third facts, we get:

| | | |
|---|---|---|
| john | $\bot_1$ | $\bot_3$ |
| john | $\bot_2$ | rome |
| mary | math | $\bot_4$ |

$\Phi_1 \wedge \Phi_2$

where $\Phi_2$ is as follows:

$$\begin{aligned}&(\bot_1 = \mathsf{math} \Rightarrow ((\bot_3 = \bot_4) \wedge (\bot_3 = \mathsf{rome} \vee \bot_4 = \mathsf{sydney}))) \wedge\\ &(\bot_1 \neq \mathsf{math} \Rightarrow (\bot_3 = \mathsf{rome} \wedge \bot_4 = \mathsf{sydney}))\end{aligned}$$

By applying $\sigma_2$ to the first two facts, we get:

| | | |
|---|---|---|
| john | $\bot_1$ | $\bot_5$ |
| john | $\bot_2$ | $\bot_6$ |
| mary | math | $\bot_4$ |

$\Phi_1 \wedge \Phi_2 \wedge \Phi_3$

where $\Phi_3$ is as follows:

$$\begin{aligned}&((\bot_1 = \bot_2) \Rightarrow ((\bot_5 = \bot_6) \wedge (\bot_5 = \bot_3 \vee \bot_6 = \mathsf{rome}))) \wedge\\ &((\bot_1 \neq \bot_2) \Rightarrow (\bot_5 = \bot_3 \wedge \bot_6 = \mathsf{rome}))\end{aligned}$$

Then, $\bot_3$ is replaced with $\bot_5$ everywhere, that is, in both $\Phi_2$ and $\Phi_3$, yielding new conditions $\Phi'_2$ and $\Phi'_3$. The result of this step is the CI consisting of the instance above and the condition $\Phi_1 \wedge \Phi'_2 \wedge \Phi'_3$.

As no further universal repair step can be applied, the conditional instance above is a universal repair.

The following theorems say that all the universal repairs of a knowledge base are equivalent in that they "represent" its repairs and computing one of them can be done in polynomial time.

**Theorem 4.** *Let $U$ be a universal repair of a knowledge base $(D, \Sigma)$. Then, $pw(U) = repair(D, \Sigma)$.*

**Theorem 5.** *Computing a universal repair is in PTIME.*

The main idea behind the proof of the previous theorem is to apply universal repair steps according to a topological sorting of the dependency graph of $\Sigma$.

## 6 Query Evaluation

In this section, we show how to evaluate queries over universal repairs and keep track of provenance information. This will be used later on to develop an approximation algorithm. In order to do that, we need to extend conditional instances to allow individual conditions for facts too.

A *conditional fact* is a pair $\langle p(\mathbf{t}), \phi\rangle$, where $p(\mathbf{t})$ is a fact and $\phi$ is a condition, also called *local condition*.

An *extended conditional instance* (ECI) is a pair $\langle E, \Phi\rangle$, where $E$ is a finite set of conditional facts and $\Phi$ is a condition, which we also call *global condition*. Given a valuation $\nu$, we define $\nu(E) = \{\nu(p(\mathbf{t})) \mid \langle p(\mathbf{t}), \phi\rangle \in E \text{ and } \nu \models \phi\}$. Thus, $\nu(E)$ is the database obtained from $E$ by applying $\nu$ to every fact and keeping only the facts whose condition is satisfied by $\nu$. The *possible worlds* of an ECI $G = \langle E, \Phi\rangle$ are $pw(G) = \{\nu(E) \mid \nu \text{ is a valuation s.t. } \nu \models \Phi\}$.

We now define how to evaluate queries over ECIs. To this end, we generalize the immediate consequence operator $T_P$ (cf. Section 3) so as to perform a "conditional" evaluation of a query in the presence of conditional facts. We point out that our conditional evaluation is a slight variant of the one for conditional tables [Grahne, 1991]. Below we use $(t_1, ..., t_n) = (u_1, ..., u_n)$ as a shorthand for $\bigwedge_{i=1}^{n}(t_i = u_i)$.

With a slight abuse of notation, given a program $P$ we overload $T_P$ so that when its input is a set $E$ of conditional facts, then $T_P(E) = E \cup E'$, where $E'$ is the following set of conditional facts:

$$\begin{array}{ll} \{\langle h(p(\mathbf{x})), \phi\rangle \mid & \text{there exist a homomorphism } h \text{ and} \\ & \text{a rule } p(\mathbf{x}) \leftarrow \bigwedge_{i=1}^{n} p_i(\mathbf{x}_i, \mathbf{y}_i), \gamma(\mathbf{z}) \text{ of } P \text{ and} \\ & \phi_1, \dots, \phi_n \text{ s.t.} \\ & \langle h(p_i(\mathbf{x}_i, \mathbf{y}_i)), \phi_i\rangle \in E \text{ for every } 1 \le i \le n \text{ and} \\ & \phi = h(\gamma(\mathbf{z})) \wedge \bigwedge_{i=1}^{n} \phi_i\} \cup \\ \{\langle h(p(\mathbf{x})), \phi\rangle \mid & \text{there exist a homomorphism } h \text{ and} \\ & \text{a rule } p(\mathbf{x}) \leftarrow p'(\mathbf{x}), \neg p''(\mathbf{x}) \text{ of } P \text{ and } \phi' \text{ s.t.} \\ & \langle h(p'(\mathbf{x})), \phi'\rangle \in E \text{ and} \\ & \phi = \phi' \wedge \bigwedge_{\langle p''(\mathbf{t}), \phi''\rangle \in E} \neg(\phi'' \wedge h(\mathbf{x}) = \mathbf{t})\}. \end{array}$$

Then, $\mathsf{CN}(P, E) = T_P^n(E)$, where $T_P^i$ is defined in exactly the same way as in Section 3.

It can be easily verified that the conditional evaluation above can be carried out in polynomial time. The main reason is that the immediate consequence operator $T_P$ above extends the classical one by just adding conditions of polynomial size to facts, so the time complexity remains polynomial.

**Proposition 6.** *Given a program $P$ and a finite set $E$ of conditional facts, computing $\mathsf{CN}(P, E)$ is in PTIME.*

Given two conditions $\phi, \phi'$, we write $\phi \models \phi'$ if for every valuation $\nu$, if $\nu \models \phi$, then $\nu \models \phi'$. Below we define the *certain answers* to a query over an extended conditional instance.

**Definition 6.** *Let $G = \langle E, \Phi\rangle$ be an ECI and $Q = (P, p)$ be a query. The* certain answers *to $Q$ over $G$ are defined as follows:*

$$\begin{aligned} cert(Q, G) = \{p(\mathbf{t}) \mid \langle p(\mathbf{t}), \phi\rangle \in \mathsf{CN}(P, E) \textit{ and} \\ \Phi \models \phi \textit{ and } \mathbf{t} \textit{ has constants only}\}. \end{aligned}$$

Thus, to compute the certain answers to $Q$ over $G$ we need to compute $\mathsf{CN}(P, E)$ and check for which conditional facts $\Phi \models \phi$ holds true. The last entailment checking can be carried out by rewriting the formula as a propositional formula and using SAT solvers [Gomes *et al.*, 2008]. Satisfiability solvers are becoming increasingly efficient and effective in solving large satisfiability problems and, despite the worst-case exponential run time of all known algorithms, they are very efficient in solving hard problems with millions of variables and tens of millions of constraints.

The tools developed thus far will be exploited in the next section to devise an approximation algorithm.

## 7 Approximation Algorithm

In light of Theorem 3, we develop a polynomial time approximation algorithm to compute a sound (but possibly incomplete) set of consistent query answers.

The basic idea of our approximation algorithm is as follows. Given a knowledge base and a query, we first compute a universal repair $U$ of the knowledge base, and then transform $U$ into an ECI $U_{\mathsf{true}}$ where each fact is associated with the trivial condition true. After that, the query is "conditionally" evaluated over $U_{\mathsf{true}}$ (i.e., as described in the previous section). The result is a set of conditional facts, whose conditions are evaluated together with the condition of $U_{\mathsf{true}}$ in order to determine a sound (but possibly incomplete) set of consistent query answers (we will illustrate the entire process in Example 9). For positive queries, a simple approximation can be obtained by deleting facts containing nulls from $U_{\mathsf{true}}$ and ignoring its global condition.

First, we need some auxiliary definitions. Given a condition $\Phi$ and two nulls $\bot_i, \bot_j \in \mathsf{Null}$, we write $\bot_i \approx_\Phi \bot_j$ if $\bot_i = \bot_j$ (or $\bot_j = \bot_i$) appears in $\Phi$. Then, $\approx_\Phi^*$ is the reflexive and transitive closure of $\approx_\Phi$. Given a term $t \in \mathsf{Const} \cup \mathsf{Null}$ and a condition $\Phi$, we denote by $\mathsf{dom}_\Phi(t)$ the *domain of $t$ w.r.t.* $\Phi$, defined as follows: $\mathsf{dom}_\Phi(c) = \{c\}$ if $c \in \mathsf{Const}$, whereas $\mathsf{dom}_\Phi(\bot_i) = \{c \mid \bot_i \approx_\Phi^* \bot_k \text{ and } \bot_k = c \text{ appears in } \Phi\}$ if $\bot_i \in \mathsf{Null}$. Intuitively, $\mathsf{dom}_\Phi(\bot_i)$ is a set of possible values that $\bot_i$ might assume in some repair.

We assume the strict ordering $\mathsf{false} < \mathsf{unknown} < \mathsf{true}$, and $\neg\mathsf{true} = \mathsf{false}$, $\neg\mathsf{false} = \mathsf{true}$, and $\neg\mathsf{unknown} = \mathsf{unknown}$. Given two conditions $\varphi$ and $\Phi$, we define $\mathsf{eval}_\Phi(\varphi)$ as follows:

- $\mathsf{eval}_\Phi((t_i = t_j)) = \begin{cases} \mathsf{true} & \text{if } t_i = t_j, \\ \mathsf{false} & \text{if } \mathsf{dom}_\Phi(t_i) \cap \mathsf{dom}_\Phi(t_j) = \varnothing, \\ \mathsf{unknown} & \text{otherwise.} \end{cases}$
- $\mathsf{eval}_\Phi((t_i \neq t_j)) = \neg\, \mathsf{eval}_\Phi((t_i = t_j))$
- $\mathsf{eval}_\Phi((\varphi_1 \wedge \varphi_2)) = \min\{\mathsf{eval}_\Phi(\varphi_1), \mathsf{eval}_\Phi(\varphi_2)\}$.

- $\mathsf{eval}_\Phi((\varphi_1 \vee \varphi_2)) = \max\{\mathsf{eval}_\Phi(\varphi_1), \mathsf{eval}_\Phi(\varphi_2)\}$.
- $\mathsf{eval}_\Phi((\neg\varphi)) = \neg\,\mathsf{eval}_\Phi(\varphi)$
- $\mathsf{eval}_\Phi(v) = v$ for $v \in \{\mathsf{true}, \mathsf{unknown}, \mathsf{false}\}$.

The following definition introduces the approximate evaluation of a query over an extended conditional instance.

**Definition 7.** *Let $G = \langle E, \Phi\rangle$ be an ECI and $Q = (P, p)$ be a query. The* approximate answers *to $Q$ over $G$ are:*

$$approx(Q, G) = \{p(\mathbf{t}) \mid \langle p(\mathbf{t}), \phi\rangle \in \mathsf{CN}(P, E) \textit{ and } \mathsf{eval}_\Phi(\phi) = \mathsf{true} \textit{ and } \mathbf{t} \textit{ has constants only}\}.$$

A conditional instance $C = \langle I, \Phi\rangle$ can be "lifted to" an ECI by setting all local conditions to true. We denote by $C_{\mathsf{true}}$ the ECI $\langle\{\langle p(\mathbf{t}), \mathsf{true}\rangle \mid p(\mathbf{t}) \in C\}, \Phi\rangle$.

The evaluation technique of Definition 7 indeed provides a way to compute a sound (but possibly incomplete) set of consistent query answers, in polynomial time.

**Definition 8.** *Let $(D, \Sigma)$ be a knowledge base and $Q$ a query. The* approximate consistent answers *to $Q$ over $(D, \Sigma)$ are $\widetilde{Q}(D, \Sigma) = approx(Q, U_{\mathsf{true}})$, where $U$ is a universal repair of $(D, \Sigma)$.*

**Example 9.** Consider the knowledge base of Example 1 and the query $Q = \langle P, \mathsf{diffCity}\rangle$, asking for the pairs of departments located in different cities, where $P$ consists of the rule:

$$\mathsf{diffCity}(\mathsf{D}, \mathsf{D}') \leftarrow \mathsf{works}(\mathsf{E}, \mathsf{D}, \mathsf{C}), \mathsf{works}(\mathsf{E}', \mathsf{D}', \mathsf{C}'), \mathsf{C} \neq \mathsf{C}'.$$

A universal repair, say $U$, is shown in Example 3. The first step of our approach consists of converting $U$ into an ECI $U_{\mathsf{true}}$ by setting all local conditions to true. Thus, $U_{\mathsf{true}}$ is as follows:

| | | | |
|---|---|---|---|
| john | cs | nyc | true |
| john | math | $\bot_1$ | true |
| mary | math | $\bot_1$ | true |

$\bot_1 = \mathsf{rome} \vee \bot_1 = \mathsf{sydney}$

The conditional evaluation of $Q$ over $U_{\mathsf{true}}$ gives the following set of conditional facts:

| | | |
|---|---|---|
| cs | cs | true ∧ true ∧ nyc ≠ nyc |
| cs | math | true ∧ true ∧ nyc ≠ $\bot_1$ |
| math | cs | true ∧ true ∧ $\bot_1$ ≠ nyc |
| math | math | true ∧ true ∧ $\bot_1 \neq \bot_1$ |

As $\mathsf{dom}_\Phi(\bot_1) = \{\mathsf{rome}, \mathsf{sydney}\}$, the approximate consistent answers are $\mathsf{diffCity}(\mathsf{cs}, \mathsf{math})$ and $\mathsf{diffCity}(\mathsf{math}, \mathsf{cs})$, which are indeed consistent query answers.

The following theorems state soundness of our approach and its complexity, which is in PTIME (data complexity).

**Theorem 7.** *Let $(D, \Sigma)$ be a knowledge base and $Q$ a query. Then, $\widetilde{Q}(D, \Sigma) \subseteq Q(D, \Sigma)$.*

**Theorem 8.** *Let $(D, \Sigma)$ be a knowledge base and $Q$ a query. Computing $\widetilde{Q}(D, \Sigma)$ is in PTIME.*

*Proof. (Sketch)* It follows from Theorem 5, Proposition 6, and because $\mathsf{eval}_\Phi(\varphi)$ can be evaluated in PTIME. □

We conclude by comparing our approach with two well-known approximation schemes that have been developed in the query answering inconsistency-tolerant area, namely the *intersection of repairs* (IAR) and the *intersection of closed repairs* (ICR) semantics [Lembo *et al.*, 2010; Bienvenu *et al.*, 2014]. The former semantics consists of querying the intersection of all repairs, while the second one consists of querying the intersection of the closure of all repairs. In our setting, since we do not consider tuple-generating dependencies, the two semantics coincide.

As stated in the following theorem, for every positive query (i.e., only positive rules and equalities are allowed), our approximation algorithm provides at least as many answers as IAR (and ICR).

**Theorem 9.** *Let $(D, \Sigma)$ be a knowledge base and $Q$ a positive query. Then, $Q(D^*) \subseteq \widetilde{Q}(D, \Sigma)$, where $D^* = \bigcap\{D' \mid D' \in repair(D, \Sigma)\}$.*

To show a simple case where our approximation algorithm yields strictly more consistent query answers, consider the knowledge base of Example 1, whose repairs are reported in Example 2. Consider also the query asking for the employees' names. Under the IAR semantics, the intersection keeps only the first fact of the original knowledge base, and thus the other two facts are lost altogether. So the only query answer is john. On the other hand, the universal repair (cf. Example 3) is much more informative: we still have the first fact of the original knowledge base, but we also have the other two facts, where consistent values are unchanged and thus preserved. Our approximation algorithm returns both john and mary.

As a further example, consider Example 6. The intersection of all repairs is empty. On the other hand, the universal repair (cf. Example 8) has valuable information, which again can be profitably exploited for (approximate) query answering.

## 8 Conclusion

We proposed a framework for query answering over inconsistent KBs based on *(i)* a notion of repair allowing values within facts to be updated, *(ii)* a compact representation of all repairs, *(iii)* an approximation algorithm to compute under-approximations of consistent query answers.

As a direction for future work, we plan to investigate further approximation algorithms based on different conditions' evaluations. Another issue to be investigated is the generalization of our framework to more general classes of dependencies, such as TGDs or arbitrary EGDs. One way of dealing with TGDs is by inserting new facts. Interestingly, by inserting facts with nulls (in a Chase-like manner) into a universal repair we can get a compact representation of all possible insertions. This also poses the issues of combining different repair primitives, such as fact updates and fact insertions (fact deletions might be considered as well).

## References

[Arenas *et al.*, 1999] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[Bertossi *et al.*, 2008] Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Information Systems*, 33(4-5):407–434, 2008.

[Bienvenu and Rosati, 2013] Meghyn Bienvenu and Riccardo Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 775–781, 2013.

[Bienvenu *et al.*, 2014] Meghyn Bienvenu, Camille Bourgaux, and Francois Goasdoué. Querying inconsistent description logic knowledge bases under preferred repair semantics. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 996–1002, 2014.

[Bienvenu, 2011] Meghyn Bienvenu. First-order expressibility results for queries over inconsistent DL-Lite knowledge bases. In *International Workshop on Description Logics (DL)*, 2011.

[Bienvenu, 2012] Meghyn Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2012.

[Bohannon *et al.*, 2005] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *International Conference on Management of Data (SIGMOD)*, pages 143–154, 2005.

[Fiorentino *et al.*, 2018] Nicola Fiorentino, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. ACID: A system for computing approximate certain query answers over incomplete databases. In *International Conference on Management of Data (SIGMOD)*, 2018. To appear.

[Flesca *et al.*, 2010] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. Querying and repairing inconsistent numerical databases. *ACM Transactions on Database Systems*, 35(2):14:1–14:50, 2010.

[Furfaro *et al.*, 2007] Filippo Furfaro, Sergio Greco, and Cristian Molinaro. A three-valued semantics for querying and repairing inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):167–193, 2007.

[Gomes *et al.*, 2008] Carla P. Gomes, Henry A. Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, pages 89–134. 2008.

[Grahne, 1991] Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.

[Greco and Molinaro, 2008] Sergio Greco and Cristian Molinaro. Approximate probabilistic query answering over inconsistent databases. In *International Conference on Conceptual Modeling (ER)*, pages 311–325, 2008.

[Greco and Molinaro, 2012] Sergio Greco and Cristian Molinaro. Probabilistic query answering over inconsistent databases. *Annals of Mathematics and Artificial Intelligence*, 64(2-3):185–207, 2012.

[Greco *et al.*, 2017] Sergio Greco, Cristian Molinaro, and Irina Trubitsyna. Computing approximate certain answers over incomplete databases. In *Alberto Mendelzon International Workshop (AMW)*, 2017.

[Guagliardo and Libkin, 2016] Paolo Guagliardo and Leonid Libkin. Making SQL queries correct on incomplete databases: A feasibility study. In *Symposium on Principles of Database Systems (PODS)*, pages 211–223, 2016.

[Imielinski and Lipski, 1984] Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[Lembo *et al.*, 2010] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant semantics for description logics. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 103–117, 2010.

[Lembo *et al.*, 2011] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Query rewriting for inconsistent DL-Lite ontologies. In *International Conference on Web Reasoning and Rule Systems (RR)*, pages 155–169, 2011.

[Lembo *et al.*, 2015] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *Journal of Web Semantics*, 33:3–29, 2015.

[Libkin, 2015] Leonid Libkin. How to define certain answers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4282–4288, 2015.

[Libkin, 2016] Leonid Libkin. Certain answers as objects and knowledge. *Artificial Intelligence*, 232:1–19, 2016.

[Lukasiewicz *et al.*, 2012a] Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Inconsistency handling in Datalog+/– ontologies. In *European Conference on Artificial Intelligence (ECAI)*, pages 558–563, 2012.

[Lukasiewicz *et al.*, 2012b] Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Inconsistency-tolerant query rewriting for linear Datalog+/–. In *Datalog 2.0*, pages 123–134, 2012.

[Lukasiewicz *et al.*, 2015] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. From classical to consistent query answering under existential rules. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 1546–1552, 2015.

[Martinez *et al.*, 2014] Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. Policy-based inconsistency management in relational databases. *International Journal of Approximate Reasoning*, 55(2):501–528, 2014.

[Rosati, 2011] Riccardo Rosati. On the complexity of dealing with inconsistency in description logic ontologies. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1057–1062, 2011.