First-Order Query Rewriting for Inconsistent Databases

Ariel D. Fuxman and Renée J. Miller

Department of Computer Science
University of Toronto
{afuxman, miller}@cs.toronto.edu

Abstract. We consider the problem of retrieving consistent answers over databases that might be inconsistent with respect to some given integrity constraints. In particular, we concentrate on sets of constraints that consist of key dependencies. Most of the existing work has focused on identifying intractable cases of this problem. In contrast, in this paper we give an algorithm that computes the consistent answers for a large and practical class of conjunctive queries. Given a query q, the algorithm returns a first-order query Q (called a *query rewriting*) such that for every (potentially inconsistent) database I, the consistent answers for q can be obtained by evaluating Q directly on I.

1 Introduction

Consistent query answering is the problem of retrieving "consistent" answers over databases that might be inconsistent with respect to some given integrity constraints. Applications that have motivated the study of this problem include data integration and data exchange. In data integration, the goal is to provide "a uniform interface to multiple autonomous data sources" [Hal01]. In data exchange, "data structured under one (source) schema must be restructured and translated into an instance of a different (target) schema" [FKMP03]. In both contexts, it is often the case that the source data does not satisfy the integrity constraints of the global or target schema. The traditional approach to deal with this situation involves "cleaning" the source instance in order to remove data that violates the target constraints. However, data cleaning is supported by semi-automatic tools at best, and it is necessarily a human-labor intensive process. An alternative approach would be to exchange an inconsistent instance, and employ the techniques of consistent query answering to resolve inconsistencies at query time. Of course, this approach becomes viable only if efficient tools for consistent query answering are available. In this paper, we present a number of results that are a step in this direction.

In addition to these long-standing problems, the trend toward autonomous computing is making the need to manage inconsistent data more acute. In autonomous environments, we can no longer assume that data are married with a single set of constraints that define their semantics. As constraints are used in an increasing number of roles (from modelling the query capabilities of a system, to defining mappings between independent sources), there is an increasing number

T. Eiter and L. Libkin (Eds.): ICDT 2005, LNCS 3363, pp. 335–349, 2005.

of applications in which data must be used with a set of independently designed constraints. In such applications, a static approach where consistency (with respect to a fixed set of constraints) is enforced by cleaning the database may not be appropriate. Rather, a dynamic approach in which data is not changed, but consistency is taken into account at query time, permits the constraints to evolve independently from the data.

The input to the consistent query answering problem is: a schema $\mathbf{R}$, a set Σ of integrity constraints, and a database instance I over $\mathbf{R}$. The database I might be *inconsistent*, in the sense that it might violate some of the constraints of Σ. In this work, we draw upon the concept of *repairs*, defined by Arenas et al. [ABC99], to give semantics to the problem. A repair $\mathcal{I}$ of I is an instance of $\mathbf{R}$ such that $\mathcal{I}$ satisfies the integrity constraints of Σ, and $\mathcal{I}$ differs minimally from I (where minimality is defined with respect to the symmetric difference between I and $\mathcal{I}$). Under this definition, repairs need not be unique. Intuitively, each repair corresponds to one possible way of "cleaning" the inconsistent database.

The notion of repairs is used to give semantics to consistent query answering in the following way. Given an instance I, a tuple $\boldsymbol{t}$ is said to be a *consistent answer* for q on I if $\mathcal{I} \models q[\boldsymbol{t}]$, *for every* repair $\mathcal{I}$ of I. This concept is similar to that of *certain answers* used in the context of data integration [AD98], but for consistent answers the set of possible worlds are the repairs of the inconsistent database, rather than the legal instances of a global database.

In this work, we focus on sets of integrity constraints that consist of key dependencies. The most commonly used constraints in database systems are keys and foreign keys. Of these, keys pose a particular challenge since instances that are inconsistent with respect to a set of key dependencies admit an exponential number of repairs in the worst case. This potentially large number of repairs leads to the question of whether it is possible to compute consistent answers efficiently. The answer to this question is known to be negative in general [CM04, CLR03a]. However, this does not necessarily preclude the existence of classes of queries for which the problem is easier to compute. Hence, we consider the following question: for what queries is the problem of computing consistent answers in polynomial time (in data complexity)?

In general, given a query q, it does not suffice to evaluate q directly on a (possibly inconsistent) instance I in order to get the consistent answers. Therefore, a related question is: does there exist some other query Q such that for every instance I, the consistent answers for q can be obtained by just evaluating Q on I? If Q is a first-order query, we say that q is *first-order rewritable*. Since first-order queries can be written in SQL, if the query is first-order rewritable, then its consistent answers can be retrieved (at query time) using existing commercial database technology. Given the desirability of such an approach, we consider the question of identifying classes of queries that are first-order rewritable.

Summary of results The main contribution of this paper is an algorithm that produces a first-order query rewriting for the problem of computing consistent answers. The algorithm, which is presented in Section 3, runs in polynomial time in the size of the query. We prove the correctness of the algorithm for a

large class of conjunctive queries. The class is defined in terms of the *join graph* of the query. The join graph is a directed graph such that: its vertices are the literals of the query; and it has an arc for each join in the query that involves some variable that is at the position of a non-key attribute. Our algorithm works for conjunctive queries without repeated relation symbols (but with any number of literals and variables) whose join graph is a forest. The class contains most queries that usually arise in practice. For example, 20 out of 22 queries in TPC-H [TPC03], the industry standard for decision support systems, are in this class.

In Section 4, we present a class of queries for which the conditions of applicability of the algorithm (which can be verified in polynomial time in the size of the query) are necessary and sufficient. That is, we show a class such that the problem of computing the consistent answers is coNP-complete for *every* query of the class whose join graph is not a forest. Notice that this type of result is much stronger than the usual approach taken in the consistent query answering literature, which consists of showing intractability of a class by exhibiting *at least one* query for which the problem is intractable. As a corollary of our result, we get a dichotomy for this class of queries: given a query q, either the problem of computing the consistent answers for q is first-order rewritable (and thus it is in PTIME), or it is a coNP-complete problem.

2 Formal Framework

A *schema* $\mathbf{R}$ is a finite collection of relation symbols, each of which has an associated arity. A set of *integrity constraints* Σ consists of sentences in some logical formalism over $\mathbf{R}$. An *instance* I over $\mathbf{R}$ is a function that associates to each relation symbol R of $\mathbf{R}$ a relation $I(R)$. Given a tuple $\boldsymbol{t}$ occurring in relation $I(R)$, we denote by $R(\boldsymbol{t})$ the association between $\boldsymbol{t}$ and R. An instance I is *consistent* with respect to a set of integrity constraints Σ if I satisfies Σ in the standard model-theoretic sense, that is $I \models \Sigma$.

We adopt a semantics for consistent query answering that was originally introduced by Arenas et al. [ABC99], and relies upon the concept of *repairs*. A repair is an instance that satisfies the integrity constraints, and which has a minimal distance to the inconsistent database. The *distance* between two database instances I and I' is defined as their symmetric difference, i.e., $\Delta(I, I') = (I - I') \cup (I' - I)$. The formal definition of repair is the following.

Definition 1 (Repair [ABC99]). *Let I be an instance. We say that an instance $\mathcal{I}$ is a* **repair** *of I with respect to Σ if:* [1]

- $\mathcal{I} \models \Sigma$, *and*
- *there is no instance I' such that $I' \models \Sigma$ and $\Delta(I, I') \subset \Delta(I, \mathcal{I})$ (i.e., $\Delta(I, \mathcal{I})$ is minimal under set inclusion in the class of instances that satisfy Σ).*

Example 1. Let $\mathbf{R}$ be a schema with one relation symbol R. Assume that R has two attributes: E (Employee) and S (Salary), and that the only constraint in Σ

[1] Whenever Σ is clear from the context, we will just say that $\mathcal{I}$ is a repair of I.

is that the attribute E is the key of R. Let $I = \{R(John, 1000), R(John, 2000), R(Mary, 3000)\}$. We can see that I is inconsistent with respect to Σ. There are two repairs: $\mathcal{I}_1 = \{(John, 1000), (Mary, 3000)\}$ and $\mathcal{I}_2 = \{(John, 2000), (Mary, 3000)\}$. We use the term "repair", as opposed to "minimal repair", because it is standard in the literature [ABC99]. However, notice that, by definition, all repairs have a minimal distance to the inconsistent database. For example, $\{(John, 2000)\}$ and $\{(Mary, 3000)\}$ are not repairs because their distance with respect to I is not minimal under set inclusion. The minimality condition for the repairs is crucial in the definition. Otherwise, the empty set would trivially be a repair of every instance.

The semantics for query answering is given in terms of *consistent answers* [ABC99], which we define next.

Definition 2 (Consistent answer [ABC99]). *Let* $\mathbf{R}$ *be a schema. Let* Σ *be a set of integrity constraints. Let* I *be an instance over* $\mathbf{R}$ *(possibly inconsistent with respect to* Σ*). Let* q *be a query over* $\mathbf{R}$*. We say that a tuple* $\boldsymbol{t}$ *is a* consistent answer *with respect to* Σ *if* $\mathcal{I} \models q[\boldsymbol{t}]$*, for every repair* $\mathcal{I}$ *of* I *with respect to* Σ*. We denote this as* $\boldsymbol{t} \in \mathbf{consistent}_{\Sigma}(q, I)$*.*

Example 1 (continued). Let $q_1(e) = \exists s : R(e, s)$. The consistent answers for q_1 on I are the tuples $(John)$ and $(Mary)$. Let $q_2(e, s) = R(e, s)$. The only consistent answer for q_2 on I is $(Mary, 3000)$. Notice that the tuples $(John, 1000)$ and $(John, 2000)$ are not consistent answers. The reason is that neither of them are present in *both* repairs. Intuitively, this reflects the fact that John's salaries are inconsistent data.

For convenience, we will use the following notation for the consistent answers to Boolean queries.

Definition 3. *Let* $\mathbf{R}$ *be a schema. Let* Σ *be a set of integrity constraints. Let* I *be an instance over* $\mathbf{R}$*. Let* q *be a Boolean query over* $\mathbf{R}$*. We say that* $\mathbf{consistent}_{\Sigma}(q, I) = \texttt{true}$ *if for every repair* $\mathcal{I}$ *of* I *with respect to* Σ*,* $\mathcal{I} \models q$*. We say that* $\mathbf{consistent}_{\Sigma}(q, I) = \texttt{false}$ *if there exists* at least one *repair* $\mathcal{I}$ *of* I *with respect to* Σ *such that* $\mathcal{I} \not\models q$*.*

Notice the asymmetry between the case for $\mathbf{consistent}_{\Sigma}(q, I) = \texttt{true}$ and $\mathbf{consistent}_{\Sigma}(q, I) = \texttt{false}$. While, for the former, every repair must satisfy the query, for the latter it suffices to have just one (non-satisfying) repair. This is not intrinsic to Boolean queries: by Definition 2, it is also the case that $\boldsymbol{t} \notin \mathbf{consistent}_{\Sigma}(q, I)$ if there exists at least one repair $\mathcal{I}$ such that $\mathcal{I} \not\models q[\boldsymbol{t}]$.

We will denote the problem of computing consistent answers as $\texttt{CONSISTENT}(q, \Sigma)$, and define it as follows.

Definition 4. *Let* $\mathbf{R}$ *be a schema. Let* q *be a query over* $\mathbf{R}$*. Let* Σ *be a set of integrity constraints. The* consistent query answering problem $\texttt{CONSISTENT}(q, \Sigma)$ *is the following: given an instance* I *over* $\mathbf{R}$*, and tuple* $\boldsymbol{t}$*, is it the case that* $\boldsymbol{t} \in \mathbf{consistent}_{\Sigma}(q, I)$*?*

We will design an algorithm that computes consistent answers *directly* from the inconsistent database, without explicitly building the repairs. In fact, given

a query q, the algorithm will return a first-order query Q such that, for every instance I, the consistent answers for q can be obtained by just evaluating Q on I. We call Q a *first-order query rewriting*, and define it next.

Definition 5 (first-order query rewriting). *Let* $\mathbf{R}$ *be a schema. Let* Σ *be a set of integrity constraints. Let* q *be a query over* $\mathbf{R}$*. We say that the problem* $\mathtt{CONSISTENT}(q, \Sigma)$ *is* first-order rewritable *if there is a first-order query* Q *such that* $I \models Q[\boldsymbol{t}]$ *iff* $\boldsymbol{t} \in \mathbf{consistent}_{\Sigma}(q, I)$*, for every instance* I *over* $\mathbf{R}$*. We also say that* Q *is a* first-order rewriting *of the problem* $\mathtt{CONSISTENT}(q, \Sigma)$*.* [2]

Notice that if $\mathtt{CONSISTENT}(q, \Sigma)$ is first-order rewritable, then it is tractable. This is because the data complexity of first-order logic is in PTIME (actually, in AC^0). Thus, it can be tested in polynomial time whether $I \models Q[\boldsymbol{t}]$. Besides this, an approach based on query rewriting is attractive because first-order queries can be written in SQL. Therefore, if the query is first-order rewritable, the consistent answers can be retrieved using existing database technology.

Throughout the paper, we will assume that the set Σ of integrity constraints consists of at most one key dependency per relation of the schema. To facilitate specifying the set of constraints each time that we give a query, we will underline the positions in each literal that correspond to key attributes. Furthermore, by convention, the key attributes will be given first. For example, the query $q = \exists x, y, z : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z)$ indicates that literals R_1 and R_2 represent binary relations whose first attribute is the key. We will use bold letters (e.g., $\boldsymbol{x}$, $\boldsymbol{y}$) to denote vectors of variables or constants from a query or tuple. In addition, when we give a tuple, we will underline the values that appear at the position of key attributes. For instance, for a tuple $R(\underline{\boldsymbol{c}}, \boldsymbol{d})$, we will say that $\underline{\boldsymbol{c}}$ is a *key value*, and $\boldsymbol{d}$ is a *non-key value*. Using this notation, the key constraints of Σ that are relevant to the query are denoted directly in the query expression.

The results in this paper concern (classes of) conjunctive queries. We will adopt the convention of using $\boldsymbol{x}$ to denote variables and constants that appear at the position of key attributes, and $\boldsymbol{y}$ for variables and constants that appear at the position of non-key attributes. Thus, conjunctive queries will be of the form:

$$q(w_1, \ldots, w_m) = \exists z_1, \ldots, z_l : R_1(\underline{\boldsymbol{x}}_1, \boldsymbol{y}_1), ..., R_n(\underline{\boldsymbol{x}}_n, \boldsymbol{y}_n)$$

where $w_1, \ldots, w_m, z_1, \ldots, z_l$ are all the variables that appear in the literals of q. We will say that $w_1, \ldots, w_m$ are the *free variables* of q. Notice that even though there are no equality symbols in q, their effect is achieved by having variables that appear in q more than once. The queries may also contain constants, which we will denote with bold letters from the beginning of the alphabet (e.g., $\mathbf{a}$ and $\mathbf{b}$). We will say that there is a *join* on a variable w if w appears in two literals $R_i(\underline{\boldsymbol{x}}_i, \boldsymbol{y}_i)$ and $R_j(\underline{\boldsymbol{x}}_j, \boldsymbol{y}_j)$ such that $i \neq j$. If w occurs in $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$, we say that there is a *non-key join* on w.

[2] On occasion, we will simply say that q is *first-order rewritable*, and that Q is a *first-order rewriting* of q.

Throughout the paper, we will focus on the class of conjunctive queries *without repeated relation symbols*. A conjunctive query without repeated relation symbols is a conjunctive query such that every relation symbol of the schema appears in q at most once. Notice that, in spite of this restriction, the query can still have any arbitrary number of literals and relation symbols, and there are no constraints on the occurrence of variables in the query.

3 A query rewriting algorithm

3.1 A Class of Tractable Queries

The problem of computing consistent answers for conjunctive queries over databases that might violate a set of key constraints is known to be coNP-complete in general [CM04, CLR03a]. This is the case even for queries with no repeated relation symbols, which is the focus of this section. However, this does not necessarily preclude the existence of classes of queries for which the problem is easier to compute. In fact, in this section we characterize a large and practical class of conjunctive queries for which the problem of computing consistent answers is indeed tractable. Even more so, we show that all queries in this class are first-order rewritable, and we give a polynomial-time algorithm that computes the first-order rewriting.

Before presenting the tractable class, let us consider the following queries for which the problem of computing consistent answers is coNP-complete, as will be shown in Section 4.

- $q_1 = \exists x, x', y : R_1(\underline{x}, y) \wedge R_2(\underline{x}', y)$
- $q_2 = \exists x, y, z : R_1(\underline{z}, \underline{x}, y) \wedge R_2(\underline{y}, x)$
- $q_3 = \exists x, y, z, w : R_1(\underline{x}, y) \wedge R_2(\underline{z}, w) \wedge R_3(\underline{y}, \underline{w})$

The queries presented above are rare in practice. The first consists of a join between *non-key* attributes; the second involves a cycle; and the third, a join with part, but not the entire key of a relation. We use these queries to provide insight into when a query is intractable. In particular, we will show in Section 4 a class of queries for which the presence of cycles and non-key joins are in fact necessary conditions for intractability. Notice that such conditions are concerned with the joins in the query where at least one non-key variable is involved. In order to define such conditions precisely, we will state them in terms of what we call the *join graph* of the query.

Definition 6 (join graph). *Let q be a conjunctive query. The* join graph *G of q is a directed graph such that:*

- *the vertices of G are the literals of q;*
- *there is an arc from R_i to R_j if $i \neq j$ and there is some variable w such that w occurs at the position of a non-key attribute in R_i and w occurs in R_j;*
- *there is a self-loop at R_i (i.e., an arc from R_i to R_i) if there is some variable w such that w occurs at the position of a non-key attribute of R_i, and w occurs at least twice in R_i.*

As we can see in Figure 1, the join graphs of q_1 and q_2 have a cycle. The join graph of q_3 does not have a cycle, but it is not a tree because the node for relation R_3 has two incoming arcs. Therefore, we will focus on queries whose join graph is a tree (or a forest). For example, the join graph of the following query is a tree. The graph is shown in Figure 1.

$$q_4 = \exists x, y, z, w : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z) \wedge R_3(\underline{z}, w) \wedge R_4(\underline{y}, \mathbf{a})$$

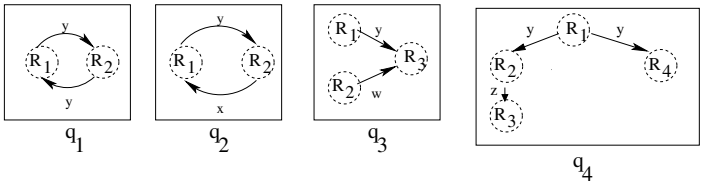

Fig. 1. Join graphs

An additional condition that we will impose on the query is that the joins from non-key to key attributes involve the *entire* key of a relation. We will call such joins *full*. For example, all the non-key to key joins of query q_4 are full. On the other hand, in the query $q = \exists x, y, z, w : R_1(\underline{x}, y) \wedge R_2(\underline{z}, \underline{y}, w)$ the join between R_1 and R_2 is not full since it does not involve the entire key of R_2.

Definition 7. *Let q be a conjunctive query. Let $R_i(\underline{\boldsymbol{x}_i}, \boldsymbol{y}_i)$ and $R_j(\underline{\boldsymbol{x}_j}, \boldsymbol{y}_j)$ be a pair of literals of q. We say that there is a* full non-key to key join *from R_i to R_j if every variable of $\boldsymbol{x}_j$ appears in $\boldsymbol{y}_i$.*

Considering queries with only full non-key to key joins, the class $\mathcal{C}_{tree}$ that we define next consists of the queries whose join graph is a forest.

Definition 8. *Let q be conjunctive query without repeated relation symbols. Let G be the join graph of q. We say that $q \in \mathcal{C}_{tree}$ if G is a forest (i.e., every connected component of G is a tree) and every non-key to key join of q is full.*

A fundamental observation about $\mathcal{C}_{tree}$ is that it is a very common, practical class of queries. Arguably, the most used form of joins are from a set of non-key attributes of one relation (which may be a foreign key)[3] to the key of another relation (which may be a primary key). Furthermore, such joins typically involve the *entire* primary key of the relation (and, hence, they are full joins in our terms). Finally, cycles are rarely present in the queries used in practice. Admittedly, the restriction not to have repeated relation symbols does rule out some common queries (those in which the same relation appears twice in the `FROM` clause of an SQL query). Still, many queries used in practice do not have

[3] Notice that in this work we are not dealing with the problem of inconsistency with respect to foreign keys, but with respect to key dependencies.

repeated relation symbols. As an empirical observation, we point out that 20 out of 22 queries in the TPC-H standard [TPC03] are in class $\mathcal{C}_{tree}$. [4]

3.2 Algorithm

The following examples highlight some of the intuition underlying our query rewriting algorithm.

Example 2. Let $q = \exists x : R_1(\underline{x}, \mathbf{a})$. First of all, notice that q itself is not a query rewriting of $\texttt{CONSISTENT}(q, \Sigma)$. Consider the instance $I_1 = \{R_1(\underline{c}_1, a), R_1(\underline{c}_1, b)\}$. It is easy to see that $I_1 \models q$. However, $\textbf{consistent}_\Sigma(q, I_1) = \texttt{false}$ because the repair $\mathcal{I} = \{R_1(\underline{c}_1, b)\}$ is such that $\mathcal{I} \not\models q$. Now, consider $I_2 = \{R_1(\underline{c}_1, a), R_1(\underline{c}_1, b), R_1(\underline{c}_2, a)\}$. It is easy to see that $\textbf{consistent}_\Sigma(q, I_2) = \texttt{true}$. This is because there is a key value in R_1 (c_2 in this case) that appears with a as its non-key value, and does not appear with any other constant a' such that $a' \neq a$. This can be checked with a formula $Q_{consist} = \forall y' : R_1(\underline{x}, y') \rightarrow y' = \mathbf{a}$. In fact, we will show that a query rewriting Q for q can be obtained as the conjunction of q and $Q_{consist}$:

$$Q = \exists x : R_1(\underline{x}, \mathbf{a}) \wedge \forall y' : R_1(\underline{x}, y') \rightarrow y' = \mathbf{a}$$

Example 3. Let $q = \exists x, y, z : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z)$. As in the previous example, q itself is not a query rewriting of $\texttt{CONSISTENT}(q, \Sigma)$. For, consider the instance $I_1 = \{R_1(\underline{c}_1, d_1), R_1(\underline{c}_1, d_2), R_2(\underline{d}_1, e_1)\}$. It is easy to see that $I_1 \models q$. However, $\textbf{consistent}_\Sigma(q, I_1) = \texttt{false}$ because the repair $\mathcal{I} = \{R_1(\underline{c}_1, d_2), R_2(\underline{d}_1, e_1)\}$ is such that $\mathcal{I} \not\models q$. Now, consider $I_2 = \{R_1(\underline{c}_1, d_1), R_1(\underline{c}_1, d_2), R_2(\underline{d}_1, e_1), R_2(\underline{d}_2, e_2)\}$. It is easy to see that $\textbf{consistent}_\Sigma(q, I_2) = \texttt{true}$. This is because *every* non-key value that appears together with c_1 in some tuple (in this case, d_1 and d_2) joins with a tuple of R_2. This can be checked with a formula $Q_{consist} = \forall y' : R_1(\underline{x}, y') \rightarrow \exists z' : R_2(\underline{y}', z')$. We will soon show that a query rewriting Q for q can be obtained as the conjunction of q and $Q_{consist}$, as follows:

$$Q = \exists x, y, z : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z) \wedge \forall y' : (R_1(\underline{x}, y') \rightarrow \exists z' : R_2(\underline{y}', z'))$$

We now proceed to present `RewriteConsistent`, our query rewriting algorithm. For convenience, it is split into three modules, which are shown in Figures 2, 3, and 4. Given a query q such that $q \in \mathcal{C}_{tree}$, and a set of key constraints Σ, $\texttt{RewriteConsistent}(q, \Sigma)$ returns a first-order rewriting Q for the problem of obtaining the consistent answers for q with respect to Σ. The algorithm `RewriteConsistent` is shown in Figure 2. The first-order rewriting Q that it returns is obtained as the conjunction of the input query q, and a new query called $Q_{consist}$. The query $Q_{consist}$ is used to ensure that q is satisfied in *every* repair (and, hence, $\textbf{consistent}_\Sigma(q, I) = \texttt{true}$). It is important to notice that $Q_{consist}$ will be applied directly to the inconsistent database (i.e., we will never generate

[4] This is considering the Select-Project-Join structure of the queries, not additional features such as aggregation or arithmetical operators. The reason that two of the queries are outside the class is just because they have repeated relation symbols.

the repairs). The query $Q_{consist}$ is obtained by recursion on the tree structure of each of the components of the join graph of q (recall that since $q \in \mathcal{C}_{tree}$, the join graph is a forest). The recursive algorithm is called `RewriteTree`, and is shown in Figure 3.

In the query $Q_{consist}$, some of the variables of q are renamed. Let us illustrate this with the query $q = \exists x, y, z : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z)$ from Example 3. In this case, $Q_{consist} = \forall y' : R_1(\underline{x}, y') \rightarrow \exists z' : R_2(\underline{y'}, z')$. Notice that the variable y of q is renamed to y' in $Q_{consist}$. In order to keep track of the renaming during the execution of the algorithm, we use a substitution δ for the variables of q.

The variables that will be renamed in $Q_{consist}$ by the substitution δ are those that are involved in some join from a non-key to a key position. Notice that these are the joins that create arcs in the join graph. The renamed variables are universally quantified in $Q_{consist}$. The intuition behind this is that the renamed variable denotes a non-key position in a literal and, as we illustrated in Example 3, the query must be satisfied by all the non-key values of a given key.

In the algorithm `RewriteConsistent`, the substitution δ is initialized to be just the identity on the variables that do not appear in non-key positions of any literal. These are the variables in the set $IdentityVars$ of the algorithm. During the recursive execution of `RewriteTree`(T_i, Σ, δ), the literal $R(\underline{\boldsymbol{x}}, \boldsymbol{y})$ at the root of T_i is selected, and the variables of $\boldsymbol{y}$ are renamed to newly-created variables from a vector $\boldsymbol{y}^*$. The substitution δ is used here to record such renamings.

At each step, `RewriteTree` produces a rewriting Q_{local} for the literal $R(\underline{\boldsymbol{x}}, \boldsymbol{y})$ at the root of the tree. This rewriting is done independently of the rest of the query, and produced by the algorithm `RewriteLocal`. We show this algorithm in Figure 4. The query Q_{local} deals with the constants that appear in $\boldsymbol{y}$ in the same way as we illustrated in Example 2.

Notice that we have presented the algorithm only for Boolean queries, but this is just for notational simplicity. In order to apply the algorithm to queries with free variables, it suffices to treat the free variables as if they were constants (using the algorithm `RewriteLocal`). For example, consider the query $q(y) = \exists x : R_1(\underline{x}, y)$. Notice that the only difference with the query of Example 2 is that the constant $\mathbf{a}$ is replaced by the free variable y. The query rewriting for q is the following:

$$Q(y) = \exists x : R_1(\underline{x}, y) \wedge \forall y' : R_1(\underline{x}, y') \rightarrow y' = y$$

The next example illustrates the application of the algorithm.

Example 4. Let q be the query q_4 introduced in Section 3.1.

$$q = \exists x, y, z, w : R_1(\underline{x}, y) \wedge R_2(\underline{y}, z) \wedge R_3(\underline{z}, w) \wedge R_4(\underline{y}, \mathbf{a})$$

The join graph T of q is shown in Figure 1. In this case, T consists of one connected component, which is a tree. Let T_2 be the subtree of T that consists of the nodes for literals R_2 and R_3. Let T_3 be a tree that consists of exactly one node, for literal R_3. Let T_4 be a tree that consists of exactly one node, for literal R_4. The first-order query rewriting Q of q is obtained by applying the algorithm `RewriteConsistent`(q, Σ) as follows.

Algorithm `RewriteConsistent`(q, Σ)
  Let G be the join graph of q
  Let $T_1, \ldots, T_m$ be the connected components of G
  Let $IdentityVars = \{x :$ there is a literal $R(\underline{\boldsymbol{x}}, \boldsymbol{y})$ in q such that x occurs in $\boldsymbol{x}$,
    and for every literal $R'(\underline{\boldsymbol{x}}', \boldsymbol{y}')$ in q, x does not occur in $\boldsymbol{y}'\}$
  Let δ be the identity substitution for the variables of $IdentityVars$
  for $i := 1$ to m **do**
    Let $Q_i = \texttt{RewriteTree}(T_i, \Sigma, \delta)$
  end for
  Let $Q_{consist} = \bigwedge_{i=1\ldots m} Q_i$
  Let $Q = q \wedge Q_{consist}$
  return Q

Fig. 2. Query rewriting algorithm

$Q = \texttt{RewriteConsistent}(q, \Sigma) = q \wedge Q_{consist}$
$Q_{consist}(x) = \texttt{RewriteTree}(T, \Sigma, \langle x/x \rangle) = \forall y' : R_1(\underline{x}, y') \rightarrow (Q_2 \wedge Q_4)$
$Q_2(y') = \texttt{RewriteTree}(T_2, \Sigma, \langle x/x, y/y' \rangle) = \exists z' : R_2(\underline{y}', z') \wedge \forall z' : R_2(\underline{y}', z') \rightarrow Q_3$
$Q_3(z') = \texttt{RewriteTree}(T_3, \Sigma, \langle x/x, y/y', z/z' \rangle) = \exists w' : R_3(\underline{z}', w')$
$Q_4(y') = \texttt{RewriteTree}(T_4, \Sigma, \langle x/x, y/y' \rangle) = \exists u' : R_4(\underline{y}', u') \wedge \forall u' : (R_4(\underline{y}', u') \rightarrow u' = \mathbf{a})$

3.3 Correctness Proof

For the correctness proof, we will refer to the query associated to a join graph. The *query q_G for a join graph G* is a conjunctive query such that the literals of q_G are the literals that appear in the nodes of G; all the variables of q_G that occur at a non-key position in a literal of q_G are existentially quantified; and the rest of the variables of q_G are free. Notice that if a variable of q_G is the cause of the existence of an arc in the join graph G (e.g., variables y and z from Example 4), then the variable is existentially-quantified in q_G.

Definition 9. *Let G be a join graph. Let $R_1(\underline{\boldsymbol{x}}_1, \boldsymbol{y}_1), \ldots, R_n(\underline{\boldsymbol{x}}_n, \boldsymbol{y}_n)$ be the literals that appear in the nodes of G. Let $W = \{w : w$ is a variable that appears in $\boldsymbol{y}_j$, for some j such that $1 \leq j \leq n\}$. Let $\boldsymbol{w}$ be the variables of W. Let $\boldsymbol{z}$ be the variables of $R_1, \ldots, R_n$ that are not in W. We say that q_G is the* query for G *if q_G is of the following form:*

$$q_G(\boldsymbol{z}) = \exists \boldsymbol{w} : R_1(\underline{\boldsymbol{x}}_1, \boldsymbol{y}_1) \wedge \ldots \wedge R_n(\underline{\boldsymbol{x}}_n, \boldsymbol{y}_n)$$

The correctness proof of `RewriteTree` is by induction on the size of the input tree, and relies on the following lemma.

Lemma 1. *Let T be a join graph such that T is a tree. Let q_T be the query for T, as in Definition 9. Let δ be a substitution for the free variables of q_T. Let Q be the first-order query returned by `RewriteTree`(T, Σ, δ).*

Let I be an instance. Let ν_q be a valuation for the free variables of q_T such that $I \models q_T[\nu_q]$. Let ν_Q be a valuation for the free variables of Q such that

Algorithm `RewriteTree`(T, Σ, δ)
  Let $R(\underline{\boldsymbol{x}}, \boldsymbol{y})$ be the literal at the root node of T
  Let $\boldsymbol{x}^* = \boldsymbol{x}[\delta]$
  Let l be the arity of $\boldsymbol{y}$
  Let $\boldsymbol{y}^* = y'_1, \ldots, y'_l$, where $y'_1, \ldots, y'_l$ are newly created variables
  if T consists of exactly one node **then**
    Let $Q_{local} =$ `RewriteLocal`$(R, \boldsymbol{x}^*, \boldsymbol{y}, \boldsymbol{y}^*, \Sigma, \delta)$
    return Q_{local}
  end if
  Let T_{local} be a tree that consists of exactly one node with literal R
  Let $Q_{local} =$ `RewriteTree`$(T_{local}, \Sigma, \delta)$
  Let $R_1, \ldots, R_m$ be the children of R in T
  Let $\delta' = \delta \cup \{y/y' :$ there exists p such that y and y' are variables that occur at position p in $\boldsymbol{y}$ and $\boldsymbol{y}^*$, respectively$\}$
  for $i := 1$ to m **do**
    Let T_i be the subtree of T rooted at R_i
    Let $Q_i =$ `RewriteTree`(T_i, Σ, δ')
  end for
  Let $Q_{subtrees} = \bigwedge_{i=1\ldots m} Q_i$
  Let $Q = Q_{local} \wedge \forall \boldsymbol{y}^* : (R(\underline{\boldsymbol{x}}^*, \boldsymbol{y}^*) \rightarrow Q_{subtrees})$
  return Q

Fig. 3. Recursive algorithm on the tree structure of the join graph

$\nu_Q(w) = \nu_q(w)$ if $\delta(w) = w$. Then, $I \models Q[\nu_Q]$ iff $\mathcal{I} \models q_T[\delta][\nu_Q]$ for every repair $\mathcal{I}$ of I.

From the previous lemma, we obtain the main result of the section proving that the rewriting algorithm is correct for all queries in $\mathcal{C}_{tree}$.

Theorem 1. *Let $\mathbf{R}$ be a schema. Let Σ be a set of integrity constraints, consisting of one key dependency per relation of $\mathbf{R}$. Let q be a conjunctive query over $\mathbf{R}$ such that $q \in \mathcal{C}_{tree}$. Let $\boldsymbol{t}$ be a tuple. Let Q be the first-order query returned by* `RewriteConsistent`(q, Σ).

Then, for every instance I over $\mathbf{R}$, $I \models Q[\boldsymbol{t}]$ iff $\boldsymbol{t} \in \mathbf{consistent}_{\Sigma}(q, I)$.

4 A Dichotomy Result

In the previous section, we presented a query rewriting algorithm which works on queries with full joins whose join graph is a forest. Clearly, this is a sufficient condition for a query to be first-order rewritable. In this section, we address the following question: for which class of queries is it also a necessary condition? In particular, we show a class of queries such that the problem of computing the consistent answers is coNP-complete for *every* query of the class which does not satisfy the conditions of our query rewriting algorithm. Notice that this establishes a dichotomy between first-order rewritability and coNP-completeness, and

Algorithm `RewriteLocal`$(R, \boldsymbol{x}^*, \boldsymbol{y}, \boldsymbol{y}^*, \Sigma, \delta)$
  if there is at least one constant in $\boldsymbol{y}$ **then**
    Let l be the arity of $\boldsymbol{y}$
    for $i := 1$ to l **do**
      Let y' be the variable that appears at position i of $\boldsymbol{y}^*$
      if there is a constant c at position i of $\boldsymbol{y}$ **then**
        Let E_i be the equality $y' = c$
      end if
    end for
    $EqualityPos = \{i :$ there is a constant a position i in $\boldsymbol{y}\}$
    Let $Q_{const} = \bigwedge_{i \in EqualityPos} E_i$
    if δ is the identity substitution on all variables of $\boldsymbol{x}$ **then**
      Let $Q_{local} = \forall \boldsymbol{y}^* : (R(\underline{\boldsymbol{x}}^*, \boldsymbol{y}^*) \rightarrow Q_{const})$
    else
      Let $Q_{local} = \exists \boldsymbol{y}^* : R(\underline{\boldsymbol{x}}^*, \boldsymbol{y}^*) \wedge \forall \boldsymbol{y}^* : (R(\underline{\boldsymbol{x}}^*, \boldsymbol{y}^*) \rightarrow Q_{const})$
    end if
  end if
  if there are no constants in $\boldsymbol{y}$ **then**
    if δ is the identity substitution on all variables of $\boldsymbol{x}$ **then**
      Let Q_{local} be an empty string
    else
      Let $Q_{local} = \exists \boldsymbol{y}^* : R(\underline{\boldsymbol{x}}^*, \boldsymbol{y}^*)$
    end if
  end if
  return Q_{local}

Fig. 4. Query rewriting for given literal

is therefore much stronger than the complexity results present in the consistent query answering literature [CM04, CLR03a]. In the literature, a class $\mathcal{C}$ is said to be coNP-hard if there is *at least one* query $q \in \mathcal{C}$ such that `CONSISTENT`(q, Σ) is a coNP-hard problem. Under such a definition, it suffices to exhibit just *one* intractable query in order to conclude that the entire class is coNP-complete. In contrast, in this section we will present a class of queries such that for *every* query q in the class, `CONSISTENT`(q, Σ) is coNP-complete.

As a first step towards proving a dichotomy for the class of conjunctive queries, we will focus on a subclass for which we can establish such a result. We call this subclass $\mathcal{C}^*$, and define it in Definition 10. In the definition, we give three conditions that are meant to rule out of the class the *only* cases of the dichotomy that we leave open. We illustrate the conditions as follows. Consider the query $q_5 = \exists x, y : R_1(\underline{x}, y) \wedge R_2(\underline{x}, y)$. The join graph of this query is not a forest; yet, it is not difficult to find a rewriting for it. What is particular about q_5 is the fact that its two literals have the *same* key. We rule out this case with the first condition of Definition 10. Now, consider the query $q_6 = \exists x : R_1(\underline{x}, x)$. Although it is easy to find a rewriting for this query, its join graph contains a self-loop. We rule out the queries whose join graph is a self-loop with the second

condition of Definition 10. Finally, our query rewriting algorithm assumes that queries have full non-key to key joins. For the moment, the case in which such joins are partial, but the join graph is still a forest, remains open. Therefore, we rule this case out of $\mathcal{C}^*$ with the third condition of the definition.

Definition 10. *We say that a conjunctive query q without repeated relation symbols is in class $\mathcal{C}^*$ if:*

- *for every literal $R(\underline{\boldsymbol{x}}, \boldsymbol{y})$ of q, there is some variable x such that x occurs in $\boldsymbol{x}$, and x does not appear in any literal R' of q such that $R' \neq R$, and*
- *the join graph of q has no self-loops.*
- *if the join graph of q is a forest, then every non-key to key join of q is full.*

We will consider a class, called $\mathcal{C}_{hard}$, of all queries of $\mathcal{C}^*$ whose join graph is not a forest. We prove that the problem of computing the consistent answers for every query of $\mathcal{C}_{hard}$ is coNP-complete as follows. In Lemma 2, we prove that, given a query q such that $q \in \mathcal{C}_{hard}$, the problem of obtaining the consistent answers for q can be reduced from the problem of obtaining the consistent answers for one of three particular query families. Queries q_1, q_2, and q_3 shown in Figure 1 are in fact examples of these three query families. In Lemma 3, we prove that the problem $\texttt{CONSISTENT}(q, \Sigma)$ is coNP-complete for each such query.

Definition 11. *We say that a query q is in class $\mathcal{C}_{hard}$ if $q \in \mathcal{C}^*$ and $q \notin \mathcal{C}_{tree}$.*

Lemma 2. *Let q be a query such that $q \in \mathcal{C}_{hard}$. Then, there is a polynomial-time reduction from $\texttt{CONSISTENT}(q', \Sigma')$ to $\texttt{CONSISTENT}(q, \Sigma)$, where q' is one of the following queries:*

- $\exists w_1, \ldots, w_{m+1} : S_1(\underline{w}_{m+1}, \underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge S_3(\underline{w}_2, w_3) \wedge \ldots \wedge S_m(\underline{w}_{m-1}, w_m)$
- $\exists x, x', y : S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$
- $\exists x, x', w_1, w_2 : S_1(\underline{x}, w_1) \wedge S_2(\underline{x}', w_2) \wedge S_3(\underline{w}_1, \underline{w}_2)$

Lemma 3. *The problem $\texttt{CONSISTENT}(q, \Sigma)$ is coNP-complete for the following queries:*

- $\exists w_1, \ldots, w_{m+1} : S_1(\underline{w}_{m+1}, \underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge S_3(\underline{w}_2, w_3) \wedge \ldots \wedge S_m(\underline{w}_{m-1}, w_m)$
- $\exists x, x', y : S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$
- $\exists x, x', w_1, w_2 : S_1(\underline{x}, w_1) \wedge S_2(\underline{x}', w_2) \wedge S_3(\underline{w}_1, \underline{w}_2)$

Theorem 2. *Let q be a query such that $q \in \mathcal{C}_{hard}$. Then, $\texttt{CONSISTENT}(q, \Sigma)$ is coNP-complete in data complexity.*

In general, by Ladner's Theorem [Lad75], there are classes of coNP problems for which there is no dichotomy between P and coNP-complete problems. However, this is not the case for the class of queries that is the focus of this section. In fact, as a corollary of Theorems 1 and 2, we get a dichotomy between membership in P and coNP-completeness. Notice that, given a query q such that $q \in \mathcal{C}^*$, it can be decided in polynomial time on which side of the dichotomy the query q

falls. Under a complexity-theoretic assumption, we also get a dichotomy between first-order rewritability and coNP-completeness. An alternative approach, which we leave as future work, would be to avoid complexity-theoretic assumptions, and appeal to games arguments in order to prove first-order inexpressibility.

Corollary 1. *Let q be a query such that $q \in \mathcal{C}^*$. Then, $\texttt{CONSISTENT}(q, \Sigma)$ is either in P, or it is coNP-complete.*

Corollary 2. *Let q be a query such that $q \in \mathcal{C}^*$. Assuming $P \neq coNP$, the problem $\texttt{CONSISTENT}(q, \Sigma)$ is first-order rewritable iff $q \in \mathcal{C}_{tree}$.*

5 Related Work

The main difference between this work and others in the consistent query answering literature is our focus on producing a *first-order* rewriting. Instead of rewriting into first-order formulas, most work in the literature is based on rewriting into logic programs (e.g., [CLR03b] and [BB03]). Their focus is on obtaining correct disjunctive logic programs for (usually large) classes of queries and constraints. However, given the high complexity of disjunctive logic programming, none of these approaches focus on tractability issues.

There are two proposals in the consistent query answering literature that are based on first-order query rewriting, but they apply to very restricted classes of queries. Arenas et al. [ABC99] consider quantifier-free conjunctive queries (i.e., queries without existential quantifiers). Chomicki and Marcinkowski [CM04] propose a rewriting for *simple* conjunctive queries, which are queries where no variables are shared between literals (and therefore, there are no joins). We have presented a query rewriting for a much larger, and practical, class of queries.

Chomicki and Marcinkowski [CM04] and Calì et al. [CLR03a] thoroughly study the decidability and complexity of consistent query answering for several classes of queries and integrity constraints. In order to show intractability of a class, they take the usual approach of exhibiting one particular query of the class for which the problem is intractable. To the best of our knowledge, ours is the first dichotomy result in the area of consistent query answering.

The work on disjunctive databases [vdM98] is relevant in our context. In particular, if Σ is a set of key dependencies, the set of all repairs of an inconsistent database can be represented as a disjunctive database D in such a way that each repair corresponds to a minimal model of D. However, there are no results in the literature for first-order query rewriting over disjunctive databases. The only tractability results in this context have been given for OR-databases [IvdMV95], which are a restricted type of disjunctive databases. However, in general, given a database I possibly inconsistent with respect to a set of key dependencies, there may be no OR-database D such that all the models of D are repairs of I.

6 Conclusions and Future Work

We presented a query-rewriting algorithm for computing consistent answers. The algorithm works on a large and practical class of conjunctive queries without

repeated relation symbols. We are currently extending the algorithm in order to take into account queries with repeated relation symbols. Our algorithm works on queries with full joins whose join graph is a forest. We showed a class of queries $\mathcal{C}^*$ in which this is in fact a necessary and sufficient condition for a query to be first-order rewritable. For this class of queries, our algorithm covers all queries which are first-order rewritable. We have mentioned that, outside the class $\mathcal{C}^*$, there are some queries whose join graph is not a forest, yet they are first-order rewritable. We are working on an extension of the algorithm that considers such queries.

In this work, we assumed that the set Σ of constraints that might be violated consists of key dependencies. It would be interesting to consider foreign key dependencies as well. In this way, we would be covering the most common constraints that are supported by commercial database systems.

Acknowledgments. We would like to thank Leonid Libkin, Marcelo Arenas, Pablo Barcelo, and Ken Pu for their comments and feedback.

References

[ABC99] M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.

[AD98] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263, 1998.

[BB03] L. Bravo and L. Bertossi. Logic programs for consistently querying data integration systems. In *IJCAI*, pages 10–15, 2003.

[CLR03a] A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over inconsistent and incomplete databases. In *PODS*, pages 260–271, 2003.

[CLR03b] A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *IJCAI*, pages 16–21, 2003.

[CM04] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. To appear in Information and Computation, 2004.

[FKMP03] R. Fagin, P. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.

[Hal01] A. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.

[IvdMV95] T. Imielinski, R. van der Meyden, and K. Vadaparty. Complexity tailored design: A new design methodology for databases with incomplete information. *J. Computer and System Sciences*, 51(3):405–432, 1995.

[Lad75] R. E. Ladner. On the structure of polynomial time reducibility. *J. of the ACM*, 22(1):155–171, 1975.

[TPC03] Transaction Processing Performance Council: TPC. TPC Benchmark H (Decision Support). Standard Specification Revision 2.1.0, 2003.

[vdM98] R. van der Meyden. Logical approaches to incomplete information: A survey. In *Logics for Databases and Inf. Systems*, pages 307–356. Kluwer, 1998.