Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 73 (2007) 610–635
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

First-order query rewriting for inconsistent databases

Ariel Fuxman *, Renée J. Miller

Department of Computer Science, University of Toronto, Canada

Received 23 January 2006; received in revised form 11 July 2006

Available online 11 December 2006

Abstract

We consider the problem of retrieving consistent answers over databases that might be inconsistent with respect to a set of integrity constraints. In particular, we concentrate on sets of constraints that consist of key dependencies, and 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.

Keywords: Consistent query answering; Inconsistent data; Uncertain data

1. Introduction

Consistent query answering is the problem of retrieving "consistent" answers over databases that might be inconsistent with respect to a set of integrity constraints. Applications that have motivated the study of this problem include data integration and data exchange. Data integration is the problem of providing a unified view of data residing at different sources [11]. Data exchange is the problem of restructuring data residing under a source schema and creating an instance of a target schema that best represents the source data [7]. 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 defined by a single set of constraints that represent 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

* Corresponding author.
E-mail addresses: afuxman@cs.toronto.edu (A. Fuxman), miller@cs.toronto.edu (R.J. Miller).

0022-0000/$ – see front matter © 2006 Published by Elsevier Inc.
doi:10.1016/j.jcss.2006.10.013

increasing number 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. [1], 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 $\vec{t}$ is said to be a *consistent answer* for q on I if $\mathcal{I} \models q[\vec{t}]$, *for every* repair $\mathcal{I}$ of I. This concept is similar to that of *certain answers* used in the context of data integration [2], 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 [4,6]. 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.

1.1. 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 linear 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 nonkey attribute. Our algorithm works for conjunctive queries without repeated relation symbols (but with any number of literals and variables) whose join graph is acyclic. The queries may have projections (that is, existentially-quantified variables), which pose a particular challenge in the context of consistent query answering. In Section 3, we argue that the class that we handle is broad enough to include many queries that arise in practice.

In Section 4, we show that the class of queries considered in Section 3 is in fact a maximal class of queries, in the sense that minimal relaxations of its conditions lead to intractability. We then embark on a more ambitious goal: 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 has a cycle. 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 in our class, 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* **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 **R**. An *instance* I over **R** is a function that associates to each relation symbol R of **R** a relation $I(R)$. Given a tuple $\vec{t}$ occurring in relation $I(R)$, we denote by $R(\vec{t})$ the association between $\vec{t}$ 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. [1], 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 [1]).* 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 **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 is that 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. In particular, there is uncertainty about what John's salary is. 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 [1]. However, notice that, by definition, all repairs have a minimal distance to the inconsistent database. For example, the instances $\{(John, 2000)\}$ 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* [1], which we define next.

Definition 2 *(Consistent answer [1]).* Let **R** be a schema. Let Σ be a set of integrity constraints. Let I be an instance over **R** (possibly inconsistent with respect to Σ). Let $q(\vec{z})$ be a query over **R**. We say that a tuple $\vec{t}$ is a *consistent answer* for q with respect to Σ if $\mathcal{I} \models q[\vec{z}/\vec{t}]$, for every repair $\mathcal{I}$ of I with respect to Σ. We denote this as $\vec{t} \in \textbf{consistent}_{\Sigma}(q, I)$.

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 of Boolean queries.

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

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

We will denote the problem of computing consistent answers as CONSISTENT(q, Σ), and define it as follows.

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

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

Notice that if CONSISTENT(q, Σ) is first-order rewritable, then it is tractable. This is because the data complexity of first-order logic is in PTIME (in fact, in AC^0, which is a subset of PTIME). Thus, it can be tested in polynomial time whether $I \models Q[\vec{z}/\vec{t}]$. 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 one key dependency per relation of the schema, where a key may consist of many attributes (in particular, it may contain all attributes). 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 vector notation (e.g., $\vec{x}$, $\vec{y}$) 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 $R(\underline{\vec{c}}, \vec{d})$, we will say that $\vec{c}$ is a *key value*, and $\vec{d}$ is a *nonkey 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 $\vec{x}$ to denote variables and constants that appear at the position of key attributes, and $\vec{y}$ for variables and constants that appear at the position of nonkey attributes. Thus, conjunctive queries will be of the form:

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

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 $z_1, \ldots, z_m$ 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., **a** and **b**). We will say that there is a *join* on a variable w if w appears in two literals $R_i(\underline{\vec{x}}_i, \vec{y}_i)$ and $R_j(\underline{\vec{x}}_j, \vec{y}_j)$ such that $i \neq j$. If w occurs in $\vec{y}_i$ and $\vec{y}_j$, we say that there is a *nonkey-to-nonkey join* on w; if w occurs in $\vec{y}_i$ and $\vec{x}_j$, we say that there is a *nonkey-to-key join*; and if w occurs in $\vec{x}_i$ and $\vec{x}_j$, we say that there is a *key-to-key join*.

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.

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

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 [4,6]. 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 linear-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:

- $q_1 = \exists x, x', y.R_1(\underline{x}, y) \wedge R_2(\underline{x'}, y)$,
- $q_2 = \exists x, y.R_1(\underline{x}, y) \wedge R_2(\underline{y}, x)$,
- $q_3 = \exists x, x', w, w', z, z', m.R_1(\underline{x}, w) \wedge R_2(\underline{m}, \underline{w}, z) \wedge R_3(\underline{x'}, w') \wedge R_4(\underline{m}, \underline{w'}, z')$.

The queries presented above are rare in practice. The first consists of a join between *nonkey* 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 nonkey-to-nonkey joins are in fact necessary and sufficient conditions for intractability. Notice that such conditions are concerned with the joins in the query where at least one nonkey 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 is existentially-quantified in q, w occurs at the position of a nonkey attribute in R_i, and w occurs in R_j.

Notice that the free variables of a query do not introduce arcs to the join graph. As a special case, if all the variables of a query are free, then its join graph has no arcs. Queries without existentially-quantified variables correspond to the class of *quantifier-free* queries, and have already been shown to be first-order rewritable [1]. Handling queries with existentially quantified variables (that is, projections), is one of the main challenges addressed in our work.

As we can see in Fig. 1, the join graphs of q_1 and q_2 have a cycle. Since computing consistent answers for these two queries is coNP-hard, we will focus on queries whose join graph is acyclic. For example, the join graph of the following query is acyclic. The graph is shown in Fig. 1,

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

Additionally, when we consider how two relations, R_1 and R_2, are joined, we will require that if any of the key attributes of R_1 are joined with a nonkey attribute of R_2, then all of the key attributes of R_1 join with nonkey attributes

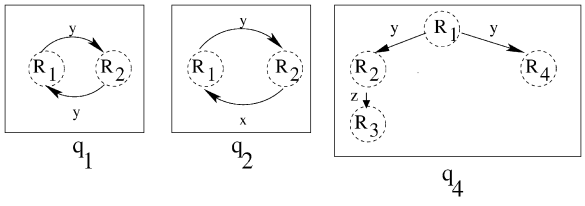

Fig. 1. Join graphs.

of R_2. We will then say that the query has *full nonkey-to-key joins*. For example, all the nonkey-to-key joins of query q_4 are full. On the other hand, in the query

$$q_3 = \exists x, x', w, w', z, z', m.R_1(\underline{x}, w) \wedge R_2(\underline{m}, \underline{w}, z) \wedge R_3(\underline{x}', w') \wedge R_4(\underline{m}, \underline{w}', z')$$

the joins between R_1 and R_2, and between R_3 and R_4, are not full since they do not involve the entire key of R_2 and R_4, respectively.

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

We observe that if G is an acyclic join graph for a query all of whose nonkey-to-key joins are full, then G must be a forest. We show this with the following proposition.

Proposition 1. *Let q be a query all of whose nonkey-to-key joins are full. Let G be the join graph of q. If G is acyclic, then G is a forest.*

Proof. Assume towards a contradiction that G is a directed acyclic graph that is not a tree. Then, there is a node v in G that receives arcs from two different nodes v_i and v_j of G. Let $R(\vec{\underline{x}}, \vec{y})$, $R_i(\vec{\underline{x}}_i, \vec{y}_i)$, and $R_j(\vec{\underline{x}}_j, \vec{y}_j)$ be the literals at the nodes of v, v_i, and v_j, respectively. Since there are arcs from v_i and v_j to v, there are variables w_i and w_j in $\vec{y}_i$ and $\vec{y}_j$, respectively, that appear in R. Since G is acyclic, w_i and w_j must appear in $\vec{x}$. Also, w_j cannot appear in a nonkey position of R_i (or, otherwise, there would be a cycle between the nodes v_i and v_j). Since there is a nonkey-to-key join from R_i to R on variable w_i, and variable w_j does not occur at a nonkey position of R_i, the join is not full; contradiction. □

We will give an algorithm for queries with an acyclic join graph and all of whose nonkey-to-key joins are full. It follows from the previous proposition that the join graph of such queries will always be a forest. We call the class of such queries $\mathcal{C}_{forest}$, and define it next.

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

A fundamental observation about $\mathcal{C}_{forest}$ is that it is a very common, practical class of queries. Arguably, the most used forms of joins are from a set of nonkey 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 repeated relation symbols.

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 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 nonkey value, and does not appear with any other constant a' such that $a' \neq a$. This can be checked with

[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.

a formula $Q_{consist}(x) = \forall y'.R_1(\underline{x}, y') \rightarrow y' = \mathbf{a}$. 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 $\texttt{CONSISTENT}(q, \Sigma)$. 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* nonkey 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 $Q_{consist}(x) = \forall y.R_1(\underline{x}, y) \rightarrow \exists z.R_2(\underline{y}, z)$. We will soon show that a query rewriting Q for q can be obtained as the conjunction of q and $Q_{consist}$, as follows:

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

We now proceed to present `RewriteConsistent`, our query rewriting algorithm (shown in Figs. 2–4). Given a query q such that $q \in \mathcal{C}_{forest}$ and a set of key constraints Σ (containing one key per relation), the algorithm $\texttt{RewriteConsistent}(q, \Sigma)$ returns a first-order rewriting Q for the problem of obtaining the consistent answers for q with respect to Σ. The main body of the algorithm `RewriteConsistent` is shown in Fig. 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 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 $q \in \mathcal{C}_{forest}$, the join graph is a forest). The recursive algorithm is called `RewriteTree`, and is shown in Fig. 3.

The first part of `RewriteTree` produces a rewriting Q_{local} for the literal $R(\vec{\underline{x}}, \vec{y})$ at the root of the tree. This rewriting is done independently of the rest of the query, and it is produced by the algorithm `RewriteLocal`. We show this algorithm in Fig. 4. The query Q_{local} deals with the constants that appear in $\vec{y}$ in the same way as we illustrated in Example 2. It also deals with the free variables of the query, as we show in the next example.

Example 4. 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 **a** is replaced by the free variable y. The algorithm `RewriteLocal` creates a new, universally-quantified variable y' for y, and equates y' to y. The resulting query rewriting for q is the following:

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

Algorithm $\texttt{RewriteConsistent}(q, \Sigma)$
Input: $q(\vec{z})$, a query of the form $\exists \vec{w}.\phi(\vec{w}, \vec{z})$
Σ, a set of key constraints, one per relation used in q
Let G be the join graph of q
Let $T_1, \ldots, T_m$ be the connected components of G
for $i := 1$ to m **do**
Let $R_i(\vec{\underline{x}}_i, \vec{y}_i)$ be the literal at the root of T_i
Let ϕ_i be the conjunction of literals of T_i
Let $\vec{w}_i = \{w\colon\ w$ is a variable that occurs in ϕ_i and $\vec{w}$, and $w \notin \vec{x}_i\}$
Let $\vec{z}_i = \{z\colon\ z$ is a variable that occurs in ϕ_i and $\vec{z}$, and $z \notin \vec{x}_i\}$
Let $q_i(\vec{x}_i, \vec{z}_i) = \exists \vec{w}_i.\phi_i(\vec{x}_i, \vec{w}_i, \vec{z}_i)$
Let $Q_i(\vec{x}_i, \vec{z}_i) = \texttt{RewriteTree}(q_i, \Sigma)$
end for
Let $Q_{consist}(\vec{w}, \vec{z}) = \bigwedge_{i=1,\ldots,m} Q_i(\vec{x}_i, \vec{z}_i)$
Let $Q(\vec{z}) = \exists \vec{w}.(\phi(\vec{w}, \vec{z}) \wedge Q_{consist}(\vec{w}, \vec{z}))$
return Q

Fig. 2. Query rewriting algorithm.

Algorithm RewriteTree(q, Σ)
Input: $q(\vec{x}, \vec{z})$, a query in $\mathcal{C}_{forest}$ of the form $\exists\vec{w}.\phi(\vec{x}, \vec{w}, \vec{z})$,
whose join graph T is a tree with root literal $R(\underline{\vec{x}}, \vec{y})$
Σ, a set of key constraints, one per relation
Let T be the join graph of q
Let $R(\underline{\vec{x}}, \vec{y})$ be the literal at the root node of T
Let $q_{local}(\vec{x}, \vec{z}) = \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})$
Let $Q_{local}(\vec{x}, \vec{z}) =$ RewriteLocal(q_{local}, Σ)
if ϕ has exactly one literal **then**
$Q = Q_{local}$
else
Let $R_1(\underline{\vec{x}}_1, \vec{y}_1), \ldots, R_m(\underline{\vec{x}}_m, \vec{y}_m)$ be the children of R in T
for $i := 1$ to m **do**
Let T_i be the subtree of T rooted at R_i
Let ϕ_i be the conjunction of literals of T_i
Let $\vec{w}_i = \{w\colon\ w$ is a variable that occurs in ϕ_i and $\vec{w}$, and $w \notin \vec{x}_i\}$
Let $\vec{z}_i = \{z\colon\ z$ is a variable that occurs in ϕ_i and $\vec{z}$, and $z \notin \vec{x}_i\}$
Let $q_i(\vec{x}_i, \vec{z}_i) = \exists\vec{w}_i.\phi_i(\vec{x}_i, \vec{w}_i, \vec{z}_i)$
Let $Q_i(\vec{x}_i, \vec{z}_i) =$ RewriteTree(q_i, Σ)
end for
Let $\vec{y}_0 = \{y\colon\ y$ is a variable that occurs in $\vec{y}$ and $\vec{w}$, and $y \notin \vec{x}\}$
Let $Q(\vec{x}, \vec{z}) = Q_{local}(\vec{x}, \vec{z}) \wedge \forall\vec{y}_0.R(\underline{\vec{x}}, \vec{y}) \rightarrow \bigwedge_{i=1,\ldots,m} Q_i(\vec{x}_i, \vec{z}_i)$
end if
return Q

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

The second part of RewriteTree recursively creates a query Q_i for each subtree T_i of T rooted at R. Let $\vec{y}_0$ be the variables at nonkey positions of R (excluding those that also appear in $\vec{x}$). Then, one of the conjuncts of the rewritten query returned by RewriteTree is of the form $\forall\vec{y}_0.R(\underline{\vec{x}}, \vec{y}) \rightarrow \bigwedge_{i=1,\ldots,m} Q_i(\vec{x}_i, \vec{z}_i)$. Notice that thevariables of $\vec{y}_0$ (i.e., the variables at nonkey positions of the root literal R) are universally quantified. The intuition behind this is that, as we illustrated in Example 3, the query must be satisfied by all the nonkey values of a given key.

The next example illustrates an application of the algorithm.

Example 5. Let q be the query q_4 introduced in Section 3.1,

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

The join graph T of q is shown in Fig. 1. In this case, T consists of one connected component, which is a tree. Let q_1 be the query $q_1(x, w) = \exists y, z.R_1(\underline{x}, y) \wedge R_2(\underline{y}, z) \wedge R_3(\underline{z}, w) \wedge R_4(\underline{y}, \mathbf{a})$; let q_2 be the query $q_2(y, w) = \exists z.R_2(\underline{y}, z) \wedge R_3(\underline{z}, w)$; let q_3 be the query $q_3(z, w) = R_3(\underline{z}, w)$; and let $q_4(y) = R_4(\underline{y}, \mathbf{a})$. The first-order query rewriting Q of q is obtained by applying the algorithm RewriteConsistent(q, Σ) as follows:

$$\begin{aligned}
Q(w) &= \texttt{RewriteConsistent}(q, \Sigma)\\
&= \exists x, y, z.R_1(\underline{x}, y) \wedge R_2(\underline{y}, z) \wedge R_3(\underline{z}, w) \wedge R_4(\underline{y}, \mathbf{a}) \wedge Q_{consist}(x, w),\\
Q_{consist}(x, w) &= \texttt{RewriteTree}(q_1, \Sigma) = \exists y.R_1(\underline{x}, y) \wedge \forall y.R_1(\underline{x}, y) \rightarrow \big(Q_2(y, w) \wedge Q_4(y)\big),\\
Q_2(y, w) &= \texttt{RewriteTree}(q_2, \Sigma) = \exists z.R_2(\underline{y}, z) \wedge \forall z.R_2\big(\underline{y}, z\big) \rightarrow Q_3(z, w),\\
Q_3(z, w) &= \texttt{RewriteTree}(q_3, \Sigma) = R_3(\underline{z}, w) \wedge \forall w'.\big(R_3(\underline{z}, w') \rightarrow w' = w\big),\\
Q_4(y) &= \texttt{RewriteTree}(q_4, \Sigma) = R_4(\underline{y}, \mathbf{a}) \wedge \forall u'.\big(R_4(\underline{y}, u') \rightarrow u' = \mathbf{a}\big).
\end{aligned}$$

Notice that we reuse variables in the rewritten queries. In particular, each existentially-quantified variable of q that appears at a nonkey position in q is first existentially quantified, and then universally quantified in the rewriting Q.

Algorithm RewriteLocal(q, Σ)
Input: $q(\vec{x}, \vec{z})$, a query of the form $\exists \vec{w}.R(\underline{\vec{x}}, \vec{y})$, where
none of the variables of $\vec{w}$ appear in $\vec{x}$
Σ, a set of key constraints
Let σ be an injective function that maps natural numbers to variables not present in R
Initialize Eq as an empty set
for each position p of $\vec{y}$ **do**
Let w be the variable that appears at position p of $\vec{y}$
Let $z = \sigma(p)$
if there is a constant d at position p of $\vec{y}$ **then**
Add the equality $z = d$ to Eq
end if
if w appears in $\vec{x}$ or w appears in $\vec{z}$ **then**
Add the equality $z = w$ to Eq
end if
for every position p' of $\vec{y}$ such that $p \neq p'$ and w occurs in $\vec{y}$ at position p' **do**
Let $z' = \sigma(p')$
Add the equality $z = z'$ to Eq
end for
end for
if $Eq \neq \emptyset$ **then**
Let $\vec{y}^*$ be a vector of variables of the same arity as $\vec{y}$, and
such that if z is at position p of $\vec{y}^*$, then $\sigma(p) = z$
Let Q_{eq} be the conjunction of the equalities of Eq
Let $Q_{local}(\vec{x}, \vec{z}) = \exists \vec{w}.R(\underline{\vec{x}}, \vec{y}) \wedge \forall \vec{y}^*.R(\underline{\vec{x}}, \vec{y}^*) \rightarrow Q_{eq}$
else
Let $Q_{local}(\vec{x}, \vec{z}) = \exists \vec{w}.R(\underline{\vec{x}}, \vec{w})$
end if
return Q_{local}

Fig. 4. Query rewriting for a given literal.

Recall that we do not allow queries with repeated relation symbols in the class $\mathcal{C}_{forest}$. We now give an example of a query with repeated relation symbols for which our algorithm fails to give the consistent answer. Although not addressed in this paper, it would be interesting to characterize the class of queries with repeated relation symbols for which our algorithm is indeed correct.

Example 6. Let **R** be a schema with one relation $r(A, B, C)$, where the attribute A is the key of the relation. Let q be the query $q = \exists x, y, z.r(\underline{x}, y, \mathbf{a}) \wedge r(\underline{y}, z, \mathbf{b})$. If we apply our query rewriting algorithm, we obtain the following:

$$Q = \exists x, y, z.r(\underline{x}, y, \mathbf{a}) \wedge r(\underline{y}, z, \mathbf{b}) \wedge \forall y', z'.\big(r(\underline{x}, y', z') \rightarrow z' = \mathbf{a}\big)$$
$$\wedge \forall y.\big(r(\underline{x}, y, \mathbf{a}) \rightarrow \exists z.r(\underline{y}, z, \mathbf{b}) \wedge \forall z', w'.\big(r(\underline{y}, z', w') \rightarrow z' = \mathbf{b}\big)\big).$$

Let $I = \{r(\underline{c}, d, a), r(\underline{d}, e, b), r(\underline{d}, f, a), r(\underline{f}, g, b)\}$. In this case, there are two repairs $\mathcal{I}_1 = \{r(\underline{c}, d, a), r(\underline{d}, e, b), r(\underline{f}, g, b)\}$ and $\mathcal{I}_2 = \{r(\underline{c}, d, a), r(\underline{d}, f, a), r(\underline{f}, g, b)\}$. Clearly, $\mathcal{I}_1 \models q$ and $\mathcal{I}_2 \models q$. However, $I \not\models Q$.

We finish this section by pointing out that the complexity of the query rewriting algorithm is linear in the number of literals of the input query. To see this, notice that the algorithm visits each node of the join graph exactly once.

3.3. Correctness proof

In this section, we show that the algorithm presented in the previous section is correct for all queries in the class $\mathcal{C}_{forest}$. In particular, we prove the following theorem.

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

Then, $I \models Q(\vec{z})[\vec{z}/\vec{t}]$ *iff* $\vec{t} \in$ **consistent**$_\Sigma(q, I)$.

Our proofs rely on a few simple properties of repairs of inconsistent databases where the set of integrity constraints contain a single key dependency per relation. We establish these properties in Section 3.3.1. We use these properties to prove the correctness of `RewriteLocal` (Section 3.3.2). `RewriteLocal` considers queries without joins (i.e., with a single literal), but with arbitrary selections, and with projections on any subset of the nonkey attributes (more precisely, any of the nonkey attributes may be projected out of the query result). We consider here only equality selections, but it is quite easy to see how to extend this algorithm to more general selection conditions.

In Section 3.3.3, we show two structural properties that are fundamental in order to guarantee the correctness of the algorithm. First, we show that for queries in $\mathcal{C}_{forest}$, the literals from distinct trees of the join graph may only share variables that appear as key attributes at the root of their trees. Second, we consider a subclass of $\mathcal{C}_{forest}$, where the join graph of the queries is a single tree, and in which the free variables are exactly the variables at the key of the root of this tree. For this class of queries, we show that for any inconsistent instance I, there is a repair $\mathcal{M}$ such that if $\mathcal{M} \models q(\vec{x})[\vec{x}/\vec{c}]$, then **consistent**$_\Sigma(q(\vec{x})[\vec{x}/\vec{c}], I) =$ `true`. This enables the algorithm to *independently* consider each instantiation of the variables for the key of the root literal. Finally, in Section 3.3.4, we prove the correctness of the `RewriteTree` and `RewriteConsistent` algorithms.

3.3.1. Properties of repairs

We first show a few important properties of repairs when the set of integrity constraints consist of one key dependency per relation.

Proposition 2. *Let* I *be an instance. Let* $\mathcal{I}$ *be a repair of* I *wrt* Σ. *Then* $\mathcal{I} \subseteq I$.

Proof. Let I' be an instance such that $I' \models \Sigma$. Assume that there is a tuple $\vec{t}$ such that $\vec{t} \in I'$ and $\vec{t} \notin I$. Let $I'' = I' - \{\vec{t}\}$. It is easy to see that by removing tuples from an instance, we do not introduce violations with respect to a set of key dependencies. Hence, $I'' \models \Sigma$. Clearly, $\Delta(I, I'') \subset \Delta(I, I')$. Therefore, I' is not a repair of I wrt Σ. □

Proposition 3. *Let* I *be an instance. Let* $\mathcal{I}$ *be a repair of* I *wrt* Σ. *Let* $R(\underline{\vec{c}}, \vec{d})$ *be a tuple of* I. *Then, there exists some* $\vec{d}'$ *such that* $R(\underline{\vec{c}}, \vec{d}')$ *is a tuple of* $\mathcal{I}$.

Proof. Let I' be an instance such that $I' \models \Sigma$ and $R(\underline{\vec{c}}, \vec{d}') \notin I'$, for every $\vec{d}'$. Let $I'' = I' \cup \{R(\underline{\vec{c}}, \vec{d})\}$. Since $R(\underline{\vec{c}}, \vec{d}') \notin I'$ for every $\vec{d}'$, $I'' \models \Sigma$. Clearly, $\Delta(I, I'') = \Delta(I, I') - \{R(\underline{\vec{c}}, \vec{d})\}$. Since $\Delta(I, I'') \subset \Delta(I, I')$, I' is not a repair of I wrt Σ. □

Proposition 4. *Let* I *be an instance. Let* $R(\underline{\vec{c}}, \vec{d})$ *be a tuple of* I. *Then, there exists some repair* $\mathcal{I}$ *of* I *such that* $R(\underline{\vec{c}}, \vec{d}) \in \mathcal{I}$.

Proof. Let $\mathcal{I}^*$ be a repair of I wrt Σ. By Proposition 3, there exists $\vec{d}'$ such that $R(\underline{\vec{c}}, \vec{d}') \in \mathcal{I}^*$. Let $I' = \mathcal{I}^* - \{R(\underline{\vec{c}}, \vec{d}')\} \cup \{R(\underline{\vec{c}}, \vec{d})\}$. Since $\mathcal{I}^*$ is a repair, $\mathcal{I}^* \models \Sigma$. Since I' does not introduce any violation to the key dependencies of Σ, $I' \models \Sigma$. Assume that I' is not a repair of I. Then, there exists a repair $\mathcal{I}^{**}$ of I such that $\Delta(I, \mathcal{I}^{**}) \subset \Delta(I, I')$. By Proposition 2, $\mathcal{I}^* \subseteq I$, and thus $I' \subset I$. Furthermore, by Proposition 2, $\mathcal{I}^{**} \subseteq I$. Thus, $I - \mathcal{I}^{**} \subset I - I'$. Therefore, $I' \subset \mathcal{I}^{**}$. Let $I'' = \mathcal{I}^{**} - \{R(\underline{\vec{c}}, \vec{d})\} \cup \{R(\underline{\vec{c}}, \vec{d}')\}$. Clearly, $\mathcal{I}^* \subset I''$. Thus, $\mathcal{I}^*$ is not a repair; contradiction. □

3.3.2. Correctness of `RewriteLocal`

We now give a correctness proof of `RewriteLocal`, the algorithm that produces a rewriting for a query with a single literal where all key attributes ($\vec{x}$) are free in the query.

Lemma 1. *Let* $q(\vec{x}, \vec{z})$ *be a query of the form* $\exists \vec{w}.R(\underline{\vec{x}}, \vec{y})$. *Let* I *be an instance. Let* $Q_{local}(\vec{x}, \vec{z})$ *be the first-order query returned by* `RewriteLocal`(q, Σ).

Then, $I \models Q_{local}(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$ *iff* **consistent**$_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) =$ `true`.

Proof. ($\Rightarrow$) Assume that $I \models Q_{local}(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, there is a tuple $R(\underline{\vec{c}},\vec{d})$ such that $\{R(\underline{\vec{c}},\vec{d})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Assume towards a contradiction that $\textbf{consistent}_{\Sigma}(\exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) =$ `false`. Then, there is some repair $\mathcal{I}$ such that $\mathcal{I} \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. By Proposition 3, there is a tuple $R(\underline{\vec{c}},\vec{d'})$ in $\mathcal{I}$.

Following the construction of Q_{local} in `RewriteLocal`, let σ be an injective function that maps natural numbers to variables not present in R. Let $\vec{y}^*$ be a vector of variables of the same arity as $\vec{y}$ and such that if z is at position p of $\vec{y}^*$, then $\sigma(p) = z$. Let ν and ν' be valuations for the variables of $\vec{x}$ and $\vec{y}^*$ such that $\nu(\vec{x}) = \vec{c}$, $\nu(\vec{y}^*) = \vec{d}$, $\nu'(\vec{x}) = \vec{c}$, and $\nu'(\vec{y}^*) = \vec{d'}$.

Since $\{R(\underline{\vec{c}},\vec{d})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$ and $\{R(\underline{\vec{c}},\vec{d'})\} \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, there is some variable z at some position p of $\vec{y}^*$ such that

(1) $\nu(z) \neq \nu'(z)$, and there is a constant at position p in $\vec{y}$; or
(2) $\nu(z) \neq \nu'(z)$, and there is some variable w such that w occurs at position p of $\vec{y}$, and w occurs in either $\vec{x}$ or $\vec{z}$; or
(3) there are variables w and z', and a position p' such that w occurs at position p of $\vec{y}$, w occurs at position p' of $\vec{y}$, $p \neq p'$, $z' = \sigma(p')$, and $\nu'(z) \neq \nu'(z')$.

Assume (1) that there is a constant d at position p in $\vec{y}$. Since $\{R(\underline{\vec{c}},\vec{d})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, $\nu(z) = d$. Since $\nu(z) \neq \nu'(z)$, there is a constant d' such that $d \neq d'$ and $\nu'(z) = d'$. Notice in the algorithm `RewriteLocal` that since $I \models Q_{local}(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, we have that $I \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = d$. Since $\mathcal{I} \subseteq I$, $R(\underline{\vec{c}},\vec{d'}) \in I$. Thus, $\{R(\underline{\vec{c}},\vec{d'})\} \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = d$. Therefore, $\nu'(z) = d$; contradiction.

Assume (2) that there is some variable w such that w occurs at position p of $\vec{y}$, and w occurs in either $\vec{x}$ or in $\vec{z}$. Let $c = \nu(w)$. Since $\{R(\underline{\vec{c}},\vec{d})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y}^*)[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, $\nu(z) = c$. Since $\nu(z) \neq \nu'(z)$, $\nu'(z) \neq c$. Notice in the algorithm `RewriteLocal` that since $I \models Q_{local}(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, we have that $I \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = w[w/c]$. Since $\mathcal{I} \subseteq I$, $R(\underline{\vec{c}},\vec{d'}) \in I$. Thus, $\{R(\underline{\vec{c}},\vec{d'})\} \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = w[w/c]$. Therefore, $\nu'(z) = c$; contradiction.

Assume (3) that there are variables w and z', and a position p' such that w occurs at position p of $\vec{y}$, w occurs at position p' of $\vec{y}$, $p \neq p'$, $z' = \sigma(p')$, and $\nu'(z) \neq \nu'(z')$. Notice in the algorithm `RewriteLocal` that since $I \models Q_{local}(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$, we have that $I \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = z'$. Since $\mathcal{I} \subseteq I$, $R(\underline{\vec{c}},\vec{d'}) \in I$. Thus, $\{R(\underline{\vec{c}},\vec{d'})\} \models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = z'$. Therefore, $\nu'(z) = \nu'(z')$; contradiction.

($\Leftarrow$) Assume that $\textbf{consistent}_{\Sigma}(q(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) =$ `true`. Assume towards a contradiction that $I \not\models Q_{local}(\vec{x},\vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, at least one of the following conditions holds:

(1) $I \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; or
(2) there is a constant d at position p in $\vec{y}$ and a variable z such that $z = \sigma(p)$ and $I \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = d[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; or
(3) there is some variable w such that w occurs at position p of $\vec{y}$, w occurs in either $\vec{x}$ or $\vec{z}$, and $I \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = w[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; or
(4) there is some variable w that occurs at position p of $\vec{y}$, and at a position p' of $\vec{y}$ such that $p \neq p'$, $\sigma(p) = z$, $\sigma(p') = z'$ and $I \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = z'[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$.

Assume that $I \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let $\mathcal{I}$ be an arbitrary repair of I. Since $\mathcal{I} \subseteq I$, $\mathcal{I} \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; contradiction.

Suppose that $I \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Furthermore, assume that there is a constant d at position p in $\vec{y}$ and a variable z such that $z = \sigma(p)$ and $I \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = d[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, there is a tuple $R(\underline{\vec{c}},\vec{d})$ in I such that $\{R(\underline{\vec{c}},\vec{d})\} \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = d[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. This means that there is some constant e at position p of $\vec{d}$ such that $d \neq e$. Thus, $\{R(\underline{\vec{c}},\vec{d})\} \not\models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. By Proposition 4, there is a repair $\mathcal{I}$ of I such that $R(\underline{\vec{c}},\vec{d}) \in I$. Assume that $\mathcal{I} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let $R(\underline{\vec{c}},\vec{d'})$ be a tuple of $\mathcal{I}$ such that $\{R(\underline{\vec{c}},\vec{d'})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Since $\mathcal{I}$ is a repair of I, $\mathcal{I}$ satisfies the key constraints of Σ. Thus, $\vec{d} = \vec{d'}$. Therefore, $\{R(\underline{\vec{c}},\vec{d})\} \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; contradiction.

Suppose that $I \models \exists\vec{w}.R(\underline{\vec{x}},\vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Furthermore, assume that there is some variable w such that w occurs at position p of $\vec{y}$, w occurs in either $\vec{x}$ or $\vec{z}$, and $I \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = w[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, there is a tuple $R(\underline{\vec{c}},\vec{d})$ in I such that $\{R(\underline{\vec{c}},\vec{d})\} \not\models \forall\vec{y}^*.R(\underline{\vec{x}},\vec{y}^*) \rightarrow z = w[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let ν be a valuation for the variables of $\vec{x}$ and

$\vec{z}$ such that $\nu(\vec{x}) = \vec{c}$ and $\nu(\vec{z}) = \vec{t}$. Let $c = \nu(w)$. Then, there is some constant e at position p of $\vec{d}$ such that $c \neq e$. Thus, $\{R(\vec{c}, \vec{d})\} \not\models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. By Proposition 4, there is a repair $\mathcal{I}$ of I such that $R(\vec{c}, \vec{d}) \in I$. Assume that $\mathcal{I} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let $R(\vec{c}, \vec{d'})$ be a tuple of $\mathcal{I}$ such that $\{R(\vec{c}, \vec{d'})\} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Since $\mathcal{I}$ is a repair of I, $\mathcal{I}$ satisfies the key constraints of Σ. Thus, $\vec{d} = \vec{d'}$. Therefore, $\{R(\vec{c}, \vec{d})\} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; contradiction.

Suppose that $I \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Furthermore, assume that there is some variable w that occurs at position p of $\vec{y}$, and at a position p' of $\vec{y}$ such that $p \neq p'$, $\sigma(p) = z$, $\sigma(p') = z'$ and $I \not\models \forall \vec{y}^*.R(\underline{\vec{x}}, \vec{y}^*) \rightarrow z = z'[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, there is a tuple $R(\vec{c}, \vec{d})$ in I such that $\{R(\vec{c}, \vec{d})\} \not\models \forall \vec{y}^*.R(\underline{\vec{x}}, \vec{y}^*) \rightarrow z = z'[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let ν be a valuation for the variables of $\vec{y}^*$ such that $\nu(\vec{y}^*) = \vec{d}$. Then, there are constants d and e at the respective positions p and p' of $\vec{d}$ such that $d \neq e$. Thus, $\{R(\vec{c}, \vec{d})\} \not\models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. By Proposition 4, there is a repair $\mathcal{I}$ of I such that $R(\vec{c}, \vec{d}) \in I$. Assume that $\mathcal{I} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let $R(\vec{c}, \vec{d'})$ be a tuple of $\mathcal{I}$ such that $\{R(\vec{c}, \vec{d'})\} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Since $\mathcal{I}$ is a repair of I, $\mathcal{I}$ satisfies the key constraints of Σ. Thus, $\vec{d} = \vec{d'}$. Therefore, $\{R(\vec{c}, \vec{d})\} \models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$; contradiction. □

3.3.3. Structural properties of $\mathcal{C}_{forest}$

In the next lemma, we show a structural property of the queries in $\mathcal{C}_{forest}$ that is fundamental in order to guarantee the correctness of the algorithm. In particular, we show that distinct trees of the join graph may only share free variables (which do not contribute arcs to the join graph) or variables that appear as key attributes at the root of their trees.

Lemma 2. *Let $q(\vec{z})$ be a query such that $q \in \mathcal{C}_{forest}$. Let G be the join graph of q. Let T_i and T_j be distinct connected components of G. Let $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ and $R_j(\underline{\vec{x}_j}, \vec{y}_j)$ be the literals at the roots of T_i and T_j, respectively. Let w be a variable that occurs in a literal of both T_i and T_j. Then, either w is free ($w \in \vec{z}$) or w is in the key of the roots of both trees ($w \in \vec{x}_i \cap \vec{x}_j$).*

Proof. Let $\vec{w}_i = \{w\colon\ w$ is a variable that occurs in some literal of T_i, $w \notin \vec{x}_i$ and $w \notin \vec{z}\}$. Let $\vec{w}_j = \{w\colon\ w$ is a variable that occurs in some literal of T_j, $w \notin \vec{x}_j$ and $w \notin \vec{z}\}$. Assume that there is some variable w such that w appears in $\vec{w}_i$ and $\vec{w}_j$. Let $S_1(\underline{\vec{u}_1}, \vec{v}_1)$ and $S_2(\underline{\vec{u}_2}, \vec{v}_2)$ be literals of T_i and T_j, respectively, such that w appears in S_1 and S_2. We must now consider the next two cases. First, suppose that w occurs in $\vec{v}_1$. Then, by definition of join graph, there is an arc from S_1 to S_2 in G. But S_1 and S_2 are in distinct connected components of G; contradiction. Second, suppose that w occurs in $\vec{u}_1$. By definition of w_i, S_1 is not at the root of T_i (i.e., $S_1 \neq R_i$). Hence, there must be a nonkey-to-key join from another literal, $S_3(\underline{\vec{u}_3}, \vec{v}_3)$, in T_i to S_1. Since q is in $\mathcal{C}_{forest}$, all the nonkey-to-key joins of q are full. Thus, the variable w also appears in a nonkey position in $\vec{v}_3$. Hence, there must be an arc in the join graph from S_3 to S_2. But S_2 and S_3 are in distinct connected components of G; contradiction. □

In Lemma 3 below, we show the following result. Let $q(\vec{x})$ be a query in $\mathcal{C}_{forest}$, whose join graph T is a tree, and where the free variables $\vec{x}$ are exactly the variables of the key of T's root. Let I be an instance. We show that there is a repair $\mathcal{M}$ of I such that for all $\vec{c}$, if $\mathcal{M} \models q(\vec{x})[\vec{x}/\vec{c}]$, then $\textbf{consistent}_{\Sigma}(q(\vec{x})[\vec{x}/\vec{c}], I) = \texttt{true}$. This is a fundamental property for the following reason. Consider a Boolean query $q = \exists \vec{x}, \vec{w}.\phi(\vec{x}, \vec{w})$ and a query $q'(\vec{x}) = \exists \vec{w}.\phi(\vec{x}, \vec{w})$. That is, q and q' have the same literals, but some of the (existentially-quantified) variables of q are free in q'. Suppose that we would like to check whether $\textbf{consistent}_{\Sigma}(q, I) = \texttt{true}$. This holds if, for every repair $\mathcal{I}$ of I, $\mathcal{I} \models q$. In particular, since $\mathcal{M}$ is a repair of I, $\mathcal{M} \models q$. Thus, there is some $\vec{c}$ such that $\mathcal{M} \models q'(\vec{x})[\vec{x}/\vec{c}]$. By Lemma 3 below, it follows that $\textbf{consistent}_{\Sigma}(q'(\vec{x})[\vec{x}/\vec{c}], I) = \texttt{true}$. This property will be exploited in the design of our algorithms in order to check the consistency of each tuple of $\vec{x}$ *independently*. Notice that the property does not hold in general for conjunctive queries, as we show in the next example. However, it does hold for the queries that satisfy the conditions of Lemma 3.

Example 7. Let q_{nk} be the Boolean query $\exists x, x', y.R_1(\underline{x}, y) \wedge R_2(\underline{x}', y)$. Notice that q_{nk} is not in $\mathcal{C}_{forest}$ because it contains a nonkey-to-nonkey join. Let I be an instance such that $I = \{R_1(\underline{a}_1, b_1), R_1(\underline{a}_1, b_2), R_1(\underline{a}_2, b_3), R_1(\underline{a}_2, b_4), R_1(\underline{a}_3, b_5), R_1(\underline{a}_3, b_3), R_2(\underline{c}_1, b_1), R_2(\underline{c}_1, b_3), R_2(\underline{c}_2, b_4), R_2(\underline{c}_2, b_5), R_2(\underline{c}_3, b_2), R_2(\underline{c}_3, b_3)\}$. It can be checked that for every repair $\mathcal{I}$ of I, $\mathcal{I} \models q_{nk}$.

Now, consider the query $q'_{nk}(x) = \exists x', y.R_1(\underline{x}, y) \wedge R_2(\underline{x}', y)$. That is, q_{nk} and q'_{nk} only differ in the fact that x is existentially-quantified in the former, and free in the latter. Let $\mathcal{I}_1$ be repair of I such that $\mathcal{I}_1 = \{R_1(\underline{a}_1, b_1), R_1(\underline{a}_2, b_3), R_1(\underline{a}_3, b_5), R_2(\underline{c}_1, b_3), R_2(\underline{c}_2, b_4), R_2(\underline{c}_3, b_3)\}$. Let $\mathcal{I}_2$ be a repair of I such that $\mathcal{I}_2 = \{R_1(\underline{a}_1, b_1), R_1(\underline{a}_2, b_3), R_1(\underline{a}_3, b_5), R_2(\underline{c}_1, b_1), R_2(\underline{c}_2, b_4), R_2(\underline{c}_3, b_2)\}$. Notice that $(a_1) \notin q'_{nk}(\mathcal{I}_1)$, $(a_2) \notin q'_{nk}(\mathcal{I}_2)$, and $(a_3) \notin q'_{nk}(\mathcal{I}_1)$. Thus, even though $\textbf{consistent}_{\Sigma}(q_{nk}, I) = \texttt{true}$, we have that $\textbf{consistent}_{\Sigma}(q'_{nk}(x)[x/a], I) = \texttt{false}$, for every a. Therefore, it is not possible to check whether $\textbf{consistent}_{\Sigma}(q_{nk}, I) = \texttt{true}$ by independently checking all instantiations of the free variables of q'_{nk}.

Lemma 3. *Let $q(\vec{x})$ be a query in $\mathcal{C}_{forest}$, whose join graph T is a tree and where $R(\underline{\vec{x}}, \vec{y})$ is the literal at the root of T. Let I be an instance. Then, there is a repair $\mathcal{M}$ such that for all $\vec{c}$ if $\mathcal{M} \models q(\vec{x})[\vec{x}/\vec{c}]$, then* $\textbf{consistent}_{\Sigma}(q(\vec{x})[\vec{x}/\vec{c}], I) = \texttt{true}$.

Algorithm `BuildRepair`
Input: $q(\vec{x})$, a query in $\mathcal{C}_{forest}$ of the form $\exists \vec{w}.\phi(\vec{w}, \vec{x})$, whose join graph T is a tree with root literal $R(\underline{\vec{x}}, \vec{y})$
Σ, a set of key constraints, one per relation
I, an instance
Initialize $\mathcal{M}$ as an empty instance
if ϕ has exactly one literal **then**
  for each $\vec{c}$ such that there is some $R(\underline{\vec{c}}, \vec{d})$ in I **do**
    if there is some $\vec{d}$ such that $R(\underline{\vec{c}}, \vec{d}) \in I$, and $\{R(\underline{\vec{c}}, \vec{d})\} \not\models \exists \vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$ **then**
      Let $\vec{t} = R(\underline{\vec{c}}, \vec{d})$
    else
      Let $\vec{t}$ be any tuple of I such that $\vec{t} = R(\underline{\vec{c}}, \vec{d})$, for some $\vec{d}$
    end if
    Add $\vec{t}$ to $\mathcal{M}$
  end for
else
  /* ϕ has more than one literal*/
  Let $S_1, \ldots, S_m$ be the children of R in T
  for $j := 1$ to m **do**
    Let T_j be the subtree of T whose root is S_j
    Let ϕ_j be the conjunction of literals of T_j
    Let $\vec{w}_j = \{w$: w is a variable that occurs in ϕ_j and $\vec{w}$, and $w \notin \vec{x}_j\}$
    Let $q_j(\vec{x}_j) = \exists \vec{w}_j.\phi_j(\vec{x}_j, \vec{w}_j)$
    Let $\mathcal{M}_j = \texttt{BuildRepair}(q_j, I)$
    Add $\mathcal{M}_j$ to $\mathcal{M}$
  end for
  for each $\vec{c}$ such that there is some $R(\underline{\vec{c}}, \vec{d})$ in I **do**
    if there is some $\vec{d}$, some j, some valuation ν for the variables of $\vec{y}$, and some $\vec{c}_j$ such that $R(\underline{\vec{c}}, \vec{d}) \in I$, $\nu(\vec{y}) = \vec{d}$, $\nu(\vec{x}_j) = \vec{c}_j$, and $\mathcal{M}_j \not\models q_j(\vec{x}_j)[\vec{x}_j/\vec{c}_j]$ **then**
      Let $\vec{t} = R(\underline{\vec{c}}, \vec{d})$
    else
      Let $\vec{t}$ be any tuple of I such that $\vec{t} = R(\underline{\vec{c}}, \vec{d})$, for some $\vec{d}$
    end if
    Add $\vec{t}$ to $\mathcal{M}$
  end for
end if

Fig. 5. Algorithm `BuildRepair`.

Proof. Let $\mathcal{M}$ be the instance built by invoking the procedure `BuildRepair`(q, I) given in Fig. 5. Assume that q is of the form $q(\vec{x}) = \exists\vec{w}.\phi(\vec{w}, \vec{x})$. We will prove the claim by induction on the number of literals of ϕ.

Base case. Assume that ϕ consists of exactly one literal $R(\underline{\vec{x}}, \vec{y})$. Let $\vec{t}$ be the tuple selected by the algorithm in the iteration for literal R and the vector of values $\vec{c}$. Assume towards a contradiction that **consistent**$_\Sigma(\exists\vec{w}.R(\vec{x}, \vec{w})[\vec{x}/\vec{c}], I) =$ `false`. Then, there is some repair $\mathcal{I}$ of I such that $\mathcal{I} \not\models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$. Since $\vec{t} \in I$ and $\mathcal{I}$ is a repair of I, by Proposition 3, there is some tuple $\vec{t}'$ in $\mathcal{I}$ and some $\vec{d}'$ such that $\vec{t}' = R(\underline{\vec{c}}, \vec{d}')$. Since $\mathcal{I} \not\models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$, we have that $\{\vec{t}'\} \not\models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$.

Notice that $\vec{t}$ and $\vec{t}'$ can be added to $\mathcal{M}$ only during the iteration for the vector of values $\vec{c}$. Since $\{\vec{t}\} \models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$ and $\{\vec{t}'\} \not\models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}]$, the algorithm never selects tuple $\vec{t}$. But $\vec{t} \in \mathcal{M}$; contradiction.

Inductive step. Assume that ϕ has more than one literal. Let $T_1, \ldots, T_m$ be the subtrees of T such that the root of T_j is a child of the root of T, for $1 \leqslant j \leqslant m$. For each $1 \leqslant j \leqslant m$, let $S_j(\underline{\vec{x}}_j, \vec{y}_j)$ be the literal at the root of T_j. Let ϕ_j be the conjunction of the literals of T_j. Let $\vec{w}_j = \{w\colon\ w$ is a variable of ϕ_j, and $w \notin \vec{x}_j\}$. Let $q_j = \phi_j(\vec{x}_j, \vec{w}_j)$. Let $\mathcal{M}_j =$ `BuildRepair`(ϕ_j, I).

Assume that $\mathcal{M} \models q(\vec{x})[\vec{x}/\vec{c}]$. Let $\vec{t}$ be the tuple of I selected by the algorithm in the iteration for literal R and the vector of values $\vec{c}$. Then, $\vec{t} \in \mathcal{M}$, and there is some $\vec{d}$ such that $\vec{t} = R(\underline{\vec{c}}, \vec{d})$. Since $\mathcal{M} \models q(\vec{x})[\vec{x}/\vec{c}]$, we have that for every j such that $1 \leqslant j \leqslant m$, there is some valuation ν for the variables of $\vec{y}$, and some $\vec{c}_j$ such that $\nu(\vec{y}) = \vec{d}$, $\nu(\vec{x}_j) = \vec{c}_j$, and $\mathcal{M}_j \models q_j(\vec{x}_j)[\vec{x}_j/\vec{c}_j]$.

Assume towards a contradiction that **consistent**$_\Sigma(q(\vec{x})[\vec{x}/\vec{c}], I) =$ `false`. Then, there is some repair $\mathcal{I}$ of I such that $\mathcal{I} \not\models q(\vec{x})[\vec{x}/\vec{c}]$. Since $\vec{t} \in I$ and $\mathcal{I}$ is a repair of I, by Proposition 3, there is some tuple $\vec{t}'$ in $\mathcal{I}$ and some $\vec{d}'$ such that $\vec{t}' = R(\underline{\vec{c}}, \vec{d}')$. By Lemma 2, none of the variables of $\vec{w}_i$ appear in $\vec{w}_j$, for every i and j such that $i \neq j$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m$. Thus, there is some j, some valuation ν for the variables of $\vec{y}$, and some tuple of values $\vec{c}'_j$ such that $1 \leqslant j \leqslant m$, $\mathcal{I} \not\models q_j(\vec{x}_j)[\vec{x}_j/\vec{c}'_j]$, $\nu(\vec{y}) = \vec{d}'$, and $\nu(\vec{x}_j) = \vec{c}'_j$. Thus, **consistent**$_\Sigma(q_j(\vec{x}_j)[\vec{x}_j/\vec{c}'_j], I) =$ `false`. By inductive hypothesis $\mathcal{M}_j \not\models q_j(\vec{x}_j)[\vec{x}_j/\vec{c}'_j]$. Since $\mathcal{M}_j \models q_j(\vec{x}_j)[\vec{x}_j/\vec{c}_j]$, the algorithm never selects $\vec{t}$ in the construction of $\mathcal{M}$. But $\vec{t} \in \mathcal{M}$; contradiction. □

3.3.4. Correctness of `RewriteTree` *and* `RewriteConsistent`

Consider a Boolean query $q = \exists\vec{x}, \vec{w}.\phi(\vec{x}, \vec{w})$ and a query $q'(\vec{x}) = \exists\vec{w}.\phi(\vec{x}, \vec{w})$. That is, q and q' have the same literals, but some of the (existentially-quantified) variables of q are free in q'. In Lemma 3 above, we showed that if q' is in a certain class of conjunctive queries, then there is a repair $\mathcal{M}$ such that for all $\vec{c}$, if $\mathcal{M} \models q'(\vec{x})[\vec{x}/\vec{c}]$, then **consistent**$_\Sigma(q'(\vec{x})[\vec{x}/\vec{c}], I) =$ `true`. We also argued that this fact implies that, in order to check whether **consistent**$_\Sigma(q, I) =$ `true`, it suffices to find some instantiation $\vec{c}$ for the free variables of q' such that **consistent**$_\Sigma(q'(\vec{x})[\vec{x}/\vec{c}], I) =$ `true`. The latter condition is fundamental in the design of our algorithm since it can be checked with a first-order query directly on the inconsistent instance I. In the next lemma, we show that the algorithm `RewriteTree`, the main building block of `RewriteConsistent`, produces a first-order query that checks precisely this condition.

Lemma 4. *Let $q(\vec{x}, \vec{z})$ be a query in $\mathcal{C}_{forest}$ whose join graph T is a tree with root literal $R(\underline{\vec{x}}, \vec{y})$. Let I be an instance. Let $Q(\vec{x}, \vec{z})$ be the first-order query returned by* `RewriteTree`(q, Σ).

Then, $I \models Q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$ iff **consistent**$_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) =$ `true`.

Proof. The proof is by induction on the number of literals of q.

Base case. Assume that q has exactly one literal. Then, $q(\vec{x}, \vec{z}) = \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})$, and $Q =$ `RewriteLocal`(q, Σ). By Lemma 1, we have that $I \models Q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$ iff **consistent**$_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) =$ `true`.

Inductive step. Let $R_1(\underline{\vec{x}}_1, \vec{y}_1), \ldots, R_m(\underline{\vec{x}}_m, \vec{y}_m)$ be the children of R in T. Assume that q is of the form $\exists\vec{w}.\phi(\vec{w}, \vec{z})$, where ϕ is a conjunction of literals. For each $1 \leqslant i \leqslant m$, let T_i be the tree whose root is R_i. Let ϕ_i be the conjunction of the literals of T_i. Let $\vec{w}_i = \{w\colon\ w$ is a variable that occurs in ϕ_i and $\vec{w}$, and $w \notin \vec{x}_i\}$. Let $\vec{z}_i = \{z\colon\ z$ is a variable that occurs in ϕ_i and $\vec{z}$, and $z \notin \vec{x}_i\}$. Let $q_i(\vec{x}_i, \vec{z}_i) = \exists\vec{w}_i.\phi_i(\vec{x}_i, \vec{w}_i, \vec{z}_i)$. Let $Q_i(\vec{x}_i, \vec{z}_i) =$ `RewriteTree`(q_i, Σ). Let $q_{local}(\vec{x}, \vec{z}) = \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})$. Let $Q_{local}(\vec{x}, \vec{z}) =$ `RewriteLocal`(q_{local}, Σ).

($\Rightarrow$) Assume that $I \models Q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, there is a valuation ν for the variables of ϕ such that:

(1) $\nu(\vec{x}) = \vec{c}$, and
(2) $\nu(\vec{z}) = \vec{t}$, and
(3) $I \models Q_{local}(\vec{x}, \vec{z})[\nu]$, and
(4) for every i such that $1 \leqslant i \leqslant m$, there are $\vec{c}_i$ and $\vec{t}_i$ such that $\nu(\vec{x}_i) = \vec{c}_i$, $\nu(\vec{z}_i) = \vec{t}_i$, and $I \models Q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$.

Let $\mathcal{I}$ be a repair of I. Assume towards a contradiction that $\mathcal{I} \not\models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Then, $\textbf{consistent}_\Sigma(\exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) = \texttt{false}$. By Lemma 1, we have that $I \not\models Q_{local}(\vec{x}, \vec{z})[\nu]$; contradiction.

Assume that $\mathcal{I} \models \exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. By Lemma 2, none of the variables of $\vec{w}_i$ appear in $\vec{w}_j$, for every i and j such that $i \neq j$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m$. Then, $\mathcal{I} \not\models q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$ for some i such that $1 \leqslant i \leqslant m$. Thus, $\textbf{consistent}_\Sigma(q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i], I) = \texttt{false}$. By inductive hypothesis, $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$; contradiction.

($\Leftarrow$) Assume that $\textbf{consistent}_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) = \texttt{true}$. Assume towards a contradiction that $I \not\models Q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}]$. Let ν be a valuation for the variables of ϕ such that $\nu(\vec{x}) = \vec{c}$ and $\nu(\vec{z}) = \vec{t}$. By Lemma 2, none of the variables of $\vec{w}_i$ appear in $\vec{w}_j$, for every i and j such that $i \neq j$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m$. Then, either (1) $I \not\models Q_{local}(\vec{x}, \vec{z})[\nu]$; or (2) there is some i such that $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\nu]$.

Assume that $I \not\models Q_{local}(\vec{z})[\nu]$. By Lemma 1, $\textbf{consistent}_\Sigma(\exists\vec{w}.R(\underline{\vec{x}}, \vec{y})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) = \texttt{false}$. Thus, it is the case that $\textbf{consistent}_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) = \texttt{false}$; contradiction. Assume that there is some i such that $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\nu]$. By inductive hypothesis, $\textbf{consistent}_\Sigma(q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i], I) = \texttt{false}$. Thus, it is the case that $\textbf{consistent}_\Sigma(q(\vec{x}, \vec{z})[\vec{x}/\vec{c}][\vec{z}/\vec{t}], I) = \texttt{false}$; contradiction. □

We are now ready to give the correctness proof of our rewriting algorithm, for all queries in class $\mathcal{C}_{forest}$. The intuition of the proof is the following. Assume that we are given a query q such that q is in $\mathcal{C}_{forest}$. Then, each of the connected components of the join graph of q is a tree. Recall that `RewriteTree`, the algorithm for which we proved correctness in the above lemma, requires that the input query satisfies the following conditions. First, the join graph of the query must be a tree. Second, the free variables of the query must include all the variables at key positions of the literal at the root of this tree. In order to be able to use `RewriteTree`, `RewriteConsistent` produces a subquery for each tree of the join graph such that the variables at the key of the corresponding tree root are free. In this way, a first-order rewriting can be produced for each subquery by invoking the algorithm `RewriteTree`. For each i, let $Q_i(\vec{x}_i, \vec{z}_i)$ be the rewriting obtained by invoking $\texttt{RewriteTree}(q_i, \Sigma)$. The query returned by `RewriteConsistent` has the form $Q(\vec{z}) = \exists\vec{w}.(\phi(\vec{w}, \vec{z}) \wedge \bigwedge_{i=1,\ldots,m} Q_i(\vec{x}_i, \vec{z}_i))$, where $\phi(\vec{w}, \vec{z})$ is the conjunction of literals of the original query q, and the variables of each $\vec{x}_i$ are in $\vec{w}$. The correctness of this formula relies on the structural properties proved in Section 3.3.3. First, by Lemma 3, it suffices to find one instantiation for the variables of each $\vec{x}_i$. Thus, the variables of $\vec{x}_i$ can be free in Q_i. Second, the subqueries do not share existentially-quantified variables. This is ensured by the structural property proved in Lemma 2.

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* $q(\vec{z})$ *be a conjunctive query over* $\mathbf{R}$ *such that* $q \in \mathcal{C}_{forest}$. *Let* $Q(\vec{z})$ *be the first-order query returned by* $\texttt{RewriteConsistent}(q, \Sigma)$. *Let* I *be an instance over* $\mathbf{R}$.

Then, $I \models Q(\vec{z})[\vec{z}/\vec{t}]$ *iff* $\vec{t} \in \textbf{consistent}_\Sigma(q, I)$.

Proof. Let G be the join graph of q. Since $q \in \mathcal{C}_{forest}$, G is a forest. Let $T_1, \ldots, T_m$ be the connected components (trees) of G. Assume that q is of the form $\exists\vec{w}.\phi(\vec{w}, \vec{z})$, where ϕ is a conjunction of literals. For each $1 \leqslant i \leqslant m$, let $R_i(\underline{\vec{x}_i}, \vec{y}_i)$ be the literal at the root of T_i. Let ϕ_i be the conjunction of the literals of T_i. Let $\vec{w}_i = \{w\colon\ w$ is a variable that occurs in ϕ_i and $\vec{w}$, and $w \notin \vec{x}_i\}$. Let $\vec{z}_i = \{z\colon\ z$ is a variable that occurs in ϕ_i and $\vec{z}$, and $z \notin \vec{x}_i\}$. Let $q_i(\vec{x}_i, \vec{z}_i) = \exists\vec{w}_i.\phi_i(\vec{x}_i, \vec{w}_i, \vec{z}_i)$. Let $Q_i(\vec{x}_i, \vec{z}_i) = \texttt{RewriteTree}(q_i, \Sigma)$.

($\Rightarrow$) Assume that $I \models Q(\vec{z})[\vec{z}/\vec{t}]$. Then, there is a valuation ν for the variables of ϕ such that:

(1) $\nu(\vec{z}) = \vec{t}$, and
(2) $I \models \phi(\vec{w}, \vec{z})[\nu]$, and
(3) for every i such that $1 \leqslant i \leqslant m$, there are $\vec{c}_i$ and $\vec{t}_i$ such that $\nu(\vec{x}_i) = \vec{c}_i$, $\nu(\vec{z}_i) = \vec{t}_i$, and $I \models Q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$.

Let $\mathcal{I}$ be a repair of I. Assume towards a contradiction that $\mathcal{I} \not\models q[\vec{z}/\vec{t}]$. Thus, $\mathcal{I} \not\models q[\nu]$. By Lemma 2, none of the variables of $\vec{w}_i$ appear in $\vec{w}_j$, for every i and j such that $i \neq j$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant m$. Then, $\mathcal{I} \not\models q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$ for some i such that $1 \leqslant i \leqslant m$. Thus, $\textbf{consistent}_{\Sigma}(q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i], I) = \texttt{false}$. By Lemma 4, $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\vec{x}_i/\vec{c}_i][\vec{z}_i/\vec{t}_i]$; contradiction.

($\Leftarrow$) Assume that $\vec{t} \in \textbf{consistent}_{\Sigma}(q, I)$. Assume towards a contradiction that $I \not\models Q(\vec{z})[\vec{z}/\vec{t}]$. Let ν be a valuation for the variables of ϕ such that $\nu(\vec{z}) = \vec{t}$. Then, either (1) $I \not\models q(\vec{z})[\nu]$; or (2) there is some i such that $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\nu]$.

We will build a repair $\mathcal{M}$ of I as follows. For each i, let I_i be the projection of I on the relation symbols of ϕ_i. By Lemma 3, there is a repair $\mathcal{M}_i$ such that if $\mathcal{M}_i \models q_i(\vec{x}_i)[\vec{x}_i/\vec{c}_i]$, then $\textbf{consistent}_{\Sigma}(q_i(\vec{x}_i)[\vec{x}_i/\vec{c}_i], I_i) = \texttt{true}$. We add all the tuples of $\mathcal{M}_i$ to $\mathcal{M}$.

We now show that $\mathcal{M} \not\models q(\vec{z})[\nu]$. Assume that $I \not\models q(\vec{z})[\nu]$. Since $\mathcal{M} \subseteq I$, $\mathcal{M} \not\models q(\vec{z})[\nu]$. Now, assume that there is some i such that $1 \leqslant i \leqslant m$ and $I \not\models Q_i(\vec{x}_i, \vec{z}_i)[\nu]$. By Lemma 4, $\textbf{consistent}_{\Sigma}(q_i(\vec{x}_i, \vec{z}_i)[\nu], I) = \texttt{false}$. By Lemma 3, $\mathcal{M}_i \not\models q_i(\vec{x}_i, \vec{z}_i)[\nu]$. Thus, $\mathcal{M} \not\models q(\vec{z})[\nu]$.

So, for every valuation ν such that $\nu(\vec{z}) = \vec{t}$, we have that $\mathcal{M} \not\models q(\vec{z})[\nu]$. Thus, $\vec{t} \notin \textbf{consistent}_{\Sigma}(q, I)$; contradiction. □

4. Intractability results

In the previous section, we presented a query rewriting algorithm that works on a class of queries with full nonkey-to-key joins and whose join graph is a forest. In this section, we show that this is a maximal class of queries. First, we show that minimal relaxations of the conditions of the class lead to intractability. Second, we embark on a more ambitious goal: for a large class of conjunctive queries, we show that the conditions of $\mathcal{C}_{forest}$ are not only sufficient, but they are also necessary conditions for a query to be first-order rewritable.

4.1. Minimal relaxations of $\mathcal{C}_{forest}$

In this section, we show that minimal relaxations of the conditions of $\mathcal{C}_{forest}$ lead to intractability. In particular, we show the intractability of the problem of computing consistent answers for: (1) a conjunctive query whose join graph is a cycle of length two; and (2) a conjunctive query whose join graph is a forest, but the query has some nonkey-to-key joins that are not full.

Chomicki and Marcinkowski [6] computing consistent answers for a query with a single nonkey-to-nonkey join is coNP-complete. Their result used a query with repeated relation symbols (specifically, a query with only two literals both for a single relation R). We can use their insight to show that the problem of computing consistent answers for the following query without repeated relation symbols, but with a single nonkey-to-nonkey join is also coNP-complete,

$$q_{nk} = \exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y).$$

Notice that q_{nk} has a cycle of length two (actually, a nonkey-to-nonkey join), and no nonkey-to-key joins. Our proof of hardness is a simple modification to the results of Chomicki and Marcinkowski [6] and uses a reduction from the problem MONOTONE-3SAT, which is well known to be NP-complete. The only difference between the MONOTONE-3SAT and 3SAT problems is that the former assumes that the input 3CNF propositional formula is *monotone*. That is, each clause Φ_i contains either positive or negative atoms, but not both. We shall say that a clause that contains only positive (negative) atoms is a *positive* (*negative*) *clause*.

Lemma 5. *Let q be the query $\exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$. Then,* CONSISTENT$(q, \Sigma)$ *is coNP-hard.*

Proof. We will prove hardness by reduction from MONOTONE-3SAT. Let $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_m$ be a 3CNF formula such that each clause Φ_i contains either positive or negative atoms, but not both. We shall build an instance I as follows:

- For each positive clause Φ_i and each atom z that occurs in Φ_i, we add a tuple $S_1(\underline{i}, \texttt{z})$ to I.
- For each negative clause Φ_i and each atom z that occurs in Φ_i, we add a tuple $S_2(\underline{i}, \texttt{z})$ to I.

We now show that $\textbf{consistent}_{\Sigma}(q, I) = \texttt{false}$ iff Φ is satisfiable.

($\Rightarrow$) Since **consistent**$_\Sigma(q, I) = \texttt{false}$, there exists a repair $\mathcal{I}$ of I such that $\mathcal{I} \not\models q$. We now build a valuation $\texttt{v}$ for the variables of Φ as follows. For each variable $\texttt{z}$, we let $\texttt{v(z)} = \texttt{true}$ if there is some i such that $S_1(\underline{i}, \texttt{z}) \in \mathcal{I}$; and we let $\texttt{v(z)} = \texttt{false}$ if there is some i such that $S_2(\underline{i}, \texttt{z}) \in \mathcal{I}$. It is easy to see that $\texttt{v}$ is a truth valuation that satisfies Φ.

($\Leftarrow$) Assume that Φ is satisfiable. Let $\texttt{v}$ be a truth assignment for the variables of Φ. We shall build a repair $\mathcal{I}$ as follows. For each positive clause Φ_i, select a variable $\texttt{z}$ that appears in Φ_i and such that $\texttt{v(z)} = \texttt{true}$. Let $S_1(\underline{i}, \texttt{z}) \in \mathcal{I}$. For each negative clause Φ_i, select a variable $\texttt{z}$ that appears in Φ_i and such that $\texttt{v(z)} = \texttt{false}$. Let $S_2(\underline{i}, \texttt{z}) \in \mathcal{I}$. It is easy to see that $\mathcal{I} \not\models q$. □

Now, we show the intractability of the problem for a conjunctive query whose join graph is a forest, but the query has nonkey-to-key joins that are not full. In particular, we focus on the following query:

$$\exists x, x', w, w', z, z', m.R_1(\underline{x}, w) \wedge R_2(\underline{m}, \underline{w}, z) \wedge R_3(\underline{x}', w') \wedge R_4(\underline{m}, \underline{w}', z').$$

We prove hardness by showing a reduction from the problem of computing the consistent answers for the query q_{nk} shown to be coNP-hard in Lemma 5.

Lemma 6. *Let q be the query $\exists x, x', w, w', z, z', m.R_1(\underline{x}, w) \wedge R_2(\underline{m}, \underline{w}, z) \wedge R_3(\underline{x}', w') \wedge R_4(\underline{m}, \underline{w}', z')$. Let q' be the query $\exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$. Then, there is a polynomial time reduction from the problem* $\texttt{CONSISTENT}(q', \Sigma')$ *to the problem* $\texttt{CONSISTENT}(q, \Sigma)$.

Proof. Let I' be an instance over the schema of q'. We shall build an instance I over the schema of q as follows:

Initialize I as the empty instance
for each tuple $S_1(\underline{c}_1, d_1) \in I'$ **do**
  Add $R_1(\underline{c}_1, d_1)$ to I
end for
for each tuple $S_2(\underline{c}_2, d_2) \in I'$ **do**
  Add $R_3(\underline{c}_2, d_2)$ to I
end for
Let c_z, $c_{z'}$ be some constants
for each valuation $\nu_{q'}$ such that $I' \models S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)[\nu_{q'}]$ **do**
  Let $\nu_q(x) = \nu_{q'}(x)$
  Let $\nu_q(x') = \nu_{q'}(x')$
  Let $\nu_q(w) = \nu_{q'}(y)$
  Let $\nu_q(w') = \nu_{q'}(y)$
  Let c_m be a newly-created constant
  Let $\nu_q(m) = c_m$
  Let $\nu_q(z) = c_z$
  Let $\nu_q(z') = c_{z'}$
  Add tuple $R_2(\underline{m}, \underline{w}, z)[\nu_q]$ to I
  Add tuple $R_4(\underline{m}, \underline{w}', z')[\nu_q]$ to I
end for

We claim that **consistent**$_\Sigma(q', I') = \texttt{true}$ iff **consistent**$_\Sigma(q, I) = \texttt{true}$.
($\Rightarrow$) Let $\mathcal{I}$ be a repair of I. We shall build an instance $\mathcal{I}'$ as follows:

for each tuple $R_1(\underline{c}_1, d_1)$ of $\mathcal{I}$ **do**
  Add a tuple $S_1(\underline{c}_1, d_1)$ to $\mathcal{I}'$
end for
for each tuple $R_3(\underline{c}_2, d_2)$ of $\mathcal{I}$ **do**
  Add a tuple $S_2(\underline{c}_2, d_2)$ to $\mathcal{I}'$
end for

Notice that R_1 and S_1 (and, similarly, R_3 and S_2) have the same extensions in I and I', respectively. Thus, since $\mathcal{I}$ is a repair of I, $\mathcal{I}'$ is a repair of I'. Since $\textbf{consistent}_\Sigma(q', I') = \texttt{true}$, $\mathcal{I}' \models q'$. Thus, there is a valuation $\nu_{q'}$ such that $\mathcal{I}' \models S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)[\nu_{q'}]$. Let $c_1 = \nu_{q'}(x)$, $c_2 = \nu_{q'}(x')$, $d = \nu_{q'}(y)$. Let c_z and $c_{z'}$ be the constants used in the algorithm that constructs I. Let c_m be the constant created in the algorithm for the iteration corresponding to $\nu_{q'}$. Let ν_q be a valuation for the variables of q such that:

- $\nu_q(x) = c_1$,
- $\nu_q(x') = c_2$,
- $\nu_q(w) = d$,
- $\nu_q(w') = d$,
- $\nu_q(m) = c_m$,
- $\nu_q(z) = c_z$,
- $\nu_q(z') = c_{z'}$.

Since $S_1(\underline{c}_1, d) \in \mathcal{I}'$, $R_1(\underline{c}_1, d) \in \mathcal{I}$. Since $S_2(\underline{c}_2, d) \in \mathcal{I}'$, $R_3(\underline{c}_2, d) \in \mathcal{I}$. By Proposition 2, $\mathcal{I}' \subseteq I'$. Thus, $S_1(\underline{c}_1, d) \in I'$ and $S_2(\underline{c}_2, d) \in I'$. Since c_m is the constant chosen in the iteration for $\nu_{q'}$ in the algorithm that constructs I, $R_2(\underline{c}_m, \underline{d}, c_z) \in I$ and $R_4(\underline{c}_m, \underline{d}, c_{z'}) \in I$. By Proposition 3, $R_2(\underline{c}_m, \underline{d}, e) \in \mathcal{I}$ and $R_4(\underline{c}_m, \underline{d}, e') \in \mathcal{I}$, for some e, e'. Thus, $\mathcal{I} \models q[\nu_q]$.

($\Leftarrow$) Let $\mathcal{I}'$ be a repair of I'. We shall build an instance $\mathcal{I}$ as follows:

for each tuple $S_1(\underline{c}_1, d_1) \in \mathcal{I}'$ **do**
  Add $R_1(\underline{c}_1, d_1)$ to $\mathcal{I}$
end for
for each tuple $S_2(\underline{c}_2, d_2) \in \mathcal{I}'$ **do**
  Add $R_3(\underline{c}_2, d_2)$ to $\mathcal{I}$
end for
for each tuple $R_2(\underline{c}_1, \underline{c}_2, d) \in I$ **do**
  Add $R_2(\underline{c}_1, \underline{c}_2, d)$ to $\mathcal{I}$
end for
for each tuple $R_4(\underline{c}_1, \underline{c}_2, d) \in I$ **do**
  Add $R_4(\underline{c}_1, \underline{c}_2, d)$ to $\mathcal{I}$
end for

We now show that $\mathcal{I}$ is a repair of I. First, notice that R_1 and S_1 (and, similarly, R_3 and S_2) have the same extensions in I and I', respectively. Second, in the construction of I, every tuple of R_2 and R_4 is given a distinct key value. Then, by Propositions 2 and 3, every tuple in the extension of R_2 in I is in the extension of R_2 in $\mathcal{I}$; and every tuple in the extension of R_4 in I is in the extension of R_4 in $\mathcal{I}$.

Since $\textbf{consistent}_\Sigma(q, I) = \texttt{true}$, $\mathcal{I} \models q$. Thus, there exists some valuation ν_q such that $\mathcal{I} \models R_1(\underline{x}, w) \wedge R_2(\underline{m}, \underline{w}, z) \wedge R_3(\underline{x}', w') \wedge R_4(\underline{m}, \underline{w}', z')[\nu_q]$. By construction of I, if R_2 and R_4 join on m, then $\nu_q(w) = \nu_q(w')$. Let $\nu_{q'}$ be such that:

- $\nu_{q'}(x) = \nu_q(x)$,
- $\nu_{q'}(x') = \nu_q(x')$,
- $\nu_{q'}(y) = \nu_q(w) = \nu_q(w')$.

It is easy to see that $\mathcal{I}' \models S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)[\nu_{q'}]$. Thus, $\mathcal{I}' \models q'$. □

4.2. A dichotomy result

In Section 3, we presented a query rewriting algorithm which works on a class of queries that we call $\mathcal{C}_{forest}$. Clearly, $\mathcal{C}_{forest}$ gives sufficient conditions for a query to be first-order rewritable. In this section, we address the following question: for which class of queries does $\mathcal{C}_{forest}$ also give necessary conditions? That is, 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 $\mathcal{C}_{forest}$. Notice that this establishes a dichotomy between first-order rewritability and coNP-completeness, and is therefore much stronger than the complexity results that we presented in Section 4.1 (and, in fact, all the complexity results present in the consistent query answering literature [4,6]). 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 $\texttt{CONSISTENT}(q, \Sigma)$ 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, $\texttt{CONSISTENT}(q, \Sigma)$ is coNP-complete.

We will focus on conjunctive queries without repeated relation symbols and all of whose nonkey-to-key joins are full. Within this class, there are some queries for which the existence of a cycle is not a sufficient condition for intractability. Consider, for example, 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 can be rewritten as follows:

$$\exists x, y.R_1(\underline{x}, y) \wedge R_2(\underline{x}, y) \wedge \forall y'.\big(R_1(\underline{x}, y') \rightarrow y' = y\big) \wedge \forall y'.\big(R_2(\underline{x}, y') \rightarrow y' = y\big).$$

Recall that the problem of computing consistent answers is intractable for the query $q_{nk} = \exists x, x', y.R_1(\underline{x}, y) \wedge R_2(\underline{x}', y)$. Notice that q_{nk} and q_5 have exactly the same join graph. The only difference between them is that in q_{nk}, the two literals are related exclusively by a nonkey-to-nonkey join; whereas in q_5, they are related by *both* a key-to-key and a nonkey-to-nonkey join. Our intuition is that a query with a cyclic join graph may be tractable only if there are literals related by more than one type of join (e.g., nonkey-to-nonkey and key-to-key). We formalize this intuition with the definition of a class $\mathcal{C}^*$, which essentially "separates" the different types of joins of the query. In $\mathcal{C}^*$, every pair of literals can be related by at most one of type of join (i.e., key-to-key, nonkey-to-nonkey, and nonkey-to-key).

Definition 9. Let q be a conjunctive query without repeated relation symbols and all of whose nonkey-to-key joins are full. We say that q is in class $\mathcal{C}^*$ if for every pair R and R' of literals of q at most one of the following conditions holds:

- there is a key-to-key join between R and R',
- there is a nonkey-to-nonkey join between R and R',
- there are literals $R_1, \ldots, R_m$ in q such that there is a nonkey-to-key join from R to R_1, from R_m to R', and from R_i to R_{i+1}, for every i such that $1 \leqslant i < m$.

We will consider a class, called $\mathcal{C}_{hard}$, of all queries of $\mathcal{C}^*$ that are not in $\mathcal{C}_{forest}$. The main result of this section, Theorem 2, proves that the problem of computing the consistent answers for every query of $\mathcal{C}_{hard}$ is coNP-complete.

Definition 10. We say that a query q is in class $\mathcal{C}_{hard}$ if $q \in \mathcal{C}^*$ and $q \notin \mathcal{C}_{forest}$.

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 [10], 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.

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.*

Under a complexity-theoretic assumption, we also get a dichotomy between first-order rewritability and first-order inexpressibility for the class $\mathcal{C}^*$. That is, for all the queries of $\mathcal{C}^*$ that are not in $\mathcal{C}_{hard}$, we can produce a first-order rewriting using our algorithm `RewriteConsistent`. For the queries of $\mathcal{C}_{hard}$, since the problem of obtaining consistent answers is coNP-complete, there is no first-order rewriting, unless P = NP (which is unlikely). An alternative approach to proving that there is no first-order rewriting, which we leave as future work, would be to avoid complexity-theoretic assumptions, and appeal directly to arguments based on game theory.

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

Notice that Corollary 2 is a stronger result than Corollary 1 since it is well known that there are problems that are tractable but not expressible in first-order logic [12]. In particular, in previous work [8], we showed examples of queries (which are outside $\mathcal{C}^*$) for which the problem of obtaining consistent answers is tractable but not first-order rewritable.

The intractability of all queries in $\mathcal{C}_{hard}$ will be shown as follows. First, we show in Lemma 7 that the problem of computing consistent answers for conjunctive queries is in coNP. This is a result known in the literature, but we briefly give a proof for our setting. For hardness, we will use a reduction from the problem of computing consistent answers for one of two particular queries to the problem of computing consistent answers for q. One of these specific queries is the query $q_{nk} = \exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$. This query has a nonkey-to-nonkey join, and was shown to be intractable in Lemma 5. The other query has a cycle of nonkey-to-key joins, and is shown to be intractable in Lemma 8.

The next lemma shows that the problem of computing consistent answers for conjunctive queries is in coNP.

Lemma 7. *Let q be a conjunctive query. The problem* CONSISTENT(q, Σ) *is in coNP.*

Proof. Let I be an instance. In order to decide whether $\vec{t} \notin \textbf{consistent}_\Sigma(q, I)$, it suffices to show a repair $\mathcal{I}$ of I such that $\mathcal{I} \not\models q[\vec{t}]$. The size of $\mathcal{I}$ is polynomially bounded by the size of I. In particular, by Proposition 2, $\mathcal{I} \subseteq I$. Furthermore, $\mathcal{I} \not\models q[\vec{t}]$ can be checked in polynomial time, since q is a conjunctive query. □

In the next lemma, we show the coNP hardness of computing consistent answers for one of the two particular queries that will be used in Lemma 11. The coNP-hardness of the other query was proven in Lemma 5.

Lemma 8. *Let $q = \exists x, y.T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)$. Then, the problem* CONSISTENT(q, Σ) *is coNP-hard.*

Proof. We will prove hardness by reduction from MONOTONE-3SAT. Let $\Phi = \Phi_1 \wedge \cdots \wedge \Phi_m$ be a monotone 3CNF formula. We shall build an instance I as follows:

- For each atom z, let $\Phi_{i_1}, \ldots, \Phi_{i_n}$ be the positive clauses where z occurs. Add tuples $T_1(\underline{\langle \Phi_{i_1}, \ldots, \Phi_{i_n} \rangle}, \mathtt{z})$ and $T_2(\underline{\mathtt{z}}, \langle \Phi_{i_1}, \ldots, \Phi_{i_n} \rangle)$ to I.
- For each atom z, let $\Phi_{i_1}, \ldots, \Phi_{i_n}$ be the negative clauses where z occurs. Add tuples $T_1(\underline{\langle \Phi_{i_1}, \ldots, \Phi_{i_n} \rangle}, \mathtt{z})$ and $T_2(\underline{\mathtt{z}}, \langle \Phi_{i_1}, \ldots, \Phi_{i_n} \rangle)$ to I.

We now show that $\textbf{consistent}_\Sigma(q, I) = \mathtt{false}$ iff Φ is satisfiable.

($\Rightarrow$) Since $\textbf{consistent}_\Sigma(q, I) = \mathtt{false}$, there exists a repair $\mathcal{I}$ of I such that $\mathcal{I} \not\models q$. Assume towards a contradiction that there are tuples $T_1(\underline{c}, \mathtt{z}) \in \mathcal{I}$ and $T_1(\underline{c'}, \mathtt{z}) \in \mathcal{I}$ such that $c \neq c'$. By construction of I, if $T_2(\underline{\mathtt{z}}, d) \in I$, then $d = c$ or $d = c'$. By Propositions 2 and 3, either $T_2(\underline{\mathtt{z}}, c) \in \mathcal{I}$ or $T_2(\underline{\mathtt{z}}, c') \in \mathcal{I}$. Thus, $\mathcal{I} \models q$; contradiction.

We now build a valuation v for the variables of Φ as follows. For each variable z, we let $\mathtt{v}(\mathtt{z}) = \mathtt{true}$ if there is some c such that $T_1(\underline{c}, \mathtt{z}) \in \mathcal{I}$ and c is a list of positive clauses; and we let $\mathtt{v}(\mathtt{z}) = \mathtt{false}$ if there is some i such that $T_1(\underline{c}, \mathtt{z}) \in \mathcal{I}$, and c is a list of negative clauses. It is easy to see that v is a truth valuation that satisfies Φ.

($\Leftarrow$) Assume that Φ is satisfiable. Let v be a truth assignment for the variables of Φ. We shall build a repair $\mathcal{I}$ as follows. For each positive clause Φ_i, select a variable z that appears in Φ_i and such that $\mathtt{v}(\mathtt{z}) = \mathtt{true}$. Add $T_1(\underline{c}, \mathtt{z})$ to $\mathcal{I}$, where c is a list of positive clauses. For each negative clause Φ_i, select a variable z that appears in Φ_i and such that $\mathtt{v}(\mathtt{z}) = \mathtt{false}$. Add $T_1(\underline{c}, \mathtt{z})$ to $\mathcal{I}$, where c is a list of negative clauses. For each variable z, if $\mathtt{v}(\mathtt{z}) = \mathtt{false}$, add $T_2(\underline{\mathtt{z}}, c)$ to $\mathcal{I}$, where c is a list of positive clauses; if $\mathtt{v}(\mathtt{z}) = \mathtt{true}$, add $T_2(\underline{\mathtt{z}}, c)$ to $\mathcal{I}$, where c is a list of negative clauses. It is easy to see that $\mathcal{I} \not\models q$. □

We now give some auxiliary results before proving Lemma 11. The next lemma generalizes Lemma 8 from cycles of length two to the case of cycles of arbitrary length.

Lemma 9. *Let q be the query $\exists w_1, \ldots, w_m.S_1(\underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge \cdots \wedge S_m(\underline{w}_{m-1}, w_m)$. Let $q' = \exists x, y.T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)$. Then, there is a polynomial time reduction from the problem* CONSISTENT(q', Σ') *to the problem* CONSISTENT(q, Σ).

Proof. Let I' be an instance over the schema of q'. We shall build an instance I over the schema of q as follows:

for each valuation $\nu_{q'}$ for the variables of q' such that $I' \models T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)[\nu_{q'}]$ **do**
  Let $\nu_q(w_m) = \nu_{q'}(x)$
  Let $\nu_q(w_1) = \nu_{q'}(y)$
  Create a new constant c_{new}
  for $i := 2$ to $m - 1$ **do**
    Let $\nu_q(w_i) = c_{new}$
  end for
  Add the tuples of $S_1(\underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge \cdots \wedge S_m(\underline{w}_{m-1}, w_m)[\nu_q]$ to I
end for

We claim that **consistent**$_\Sigma(q', I') =$ `true` iff **consistent**$_\Sigma(q, I) =$ `true`.

($\Rightarrow$) Let $\mathcal{I}$ be a repair of I over the schema of q. We shall build a repair $\mathcal{I}'$ over the schema of q' as follows:

for each tuple $S_1(\underline{c}_m, c_1)$ of $\mathcal{I}$ **do**
  Add a tuple $T_1(\underline{c}_m, c_1)$ to $\mathcal{I}'$
  for each c_{new} such that $S_2(\underline{c}_1, c_{new}) \in \mathcal{I}$ and $S_m(\underline{c}_{new}, c_m) \in \mathcal{I}$ **do**
    Add a tuple $T_2(\underline{c}_1, c_m)$ to $\mathcal{I}'$
  end for
end for

Since **consistent**$_\Sigma(q', I') =$ `true`, $\mathcal{I}' \models q'$. Thus, there is a valuation $\nu_{q'}$ such that $\mathcal{I}' \models T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)[\nu_{q'}]$. Let $c_m = \nu_{q'}(x)$, $c_1 = \nu_{q'}(y)$. Since $T_2(\underline{c}_1, c_m) \in \mathcal{I}'$, there exists c_{new} such that $S_2(\underline{c}_1, c_{new}) \in \mathcal{I}$ and $S_m(\underline{c}_{new}, c_m) \in \mathcal{I}$. Let ν_q be a valuation for the variables of q such that:

- $\nu_q(w_m) = c_m$,
- $\nu_q(w_1) = c_1$,
- $\nu_q(w_i) = c_{new}$, for $1 < i < m$.

Since $T_1(\underline{c}_m, c_1) \in \mathcal{I}'$, $S_1(\underline{c}_m, c_1) \in \mathcal{I}$. By construction of ν_q, $S_2(\underline{c}_1, c_{new}) \in \mathcal{I}$ and $S_m(\underline{c}_{new}, c_m) \in \mathcal{I}$. For $2 < i \leqslant m$, notice that by construction of I, there are no tuples $S_i(\underline{c}_i, d_i)$ and $S_i(\underline{c}_i, d_i')$ in I such that $d_i \neq d_i'$. Therefore, by Propositions 2 and 3, every tuple in the extension of S_i in I appears in the extension of S_i in $\mathcal{I}$. By construction of I, $S_i(\underline{c}_{new}, c_{new}) \in I$, for $3 \leqslant i \leqslant m - 1$. Thus, $S_i(\underline{c}_{new}, c_{new}) \in \mathcal{I}$. We conclude that $\mathcal{I} \models S_1(\underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge \cdots \wedge S_m(\underline{w}_{m-1}, w_m)[\nu_q]$. Thus, $\mathcal{I} \models q$.

($\Leftarrow$) Let $\mathcal{I}'$ be a repair of I'. We shall build an instance $\mathcal{I}$ as follows:

for each tuple $T_1(\underline{c}_m, c_1)$ of $\mathcal{I}'$ **do**
  Add a tuple $S_1(\underline{c}_m, c_1)$ to $\mathcal{I}$
  Let c_{new} be a constant such that $S_2(\underline{c}_1, c_{new}) \in I$ and $S_m(\underline{c}_{new}, c_m) \in I$
  Add a tuple $S_2(\underline{c}_1, c_{new})$ to $\mathcal{I}$
  for $i := 3$ to $m - 1$ **do**
    Add a tuple $S_i(\underline{c}_{new}, c_{new})$ to $\mathcal{I}$
  end for
  Add a tuple $S_m(\underline{c}_{new}, c_m)$ to $\mathcal{I}$
end for

It is easy to see that $\mathcal{I}$ is a repair of I. Since $\textbf{consistent}_{\Sigma}(q, I) = \texttt{true}$, $\mathcal{I} \models q$. Thus, there exists some valuation v_q such that $\mathcal{I} \models S_1(\underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge \cdots \wedge S_m(\underline{w}_{m-1}, w_m)[v_q]$. Let $v_{q'}$ be such that:

- $v_{q'}(x) = v_q(w_m)$,
- $v_{q'}(y) = v_q(w_{m_1})$.

It is easy to see that $\mathcal{I}' \models T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)[v_{q'}]$. Thus, $\mathcal{I}' \models q'$. □

Our strategy for proving the dichotomy will be to show that if q has a subquery q' that is known to be intractable (in particular, a cycle), then q is not tractable. This does not hold in general, but as we show with the next auxiliary result, it holds for the queries in $\mathcal{C}^*$.

Lemma 10. *Let q be a Boolean query such that $q \in \mathcal{C}^*$. Let $R_1(\vec{\underline{x}}_1, \vec{y}_1), \ldots, R_n(\vec{\underline{x}}_n, \vec{y}_n)$ be the literals of q. Let q' be a Boolean query. Let $S_1(\underline{x}_1, y_1), \ldots, S_m(\underline{x}_m, y_m)$ be the literals of q', where $m \leqslant n$. Assume that the join graph of q' is a cycle. Let $L = \{x_1, y_1, \ldots, x_m, y_m\}$. Assume that*:

- *x_i occurs in $\vec{x}_i$, for $1 \leqslant i \leqslant m$, and*
- *y_i occurs in $\vec{y}_i$, for $1 \leqslant i \leqslant m$, and*
- *for $1 \leqslant i \leqslant m$, if $w \in L$ and w occurs in R_i, then w occurs in S_i.*

Then, there is a polynomial-time reduction from the problem $\texttt{CONSISTENT}(q', \Sigma')$ *to* $\texttt{CONSISTENT}(q, \Sigma)$.

Proof. Let $F = \{w\colon\ w \text{ occurs in } R_i, \text{ and } 1 \leqslant i \leqslant m\} - L$. Let $U = \{w\colon\ w \text{ occurs in } q\} - F - L$.

Let I' be an instance over the schema of q'. We shall build an instance I over the schema of q as follows:

for each variable w such that $w \in F$ **do**
  Create a new constant c_{new}
  Let $v_F(w) = c_{new}$
end for
for each valuation $v_{q'}$ for the variables of q' such that $I' \models S_1(\underline{x}_1, y_1) \wedge \cdots \wedge S_m(\underline{x}_m, y_m)[v_{q'}]$ **do**
  for each variable w such that $w \in F$ **do**
    Let $v_q(w) = v_F(w)$
  end for
  for each variable w such that $w \in U$ **do**
    Create a new constant c_{new}
    Let $v_q(w) = c_{new}$
  end for
  for $i := 1$ to m **do**
    Let $v_q(x_i) = v_{q'}(x_i)$
    Let $v_q(y_i) = v_{q'}(y_i)$
  end for
  Add the tuples of $R_1(\vec{\underline{x}}_1, \vec{y}_1) \wedge \cdots \wedge R_n(\vec{\underline{x}}_n, \vec{y}_n)[v_q]$ to I
end for

We claim that $\textbf{consistent}_{\Sigma}(q', I') = \texttt{true}$ iff $\textbf{consistent}_{\Sigma}(q, I) = \texttt{true}$.

($\Rightarrow$) Let $\mathcal{I}$ be a repair of I over the schema of q. We shall build an instance $\mathcal{I}'$ over the schema of q' as follows:

for $i := 1$ to m **do**
  for each tuple $R_i(\vec{c}_i, \vec{d}_i)$ of $\mathcal{I}$ **do**
    Let c_i be the constant that appears in $\vec{c}_i$ at the position of one of the occurrences of x_i in $\vec{x}_i$
    Let d_i be the constant that appears in $\vec{d}_i$ at the position of y_i in $\vec{y}_i$
    Add $S_i(\underline{c}_i, d_i)$ to $\mathcal{I}'$
  end for
end for

We make the following observations with respect to the construction of $\mathcal{I}'$. By construction of I, if $R_i(\vec{c}_i, \vec{d}_i) \in I$, the same constant appears in $\vec{c}_i$ at all the positions where x_i appears in $\vec{x}_i$. By Proposition 2, $\mathcal{I} \subseteq I$. Thus, in the construction of $\mathcal{I}'$, it suffices to choose the constant that occurs in $\vec{c}_i$ at any of the positions where x_i occurs in $\vec{x}_i$.

Assume that $\mathcal{I}'$ is not a repair of I'. Then, there are constants c_i, d_i and d_i' such that $d_i \neq d_i'$, $S_i(\underline{c}_i, d_i) \in \mathcal{I}'$ and $S_i(\underline{c}_i, d_i') \in \mathcal{I}'$. By construction of $\mathcal{I}'$, there are tuples $R_i(\vec{c}_i, \vec{d}_i) \in \mathcal{I}$ and $R_i(\vec{c}_i', \vec{d}_i') \in \mathcal{I}$ such that c_i appears in $\vec{c}_i$ and $\vec{c}_i'$ at all the positions where x_i appears in $\vec{x}_i$; and d_i and d_i' appear in $\vec{d}_i$ and $\vec{d}_i'$, respectively, at the position of y_i in $\vec{y}_i$. Clearly, $\vec{d}_i \neq \vec{d}_i'$. By construction of I, if w is a variable such that $w \notin L$, w is assigned the value $\nu_F(w)$ in every tuple of I. By Proposition 2, $\mathcal{I} \subseteq I$. Thus, $\vec{c}_i = \vec{c}_i'$. Since $\vec{d}_i \neq \vec{d}_i'$, $\mathcal{I}$ does not satisfy the key constraints of Σ. Thus $\mathcal{I}$ is not a repair; contradiction. We conclude that $\mathcal{I}'$ is a repair of I'.

Since $\textbf{consistent}_{\Sigma}(q', I') = \texttt{true}$, $\mathcal{I}' \models q'$. Thus, there is some valuation $\nu_{q'}$ such that $\mathcal{I}' \models S_1(\underline{x}_1, y_1) \wedge \cdots \wedge S_m(\underline{x}_m, y_m)[\nu_{q'}]$. Let ν_m be a valuation for the variables of $R_1, \ldots, R_m$ such that:

- $\nu_m(x_i) = \nu_{q'}(x_i)$, for $1 \leqslant i \leqslant m$,
- $\nu_m(y_i) = \nu_{q'}(y_i)$, for $1 \leqslant i \leqslant m$,
- $\nu_m(w) = \nu_F(w)$ if $w \in F$.

Let w be a variable that appears in R_i, for $1 \leqslant i \leqslant m$. If $w \in L$ and w occurs in R_i, by hypothesis, w occurs in S_i. If $w \notin L$, then $w \in F$, by definition of F. Since $\mathcal{I}' \models S_1(\underline{x}_1, y_1) \wedge \cdots \wedge S_m(\underline{x}_m, y_m)[\nu_{q'}]$, and $\nu_m(w) = \nu_F(w)$ if $w \in F$, we conclude that $\mathcal{I} \models R_1(\vec{\underline{x}}_1, \vec{y}_1) \wedge \cdots \wedge R_m(\vec{\underline{x}}_m, \vec{y}_m)[\nu_m]$.

By construction of I, there is a valuation ν_q for the variables of q such that:

- $\nu_m(w) = \nu_q(w)$ if w appears in R_i, for $1 \leqslant i \leqslant m$; and
- $I \models R_1(\vec{\underline{x}}_1, \vec{y}_1) \wedge \cdots \wedge R_n(\vec{\underline{x}}_n, \vec{y}_n)[\nu_q]$.

Let $R_i(\vec{\underline{x}}_i, \vec{y}_i)$ be a literal of q such that $i > m$. Notice that we assume that the join graph of q' is a cycle. Since q is in $\mathcal{C}^*$, there exists some variable w such that w occurs in $\vec{x}_i$ and w does not occur in any of $R_1, \ldots, R_m$. Thus, $w \in U$. Since the variables of U are assigned a distinct constant in every iteration of the algorithm that constructs I, if two tuples $R_i(\vec{c}_i, \vec{d}_i)$ and $R_i(\vec{c}_i', \vec{d}_i')$ are added at different iterations, then $\vec{c}_i \neq \vec{c}_i'$. Therefore, by Propositions 2 and 3, every tuple in the extension of R_i in I is in the extension of R_i in $\mathcal{I}$. Therefore, $\mathcal{I} \models R_1(\vec{\underline{x}}_1, \vec{y}_1) \wedge \cdots \wedge R_n(\vec{\underline{x}}_n, \vec{y}_n)[\nu_q]$.

($\Leftarrow$) Let $\mathcal{I}'$ be a repair of I'. We shall build an instance $\mathcal{I}$ as follows:

for $i := 1$ to m **do**
  for each tuple $S_i(\underline{c}_i, d_i)$ of $\mathcal{I}'$ **do**
    Let $R_i(\vec{c}_i, \vec{d}_i)$ be a tuple of I such that c_i appears in $\vec{c}_i$ at all the positions of x_i in $\vec{x}_i$, and
      d_i appears in $\vec{d}_i$ at the position of y_i in $\vec{y}_i$
    Add $R_i(\vec{c}_i, \vec{d}_i)$ to $\mathcal{I}$
  end for
end for
for $i := m + 1$ to n **do**
  for each tuple $R_i(\vec{c}_i, \vec{d}_i)$ in I **do**
    Add $R_i(\vec{c}_i, \vec{d}_i)$ to $\mathcal{I}$
  end for
end for

We will now show that $\mathcal{I}$ is a repair of I. Towards a contradiction, assume that $\mathcal{I}$ is not a repair of I. Then, there are values $\vec{c}_i$, $\vec{d}_i$, and $\vec{d}_i'$ such that $\vec{d}_i \neq \vec{d}_i'$, $R_i(\vec{c}_i, \vec{d}_i) \in \mathcal{I}$, and $R_i(\vec{c}_i, \vec{d}_i') \in \mathcal{I}$.

First, assume that $1 \leqslant i \leqslant m$. For every variable w such that $w \notin L$ and w occurs in R_i, $w \in F$. Thus, w is assigned the same constant $\nu_F(w)$ in every tuple of I. By Proposition 2, $\mathcal{I} \subseteq I$. Therefore, there are constants c_i, d_i and d_i' such that $d_i \neq d_i'$, c_i appears in $\vec{c}_i$ at the positions of x_i in $\vec{x}_i$, and d_i and d_i' appears in $\vec{d}_i$ and $\vec{d}_i'$, respectively, at the position of y_i in $\vec{y}_i$. By construction of $\mathcal{I}$, there are tuples $S_i(\underline{c}_i, d_i)$ and $S_i(\underline{c}_i, d_i')$ in $\mathcal{I}'$. Since $d_i \neq d_i'$, $\mathcal{I}'$ does not satisfy the key constraints of Σ'. Thus, $\mathcal{I}'$ is not a repair; contradiction.

Now, assume that $m < i \leqslant n$. Notice that we assume that the join graph of q' is a cycle. Since q is in $\mathcal{C}^*$, there exists some variable w such that w occurs in $\vec{x}_i$ and w does not occur in any of $R_1, \ldots, R_m$. Thus, $w \in U$. Since the variables of U are assigned a different constant in every iteration of the algorithm that constructs I, if two tuples $R_i(\underline{\vec{c}_i}, \vec{d}_i)$ and $R_i(\underline{\vec{c}'_i}, \vec{d}'_i)$ are added at different iterations, then $\vec{c}_i \neq \vec{c}'_i$. Therefore, the extension of R_i in I satisfies the key dependencies of Σ. Thus, by construction of $\mathcal{I}$, the extension of R_i in $\mathcal{I}$ satisfies the key constraints of Σ. Thus, $\mathcal{I}$ is a repair of I; contradiction.

We conclude that $\mathcal{I}$ is a repair of I. Since $\textbf{consistent}_{\Sigma}(q, I) = \texttt{true}$, $\mathcal{I} \models q$. Thus, there exists some valuation ν_q such that $\mathcal{I} \models R_1(\underline{\vec{x}}_1, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}}_n, \vec{y}_n)[\nu_q]$. Let $\nu_{q'}$ be a valuation for the variables of q' such that, for $1 \leqslant i \leqslant m$:

- $\nu_{q'}(x_i) = \nu_q(x_i)$,
- $\nu_{q'}(y_i) = \nu_q(y_i)$.

It is easy to see that $\mathcal{I}' \models S_1(\underline{x}_1, y_1) \wedge \cdots \wedge S_m(\underline{x}_m, y_m)[\nu_{q'}]$. Thus, $\mathcal{I}' \models q'$. □

We are now ready to prove Lemma 11, which gives a polynomial-time reduction from the problem of computing consistent answers for the queries of Lemmas 5 or 8 to every query in $\mathcal{C}_{hard}$. From this, Theorem 2 follows directly.

Lemma 11. *Let q be a query s.t. $q \in \mathcal{C}_{hard}$. 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*:

- $\exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$,
- $\exists x, y.T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)$.

Proof. Let G be the join graph of q. Let G' be an induced subgraph of G such that:

- G' is connected, and
- G' is not a tree, and
- if G'' is a proper induced subgraph of G', and G'' is connected, then G'' is a tree.

Let $P = \langle R_1, R_2, R_1 \rangle$ be a cycle of G'. Let $R_1(\underline{\vec{x}}_1, \vec{y}_1)$ and $R_2(\underline{\vec{x}}_2, \vec{y}_2)$ be the literals in G'. Assume that there is some variable y such that y occurs in $\vec{y}_1$ and $\vec{y}_2$. By Definition of $\mathcal{C}^*$, there is no key-to-key join between R_1 and R_2. Therefore, there exists a variable x such that x occurs in $\vec{x}_1$, and x does not occur in $\vec{x}_2$; and a variable x' such that x' occurs in $\vec{x}_2$ and x' does not occur in $\vec{x}_1$. Let $q' = S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$. By Lemma 10, there is a polynomial-time reduction from $\texttt{CONSISTENT}(q', \Sigma')$ to $\texttt{CONSISTENT}(q, \Sigma)$.

Let $P = \langle R_1, \ldots, R_m, R_1 \rangle$ be a cycle of G'. Let $R_1(\underline{\vec{x}}_1, \vec{y}_1), \ldots, R_m(\underline{\vec{x}}_m, \vec{y}_m)$ be the literals of P. Let $w_1, w_2, \ldots, w_m$ be variables such that w_i occurs in $\vec{y}_i$ and in $R_{(i \bmod m)+1}$, for every $1 \leqslant i \leqslant m$. Assume that there is some w_i such that $1 \leqslant i \leqslant m$ and w_i occurs in some literal R_j of q such that $j \neq i$ and $j \neq (i \bmod m) + 1$. Then $\{R_1, \ldots, R_i, R_j, \ldots, R_1\}$ is a cycle. Therefore G' contains a proper induced subgraph G'' such that G'' is connected, and G'' is not a tree; contradiction. Let $q'' = S_1(\underline{w}_m, w_1) \wedge S_2(\underline{w}_1, w_2) \wedge \cdots \wedge S_m(\underline{w}_{m-1}, w_m)$. It can be checked that q and q'' satisfy the conditions of Lemma 10. Consequently, there is a polynomial-time reduction from $\texttt{CONSISTENT}(q,'' \Sigma'')$ to $\texttt{CONSISTENT}(q, \Sigma)$. Let $q' = \exists x, y.T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)$. By Lemma 9, there is a polynomial-time reduction from $\texttt{CONSISTENT}(q', \Sigma')$ to $\texttt{CONSISTENT}(q,'' \Sigma'')$. □

Finally, we give the proof for Theorem 2, the main result of this section.

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.*

Proof. By Lemma 7, $\texttt{CONSISTENT}(q, \Sigma)$ is in coNP. In order to prove hardness, let q' be one of the following queries:

- $\exists x, x', y.S_1(\underline{x}, y) \wedge S_2(\underline{x}', y)$,
- $\exists x, y.T_1(\underline{x}, y) \wedge T_2(\underline{y}, x)$.

By Lemma 11, there is a polynomial-time reduction from CONSISTENT(q', Σ') to CONSISTENT(q, Σ). By Lemmas 5 and 8, CONSISTENT(q', Σ') is coNP-hard. Thus, CONSISTENT(q, Σ) is coNP-hard. □

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., [5] and [3]). 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.

The work on disjunctive databases [13] 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 [9], 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.

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. [1] consider quantifier-free conjunctive queries (i.e., queries without existential quantifiers). Chomicki and Marcinkowski [6] 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 [6] and Calì et al. [4] 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.

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 nonkey-to-key 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 our algorithm that considers such queries.

The focus of this paper is on producing first-order rewritings. For the queries in $\mathcal{C}^*$, every tractable query is first-order rewritable. However, in general, a polynomial-time computable query may not be first-order rewritable. Indeed, in our earlier work, we have shown examples of conjunctive queries with repeated relation symbols that are tractable but not first-order rewritable [8]. It is still open, however, whether there are such examples for the class of conjunctive queries without repeated relations symbols.

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 thank Marcelo Arenas, Pablo Barcelo, Leonid Libkin, and Ken Pu for their comments and feedback.

References

[1] M. Arenas, L. Bertossi, J. Chomicki, Consistent query answers in inconsistent databases, in: Principles of Database Systems (PODS), 1999, pp. 68–79.

[2] S. Abiteboul, O.M. Duschka, Complexity of answering queries using materialized views, in: Principles of Database Systems (PODS), 1998, pp. 254–263.
[3] L. Bravo, L. Bertossi, Logic programs for consistently querying data integration systems, in: International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 10–15.
[4] A. Calì, D. Lembo, R. Rosati, On the decidability and complexity of query answering over inconsistent and incomplete databases, in: Principles of Database Systems (PODS), 2003, pp. 260–271.
[5] A. Calì, D. Lembo, R. Rosati, Query rewriting and answering under constraints in data integration systems, in: International Joint Conference on Artificial Intelligence (IJCAI), 2003, pp. 16–21.
[6] J. Chomicki, J. Marcinkowski, Minimal-change integrity maintenance using tuple deletions, Inform. and Comput. 1–2 (197) (2005) 90–121.
[7] R. Fagin, P. Kolaitis, R. Miller, L. Popa, Data exchange: Semantics and query answering, Theoret. Comput. Sci. 336 (1) (2005) 89–124.
[8] A. Fuxman, R.J. Miller, Towards inconsistency management in data integration systems, in: Workshop on Information Integration on the Web, 2003, pp. 143–148.
[9] T. Imielinski, R. van der Meyden, K. Vadaparty, Complexity tailored design: A new design methodology for databases with incomplete information, J. Comput. System Sci. 51 (3) (1995) 405–432.
[10] R.E. Ladner, On the structure of polynomial time reducibility, J. ACM 22 (1) (1975) 155–171.
[11] M. Lenzerini, Data integration: A theoretical perspective, in: Principles of Database Systems (PODS), 2002, pp. 233–246.
[12] L. Libkin, Elements Of Finite Model Theory, Springer, 2004.
[13] R. van der Meyden, Logical approaches to incomplete information: A survey, in: Logics for Databases and Information Systems, Kluwer, 1998, pp. 307–356.