Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints

PARASCHOS KOUTRIS, University of Washington
JEF WIJSEN, University of Mons

A relational database is said to be uncertain if primary key constraints can possibly be violated. A repair (or possible world) of an uncertain database is obtained by selecting a maximal number of tuples without ever selecting two distinct tuples with the same primary key value. For any Boolean query q, CERTAINTY(q) is the problem that takes an uncertain database **db** as input and asks whether q is true in every repair of **db**. The complexity of this problem has been particularly studied for q ranging over the class of self-join-free Boolean conjunctive queries. A research challenge is to determine, given q, whether CERTAINTY(q) belongs to complexity classes **FO**, **P**, or **coNP**-complete.

In this article, we combine existing techniques for studying this complexity classification task. We show that, for any self-join-free Boolean conjunctive query q, it can be decided whether or not CERTAINTY(q) is in **FO**. We additionally show how to construct a single SQL query for solving CERTAINTY(q) if it is in **FO**. Further, for any self-join-free Boolean conjunctive query q, CERTAINTY(q) is either in **P** or **coNP**-complete and the complexity dichotomy is effective. This settles a research question that has been open for 10 years.

CCS Concepts: • **Information systems → Relational database query languages**; • **Theory of computation → Database theory;**

Additional Key Words and Phrases: Attack graph, complexity dichotomy, conjunctive queries, consistent query answering, primary keys

ACM Reference Format:
Paraschos Koutris and Jef Wijsen. 2017. Consistent query answering for self-join-free conjunctive queries under primary key constraints. ACM Trans. Database Syst. 42, 2, Article 9 (April 2017), 45 pages.
DOI: http://dx.doi.org/10.1145/3068334

1. INTRODUCTION

A database is inconsistent if it violates one or more integrity constraints that the data is required to obey. Inconsistency in the data can arise in various settings. For example, if data is integrated from different sources, then the resulting integrated database can be inconsistent even if each individual source was consistent.

The standard method of dealing with inconsistency is *data cleaning*, in which the database is first *repaired* to satisfy the integrity constraints and then the cleaned version is used to answer queries. Since an inconsistent database can typically be

This article extends an earlier publication presented at the *2015 ACM Symposium on Principles of Database Systems* [24]. A modified version entitled "Consistent Query Answering for Primary Keys" appeared in the March 2016 issue of *ACM SIGMOD Record* [25]. This work was supported in part by the NSF through NSF grant NSF IIS-1115188.
Authors' addresses: P. Koutris, University of Wisconsin-Madison, Department of Computer Sciences, 1210 W. Dayton St., Madison, WI 53706-1613, USA; email: paris@cs.wisc.edu; J. Wijsen, University of Mons (UMONS), Departement d'Informatique, Avenue du Champ de Mars, 6, B-7000 Mons, Belgium; email: jef.wijsen@umons.ac.be.

DOI: http://dx.doi.org/10.1145/3068334

E	EID	ENAME	CITY	DNAME
	E1	Smith	London	Training
	E2	Jones	Paris	Training
	E3	Blake	Paris	HR
	E3	Blake	London	HR
	E4	Clark	London	HR
	E5	Adams	Athens	HR

D	DNAME	BUDGET	CITY	MGR
	Training	120	London	E3
	HR	300	Paris	E3
	HR	310	Paris	E5

Fig. 1. Example of an inconsistent database that violates the primary key constraints.

repaired in many different ways, it happens that data cleaners make arbitrary choices about which data to keep and which data to delete, which means that information may be lost. An alternative to data cleaning is *consistent query answering* (CQA), which was first introduced in [2]. In this paradigm, no data is lost and queries are answered by considering all possible repairs of the inconsistent database.

In this article, we focus on integrity constraints that are *primary key constraints*. Consider the database in Figure 1, which includes the table E that stores employee information and the table D that stores department information. The first line in table E stores that Smith was born in London and works for the Training department. The first line in table D stores that the Training department is located in London and managed by employee E3, with a budget of 120. The primary keys of E and D are EID and DNAME, respectively. There are two foreign keys: every DNAME-value in E must occur in the DNAME-column of D, and every MGR-value in D must occur in the EID-column of E. Employees can manage departments that are distinct from the department they are working for. This typically happens when one department is subordinate to another department. For example, Training may be managed by an employee of Human Resources (HR).

Both tables in the database contain primary key violations: for example, the employee with EID E3 has two entries in table E. Two or more tuples that agree on their primary key represent mutually exclusive possibilities. Such tuples are said to form a *block*; in Figure 1, blocks are separated by dashed lines for readability. The reader should note that even though we do not know the exact city where Blake was born, we still know that it is either Paris or London; contrast this with the case in which we would represent Blake's city of birth by a single uninformative NULL. Likewise, the uncertain database represents that the HR department is either managed by E3 with a budget of 300 or by E5 with a budget of 310.

Blocks with two or more tuples model uncertainty: exactly one of the tuples is true, but we do not know which one is true. Therefore, we use the term *uncertain database* to refer to databases that can contain primary key violations. A *repair* (or possible world) of an uncertain database is obtained by selecting exactly one tuple from each block. Although we will not deal with foreign key constraints from here, we incidentally note that if an uncertain database satisfies some foreign key constraint, then each repair will satisfy this foreign key constraint as well, because all primary-key values that are present in the uncertain database will be present in each repair.

In this article, we study how to answer conjunctive queries on uncertain databases. We allow joins, but we disallow that a table be joined with itself (called a *self-join*). Conjunctive queries without self-joins are widely used database queries. They correspond to SQL queries of the form `SELECT...FROM...WHERE...`, in which no relation occurs more than once after the `FROM`-clause and in which the `WHERE`-clause equates attributes with each other or with constant values. It is natural to distinguish between possible and certain answers: the *possible* answer to a query consists of the tuples that are in the answer to the query on some repair, while the *certain* (or consistent) answer consists of the tuples that are in the answer to the query on every repair. Consider, for example, the query *"Get names for employees who were born in London,"* which is encoded in SQL as follows.

```
SELECT E1.ENAME
FROM   E AS E1
WHERE  E1.CITY='London';
```

The possible answer to this query consists of Smith, Clark, and Blake. The certain answer consists of Smith and Clark. Blake does not belong to the certain answer since the database leaves open the possibility that Blake was born in Paris. For conjunctive queries without self-joins, it is easy to see that the possible answer is obtained by executing the query on the uncertain database. Computing certain answers is a more difficult task. Interestingly, the certain answer to the preceding query is computed by the following SQL query.

```
SELECT E1.ENAME
FROM   E AS E1
WHERE  E1.CITY='London'
AND    NOT EXISTS ( SELECT *
                    FROM   E AS E2
                    WHERE  E2.EID=E1.EID
                    AND    E2.CITY≠'London' );
```

The NOT EXISTS subquery checks the nonexistence of a city of birth other than London and thus excludes Blake from the answer. Unfortunately, it is not always possible to obtain certain answers directly in SQL. We show in this article that, for all conjunctive queries without self-joins, CQA with respect to primary keys can be classified into one of three classes of increasing complexity:

(1) For some queries, the certain answer can be expressed in relational calculus (or first-order logic), thus can be written in SQL. One such query was shown before; as we will see in Section 3, another example is the query *"Get names for departments that are self-managed (i.e., are managed by one of their own employees)."*

```
SELECT D.DNAME
FROM   E, D
WHERE  E.EID=D.MGR
AND    E.DNAME=D.DNAME;
```

The certain answer consists of HR. Notice that although there are two possibilities for the manager of HR, we know for certain that HR is self-managed. The SQL query that computes certain answers is shown at the end of Section 5.3.

(2) For some queries, the certain answer can be computed in polynomial time (with respect to the database size) but cannot be expressed in relational calculus. We will see in Section 3 that an example of such a query is *"Get names for employees who manage the department for which they work."*

```
SELECT E.ENAME
FROM   E, D
WHERE  E.EID=D.MGR
AND    E.DNAME=D.DNAME;
```

The certain answer is empty. Note that the only difference between the latter query and the previous one is the attribute that occurs after SELECT (E.ENAME versus D.DNAME).

(3) For some queries, the certain answer cannot be obtained in polynomial time (unless **P** = **NP**), since it is **coNP**-hard to compute the certain answer. We will see in Section 3 that an example of such a query is *"Get names for employees who work in the city of their birth."*

```
SELECT E.ENAME
FROM   E, D
WHERE  E.CITY=D.CITY
AND    E.DNAME=D.DNAME;
```

The certain answer consists only of Smith.

This threefold complexity classification gives rise to the following contributions of practical interest.

- In Section 3, we exhibit an easy-to-implement decision procedure that takes as input a conjunctive query q without self-joins and decides to which of the three aforementioned classes q belongs.
- In Section 5, we outline an easy-to-implement procedure whose input is a query q belonging to the first class and whose output is an SQL query that computes the certain answers to q.
- In Sections 7 and 8, we specify a procedure whose input is a query q belonging to the first or second class and whose output is a polynomial-time program (but not an SQL query) that computes the certain answers to q.

For every conjunctive query q, practitioners can always fall back on procedures that build a Disjunctive Logic Program [19] or a Binary Integer Program [22] whose executions yield the certain answers to q. These executions will in the worst case take exponential time in the size of the database, which is unnecessarily expensive for queries q falling in the first or second class. As summarized in [38], with our results, "it is now possible to determine and delegate the computation of CQA to the right engine; pushing computations to the RDBMS whenever possible, and relying on more algorithmic or expressive engines otherwise."

The remainder of this section pinpoints the theoretical contribution of this article. For every Boolean query q, define CERTAINTY(q) as the following computational problem.

PROBLEM:	CERTAINTY(q)
INPUT:	uncertain database **db**
QUESTION:	Does every repair of **db** satisfy q?

Significantly, the Boolean query q is not part of the input. Every Boolean query q thus gives rise to a new problem. Since the input to CERTAINTY(q) is an uncertain database, we consider the *data complexity* of the problem. The restriction to Boolean queries q in the complexity study of CERTAINTY(q) eases the technical treatment but is not fundamental. In Section 3.3, we will show how our complexity results carry over to queries with free variables.

Although the problem CERTAINTY(q) is defined for arbitrary Boolean queries q, its complexity will only be studied for queries q that are expressible in particular fragments of first-order logic. For every first-order query q, the problem CERTAINTY(q) is in **coNP**, because a nondeterministic Turing machine can guess a repair and verify in polynomial time that it falsifies q. Further, it has been known for several years [9, 17, 42] that there exist Boolean conjunctive queries q_1, q_2, and q_3 such that CERTAINTY(q_1)

is in **FO**, CERTAINTY(q_2) is in $\mathbf{P} \setminus \mathbf{FO}$, and CERTAINTY($q_3$) is **coNP**-hard. It is then significant to ask whether we can determine the complexity of CERTAINTY(q) for every Boolean conjunctive query q:

COMPLEXITY CLASSIFICATION TASK	
INPUT:	A Boolean conjunctive query q
QUESTION:	What complexity classes does CERTAINTY(q) belong to, for which complexity classes of interest are **FO**, **P**, and **coNP**-complete?

We recall from descriptive complexity theory (see, e.g., [20]) that membership of CERTAINTY(q_1) in **FO** is equivalent to the existence of a Boolean first-order query φ_1, called *consistent first-order rewriting* in our setting, such that, for every uncertain database **db**, the answer to CERTAINTY(q_1) is "yes" on input **db** if and only if **db** satisfies φ_1. If CERTAINTY(q_1) is in **FO**, then CERTAINTY(q_1) is in **P**, and thus "efficiently solvable." Of more interest to database practitioners perhaps, if the problem CERTAINTY(q_1) is in **FO**, then the problem is "solvable in SQL" using standard database technology by executing an SQL encoding of a consistent first-order rewriting.

In this article, we make significant progress in the complexity classification task under the restriction that conjunctive queries are self-join-free (i.e., contain no self-join) and contain no built-in predicates. Our main contribution is that, for every self-join-free Boolean conjunctive query q,

(1) it can be decided whether or not CERTAINTY(q) is in **FO**; and
(2) CERTAINTY(q) is either in **P** or **coNP**-complete and it can be decided which of the two cases applies.

In connection with the latter item, we recall that under the widely believed conjecture $\mathbf{P} \neq \mathbf{coNP}$, there exist problems in **coNP** that are neither in **P** nor **coNP**-complete [26]. It is therefore significant that a dichotomy between **P** and **coNP**-complete exists in the class of problems that contains CERTAINTY(q) whenever q is a self-join-free Boolean conjunctive query.

The complexity classification task of CERTAINTY(q) for self-join-free Boolean conjunctive queries q has been studied since 2005 [17] and the following results were obtained before the current article: CERTAINTY(q) is either in **P** or **coNP**-complete if q joins at most two relations [21] or if all primary keys consist of a single attribute [23]; Wijsen [41, 43] showed that membership of CERTAINTY(q) in **FO** is decidable if q is α-acyclic in the sense of [13]. It is noteworthy that, for Boolean conjunctive queries q with self-joins or built-in predicates, the complexity classification task for CERTAINTY(q) remains largely open today. On the other hand, the extension from Boolean queries to queries with free variables is well understood and explained in Section 3.3.

This article is organized as follows. Section 2 introduces our data and query model. Section 3 defines attack graphs for Boolean conjunctive queries, extending an older notion of attack graph [43] that was defined exclusively for α-acyclic Boolean conjunctive queries. Section 3 also states the main result of the article, Theorem 3.2. Section 4 establishes an effective procedure that takes in a self-join-free Boolean conjunctive query q, and decides whether CERTAINTY(q) is in **FO**. Section 5 explains how to express CERTAINTY(q) in SQL if CERTAINTY(q) is in **FO**. Section 6 provides a sufficient condition for **coNP**-hardness of CERTAINTY(q) for any self-join-free Boolean conjunctive query q. Sections 7 and 8 show that if the condition is not satisfied, then CERTAINTY(q) is in **P**. Section 9 discusses related work. The appendix contains proofs of lemmas and theorems.

2. PRELIMINARIES

We assume disjoint sets of *variables* and *constants*. If $\vec{x}$ is a sequence containing variables and constants, then $\mathsf{vars}(\vec{x})$ denotes the set of variables that occur in $\vec{x}$. A *valuation* over a set U of variables is a total mapping θ from U to the set of constants. At several places, it is implicitly understood that such a valuation θ is extended to be the identity on constants and on variables not in U. If $V \subseteq U$, then $\theta[V]$ denotes the restriction of θ to V.

If θ is a valuation over a set U of variables, x is a variable, and a is a constant, then $\theta_{[x \mapsto a]}$ is the valuation over $U \cup \{x\}$ such that $\theta_{[x \mapsto a]}(x) = a$ and for every variable y such that $y \neq x$, $\theta_{[x \mapsto a]}(y) = \theta(y)$. Note that $x \in U$ is allowed.

Atoms and key-equal facts. Each *relation name* R of arity n, $n \geq 1$, has a unique *primary key* that is a set $\{1, 2, \ldots, k\}$, where $1 \leq k \leq n$. We say that R has *signature* $[n, k]$ if R has arity n and primary key $\{1, 2, \ldots, k\}$. We say that R is *simple-key* if $k = 1$. Elements of the primary key are called *primary-key positions*, while $k+1, k+2, \ldots, n$ are *nonprimary-key positions*. For all positive integers n, k such that $1 \leq k \leq n$, we assume denumerably many relation names with signature $[n, k]$. For example, the relation name E from Figure 1 has signature $[4, 1]$ and is simple-key.

If R is a relation name with signature $[n, k]$, then $R(s_1, \ldots, s_n)$ is called an *R-atom* (or simply atom), where each s_i is either a constant or a variable ($1 \leq i \leq n$). Such an atom is commonly written as $R(\underline{\vec{x}}, \vec{y})$, where the primary-key value $\vec{x} = s_1, \ldots, s_k$ is underlined and $\vec{y} = s_{k+1}, \ldots, s_n$. An *$R$-fact* (or simply fact) is an R-atom in which no variable occurs. Two facts $R_1(\underline{\vec{a}_1}, \vec{b}_1)$, $R_2(\underline{\vec{a}_2}, \vec{b}_2)$ are *key-equal* if $R_1 = R_2$ and $\vec{a}_1 = \vec{a}_2$. An R-atom or an R-fact is *simple-key* if R is simple-key.

We will use the letters F, G, H for atoms. For an atom $F = R(\underline{\vec{x}}, \vec{y})$, we denote by $\mathsf{key}(F)$ the set of variables that occur in $\vec{x}$ and by $\mathsf{vars}(F)$ the set of variables that occur in F, that is, $\mathsf{key}(F) = \mathsf{vars}(\vec{x})$ and $\mathsf{vars}(F) = \mathsf{vars}(\vec{x}) \cup \mathsf{vars}(\vec{y})$.

Uncertain databases, blocks, and repairs. A *database schema* is a finite set of relation names. All constructs that follow are defined relative to a fixed database schema.

An *uncertain database* is a finite set $\mathbf{db}$ of facts using only the relation names of the schema. We refer to databases as "uncertain databases" to stress that such databases can violate primary key constraints.

We write $\mathsf{adom}(\mathbf{db})$ for the active domain of $\mathbf{db}$ (i.e., the set of constants that occur in $\mathbf{db}$). A *block* of $\mathbf{db}$ is a maximal set of key-equal facts of $\mathbf{db}$. The term R-block refers to a block of R-facts, that is, facts with relation name R. If A is a fact of $\mathbf{db}$, then $\mathsf{block}(A, \mathbf{db})$ denotes the block of $\mathbf{db}$ that contains A. An uncertain database $\mathbf{db}$ is *consistent* if no two distinct facts are key-equal (i.e., if every block of $\mathbf{db}$ is a singleton). A *repair* of $\mathbf{db}$ is a maximal (with respect to set inclusion) consistent subset of $\mathbf{db}$. We write $\mathsf{rset}(\mathbf{db})$ for the set of repairs of $\mathbf{db}$.

Example 2.1. In the uncertain database of Figure 1, blocks are separated by dashed lines. This uncertain database contains one E-block of cardinality 2 and one D-block of cardinality 2; all other blocks have cardinality 1. This uncertain database has four repairs, each of which can be obtained by selecting exactly one fact from each block.

Boolean conjunctive queries. A *Boolean query* is a mapping q that associates a Boolean (true or false) to each uncertain database, such that q is closed under isomorphism [28]. We write $\mathbf{db} \models q$ to denote that q associates true to $\mathbf{db}$, in which case $\mathbf{db}$ is said to *satisfy* q. A Boolean query q can be viewed as a decision problem that takes an uncertain database as input and asks whether $\mathbf{db}$ satisfies q. In this article, the complexity class $\mathbf{FO}$ stands for the set of Boolean queries that can be defined in first-order logic with equality and constants, but without other built-in predicates or function symbols.

A *Boolean conjunctive query* is a finite set $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \ldots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$ of atoms, without equality or built-in predicates. We denote by $\mathsf{vars}(q)$ the set of variables that occur in q. The set q represents the first-order sentence

$$\exists u_1 \cdots \exists u_k \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n)\right),$$

where $\{u_1, \ldots, u_k\} = \mathsf{vars}(q)$. This query q is satisfied by an uncertain database $\mathbf{db}$ if there exists a valuation θ over $\mathsf{vars}(q)$ such that, for each $i \in \{1, \ldots, n\}$, $R_i(\underline{\vec{a}}, \vec{b}) \in \mathbf{db}$ with $\vec{a} = \theta(\vec{x}_i)$ and $\vec{b} = \theta(\vec{y}_i)$.

We say that a Boolean conjunctive query q has a *self-join* if some relation name occurs more than once in q. If q has no self-join, then it is called *self-join-free*. We write BCQ for the class of Boolean conjunctive queries, and sjfBCQ for the class of self-join-free Boolean conjunctive queries. If q is an sjfBCQ query with an R-atom, then, by an abuse of notation, we write R to mean the R-atom of q.

If q is a Boolean conjunctive query, $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ is a sequence of distinct variables that occur in q, and $\vec{a} = \langle a_1, \ldots, a_\ell \rangle$ is a sequence of constants, then $q_{[\vec{x} \mapsto \vec{a}]}$ denotes the query obtained from q by replacing all occurrences of x_i with a_i for all $1 \leq i \leq \ell$.

We now introduce some notions that will facilitate the complexity study of $\mathsf{CERTAINTY}(q)$, as expressed by Lemmas 2.4 and 2.5.

Typed uncertain databases. For every variable x, we assume an infinite set of constants, denoted $\mathsf{type}(x)$, such that $x \neq y$ implies $\mathsf{type}(x) \cap \mathsf{type}(y) = \emptyset$. Let $q \in \mathsf{sjfBCQ}$ and let $\mathbf{db}$ be an uncertain database. We say that $\mathbf{db}$ is *typed relative to* q if, for every atom $R(x_1, \ldots, x_n)$ in q, for every $i \in \{1, \ldots, n\}$, if x_i is a variable, then for every fact $R(a_1, \ldots, a_n)$ in $\mathbf{db}$, $a_i \in \mathsf{type}(x_i)$ and the constant a_i does not occur in q.

An uncertain database $\mathbf{db}$ can be trivially transformed into an uncertain database $\mathbf{db}'$ that is typed relative to q such that the problem $\mathsf{CERTAINTY}(q)$ yields the same answer on problem instances $\mathbf{db}$ and $\mathbf{db}'$. Indeed, we can take $\mathbf{db}'$ to be the smallest database such that, for every atom $R(x_1, \ldots, x_n)$ in q, if $\mathbf{db}$ contains $R(a_1, \ldots, a_n)$, then $\mathbf{db}'$ contains $R({a_1}^{x_1}, \ldots, {a_n}^{x_n})$. Here, each c^s denotes a constant such that (i) ${c_1}^{s_1} = {c_2}^{s_2}$ if and only if both $c_1 = c_2$ and $s_1 = s_2$ and (ii) $c^s = c$ if and only if $c = s$. Further, if x is a variable, then $\mathsf{type}(x)$ contains all (and only) constants of the form c^x. Intuitively, occurrences of constants in $\mathbf{db}$ are "tagged" by the variable or constant that occurs at the same position in q, and these tags are unique because q is self-join-free. Because of this transformation, the assumption that uncertain databases are typed can be made without loss of generality in the complexity classification task of $\mathsf{CERTAINTY}(q)$ for sjfBCQ queries q. This assumption is useful because it simplifies the technical treatment, especially in Section 8.

Purified uncertain databases. Let q be a Boolean conjunctive query and let $\mathbf{db}$ be an uncertain database. We say that a fact $A \in \mathbf{db}$ is *relevant for q in* ***db*** if, for some valuation θ over $\mathsf{vars}(q)$, $A \in \theta(q) \subseteq \mathbf{db}$. We say that $\mathbf{db}$ is *purified relative to* q if every fact $A \in \mathbf{db}$ is relevant for q in $\mathbf{db}$.

Example 2.2. Let q be the query $\{\mathsf{E}(\underline{m}, n, c_1, d), \mathsf{D}(\underline{d}, b, c_2, m)\}$, where m, n, c_1, d, b, c_2 are variables. This query asks whether some department is managed by one of its own employees. The database of Figure 1 is not purified relative to q, because the fact $\mathsf{D}(\underline{\text{Training}}, 120, \text{London}, \text{E3})$ is not relevant (there is no tuple in E containing both E3 and Training). On the other hand, the database of Figure 1 is purified relative to $\{\mathsf{E}(\underline{e}, n, c_1, d), \mathsf{D}(\underline{d}, b, c_2, m)\}$, because every department name in either table is present in the other table.

Note, incidentally, that the uncertain database of Figure 1 is not typed relative to q because $c_1 \neq c_2$ but the value 'Paris' occurs both in the position of c_1 and the position of c_2.

Frugal repairs. For every uncertain database **db**, Boolean conjunctive query q, and $X \subseteq \mathsf{vars}(q)$, we define a preorder $\preceq_q^X$ on $\mathsf{rset}(\mathbf{db})$, as follows. For every two repairs $\mathbf{r}_1, \mathbf{r}_2$, we define $\mathbf{r}_1 \preceq_q^X \mathbf{r}_2$ if, for every valuation θ over X, $\mathbf{r}_1 \models \theta(q)$ implies that $\mathbf{r}_2 \models \theta(q)$. Here, $\theta(q)$ is the query obtained from q by replacing all occurrences of each $x \in X$ with $\theta(x)$; variables not in X remain unaffected (i.e., θ is understood to be the identity on variables not in X). Clearly, $\preceq_q^X$ is a preorder (i.e., it is reflexive and transitive); its minimal elements are called $\preceq_q^X$-*frugal repairs*.[1]

Example 2.3. The relation D of Figure 1 has two repairs, denoted $\mathbf{r}_1$ and $\mathbf{r}_2$, containing D($\underline{\text{HR}}$, 300, Paris, E3) and D($\underline{\text{HR}}$, 310, Paris, E5), respectively. Both repairs contain D($\underline{\text{Training}}$, 120, London, E3). Consider the query $q = \{\mathsf{D}(\underline{u}, v, w, x)\}$. The repair $\mathbf{r}_1$ satisfies q when x is mapped to E3, but not when x is mapped to E5; the repair $\mathbf{r}_2$ satisfies q when x is mapped to E5 or E3. It is then correct to write $\mathbf{r}_1 \preceq_q^{\{x\}} \mathbf{r}_2$, but $\mathbf{r}_2 \not\preceq_q^{\{x\}} \mathbf{r}_1$. The repair $\mathbf{r}_1$ is $\preceq_q^{\{x\}}$-frugal but $\mathbf{r}_2$ is not.

Functional dependencies. Let q be a Boolean conjunctive query. A *functional dependency for* q is an expression $X \rightarrow Y$, where $X, Y \subseteq \mathsf{vars}(q)$. Let $\mathcal{V}$ be a finite set of valuations over $\mathsf{vars}(q)$. We say that $\mathcal{V}$ satisfies $X \rightarrow Y$ if, for all $\theta, \mu \in \mathcal{V}$, if $\theta[X] = \mu[X]$, then $\theta[Y] = \mu[Y]$. Let Σ be a set of functional dependencies for q. We write $\Sigma \models X \rightarrow Y$ if, for every set $\mathcal{V}$ of valuations over $\mathsf{vars}(q)$, if $\mathcal{V}$ satisfies each functional dependency in Σ, then $\mathcal{V}$ satisfies $X \rightarrow Y$. Note that the foregoing conforms with standard dependency theory if variables are viewed as attributes and valuations as tuples.

Consistent query answering. For every Boolean query q, the decision problem $\mathsf{CERTAINTY}(q)$ takes as input an uncertain database **db** and asks whether q is satisfied by every repair of **db**. It is straightforward that, for every Boolean first-order query q, $\mathsf{CERTAINTY}(q)$ is in **coNP**.

The following two lemmas are useful in the study of the complexity of $\mathsf{CERTAINTY}(q)$.

LEMMA 2.4 ([44]). *Let q be a Boolean conjunctive query. Let* **db** *be an uncertain database. It is possible to compute in polynomial time an uncertain database* $\mathbf{db}'$ *that is purified relative to q such that every repair of* **db** *satisfies q if and only if every repair of* $\mathbf{db}'$ *satisfies q.*

LEMMA 2.5. *Let q be a Boolean conjunctive query and let $X \subseteq \mathsf{vars}(q)$. Let* **db** *be an uncertain database. Then, every repair of* **db** *satisfies q if and only if every $\preceq_q^X$-frugal repair of* **db** *satisfies q.*

First-order reduction. In this article, all considered decision problems take as input databases over some fixed schema. Let P_1 and P_2 be two such decision problems over schemas S_1 and S_2, respectively. A many-one reduction ρ from P_1 to P_2 is said to be *first-order* if for every n-ary relation name R in S_2, there exists a first-order query $\varphi_R(x_1, \ldots, x_n)$ over S_1 such that, for every database I over S_1, for all constants $a_1, \ldots, a_n$, we have that $R(a_1, \ldots, a_n) \in \rho(I)$ if and only if $I \models \varphi_R((a_1, \ldots, a_n)$.

3. ATTACK GRAPHS AND MAIN RESULTS

Wijsen [41] introduced attack graphs for studying first-order expressibility of $\mathsf{CERTAINTY}(q)$ for sjfBCQ queries q that are α-acyclic. Here, we extend the notion of attack graph to all sjfBCQ queries.

This section contains five subsections. Section 3.1 defines attack graphs. Section 3.2 presents Theorem 3.2, which is the main theorem of the article, settling the complexity classification task introduced in Section 1. This theorem is stated only for queries that

[1] $\mathbf{r}_1$ is minimal if, for all $\mathbf{r}_2$, if $\mathbf{r}_2 \preceq_q^X \mathbf{r}_1$, then $\mathbf{r}_1 \preceq_q^X \mathbf{r}_2$.

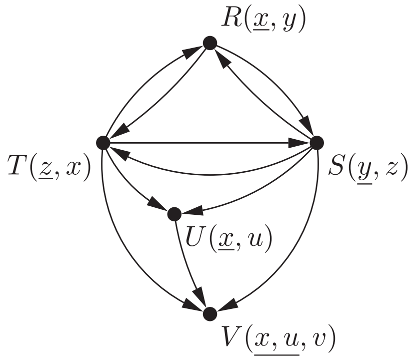


Fig. 2. Attack graph of the query in Example 3.1.

are Boolean. Since non-Boolean queries are common in practice, we explain in Section 3.3 how the theorem can be applied to non-Boolean queries. Section 3.4 illustrates the complexity classification by means of the queries introduced in Section 1. Finally, Section 3.5 introduces some helpful constructs and lemmas that will be used later in the proof of Theorem 3.2.

3.1. The Attack Graph Tool

Let $q \in \mathsf{sjfBCQ}$. We define $\mathcal{K}(q)$ as the following set of functional dependencies:

$$\mathcal{K}(q) := \{\mathsf{key}(F) \to \mathsf{vars}(F) \mid F \in q\}.$$

For every atom $F \in q$, we define $F^{+,q}$ and $F^{\boxplus,q}$ as the following sets of variables:

$$\begin{aligned} F^{+,q} &:= \{x \in \mathsf{vars}(q) \mid \mathcal{K}(q \setminus \{F\}) \models \mathsf{key}(F) \to x\}, \\ F^{\boxplus,q} &:= \{x \in \mathsf{vars}(q) \mid \mathcal{K}(q) \models \mathsf{key}(F) \to x\}. \end{aligned}$$

The *attack graph of* q is a directed graph whose vertices are the atoms of q. There is a directed edge from F to G ($F \neq G$) if there exists a sequence

$$F_0, F_1, \ldots, F_n \tag{1}$$

of (not necessarily distinct) atoms of q such that

- $F_0 = F$ and $F_n = G$; and
- for all $i \in \{0, \ldots, n-1\}$, $\mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1}) \not\subseteq F^{+,q}$.

A directed edge from F to G in the attack graph of q is also called an *attack from F to G*, denoted by $F \overset{q}{\rightsquigarrow} G$. The sequence (1) is called a *witness* for the attack $F \overset{q}{\rightsquigarrow} G$. We will often add variables to a witness: if we write $F_0 \overset{z_1}{\frown} F_1 \overset{z_2}{\frown} F_2 \cdots \overset{z_n}{\frown} F_n$, then it is understood that, for $i \in \{1, \ldots, n\}$, $z_i \in \mathsf{vars}(F_{i-1}) \cap \mathsf{vars}(F_i)$ and $z_i \notin {F_0}^{+,q}$. If $F \overset{q}{\rightsquigarrow} G$, then we also say that F *attacks* G (or that G is attacked by F).

An attack from F to G is called *weak* if $\mathcal{K}(q) \models \mathsf{key}(F) \to \mathsf{key}(G)$; otherwise, it is *strong*. A directed cycle in the attack graph of q is called *weak* if all attacks in the cycle are weak; otherwise, the cycle is called *strong*.

Example 3.1. Let $q = \{R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x), U(\underline{x}, u), V(\underline{x, u}, v)\}$. By a little abuse of notation, we denote each atom by its relation name (e.g., R is used to denote the atom $R(\underline{x}, y)$). We have that $\mathcal{K}(q \setminus \{R\}) = \{y \to z, z \to x, x \to u, \{x, u\} \to v\}$. We have that $R^{+,q} = \{x, u, v\}$, the closure of $\mathsf{key}(R) = \{x\}$ with respect to $\mathcal{K}(q \setminus \{R\})$. A witness for $R \overset{q}{\rightsquigarrow} T$ is $R \overset{y}{\frown} S \overset{z}{\frown} T$. The complete attack graph is shown in Figure 2. All attacks are weak. Since $\mathcal{K}(q) = \{x \rightarrow y, y \rightarrow z, z \rightarrow x, x \rightarrow u, \{x, u\} \rightarrow v\}$, we have that $R^{\boxplus,q} = \{x, y, z, u, v\}$.

3.2. Main Theorem

Equipped with the notion of attack graph, we can now present our threefold solution to the complexity classification task for $\mathsf{CERTAINTY}(q)$ with $q \in \mathsf{sjfBCQ}$. In the statement of the following theorem, **L** is the class of decision problems solvable in deterministic logarithmic space.

THEOREM 3.2. *For every $q \in \mathsf{sjfBCQ}$,*

(1) *if the attack graph of q is acyclic, then* $\mathsf{CERTAINTY}(q)$ *is in* **FO**;
(2) *if the attack graph of q is cyclic but contains no strong cycle, then* $\mathsf{CERTAINTY}(q)$ *is in* **P** *and is* **L**-*hard (under first-order reductions); and*
(3) *if the attack graph of q contains a strong cycle, then* $\mathsf{CERTAINTY}(q)$ *is* **coNP**-*complete.*

Furthermore, it can be decided in quadratic time in the size of q which of these three cases applies.

The following lemma establishes that the three if conditions in Theorem 3.2 can be decided in quadratic time in the size of q; its proof is given in Appendix A and relies on an existing algorithm [43].

LEMMA 3.3. *Given* sjfBCQ *query q, the following questions can be answered in quadratic time in the size of q:*

(1) *Does the attack graph of q contain a strong cycle?*
(2) *Does the attack graph of q contain a weak cycle?*

The rest of the article completes the proof of Theorem 3.2. We conclude this section by pointing out that our results are valid only for self-join-free conjunctive queries. The presence of self-joins complicates consistent query answering in a way that is currently not well understood. For example, for the sjfBCQ query $q_0 = \{R(\underline{x}, y), S(\underline{y}, a)\}$, where a is a constant, we have that $\mathsf{CERTAINTY}(q_0)$ is in **FO**. On the other hand, for the Boolean conjunctive query $q_1 = \{R(\underline{x}, y), R(\underline{y}, a)\}$, with a self-join, it is known [40] that $\mathsf{CERTAINTY}(q_1)$ is not in **FO**.

3.3. Certain Answers to Non-Boolean Queries

If $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \ldots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$ is a finite set of atoms and $\vec{f}$ is a sequence of distinct variables of $\mathsf{vars}(q)$, then $\vec{f} \leftarrow q$ is a *conjunctive query* whose set of *free variables* is $\mathsf{vars}(\vec{f})$. The conjunctive query $\vec{f} \leftarrow q$ represents the following query in relational calculus:

$$\left\{\vec{f} \mid \exists u_1 \cdots \exists u_k \left(R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_1(\underline{\vec{x}_n}, \vec{y}_n)\right)\right\},$$

where $\{u_1, \ldots, u_k\} = \mathsf{vars}(q) \setminus \mathsf{vars}(\vec{f})$. The conjunctive query $\vec{f} \leftarrow q$ is self-join-free if q is self-join-free. The syntax $\vec{f} \leftarrow q$ is adopted from rule-based conjunctive queries [1, page 41] and datalog.

The decision problem $\mathsf{CERTAINTY}(q)$ is naturally generalized to queries with free variables, as follows. Given an uncertain database **db**, the *certain answer* to the conjunctive query $\vec{f} \leftarrow q$ is the set of all sequences $\vec{a}$ of constants (of the same length as $\vec{f}$) such that $q_{[\vec{f} \mapsto \vec{a}]}$ is satisfied by every repair of **db**. That is, given an uncertain database **db**, the certain answer contains $\vec{a}$ if and only if the answer to the decision problem $\mathsf{CERTAINTY}(q_{[\vec{f} \mapsto \vec{a}]})$ is "yes." Intuitively, the certain answer contains each tuple that belongs to the output of $\vec{f} \leftarrow q$ for every repair of **db**. Then, for any fixed conjunctive query $\vec{f} \leftarrow q$, consistent query answering is the function problem that takes an uncertain database **db** as input and asks to compute the certain answer to $\vec{f} \leftarrow q$.

We now argue that our results in the complexity classification task of $\mathsf{CERTAINTY}(q)$ still stand when free variables are introduced, by treating the free variables as constants. Let $\vec{f} \leftarrow q$ with $\vec{f} = \langle x_1, \ldots, x_\ell \rangle$ be a self-join-free conjunctive query and let $\vec{c} = \langle c_1, \ldots, c_\ell \rangle$ be a sequence of constants. We ask whether $\vec{c}$ belongs to the certain answer, that is, whether the answer to the decision problem $\mathsf{CERTAINTY}(q_{[\vec{f} \mapsto \vec{c}]})$ is "yes." The complexity of the latter problem does not depend on the choice of $\vec{c}$, since uncertain databases can be assumed to be typed (see Section 2; in particular, $c_1, \ldots, c_\ell$ can be assumed to be mutually distinct) and since constants are generic. Therefore, the following implications hold:

- If $\mathsf{CERTAINTY}(q_{[\vec{f} \mapsto \vec{c}]})$ is in **FO**, then there exists a relational calculus query that takes in an uncertain database and outputs the certain answer to $\vec{f} \leftarrow q$. This case is illustrated by Example 3.4.
- If $\mathsf{CERTAINTY}(q_{[\vec{f} \mapsto \vec{c}]})$ is in **P**, then there exists a polynomial-time algorithm that takes in an uncertain database and outputs the certain answer to $\vec{f} \leftarrow q$.

The converse implications are obviously true. The following is also true:

- If $\mathsf{CERTAINTY}(q_{[\vec{f} \mapsto \vec{c}]})$ is **coNP**-hard, then, unless **P** = **NP**, there exists no polynomial-time algorithm that takes in an uncertain database and outputs the certain answer to $\vec{f} \leftarrow q$.

Then, since the constants in the sequence $\vec{c}$ are arbitrary, the complexity of computing the certain answer to $\vec{f} \leftarrow q$ can be obtained from Theorem 3.2 by treating free variables like constants in the computation of the attack graph. This is illustrated in Section 3.4.

Example 3.4. The following conjunctive query with free variables e and c asks for identifiers and cities of employees:

$$\langle e, c \rangle \leftarrow \mathsf{E}(\underline{e}, n, c, d). \tag{2}$$

If we replace in (2) the variables e and c by two arbitrary constants—say, E1 and Rome—we obtain the Boolean conjunctive query $\{\mathsf{E}(\underline{\text{'E1'}}, n, \text{'Rome'}, d)\}$, asking whether employee E1 was born in Rome. It can be easily seen that the latter query is true in every repair of an uncertain database **db** if and only if **db** satisfies the following Boolean first-order query:

$$\exists n \exists d \left(\mathsf{E}(\underline{\text{'E1'}}, n, \text{'Rome'}, d) \wedge \forall n \forall c' \forall d \left(\mathsf{E}(\underline{\text{'E1'}}, n, c', d) \rightarrow c' = \text{'Rome'} \right)\right).$$

If we substitute back e and c for E1 and Rome in the latter query, we obtain a relational calculus query that computes the certain answer to (2):

$$\left\{ e, c \mid \exists n \exists d \left(\mathsf{E}(\underline{e}, n, c, d) \wedge \forall n \forall c' \forall d \left(\mathsf{E}(\underline{e}, n, c', d) \rightarrow c' = c \right)\right)\right\}.$$

3.4. Complexity Classification Examples: Non-Boolean Queries

The query *"Get names for departments that are self-managed (i.e., are managed by one of their own employees),"* introduced in Section 1, is expressed by the following conjunctive query:

$$q_1 : d \leftarrow \mathsf{E}(\underline{m}, n, c_1, d), \mathsf{D}(\underline{d}, b, c_2, m).$$

When we treat the free variable d as a constant, we obtain $\mathsf{E}(\underline{m}, n, c_1, d)^{+,q_1} = \{m, b, c_2\}$ and $\mathsf{D}(\underline{d}, b, c_2, m)^{+,q_1} = \{\}$. Note that since d is treated like a constant, the D-atom gives rise to a functional dependency $\{\} \rightarrow \{b, c_2, m\}$, whose left-hand side is the empty set. The only attack is from the D-atom to the E-atom. Since the attack graph is acyclic, the certain answer to this query can be obtained by a single SQL query whose construction will be given in Section 5.3.

Next, consider the query *"Get names for employees who manage the department for which they work."*

$$q_2 : n \leftarrow \mathsf{E}(\underline{m}, n, c_1, d), \mathsf{D}(\underline{d}, b, c_2, m).$$

We have that $\mathsf{E}(\underline{m}, n, c_1, d)^{+,q_2} = \{m\}$ and $\mathsf{D}(\underline{d}, b, c_2, m)^{+,q_2} = \{d\}$. The E-atom attacks the D-atom because of the shared variable d, and the D-atom attacks the E-atom because of the shared variable m. The attack graph is cyclic, but contains no strong cycle. Therefore, the certain answer to this query can be computed in polynomial time but not in first-order logic or SQL.

Finally, consider the query *"Get names for employees who work in the city of their birth."*

$$q_3 : n \leftarrow \mathsf{E}(\underline{e}, n, c, d), \mathsf{D}(\underline{d}, b, c, m).$$

We have that $\mathsf{E}(\underline{e}, n, c, d)^{+,q_3} = \{e\}$ and $\mathsf{D}(\underline{d}, b, c, m)^{+,q_3} = \{d\}$. Both atoms attack each other because of the shared variable c. Moreover, the attack from the D-atom to the E-atom is strong. Since the attack graph contains a strong cycle, there exists no polynomial-time algorithm for computing the certain answer to this query (unless $\mathbf{P} = \mathbf{NP}$).

3.5. Auxiliary Lemmas and Constructs

The following lemmas are proved in Appendix A.

LEMMA 3.5. *For every query $q \in \mathsf{sjfBCQ}$, if $F \overset{q}{\rightsquigarrow} G$ and $G \overset{q}{\rightsquigarrow} H$, then either $F \overset{q}{\rightsquigarrow} H$ or $G \overset{q}{\rightsquigarrow} F$ (or both).*

LEMMA 3.6. *For every query $q \in \mathsf{sjfBCQ}$,*

(1) *if the attack graph of q contains a cycle, then it contains a cycle of size two; and*
(2) *if the attack graph of q contains a strong cycle, then it contains a strong cycle of size two.*

LEMMA 3.7. *Let $q \in \mathsf{sjfBCQ}$. Let $x \in \mathsf{vars}(q)$ and let a be an arbitrary constant.*

(1) *If the attack graph of q is acyclic, then the attack graph of $q_{[x \mapsto a]}$ is acyclic.*
(2) *If the attack graph of q contains no strong cycle, then the attack graph of $q_{[x \mapsto a]}$ contains no strong cycle.*

We conclude this section with three definitions. The following definition is taken from [5] and applies to directed graphs in general.

Definition 3.8 (*Strongly Connected Components*). A directed graph is *strongly connected* if there is a directed path from any vertex to any other. The maximal strongly connected subgraphs of a graph are vertex-disjoint and are called its *strong components*. If S_1 and S_2 are strong components such that an edge leads from a vertex in S_1 to a vertex in S_2, then S_1 is a *predecessor* of S_2 and S_2 is a *successor* of S_1. A strong component is called *initial* if it has no predecessor.

Strong components in the attack graph should not be confused with strong attacks.

Example 3.9. In the attack graph of Figure 2, the atoms $R(\underline{x}, y)$, $S(\underline{y}, z)$, and $T(\underline{z}, x)$ together constitute an initial strong component.

So far, we have defined an attack from an atom to another atom. The following definition introduces attacks from an atom to a variable.

Definition 3.10. Let $q \in \mathsf{sjfBCQ}$. Let R be a relation name with signature [1, 1] such that R does not occur in q. For $F \in q$ and $z \in \mathsf{vars}(q)$, we say that F *attacks* z, denoted $F \overset{q}{\rightsquigarrow} z$, if $F \overset{q'}{\rightsquigarrow} R(\underline{z})$ where $q' = q \cup \{R(\underline{z})\}$.

Example 3.11. Clearly, if $F_0 \overset{z_1}{\frown} F_1 \cdots \overset{z_n}{\frown} F_n$ is a witness for $F_0 \overset{q}{\rightsquigarrow} F_n$, then $F_0 \overset{q}{\rightsquigarrow} z_i$ for every $i \in \{1, \dots, n\}$. Note also that if $q = \{R(\underline{x}, y)\}$, then the attack graph of q contains no edge, yet R attacks y, denoted $R \overset{q}{\rightsquigarrow} y$.

Finally, we introduce the notion of *sequential proof*, which mimics an algorithm for testing logical implication for functional dependencies [1, Algorithm 8.2.7].

Definition 3.12. Let q be a self-join-free Boolean conjunctive query. Let $X \subseteq \mathsf{vars}(q)$ and $y \in \mathsf{vars}(q)$. A *sequential proof* of $\mathcal{K}(q) \models X \rightarrow y$ is a sequence $H_0, H_1, \dots, H_\ell$ of atoms of q such that

- $y \in X \cup \bigcup_{i=0}^{\ell} \mathsf{vars}(H_i)$; and
- for $i \in \{0, \dots, \ell\}$, $\mathsf{key}(H_i) \subseteq X \cup \bigcup_{j=0}^{i-1} \mathsf{vars}(H_j)$.

Note that if $y \in X$, then the empty sequence is a sequential proof of $\mathcal{K}(q) \models X \rightarrow y$.

4. FIRST-ORDER EXPRESSIBILITY

In this section, we prove the first item in the statement of Theorem 3.2 as well as the **L**-hard lower complexity bound stated in the second item. Taken together, this leads to the following characterization of first-order expressibility of CERTAINTY(q).

THEOREM 4.1. *For every $q \in \mathsf{sjfBCQ}$, the following are equivalent:*

(*1*) CERTAINTY(q) *is in* **FO***;*
(*2*) *the attack graph of q is acyclic.*

In [41], this theorem was proved under the assumption that queries are α-acyclic. In Section 4.1, we show the contrapositive of the implication $1 \Rightarrow 2$. In Section 4.2, we show the implication $2 \Rightarrow 1$.

4.1. Cyclic Attack Graph

Let $q_0 = \{R_0(\underline{x}, y), S_0(\underline{y}, x)\}$. In [42], it was shown that CERTAINTY(q_0) is not in **FO**. The following lemma shows a stronger result.

LEMMA 4.2. *Let $q_0 = \{R_0(\underline{x}, y), S_0(\underline{y}, x)\}$. Then,* CERTAINTY($q_0$) *is* **L***-hard.*

LEMMA 4.3. *For every $q \in \mathsf{sjfBCQ}$, if the attack graph of q is cyclic, then* CERTAINTY(q) *is* **L***-hard (and thus not in* **FO***).*

PROOF. Assume that the attack graph of q is cyclic. We show next that there exists a first-order many-one reduction from CERTAINTY(q_0) to CERTAINTY(q). The desired result then follows from Lemma 4.2.

By Lemma 3.6, we can assume two distinct atoms $F, G \in q$ such that $F \overset{q}{\rightsquigarrow} G \overset{q}{\rightsquigarrow} F$ is an attack cycle of size two. We will assume from here that the relation names in F and G are R and S, respectively.

For all constants a, b, we define the valuation Θ_b^a over $\mathsf{vars}(q)$ as follows. Let $\perp$ be a fixed constant not occurring elsewhere. For every variable $u \in \mathsf{vars}(q)$,

(1) if $u \in F^{+,q} \setminus G^{+,q}$, then $\Theta^a_b(u) = a$;
(2) if $u \in G^{+,q} \setminus F^{+,q}$, then $\Theta^a_b(u) = b$;
(3) if $u \in F^{+,q} \cap G^{+,q}$, then $\Theta^a_b(u) = \bot$;
(4) if $u \in \mathsf{vars}(q) \setminus (F^{+,q} \cup G^{+,q})$, then $\Theta^a_b(u) = \langle a, b\rangle$.

Sublemma 1. *For all constants a, b, a', b', if $H \in q \setminus \{F, G\}$, then $\{\Theta^a_b(H), \Theta^{a'}_{b'}(H)\}$ is consistent.*

Proof of Sublemma 1. Assume that for all $u \in \mathsf{key}(H)$, $\Theta^a_b(u) = \Theta^{a'}_{b'}(u)$. We distinguish four cases.

Case $a = a'$ and $b = b'$. Then, $\Theta^a_b(H) = \Theta^{a'}_{b'}(H)$.
Case $a = a'$ and $b \neq b'$. Then, $\mathsf{key}(H) \subseteq F^{+,q}$; thus, $\mathsf{vars}(H) \subseteq F^{+,q}$. Then, $\Theta^a_b(H) = \Theta^{a'}_{b'}(H)$.
Case $a \neq a'$ and $b = b'$. Then, $\mathsf{key}(H) \subseteq G^{+,q}$; thus, $\mathsf{vars}(H) \subseteq G^{+,q}$. Then, $\Theta^a_b(H) = \Theta^{a'}_{b'}(H)$.
Case $a \neq a'$ and $b \neq b'$. Then, $\mathsf{key}(H) \subseteq F^{+,q} \cap G^{+,q}$; thus, $\mathsf{vars}(H) \subseteq F^{+,q} \cap G^{+,q}$. Then, $\Theta^a_b(H) = \Theta^{a'}_{b'}(H)$.

This concludes the proof of Sublemma 1. $\lhd$

Sublemma 2. *For all constants a, b, a', b',*

(1) *$\Theta^a_b(F)$ and $\Theta^{a'}_{b'}(F)$ are key-equal if and only if $a = a'$.*
(2) *$\Theta^a_b(F) = \Theta^{a'}_{b'}(F)$ if and only if $a = a'$ and $b = b'$.*
(3) *$\Theta^a_b(G)$ and $\Theta^{a'}_{b'}(G)$ are key-equal if and only if $b = b'$.*
(4) *$\Theta^a_b(G) = \Theta^{a'}_{b'}(G)$ if and only if $a = a'$ and $b = b'$.*

Proof of Sublemma 2. $\boxed{1. \Rightarrow}$ Consequence of $\mathsf{key}(F) \nsubseteq G^{+,q}$ (because $G \overset{q}{\rightsquigarrow} F$). $\boxed{1. \Longleftarrow}$ Consequence of $\mathsf{key}(F) \subseteq F^{+,q}$. $\boxed{2. \Rightarrow}$ Consequence of $\mathsf{vars}(F) \nsubseteq F^{+,q}$ (because $F \overset{q}{\rightsquigarrow} G$). $\boxed{2. \Longleftarrow}$ Trivial.
The proof of the remaining items is analogous. $\lhd$

For every uncertain database **db** with R_0-facts and S_0-facts, we define $f(\mathbf{db})$ as the following uncertain database:

(1) for every $R_0(\underline{a}, b)$ in **db**, $f(\mathbf{db})$ contains $\Theta^a_b(q \setminus \{G\})$; and
(2) for every $S_0(\underline{b}, a)$ in **db**, $f(\mathbf{db})$ contains $\Theta^a_b(q \setminus \{F\})$.

It is easy to see that f is computable in **FO**.
In what follows, we assume that **db** is typed, as explained in Section 2. It will be understood that $a, a_1, a_2, \ldots$ belong to $\mathsf{type}(x)$, and that $b, b_1, b_2, \ldots$ belong to $\mathsf{type}(y)$.
Let us define $g(\mathbf{db})$ as follows:

$$g(\mathbf{db}) := f(\mathbf{db}) \setminus \left(\{\Theta^a_b(F) \mid R_0(\underline{a}, b) \in \mathbf{db}\} \cup \{\Theta^a_b(G) \mid S_0(\underline{b}, a) \in \mathbf{db}\}\right).$$

That is, $g(\mathbf{db})$ contains all facts of $f(\mathbf{db})$ that are neither R-facts nor S-facts.
By Sublemmas 1 and 2,

$$\mathsf{rset}(f(\mathbf{db})) = \{f(\mathbf{r}) \cup g(\mathbf{db}) \mid \mathbf{r} \in \mathsf{rset}(\mathbf{db})\}. \tag{3}$$

Let **db** be an arbitrary database with R_0-facts and S_0-facts. It suffices to show that the following are equivalent for every repair **r** of **db**:

(1) **r** satisfies q_0;
(2) $f(\mathbf{r}) \cup g(\mathbf{db})$ satisfies q.

$\boxed{1 \Rightarrow 2}$ Assume that $\mathbf{r}$ satisfies q_0. We can assume constants a, b such that $R_0(\underline{a}, b), S_0(\underline{b}, a) \in \mathbf{r}$. Then, $f(\mathbf{r})$ contains $\Theta_b^a(q \setminus \{G\}) \cup \Theta_b^a(q \setminus \{F\}) = \Theta_b^a(q)$. It follows that $f(\mathbf{r})$ satisfies q. $\boxed{2 \Rightarrow 1}$ Let θ be a substitution over $\mathsf{vars}(q)$ such that $\theta(q) \subseteq f(\mathbf{r}) \cup g(\mathbf{db})$. By our construction, we can assume some $R_0(\underline{a}, b) \in \mathbf{r}$ such that $\theta(F) \in \Theta_b^a(q \setminus \{G\})$. Likewise, we can assume some $S_0(\underline{b'}, a') \in \mathbf{r}$ such that $\theta(G) \in \Theta_{b'}^{a'}(q \setminus \{F\})$. It suffices to show that $a = a'$ and $b = b'$.

Before giving the proof, we provide some intuition. For every fact $A \in f(\mathbf{db})$, we can assume an atom in q, denoted H_A, such that $A = \Theta_b^a(H_A)$ for some constant $a \in \mathsf{type}(x)$ and some constant $b \in \mathsf{type}(y)$. Then, for all $z \in \mathsf{vars}(H_A)$, $\Theta_b^a(z) \in \{\bot, a, b, \langle a, b\rangle\}$. The constants in the latter set allow one to "trace back" A to some facts $R_0(\underline{a}, b)$ or $S_0(\underline{b}, a)$ in $\mathbf{db}$.

With this intuition in mind, it is easy to show that $b = b'$ (the proof of $a = a'$ is symmetrical). Since $F \overset{q}{\rightsquigarrow} G$, there exists a sequence $F_0, F_1, \dots, F_n$ of atoms of q such that

- $F_0 = F$ and $F_n = G$; and
- for all $i \in \{0, \dots, n-1\}$, we can assume some $u_i \in \mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1})$ such that $u_i \notin F^{+,q}$.

We show by induction on increasing i that for all $i \in \{0, \dots, n-1\}$, there exists constant a_i such that for all $w_i \in \mathsf{vars}(F_i)$, we have that $\theta(w_i) \in \{\bot, a_i, b, \langle a_i, b\rangle\}$.

Basis $i = 0$. Since $\theta(F) \in \Theta_b^a(q \setminus \{G\})$, for all $w_0 \in \mathsf{vars}(F_0)$, we have that $\theta(w_0) \in \{\bot, a, b, \langle a, b\rangle\}$.

Step $i \to i+1$. By the induction hypothesis, there exists constant a_i such that for all $w_i \in \mathsf{vars}(F_i)$, we have that $\theta(w_i) \in \{\bot, a_i, b, \langle a_i, b\rangle\}$. From $u_i \notin F^{+,q}$, it follows that $\theta(u_i) \in \{b, \langle a_i, b\rangle\}$. Since $u_i \in \mathsf{vars}(F_{i+1})$, it follows that there exists constant a_{i+1} such that for all $w_{i+1} \in \mathsf{vars}(F_{i+1})$, we have that $\theta(w_{i+1}) \in \{\bot, a_{i+1}, b, \langle a_{i+1}, b\rangle\}$.

It follows that for $u_{n-1} \in \mathsf{vars}(G)$, there exists constant a_{n-1} such that $\theta(u_{n-1}) \in \{b, \langle a_{n-1}, b\rangle\}$. From $\theta(G) \in \Theta_{b'}^{a'}(q \setminus \{F\})$, it follows that $\theta(u_{n-1}) \in \{b', \langle a', b'\rangle\}$. Consequently, $b = b'$. This concludes the proof of Lemma 4.3. □

4.2. Acyclic Attack Graph

In this section, we show that $\mathsf{CERTAINTY}(q)$ is in $\mathbf{FO}$ if the attack graph of q is acyclic.

LEMMA 4.4. *Let q be a* sjfBCQ *query. Let F be an atom of q such that in the attack graph of q, the indegree of F is zero. Let $k = |\mathsf{key}(F)|$ and let $\vec{x} = \langle x_1, \dots, x_k\rangle$ be a sequence containing (exactly once) each variable of* $\mathsf{key}(F)$. *Then, the following are equivalent for every uncertain database* $\mathbf{db}$:

(*1*) *q is true in every repair of* $\mathbf{db}$;
(*2*) *for some $\vec{a} \in (\mathsf{adom}(\mathbf{db}))^k$, it is the case that $q_{[\vec{x} \mapsto \vec{a}]}$ is true in every repair of* $\mathbf{db}$.

Lemma 4.4 immediately leads to the following result.

LEMMA 4.5. *For every query $q \in$* sjfBCQ, *if the attack graph of q is acyclic, then* $\mathsf{CERTAINTY}(q)$ *is in* $\mathbf{FO}$.

PROOF. Assume that the attack graph of q is acyclic. The proof runs by induction on $|q|$. If $|q| = 0$, then $\mathsf{CERTAINTY}(q)$ is obviously in $\mathbf{FO}$. Assume next that $|q| > 0$.

Let $\mathbf{db}$ be an uncertain database that is input to $\mathsf{CERTAINTY}(q)$. Since the attack graph of q is acyclic, we can assume an atom $R(\underline{\vec{x}}, \vec{y})$ that is not attacked in the attack graph of q. By Lemma 4.4, the following are equivalent:

(1) q is true in every repair of **db**.
(2) For some fact $R(\underline{\vec{a}}, \vec{b}) \in \mathbf{db}$, there exists a valuation θ over $\mathsf{vars}(\vec{x})$ such that $\theta(\vec{x}) = \vec{a}$ and such that for all key-equal facts $R(\underline{\vec{a}}, \vec{b}')$ in **db**, the valuation θ can be extended to a valuation θ^+ over $\mathsf{vars}(\vec{x}) \cup \mathsf{vars}(\vec{y})$ such that $\theta^+(\vec{y}) = \vec{b}'$ and $\theta^+(q')$ is true in every repair of **db**, where $q' = q \setminus \{R(\underline{\vec{x}}, \vec{y})\}$.

From Lemma 3.7, it follows that the attack graph of $\theta^+(q')$ is acyclic; thus, CERTAINTY($\theta^+(q')$) is in **FO** by the induction hypothesis. It is then clear that the latter condition 2 can be checked in **FO**. □

5. CONSISTENT FIRST-ORDER REWRITING IN SQL

We explain in this section how to solve CERTAINTY(q) in SQL when q is a sjfBCQ query such that CERTAINTY(q) is in **FO**. Section 5.1 defines the construct of consistent first-order rewriting. Section 5.2 shows how to construct such a rewriting in SQL. The overall construction is summarized and illustrated in Section 5.3.

5.1. Consistent First-Order Rewriting: Definition

A *consistent first-order rewriting* for a Boolean conjunctive query q is a Boolean first-order query φ such that, for every uncertain database **db**, the following equivalence holds: every repair of **db** satisfies q if and only if **db** satisfies φ. If $q \in$ sjfBCQ such that CERTAINTY(q) is in **FO**, then there exists a consistent first-order rewriting for q (by definition of the complexity class **FO**). Such a first-order rewriting is actually an implementation, in first-order logic, of the algorithm in the proof of Lemma 4.5. This is illustrated next.

Example 5.1. Let $q = \{R(\underline{x}, y), S(\underline{y}, b)\}$, where b is a constant. The attack graph of q contains a single directed edge, from the R-atom to the S-atom. The following first-order sentence is a consistent first-order rewriting for q:

$$\exists x \exists y (R(\underline{x}, y) \wedge \forall y (R(\underline{x}, y) \rightarrow (S(\underline{y}, b) \wedge \forall z (S(\underline{y}, z) \rightarrow z = b)))).$$

This definition is naturally generalized to non-Boolean queries, as follows. A *consistent first-order rewriting* for a conjunctive query $\vec{f} \leftarrow q$ is a first-order query $\varphi(\vec{f})$ with free variables $\vec{f}$ such that for every sequence $\vec{a}$ of constants (of the same length as $\vec{f}$), for every uncertain database **db**, the following are equivalent:

(1) the certain answer to $\vec{f} \leftarrow q$ contains $\vec{a}$; and
(2) $\mathbf{db} \models \varphi(\vec{a})$.

Our theoretical development in Section 4 is focused on queries without free variables. Nevertheless, if conjunctive queries are self-join-free, then the presence of free variables causes no difficulty. According to the argumentation in Section 3.3, in the construction of a consistent first-order rewriting for $\vec{f} \leftarrow q$, free variables can be treated like constants.

5.2. Consistent First-Order Rewriting: Construction

We now put the theoretical results of Section 4 into practice. We explain how to construct a consistent first-order rewriting in a subset of tuple relational calculus (TRC) [39, page 157]. The subset is chosen such that the translation from TRC queries into SQL will be straightforward.

In TRC, variables represent tuples, which is different from first-order logic, in which variables represent atomic values. For our purpose, it is sufficient to have tuple variables that range over the tuples of a single relation in the same way as aliases in SQL range over the rows of one table. We use TRC rather than pure SQL because TRC allows universal quantifiers ($\forall$) and implication ($\rightarrow$), which leads to better readability.

It is straightforward to translate conjunctive queries into TRC. For example, the conjunctive query

$$\langle x, z\rangle \leftarrow \{R_1(\underline{x}, y), R_2(\underline{x, y}, z, a)\},$$

where a is a constant, reads as follows in TRC:

$$\left\{ f^{(2)} \mid \exists r_1 \in R_1 \exists r_2 \in R_2 \begin{pmatrix} r_1[1] = f[1] \wedge r_1[1] = r_2[1] \wedge r_1[2] = r_2[2] \wedge \\ r_2[3] = f[2] \wedge r_2[4] = a \end{pmatrix} \right\}.$$

The notation $f^{(2)}$ indicates that f is a free tuple variable that represents a tuple of arity 2. The notation $r_1 \in R_1$ introduces a tuple variable r_1 that ranges over the tuples of the relation R_1. The notation $r_1[1]$ refers to the first argument of r_1 and so on. The previous query equates $f[2]$ to $r_2[3]$ and equates $f[1]$ to both $r_1[1]$ and $r_2[1]$ (by transitivity and symmetry of equality).

Formally, a *term* is either a constant or a component reference of the form $t[i]$, where t is a tuple variable of arity n and $1 \leq i \leq n$. A self-join-free conjunctive query in TRC has the form

$$\{f^{(m)} \mid \exists r_1 \in R_1 \cdots \exists r_\ell \in R_\ell\,(\varphi)\}, \tag{4}$$

where

(1) f is the only free tuple variable, whose arity is m;
(2) $R_1, \ldots, R_\ell$ are distinct relation names;
(3) for $i \in \{1, \ldots, \ell\}$, r_i is a tuple variable of the same arity as the relation name R_i; and
(4) φ is a satisfiable conjunction of equalities between terms, such that, for every constant a and $j \in \{1, \ldots, m\}$, $\varphi \not\models (f[j] = a)$ and for all $j_1, j_2 \in \{1, \ldots, m\}$ such that $j_1 \neq j_2$, $\varphi \not\models (f[j_1] = f[j_2])$. That is, φ equates no component of the free tuple f to a constant or to another component. Further, to guarantee safety, for every $j \in \{1, \ldots, m\}$, φ must equate $f[j]$ to some component of some r_i ($1 \leq i \leq \ell$).

Now, assume that the attack graph of the self-join-free conjunctive query (4) is acyclic and that the relation names occur in a topological ordering of the attack graph. That is, the R_1-atom precedes the R_2-atom in the topological ordering, the R_2-atom precedes the R_3-atom, and so on. For every $i \in \{1, \ldots, \ell\}$, we assume a tuple variable s_i with the same arity as R_i. Further, for $i \in \{1, \ldots, \ell\}$, we write $s_i \sim r_i$ as shorthand for $\bigwedge_{j=1}^{k} s_i[j] = r_i[j]$, where k is the cardinality of the primary key of R_i. That is, $s_i \sim r_i$ expresses that s_i and r_i are key-equal. Then, the following is a consistent first-order rewriting, in prenex normal form, for the query (4):

$$\left\{ f^{(m)} \mid \exists s_1 \in R_1 \forall r_1 \in R_1 \cdots \exists s_\ell \in R_\ell \forall r_\ell \in R_\ell \left(\left(\bigwedge_{i=1}^{\ell} s_i \sim r_i \right) \rightarrow \varphi \right) \right\}. \tag{5}$$

Intuitively, each $\exists s_i$ chooses an R_i-block, and the successive $\forall r_i$ can be assumed to range only over the tuples of that block (because of the condition $s_i \sim r_i$ in the premise). The conclusion φ, which is copied from (4), does not refer to any s_i.

Example 5.2. The conjunctive query $y \leftarrow \{R_1(\underline{x}, y)\}$ is equivalent to the TRC query

$$\{f^{(1)} \mid \exists r_1 \in R_1\,(r_1[2] = f[1])\},$$

whose consistent first-order rewriting according to (5) is:

$$\{f^{(1)} \mid \exists s_1 \in R_1 \forall r_1 \in R_1\,(s_1[1] = r_1[1] \rightarrow r_1[2] = f[1])\}.$$

If we eliminate $\forall$ and $\rightarrow$ from the latter query, we obtain:

$$\{f^{(1)} \mid \exists s_1 \in R_1 \neg \exists r_1 \in R_1 (s_1[1] = r_1[1] \wedge r_1[2] \neq f[1])\}.$$

The translation of TRC queries of the form (5) into SQL is fairly straightforward. As illustrated in Example 5.2, the query can be easily rewritten to eliminate the usage of universal quantifiers ($\forall$) and implications ($\rightarrow$), which are not directly available in SQL. Then, the only significant difference between our TRC and SQL is the free tuple variable $f^{(m)}$, which has no direct counterpart in SQL. In general, this difference can be overcome by translating (5) into an SQL query of the following form:

```
SELECT  DISTINCT $t_{i_1}.A_1, \ldots, t_{i_m}.A_m$
FROM    $R_1$ AS $t_1, \ldots, R_\ell$ AS $t_\ell$                    (6)
WHERE   $\psi$ ;
```

For every $j \in \{1, \ldots, m\}$, the component reference $f[j]$ is translated into $t_{i_j}.A_j$, where i_j and A_j are picked such that $\varphi \models (f[j] = r_{i_j}[A_j])$. That is, each A_j is a column position in the table R_{i_j}, and the domain of interpretation for the jth component of f can be safely restricted to the values in that column. The expression (6) is not a genuine SQL query, because it uses column positions in the SELECT clause. In the final SQL query, these positions are to be replaced by their corresponding column names. As each of the SQL aliases t_i ranges over the tuples of its table R_i, all instantiations for the free tuple f will be considered and the ones that make ψ true are selected. Finally, the formula ψ is a straightforward translation into SQL of the TRC formula in (5) (i.e., of the formula starting with $\exists s_1 \in R_1$), where each component reference $f[j]$ is replaced by its translation $t_{i_j}.A_j$.

Example 5.3. For the query in Example 5.2, assume the relation schema $R_1[A, B]$. The only possible translation for $f[1]$ is $t_1.B$. The translation in SQL is as follows:

```
SELECT DISTINCT t1.B
FROM    R1 AS t1
WHERE EXISTS ( SELECT *
               FROM    R1 AS s1
               WHERE NOT EXISTS ( SELECT *
                                  FROM    R1 AS r1
                                  WHERE   s1.A = r1.A
                                  AND     r1.B ≠ t1.B ) );
```

The subquery that occurs after WHERE EXISTS is a straightforward translation of the TRC formula in the last query in Example 5.2. The preceding SQL query can be simplified as follows.

```
SELECT DISTINCT s1.B
FROM    R1 AS s1
WHERE NOT EXISTS ( SELECT *
                   FROM    R1 AS r1
                   WHERE   s1.A = r1.A
                   AND     r1.B ≠ s1.B );
```

Simplification of consistent first-order rewriting in SQL has been studied in [12].

5.3. Elaborated Example

In four steps, we build a consistent first-order rewriting in SQL for the query *"Get names for departments that are self-managed (i.e., are managed by one of their own* *employees).”* This query was introduced in Section 1 and is expressed by the following conjunctive query:

$$q_1 : d \leftarrow \mathsf{E}(\underline{m}, n, c_1, d), \mathsf{D}(\underline{d}, b, c_2, m).$$

As argued in Section 3.4, the only attack is from the D-atom to the E-atom. The first step is to write this query in the form (4):

$$\left\{ f^{(1)} \mid \exists r_1 \in \mathsf{D} \exists r_2 \in \mathsf{E}\,(r_1[1] = f[1] \wedge r_1[1] = r_2[4] \wedge r_1[4] = r_2[1]) \right\}.$$

Significantly, $\exists r_1 \in \mathsf{D}$ precedes $\exists r_2 \in \mathsf{E}$ because the D-atom attacks the E-atom. In the second step, we construct a consistent first-order rewriting in the form (5), where we use that the primary key of both D and E consists of the first position:

$$\left\{ f^{(1)} \mid \exists s_1 \in \mathsf{D} \forall r_1 \in \mathsf{D} \exists s_2 \in \mathsf{E} \forall r_2 \in \mathsf{E} \left(\left(\begin{array}{c} s_1[1] = r_1[1] \\ \wedge\ s_2[1] = r_2[1] \end{array} \right) \rightarrow \left(\begin{array}{c} r_1[1] = f[1] \\ \wedge\ r_1[1] = r_2[4] \\ \wedge\ r_1[4] = r_2[1] \end{array} \right) \right) \right\}.$$

In the third step, we equivalently rewrite this query without universal quantification and implication:

$$\left\{ f^{(1)} \mid \exists s_1 \in \mathsf{D} \neg \exists r_1 \in \mathsf{D} \neg \exists s_2 \in \mathsf{E} \neg \exists r_2 \in \mathsf{E} \left(\left(\begin{array}{c} s_1[1] = r_1[1] \\ \wedge\ s_2[1] = r_2[1] \end{array} \right) \wedge \left(\begin{array}{c} r_1[1] \neq f[1] \\ \vee\ r_1[1] \neq r_2[4] \\ \vee\ r_1[4] \neq r_2[1] \end{array} \right) \right) \right\}.$$

Note that we have chosen to push negation inward using De Morgan’s laws. In the fourth step, we translate the preceding query in an SQL query of the form (6), where attribute positions are replaced with their names, and $f[1]$ is translated by t_1.DNAME.

```
SELECT DISTINCT t1.DNAME FROM D AS t1, E AS t2
WHERE EXISTS (
          SELECT * FROM D AS s1
          WHERE NOT EXISTS (
                    SELECT * FROM D AS r1
                    WHERE NOT EXISTS (
                              SELECT * FROM E AS s2
                              WHERE NOT EXISTS (
                                        SELECT *  FROM E AS r2
                                        WHERE s1.DNAME = r1.DNAME
                                        AND      s2.EID = r2.EID
                                        AND     (       r1.DNAME ≠ t1.DNAME
                                                  OR r1.DNAME ≠ r2.DNAME
                                                  OR r1.MGR ≠ r2.EID          ) ) ) ) );
```

Eventually, it is possible to simplify this SQL query, as in Example 5.3.

6. INTRACTABILITY RESULT

In this section, we prove the **coNP**-hard lower complexity bound stated in the third item of Theorem 3.2.

THEOREM 6.1. *For every $q \in$ sjfBCQ, if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP**-*hard.*

PROOF. Let $q \in$ sjfBCQ. Assume that the attack graph of q contains a strong cycle. By Lemma 3.6, we can assume some $F, G \in q$ such that $F \overset{q}{\rightsquigarrow} G \overset{q}{\rightsquigarrow} F$ and the attack $F \overset{q}{\rightsquigarrow} G$ is strong. We will assume from here that the relation names in F and G are R and S, respectively.

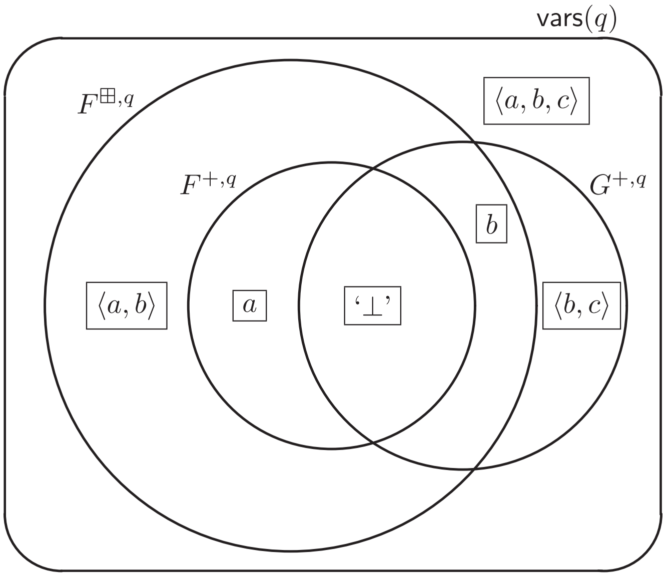


Fig. 3. Help for the proof of Theorem 6.1.

Let $q_1 = \{R_1(\underline{x}, y), S_1(\underline{y, z}, x)\}$. We show next that there exists a polynomial-time (and even first-order) many-one reduction from CERTAINTY(q_1) to CERTAINTY(q). Since it is known [21] that CERTAINTY(q_1) is **coNP**-hard, it follows that CERTAINTY(q) is **coNP**-hard.

For all constants a, b, c, we define $\Theta^a_{b,c}$ as the following valuation over vars(q) (see Figure 3 for a mnemonic). Let $\perp$ be some fixed constant.

(1) If $u \in F^{+,q} \cap G^{+,q}$, then $\Theta^a_{b,c}(u) = \perp$;
(2) if $u \in F^{+,q} \setminus G^{+,q}$, then $\Theta^a_{b,c}(u) = a$;
(3) if $u \in G^{+,q} \setminus F^{\boxplus,q}$, then $\Theta^a_{b,c}(u) = \langle b, c\rangle$;
(4) if $u \in \left(G^{+,q} \cap F^{\boxplus,q}\right) \setminus F^{+,q}$, then $\Theta^a_{b,c}(u) = b$;
(5) if $u \in F^{\boxplus,q} \setminus (F^{+,q} \cup G^{+,q})$, then $\Theta^a_{b,c}(u) = \langle a, b\rangle$; and
(6) if $u \notin F^{\boxplus,q} \cup G^{+,q}$, then $\Theta^a_{b,c}(u) = \langle a, b, c\rangle$.

SUBLEMMA 3. *For all constants a, b, c, a', b', c', if $H \in q \setminus \{F, G\}$, then $\{\Theta^a_{b,c}(H), \Theta^{a'}_{b',c'}(H)\}$ is consistent.*

PROOF OF SUBLEMMA 3. Assume that for all $u \in$ key(H),

$$\Theta^a_{b,c}(u) = \Theta^{a'}_{b',c'}(u). \tag{7}$$

We distinguish four cases.

Case $a = a'$ and $b = b'$. If $c = c'$, then $\Theta^a_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$. Assume next that $c \neq c'$. From Equation (7), it follows that key(H) $\subseteq F^{\boxplus,q}$. Consequently, vars(H) $\subseteq F^{\boxplus,q}$. Since c does not occur inside $F^{\boxplus,q}$ in the Venn diagram of Figure 3, we have that $\Theta^a_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$.

Case $a = a'$ and $b \neq b'$. From Equation (7), it follows that key(H) $\subseteq F^{+,q}$; thus, vars(H) $\subseteq F^{+,q}$. Since b and c do not occur inside $F^{+,q}$ in the Venn diagram, $\Theta^a_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$.

Case $a \neq a'$ and $b = b'$. First, assume that $c = c'$. From Equation (7), it follows that $\mathsf{key}(H) \subseteq G^{+,q}$; thus, $\mathsf{vars}(H) \subseteq G^{+,q}$. Since a does not occur inside $G^{+,q}$ in the Venn diagram, $\Theta^{a}_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$.

Next, assume that $c \neq c'$. From Equation (7), it follows that $\mathsf{key}(H) \subseteq F^{\boxplus,q} \cap G^{+,q}$; thus, $\mathsf{vars}(H) \subseteq F^{\boxplus,q} \cap G^{+,q}$. Since a and c do not occur inside $F^{\boxplus,q} \cap G^{+,q}$ in the Venn diagram, $\Theta^{a}_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$.

Case $a \neq a'$ and $b \neq b'$. From Equation (7), it follows that $\mathsf{key}(H) \subseteq F^{+,q} \cap G^{+,q}$; thus, $\mathsf{vars}(H) \subseteq F^{+,q} \cap G^{+,q}$. Since a, b, c do not occur inside $F^{+,q} \cap G^{+,q}$ in the Venn diagram, $\Theta^{a}_{b,c}(H) = \Theta^{a'}_{b',c'}(H)$.

This concludes the proof of Sublemma 3. $\triangleleft$

SUBLEMMA 4. *For all constants a, b, c, a', b', c',*

(1) $\Theta^{a}_{b,c}(F)$ *and* $\Theta^{a'}_{b',c'}(F)$ *are key-equal if and only if* $a = a'$.
(2) $\Theta^{a}_{b,c}(F) = \Theta^{a'}_{b',c'}(F)$ *if and only if* $a = a'$ *and* $b = b'$.
(3) $\Theta^{a}_{b,c}(G)$ *and* $\Theta^{a'}_{b',c'}(G)$ *are key-equal if and only if* $b = b'$ *and* $c = c'$.
(4) $\Theta^{a}_{b,c}(G) = \Theta^{a'}_{b',c'}(G)$ *if and only if* $a = a'$ *and* $b = b'$ *and* $c = c'$.

PROOF OF SUBLEMMA 4. $\boxed{1. \Rightarrow}$ Consequence of $\mathsf{key}(F) \not\subseteq G^{+,q}$ (because $G \overset{q}{\rightsquigarrow} F$). $\boxed{1. \Longleftarrow}$ Consequence of $\mathsf{key}(F) \subseteq F^{+,q}$. $\boxed{2. \Rightarrow}$ Consequence of $\mathsf{vars}(F) \not\subseteq F^{+,q}$ (because $F \overset{q}{\rightsquigarrow} G$). $\boxed{2. \Longleftarrow}$ Consequence of $\mathsf{vars}(F) \subseteq F^{\boxplus,q}$. $\boxed{3. \Rightarrow}$ Consequence of $\mathsf{key}(G) \not\subseteq F^{\boxplus,q}$ (because $F \overset{q}{\rightsquigarrow} G$ is a strong attack). $\boxed{3. \Longleftarrow}$ Consequence of $\mathsf{key}(G) \subseteq G^{+,q}$. $\boxed{4. \Rightarrow}$ Consequence of item 3 and $\mathsf{vars}(G) \not\subseteq G^{+,q}$ (because $G \overset{q}{\rightsquigarrow} F$). $\boxed{4. \Longleftarrow}$ Trivial. $\triangleleft$

Let $\mathbf{db}$ be an uncertain database with R_1-facts and S_1-facts. In what follows, we assume that $\mathbf{db}$ is typed, as explained in Section 2. It will be understood that $a, a_1, a_2, \ldots$ belong to $\mathsf{type}(x)$, that $b, b_1, b_2, \ldots$ belong to $\mathsf{type}(y)$, and that $c, c_1, c_2, \ldots$ belong to $\mathsf{type}(z)$.

We define $f(\mathbf{db})$ as the following uncertain database:

(1) for every $R_1(\underline{a}, b)$ in $\mathbf{db}$, $f(\mathbf{db})$ contains $\Theta^{a}_{b,c}(F)$ for some arbitrary constant c; note that the choice of c does not matter because of the second item in Sublemma 4;
(2) for every $S_1(\underline{b, c}, a)$ in $\mathbf{db}$, $f(\mathbf{db})$ contains $\Theta^{a}_{b,c}(q \setminus \{F\})$.

It is easy to see that f is computable in **FO**.

Let $g(\mathbf{db})$ be the subset of $f(\mathbf{db})$ containing all facts of $f(\mathbf{db})$ that are neither R-facts nor S-facts. By Sublemmas 3 and 4,

$$\mathsf{rset}(f(\mathbf{db})) = \{f(\mathbf{r}) \cup g(\mathbf{db}) \mid \mathbf{r} \in \mathsf{rset}(\mathbf{db})\}. \tag{8}$$

Let $\mathbf{db}$ be an arbitrary database with R_1-facts and S_1-facts. It suffices to show that the following are equivalent for every repair $\mathbf{r}$ of $\mathbf{db}$:

(1) $\mathbf{r}$ satisfies q_1;
(2) $f(\mathbf{r}) \cup g(\mathbf{db})$ satisfies q.

$\boxed{1 \Rightarrow 2}$ Assume that $\mathbf{r}$ satisfies q_1. Then, we can assume constants a, b, c such that $R_1(\underline{a}, b), S_1(\underline{b, c}, a) \in \mathbf{r}$. Then, $f(\mathbf{r})$ contains $\{\Theta^{a}_{b,c'}(F)\} \cup \Theta^{a}_{b,c}(q \setminus \{F\})$ for some constant c'. Since $\Theta^{a}_{b,c'}(F) = \Theta^{a}_{b,c}(F)$, we have that $f(\mathbf{r})$ contains $\Theta^{a}_{b,c}(q)$. Consequently, $f(\mathbf{r})$ satisfies q. $\boxed{2 \Rightarrow 1}$ Let θ be a substitution over $\mathsf{vars}(q)$ such that $\theta(q) \subseteq f(\mathbf{r}) \cup g(\mathbf{db})$. By our construction, we can assume some $R_1(\underline{a}, b) \in \mathbf{r}$ and some constant c such that $\theta(F) = \Theta^{a}_{b,c}(F)$. Likewise, we can assume some $S_1(\underline{b', c'}, a') \in \mathbf{r}$ such that $\theta(G) \in \Theta^{a'}_{b',c'}(q \setminus \{F\})$. It suffices to show that $a = a'$ and $b = b'$.

$\boxed{b = b'}$ Since $F \overset{q}{\rightsquigarrow} G$, there exists a sequence $F_0, F_1, \ldots, F_n$ of distinct atoms of q such that

- $F_0 = F$ and $F_n = G$; and
- for all $i \in \{0, \ldots, n-1\}$, we can assume some $u_i \in \mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1})$ such that $u_i \notin F^{+,q}$.

We show by induction on increasing i that for all $i \in \{0, \ldots, n-1\}$, there exist constants a_i and c_i such that for all $w_i \in \mathsf{vars}(F_i)$, we have that $\theta(w_i) \in \{\perp, a_i, b, \langle a_i, b\rangle, \langle b, c_i\rangle, \langle a_i, b, c_i\rangle\}$.

Basis $i = 0$. Since $\theta(F) = \Theta^{a}_{b,c}(F)$, for all $w_0 \in \mathsf{vars}(F_0)$, we have that $\theta(w_0) \in \{\perp, a, b, \langle a, b\rangle, \langle b, c\rangle, \langle a, b, c\rangle\}$.

Step $i \to i+1$. By the induction hypothesis, there exist constants a_i and c_i such that for all $w_i \in \mathsf{vars}(F_i)$, we have that $\theta(w_i) \in \{\perp, a_i, b, \langle a_i, b\rangle, \langle b, c_i\rangle, \langle a_i, b, c_i\rangle\}$. From $u_i \notin F^{+,q}$, it follows that $\theta(u_i) \in \{b, \langle a_i, b\rangle, \langle b, c_i\rangle, \langle a_i, b, c_i\rangle\}$. Since $u_i \in \mathsf{vars}(F_{i+1})$, it follows that there exist constants a_{i+1} and c_{i+1} such that for all $w_{i+1} \in \mathsf{vars}(F_{i+1})$, we have that $\theta(w_{i+1}) \in \{\perp, a_{i+1}, b, \langle a_{i+1}, b\rangle, \langle b, c_{i+1}\rangle, \langle a_{i+1}, b, c_{i+1}\rangle\}$.

It follows that for $u_{n-1} \in \mathsf{vars}(G)$, there exist constants a_{n-1} and c_{n-1} such that $\theta(u_{n-1}) \in \{b, \langle a_{n-1}, b\rangle, \langle b, c_{n-1}\rangle, \langle a_{n-1}, b, c_{n-1}\rangle\}$. From $\theta(G) \in \Theta^{a'}_{b',c'}(q \setminus \{F\})$, it follows that $\theta(u_{n-1}) \in \{b', \langle a', b'\rangle, \langle b', c'\rangle, \langle a', b', c'\rangle\}$. Consequently, $b = b'$.

$\boxed{a = a'}$ Analogous. This concludes the proof of Theorem 6.1. □

7. POLYNOMIAL-TIME TRACTABILITY

In this section, we prove the **P** upper complexity bound stated in the second item of Theorem 3.2.

THEOREM 7.1. *For every $q \in$ sjfBCQ, if the attack graph of q contains no strong cycle, then* CERTAINTY(q) *is in* **P**.

A very high-level outline of the proof of Theorem 7.1 is as follows. Let q be a sjfBCQ query such that the attack graph of q contains no strong cycle. The proof will run by structural induction. If the attack graph of q contains an atom F without incoming attacks, then Lemma 4.4 tells us that the answer to CERTAINTY(q) can be obtained in polynomial time from the answers to a polynomial number of problems CERTAINTY(q'), all of which are in **P** by the induction hypothesis. The more difficult case is if all atoms have incoming attacks in the attack graph of q. We will show that, in this case, CERTAINTY(q) can be reduced in polynomial time to some problem CERTAINTY(q''), which is in **P** by the induction hypothesis. The query q'' is obtained from q by a technique called *"dissolution of Markov cycles."*

The proof of Theorem 7.1 is technically involved. We start by introducing in Section 7.1 an extension of the data model that allows some syntactic simplifications, expressed in Section 7.2. In Section 7.3, we introduce the notion of *Markov cycle*, and show how the dissolution of Markov cycles is helpful in the proof of Theorem 7.1, which is given in Section 7.4. The dissolution of Markov cycles will be explained in detail in Section 8.

7.1. Relations Known to Be Consistent

We conservatively extend our data model. We first distinguish between two kinds of relation names: those that can be inconsistent and those that cannot.

The mode of a relation name. Every relation name has a unique and fixed *mode*, which is an element in $\{i, c\}$. It will come in handy to think of i and c as inconsistent and consistent respectively. We often write R^c to denote that R is a relation name with mode c. If $q \in \mathsf{sjfBCQ}$, then $[\![q]\!]$ denotes the subset of q containing each atom whose relation name has mode c. The *inconsistency count* of q, denoted $\mathsf{incnt}(q)$, is the number of relation names with mode i in q. Modes carry over to atoms and facts: the mode of an atom $R(\underline{\vec{x}}, \vec{y})$ or a fact $R(\underline{\vec{a}}, \vec{b})$ is the mode of R.

The intended semantics is that if a relation name R has mode c, then the set of R-facts of an uncertain database will always be consistent.

Certain query answering with consistent and inconsistent relations. The problem $\mathsf{CERTAINTY}(q)$ now takes as input an uncertain database **db** such that for every relation name R in q, if R has mode c, then the set of R-facts of **db** is consistent. The problem is to determine whether every repair of **db** satisfies q.

All constructs and results shown in previous sections assumed that all relation names had mode i. Nevertheless, Proposition 7.2 indicates that in the tractability study of $\mathsf{CERTAINTY}(q)$, relation names with mode c can be simulated by means exclusively of relation names with mode i. Therefore, having relation names with mode c will be convenient, but is not fundamental.

PROPOSITION 7.2. *Let q be a self-join-free Boolean conjunctive query. Let $R^c(\underline{\vec{x}}, \vec{y})$ be an atom with mode c in q. Let R_1 and R_2 be two relation names, both with mode i and with the same signature as R, such that neither R_1 nor R_2 occurs in q. Let $q' = (q \setminus \{R^c(\underline{\vec{x}}, \vec{y})\}) \cup \{R_1(\underline{\vec{x}}, \vec{y}), R_2(\underline{\vec{x}}, \vec{y})\}$. Then,* $\mathsf{CERTAINTY}(q)$ *and* $\mathsf{CERTAINTY}(q')$ *are equivalent under first-order reductions.*

If relation names with mode c are allowed for syntactic convenience, the definition of $F^{+,q}$ needs a slight change:

$$F^{+,q} := \{x \in \mathsf{vars}(q) \mid \mathcal{K}((q \setminus F) \cup [\![q]\!]) \models \mathsf{key}(F) \rightarrow x\}.$$

Modulo this redefinition, the notion of attack graph remains unchanged.

Proposition 7.2 explains how to replace atoms with mode c. Conversely, the following lemma states that in pursuing a proof for Theorem 7.1, there are cases in which a sjfBCQ query can be extended with atoms of mode c.

LEMMA 7.3. *Let $q \in \mathsf{sjfBCQ}$. Let $x, z \in \mathsf{vars}(q)$ such that $\mathcal{K}(q) \models x \rightarrow z$ and for every $F \in q$, if $\mathcal{K}(q) \models x \rightarrow \mathsf{key}(F)$, then $F \overset{q}{\not\rightsquigarrow} x$ and $F \overset{q}{\not\rightsquigarrow} z$. Let $q' = q \cup \{T^c(\underline{x}, z)\}$, where T is a fresh relation name with mode c. Then,*

(1) *there exists a polynomial-time many-one reduction from* $\mathsf{CERTAINTY}(q)$ *to* $\mathsf{CERTAINTY}(q')$*; and*
(2) *if the attack graph of q contains no strong cycle, then the attack graph of q' contains no strong cycle either.*

Saturated queries. Given a query in sjfBCQ, the reduction of Lemma 7.3 can be repeated until it can no longer be applied. The query so obtained will be called *saturated*.

Definition 7.4. We say that $q \in \mathsf{sjfBCQ}$ is *saturated* if whenever $x, z \in \mathsf{vars}(q)$ such that $\mathcal{K}(q) \models x \rightarrow z$ and $\mathcal{K}([\![q]\!]) \not\models x \rightarrow z$, then there exists an atom $F \in q$ with $\mathcal{K}(q) \models x \rightarrow \mathsf{key}(F)$ such that $F \overset{q}{\rightsquigarrow} x$ or $F \overset{q}{\rightsquigarrow} z$.

Example 7.5. Consider the query $q = \{R(\underline{x}, y), S_1(\underline{y}, z), S_2(\underline{y}, z), T^c(\underline{x, z}, w), U(\underline{w}, x)\}$. We have that $\mathcal{K}(q) \models y \rightarrow z$ and $\mathcal{K}(\llbracket q \rrbracket) \not\models y \rightarrow z$. The set $\{F \in q \mid \mathcal{K}(q) \models y \rightarrow \mathsf{key}(F)\}$ equals $\{S_1, S_2\}$. We have neither $S_1 \overset{q}{\rightsquigarrow} y$ nor $S_1 \overset{q}{\rightsquigarrow} z$. Likewise, neither $S_2 \overset{q}{\rightsquigarrow} y$ nor $S_2 \overset{q}{\rightsquigarrow} z$. Thus, q is not saturated. By Lemma 7.3, there exists a polynomial-time many-one reduction from $\mathsf{CERTAINTY}(q)$ to $\mathsf{CERTAINTY}(q')$ with $q' = q \cup \{S^c(\underline{y}, z)\}$, where S is a fresh relation name with mode c. It can be verified that the query q' is saturated.

7.2. Syntactic Simplifications

The following lemma shows that any proof of Theorem 7.1 can assume some syntactic simplifications without loss of generality.

LEMMA 7.6. *For every $q \in \mathsf{sjfBCQ}$, there exists a polynomial-time many-one reduction from $\mathsf{CERTAINTY}(q)$ to $\mathsf{CERTAINTY}(q')$ for some $q' \in \mathsf{sjfBCQ}$ with the following properties:*

- $\mathsf{incnt}(q') \leq \mathsf{incnt}(q)$*;*
- *no atom in q' contains two occurrences of the same variable;*
- *constants occur in q' exclusively at the primary-key position of simple-key atoms;*
- *every atom with mode i in q' is simple-key;*
- *q' is saturated; and*
- *if the attack graph of q contains no strong cycle, then the attack graph of q' contains no strong cycle either.*

The proof of Lemma 7.6 is given in Appendix C and proceeds as follows. For $q \in \mathsf{sjfBCQ}$, we first exhibit a family of polynomial-time many-one reductions from $\mathsf{CERTAINTY}(q)$ to $\mathsf{CERTAINTY}(q')$, where q' possesses more of the desirable properties in the statement of the lemma. Significantly, all these reductions are such that if the attack graph of q contains no strong cycle, then the attack graph of q' contains no strong cycle either. The overall result then follows from composing these reductions.

7.3. Dissolving Markov Cycles

The following definition introduces Markov graphs. The notion of Markov graph will only be defined for sjfBCQ queries in which all atoms of mode i are simple-key. It is a (simple) directed graph whose vertices are the variables of the query. If a query contains an atom $R(\underline{x}, \vec{y})$ of mode i, then its Markov graph will contain directed edges from x to each variable y in $\vec{y}$ with $y \neq x$. Moreover, if a query contains an atom $S^c(\underline{\vec{u}}, \vec{w})$ of mode c and the Markov graph already contains edges from x to each variable u in $\vec{u}$ with $u \neq x$, then the Markov graph contains directed edges from x to each variable w in $\vec{w}$ with $w \neq x$.

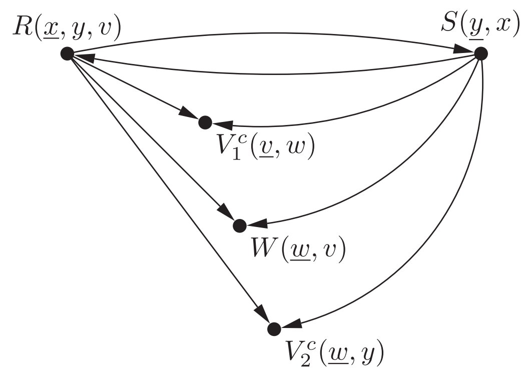
Definition 7.7. Let $q \in \mathsf{sjfBCQ}$ such that every atom with mode i in q is simple-key. For every $x \in \mathsf{vars}(q)$, we define

$$\mathsf{C}_q(x) := \{F \in q \mid F \text{ has mode } i \text{ and } \mathsf{key}(F) = \{x\}\}.$$

Note that $\mathsf{C}_q(x)$ can be empty.

The *Markov graph* of q is a directed graph whose vertex set is $\mathsf{vars}(q)$. There is a directed edge from x to y, denoted $x \xrightarrow{q,\mathsf{M}} y$, if $x \neq y$ and $\mathcal{K}(\mathsf{C}_q(x) \cup \llbracket q \rrbracket) \models x \rightarrow y$. If the query q is clear from the context, then $x \xrightarrow{q,\mathsf{M}} y$ can be shortened into $x \xrightarrow{\mathsf{M}} y$.[2]

[2]The term Markov refers to the intuition that along a path in the Markov graph, each variable functionally determines the next variable in the path independently of preceding variables.



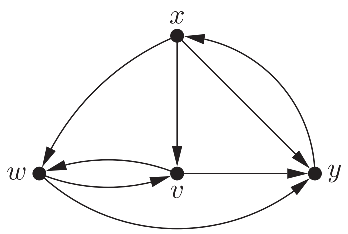


Fig. 4. Attack graph (left) and Markov graph (right) of the query $\{R(\underline{x}, y, v),\ S(\underline{y}, x),\ V_1^c(\underline{v}, w),\ W(\underline{w}, v)\ V_2^c(\underline{w}, y)\}$.

An elementary directed cycle $\mathcal{C}$ in the Markov graph of q is said to be *premier* if there exists a variable $x \in \mathsf{vars}(q)$ such that

(1) $\{x\} = \mathsf{key}(F_0)$ for some atom F_0 with mode i that belongs to an initial strong component of the attack graph of q; and
(2) for some y in $\mathcal{C}$, we have that $\mathcal{K}(q) \models y \rightarrow x$ and the Markov graph contains a directed path from x to y.

The term *Markov path* is used for a path in the Markov graph; the term *Markov cycle* is used for a cycle in the Markov graph.

Example 7.8. Let $q = \{R(\underline{x}, y, v),\ S(\underline{y}, x),\ V_1^c(\underline{v}, w),\ W(\underline{w}, v)\ V_2^c(\underline{w}, y)\}$. All atoms in q are simple-key. Then, $[\![q]\!] = \{V_1^c(\underline{v}, w),\ V_2^c(\underline{w}, y)\}$. We have that $\mathsf{C}_q(x) = \{R(\underline{x}, v, y)\}$. Since $\mathcal{K}(\mathsf{C}_q(x) \cup [\![q]\!]) \models x \rightarrow \{y, v, w\}$, the Markov graph of q contains directed edges from x to each of y, v, and w. We have that $\mathsf{C}_q(v) = \emptyset$. Since $\mathcal{K}(\mathsf{C}_q(v) \cup [\![q]\!]) \models v \rightarrow \{y, w\}$, the Markov graph of q contains directed edges from v to both y and w. The complete Markov graph of q is shown in Figure 4 (right).

The attack graph of q is shown in Figure 4 (left). The atoms $R(\underline{x}, y, v)$ and $S(\underline{y}, x)$ together constitute an initial strong component of the attack graph. It is then straightforward that each cycle in the Markov graph of q that contains x or y must be premier. Further, the cycle $\langle v, w, v\rangle$ in the Markov graph of q is also premier because there is a Markov path from x to v, and $\mathcal{K}(q) \models v \rightarrow x$.

Semantically, Definition 7.7 means the following for sjfBCQ query q and variable x. Let $\mathbf{db}$ be an uncertain database, and let θ_1, θ_2 be two valuations over $\mathsf{vars}(q)$ such that $\theta_1(q) \subseteq \mathbf{db}$ and $\theta_2(q) \subseteq \mathbf{db}$. If $\theta_1(F) = \theta_2(F)$ for all atoms $F \in q$ with mode i such that $\mathsf{key}(F) = \{x\}$, then $\theta_1(y) = \theta_2(y)$ for every variable y that is reachable from x in the Markov graph of q. That is, if y is reachable from x and θ is a valuation such that $\theta(q) \subseteq \mathbf{db}$, then $\theta(y)$ is fully determined if we know $\theta(F)$ for all atoms $F \in q$ with mode i such that $\mathsf{key}(F) = \{x\}$.

Let q be as in Definition 7.7 and assume that the Markov graph of q contains an elementary directed cycle $\mathcal{C}$. Lemma 7.11 states that $\mathsf{CERTAINTY}(q)$ can be reduced in polynomial time to $\mathsf{CERTAINTY}(q^*)$, where q^* is obtained from q by "dissolving" the Markov cycle $\mathcal{C}$ as defined in Definition 7.9. Moreover, we will show (Lemma 7.12) that if $\mathcal{C}$ is premier and the attack graph of q contains no strong cycle, then the attack graph of q^* will contain no strong cycle either. The reduction that "dissolves" Markov cycles

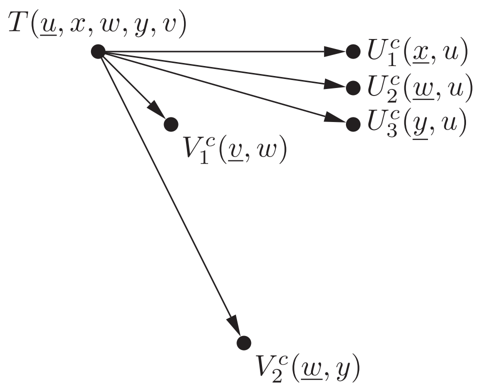

$T(\underline{u}, x, w, y, v)$
$U_1^c(\underline{x}, u)$
$U_2^c(\underline{w}, u)$
$U_3^c(\underline{y}, u)$
$V_1^c(\underline{v}, w)$
$V_2^c(\underline{w}, y)$

Fig. 5. Attack graph of the query that results from dissolving the Markov cycle $\langle x, w, y, x\rangle$ in the query of Figure 4.

will be the central idea in our polynomial-time algorithm for $\mathsf{CERTAINTY}(q)$ when the attack graph of q contains no strong cycle.

Definition 7.9. Let $q \in \mathsf{sjfBCQ}$ such that every atom with mode i in q is simple-key. Let $\mathcal{C}$ be an elementary directed cycle of length $k \geq 2$ in the Markov graph of q. Then, $\mathsf{dissolve}(\mathcal{C}, q)$ denotes the sjfBCQ query defined next. Let $x_0, \ldots, x_{k-1}$ be the variables in $\mathcal{C}$, and let $q_0 = \bigcup_{i=0}^{k-1} \mathsf{C}_q(x_i)$. Let $\vec{y}$ be a sequence of variables containing exactly once each variable of $\mathsf{vars}(q_0) \setminus \{x_0, \ldots, x_{k-1}\}$. Let $q_1 = \{T(\underline{u}, x_0, \ldots, x_{k-1}, \vec{y})\} \cup \{U_i^c(\underline{x_i}, u)\}_{i=0}^{k-1}$, where u is a fresh variable, T is a fresh relation name with mode i, and $U_1, \ldots, U_{k-1}$ are fresh relation names with mode c. Then, we define

$$\mathsf{dissolve}(\mathcal{C}, q) := (q \setminus q_0) \cup q_1.$$

Note that $\mathsf{dissolve}(\mathcal{C}, q)$ is unique up to a renaming of the variable u and the relation names in q_1.

Example 7.10. Let q be the query of Figure 4. Let $\mathcal{C}$ be the cycle $\langle x, w, y, x\rangle$ in the Markov graph of q. Using the notation of Definition 7.9, we have that

$$\begin{aligned} q_0 &= \{R(\underline{x}, y, v), S(\underline{y}, x), W(\underline{w}, v)\},\\ q_1 &= \{T(\underline{u}, x, w, y, v), U_1^c(\underline{x}, u), U_2^c(\underline{w}, u), U_3^c(\underline{y}, u)\}. \end{aligned}$$

Thus, $\mathsf{dissolve}(\mathcal{C}, q) = \{V_1^c(\underline{v}, w), V_2^c(\underline{w}, y), T(\underline{u}, x, w, y, v), U_1^c(\underline{x}, u), U_2^c(\underline{w}, u), U_3^c(\underline{y}, u)\}$. The attack graph of the latter query is shown in Figure 5.

LEMMA 7.11 (DISSOLUTION LEMMA). *Let $q \in \mathsf{sjfBCQ}$ such that every atom with mode i in q is simple-key. Let $\mathcal{C}$ be an elementary directed cycle in the Markov graph of q and let $q^* = \mathsf{dissolve}(\mathcal{C}, q)$. Then, there exists a polynomial-time many-one reduction from* $\mathsf{CERTAINTY}(q)$ *to* $\mathsf{CERTAINTY}(q^*)$.

The reduction of Lemma 7.11 will be explained in Section 8. To use the reduction in a proof of Theorem 7.1, two more results are needed:

- First, we need to show that the dissolution of Markov cycles can be done while keeping the attack graph free of strong cycles (this is Lemma 7.12). This turns out to be true only for Markov cycles that are premier (as defined in Definition 7.7).
- Second, we need to show the existence of premier Markov cycles that can be "dissolved" (this is Lemma 7.13).

LEMMA 7.12. *Let $q \in$ sjfBCQ such that every atom with mode i in q is simple-key. Let $\mathcal{C}$ be an elementary directed cycle in the Markov graph of q such that $\mathcal{C}$ is premier and let $q^* = \mathsf{dissolve}(\mathcal{C}, q)$. If the attack graph of q contains no strong cycle, then the attack graph of q^* contains no strong cycle either.*

LEMMA 7.13. *Let $q \in$ sjfBCQ such that*

- *for every atom $F \in q$, if F has mode i, then F is simple-key and $\mathsf{key}(F) \neq \emptyset$;*
- *q is saturated;*
- *the attack graph of q contains no strong cycle; and*
- *the attack graph of q contains an initial strong component with two or more atoms.*

Then, the Markov graph of q contains an elementary directed cycle that is premier and such that for every y in $\mathcal{C}$, $\mathsf{C}_q(y) \neq \emptyset$.

The condition $\mathsf{C}_q(y) \neq \emptyset$ for every y in $\mathcal{C}$ guarantees that $\mathsf{dissolve}(\mathcal{C}, q)$ will contain strictly less atoms of mode i than q. This condition will be used in the proof of Theorem 7.1, which runs by induction on the number of atoms with mode i. The following example shows that Lemma 7.13 is no longer true if q is not saturated.

Example 7.14. Continuing Example 7.5. The query q of Example 7.5 is not saturated but satisfies all other conditions in the statement of Lemma 7.13. In particular, the attack graph of q contains a weak cycle $R \overset{q}{\rightsquigarrow} U \overset{q}{\rightsquigarrow} R$, which is part of an initial strong component. The Markov graph of q consists of a single path $w \xrightarrow{q,\mathsf{M}} x \xrightarrow{q,\mathsf{M}} y \xrightarrow{q,\mathsf{M}} z$, thus is acyclic.

The query q' of Example 7.5 is saturated and we have $x \xrightarrow{q',\mathsf{M}} w \xrightarrow{q',\mathsf{M}} x$, a Markov cycle which can be shown to be premier.

7.4. The Proof of Theorem 7.1

PROOF OF THEOREM 7.1. Assume that the attack graph of q contains no strong cycle. The proof runs by induction on increasing $\mathsf{incnt}(q)$. The desired result is obvious if $\mathsf{incnt}(q) = 0$. Assume that $\mathsf{incnt}(q) > 0$ in the remainder of the proof. Let **db** be an uncertain database that is input to CERTAINTY(q).

First, we reduce in polynomial time CERTAINTY(q) to CERTAINTY(q') with q' as in Lemma 7.6. We now distinguish two cases.

Case: Some atom F of mode i in q' has zero indegree in the attack graph of q'.

We can assume that either $F = R(\underline{x}, \vec{y})$ or $F = R(\underline{a}, \vec{y})$, where $\vec{y}$ is a sequence of distinct variables. In the remainder, we treat the case $F = R(\underline{x}, \vec{y})$ (the case $F = R(\underline{a}, \vec{y})$ is even simpler).

Let $q'' = q' \setminus \{R(\underline{x}, \vec{y})\}$. By Lemma 4.4, every repair of **db** satisfies q' if and only if **db** includes an R-block **b** (there are only polynomially many such blocks) such that for every $R(\underline{a}, \vec{b}) \in \mathbf{b}$, every repair of **db** satisfies $q''_{[x,\vec{y}\mapsto a,\vec{b}]}$. By Lemma 3.7, the attack graph of $q''_{[x,\vec{y}\mapsto a,\vec{b}]}$ contains no strong cycle. From $\mathsf{incnt}(q''_{[x,\vec{y}\mapsto a,\vec{b}]}) = \mathsf{incnt}(q') - 1 < \mathsf{incnt}(q)$, it follows that CERTAINTY($q''_{[x,\vec{y}\mapsto a,\vec{b}]}$) is in **P** by the induction hypothesis. It follows that CERTAINTY(q) is in **P** as well.

Case: Each atom F of mode i in q' has an incoming attack in the attack graph of q'.

It will be the case that no constant occurs in an atom of mode i in q'.

Then, the attack graph of q' must contain an initial strong component with two or more atoms. By Lemma 7.13, the Markov graph of q' contains an elementary directed cycle $\mathcal{C}$ that is premier and such that for every y in $\mathcal{C}$, $\mathsf{C}_{q'}(y) \neq \emptyset$. By Lemma 7.11, we can reduce in polynomial time CERTAINTY(q') to CERTAINTY(q^*), where $q^* = \mathsf{dissolve}(\mathcal{C}, q')$.

Since the attack graph of q' contains no strong cycle, it follows by Lemma 7.12 that the attack graph of q^* contains no strong cycle either.

Let $k \geq 2$ be the size of $\mathcal{C}$. It can be easily verified that $\mathsf{incnt}(q^*) \leq \big(\mathsf{incnt}(q') - k\big) + 1 < \mathsf{incnt}(q')$. By the induction hypothesis, CERTAINTY(q^*) is in **P**. Since there exists a polynomial-time reduction from CERTAINTY(q) to CERTAINTY(q^*), we conclude that CERTAINTY(q) is in **P** as well.

This concludes the proof that CERTAINTY(q) is in **P**. □

8. PROOF OF THE DISSOLUTION LEMMA (LEMMA 7.11)

In this section, we spell out the reduction of Lemma 7.11, called the Dissolution Lemma, and then prove the lemma. The example in Section 8.1 illustrates the main ideas of the reduction. Sections 8.2 and 8.3 generalize the notions of relevant facts and purified databases, respectively, which were introduced in Section 2. Given an sjfBCQ query q, these notions serve to remove from an uncertain database **db** some blocks that are not relevant to the problem CERTAINTY(q). Section 8.4 introduces the graph-theoretical representation of database facts that underlies the reduction of the Dissolution Lemma. The reduction itself is specified in Section 8.5. Finally, the Dissolution Lemma is shown in Section 8.6.

8.1. Introductory Example

We present an example to illustrate the main ideas behind the reduction in the Dissolution Lemma. Let $q \in$ sjfBCQ such that q includes $q_0 = \{R(\underline{x}, y),\ S(\underline{y}, z),\ V(\underline{z}, x)\}$. Then, the Markov graph of q contains a cycle $x \xrightarrow{\mathsf{M}} y \xrightarrow{\mathsf{M}} z \xrightarrow{\mathsf{M}} x$. Let **db** be an uncertain database that is purified relative to q. Let $\mathbf{db}_0$ be the subset of **db** containing all R-facts, S-facts, and V-facts of **db**. Assume that the following three tables represent all facts of $\mathbf{db}_0$ (for convenience, we use variables as attribute names and we blur the distinction between a relation name R and a table representing a set of R-facts).

R		S		V		
$\underline{x}$	y	$\underline{y}$	z	$\underline{z}$	x	
1	a	a	α	α	1	$\mathbf{db}_{01}$
		a	κ	κ	1	
2	b	b	β	β	2	$\mathbf{db}_{02}$
2	c	c	γ	γ	2	
3	d	d	δ	δ	3	$\mathbf{db}_{03}$
3	e	e	ϵ	ϵ	3	
4	e	e	δ	δ	4	
4	f	f	ϕ	ϕ	4	

As indicated, we can partition $\mathbf{db}_0$ into three subsets $\mathbf{db}_{01}$, $\mathbf{db}_{02}$, and $\mathbf{db}_{03}$ whose active domains have, pairwise, no constants in common. Consider each of these three subsets in turn.

(1) $\mathbf{db}_{01}$ has two repairs, each of which satisfies q_0. For every repair $\mathbf{r}$ of **db**, either $\mathbf{r} \models q_{0[x,y,z \mapsto 1,a,\alpha]}$ or $\mathbf{r} \models q_{0[x,y,z \mapsto 1,a,\kappa]}$.
(2) $\mathbf{db}_{02}$ has two repairs, each of which satisfies q_0. For every repair $\mathbf{r}$ of **db**, either $\mathbf{r} \models q_{0[x,y,z \mapsto 2,b,\beta]}$ or $\mathbf{r} \models q_{0[x,y,z \mapsto 2,c,\gamma]}$.
(3) $\mathbf{db}_{03}$ has 16 repairs, and for $\mathbf{s} := \{R(\underline{3}, d),\ S(\underline{d}, \delta),\ V(\underline{\delta}, 4),\ R(\underline{4}, e),\ S(\underline{e}, \epsilon),\ V(\underline{\epsilon}, 3),\ S(\underline{f}, \phi),\ V(\underline{\phi}, 4)\}$, we have that $\mathbf{s}$ is a repair of $\mathbf{db}_{03}$ that falsifies q_0. It can be easily seen that every repair of **db** satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{db}_{03}$ satisfies q. That is, $\mathbf{db}_{03}$ can be ignored from this point on.

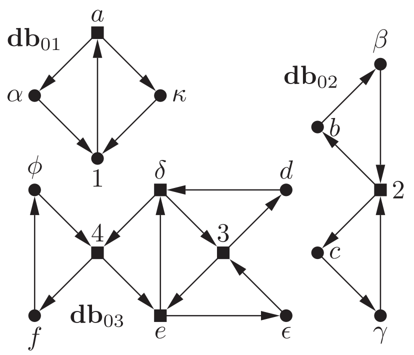


Fig. 6. 3-partite digraph representing the database facts of the example in Section 8.1. Square vertices have outdegree 2. Round vertices have outdegree 1.

The following table T summarizes our findings. In the first column (named with a fresh variable u), the values 01 and 02 refer to $\mathbf{db}_{01}$ and $\mathbf{db}_{02}$, respectively. The table includes two blocks (separated by a dashed line for clarity). The first block indicates that for every repair $\mathbf{r}$ of $\mathbf{db}$, either $\mathbf{r} \models q_{0[x,y,z \mapsto 1,a,\alpha]}$ or $\mathbf{r} \models q_{0[x,y,z \mapsto 1,a,\kappa]}$. Likewise for the second block.

T	$\underline{u}$	x	y	z
	01	1	a	α
	01	1	a	κ
	02	2	b	β
	02	2	c	γ

The following table U_x is the projection of T on attributes x and u. This table must be consistent because, by construction, the active domains of $\mathbf{db}_{01}$ and $\mathbf{db}_{02}$ are disjoint. Likewise for U_y and U_z.

U_x^c	$\underline{x}$	u
	1	01
	2	02

U_y^c	$\underline{y}$	u
	a	01
	b	02
	c	02

U_z^c	$\underline{z}$	u
	α	01
	κ	01
	β	02
	γ	02

Let $\mathbf{db}'$ be the database that extends $\mathbf{db}$ with all facts shown in tables T, U_x, U_y, and U_z.[3] Let $q^* = (q \setminus q_0) \cup \{T(\underline{u}, x, y, z), U_x^c(\underline{x}, u), U_y^c(\underline{y}, u), U_z^c(\underline{z}, u)\}$. From our construction, it follows that every repair of $\mathbf{db}$ satisfies q if and only if every repair of $\mathbf{db}'$ satisfies q^*.

We now give a graph-theoretical interpretation of the example. The directed graph of Figure 6 is a straightforward representation of the database facts of $\mathbf{db}_0$: the vertices are the constants in $\mathbf{db}$ and a directed edge from s to t means that $\mathbf{db}$ contains a fact whose primary-key position contains s, and whose nonprimary-key position contains t. For example, $R(\underline{a}, \alpha)$ gives rise to the directed edge (a, α). The graph is 3-partite because every constant in $\mathbf{db}_0$ belongs to exactly one set among $\mathsf{type}(x)$, $\mathsf{type}(y)$, and $\mathsf{type}(z)$. The square nodes have outdegree 2 and correspond to primary-key violations in the database. The directed graph has three strong components corresponding to $\mathbf{db}_{01}$, $\mathbf{db}_{02}$, and $\mathbf{db}_{03}$.

[3]Facts of $\mathbf{db}_0$ can be omitted from $\mathbf{db}'$, but that is not important.

In this graphical representation, a repair is any subgraph that can be obtained by selecting exactly one outgoing edge for each vertex. Such a repair will satisfy q_0 if and only if it contains an elementary cycle of length 3. The component $\mathbf{db}_{03}$ has a cycle $\langle 3, d, \delta, 4, e, \epsilon, 3\rangle$ of length 6; if we add the edges (f, ϕ) and $(\phi, 4)$ to the edges of this cycle, then we obtain a repair of $\mathbf{db}_{03}$ without cycles of length 3. This subgraph corresponds to the repair $\mathbf{s}$ found before.

When we repair the strong component $\mathbf{db}_{01}$, we cannot avoid creating either the cycle $\langle 1, a, \alpha\rangle$ or the cycle $\langle 1, a, \kappa\rangle$, both of length 3. Likewise, the strong component $\mathbf{db}_{02}$ cannot be repaired without creating one of the cycles $\langle 2, b, \beta\rangle$ or $\langle 2, c, \gamma\rangle$. These unavoidable cycles of length 3 are encoded in the table T.

8.2. Relevance of Subsets of Repairs

In Section 2, we distinguished database facts that are relevant for a query from those that are not. This notion is extended next.

Definition 8.1. Let $q \in \mathsf{sjfBCQ}$ and let $\mathbf{db}$ be an uncertain database. A consistent subset $\mathbf{s}$ of $\mathbf{db}$ is said to be *grelevant for q in* $\mathbf{db}$ (generalized relevant) if it can be extended into a repair $\mathbf{r}$ of $\mathbf{db}$ such that some fact of $\mathbf{s}$ is relevant for q in $\mathbf{r}$.

It can be seen that $A \in \mathbf{db}$ is relevant for q in $\mathbf{db}$ if and only if $\{A\}$ is grelevant for q in $\mathbf{db}$. Therefore, "grelevant" is a notion that generalizes "relevant."

Example 8.2. In the example of Section 8.1, the repair $\mathbf{s}$ of $\mathbf{db}_{03}$ is not grelevant for q in $\mathbf{db}$. As a consequence, every repair of $\mathbf{db}$ satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{db}_{03}$ satisfies q.

LEMMA 8.3. *Let $q \in \mathsf{sjfBCQ}$ and let $\mathbf{db}$ be an uncertain database. Let $\mathbf{s}$ be a consistent subset of $\mathbf{db}$ that is not grelevant for q in $\mathbf{db}$. Let $\mathbf{db}_0 = \bigcup\{\mathsf{block}(A, \mathbf{db}) \mid A \in \mathbf{s}\}$. Then, the following are equivalent:*

(1) *every repair of $\mathbf{db}$ satisfies q;*
(2) *every repair of $\mathbf{db} \setminus \mathbf{db}_0$ satisfies q.*

PROOF. $\boxed{1 \Rightarrow 2}$ By contraposition. Let $\mathbf{r}$ be a repair of $\mathbf{db} \setminus \mathbf{db}_0$ that falsifies q. Then, $\mathbf{r} \cup \mathbf{s}$ is a repair of $\mathbf{db}$. If $\mathbf{r} \cup \mathbf{s} \models q$, then it must be the case that $\mathbf{s}$ is grelevant for q in $\mathbf{db}$, a contradiction. We conclude by contradiction that $\mathbf{r} \cup \mathbf{s} \not\models q$. $\boxed{2 \Rightarrow 1}$ Trivial. □

8.3. Gblocks and Gpurification

The following definition strengthens the notion of purification introduced earlier in Section 2.

Definition 8.4. Let $q \in \mathsf{sjfBCQ}$ such that all atoms with mode i in q are simple-key. Let $\mathbf{db}$ be an uncertain database that is purified and typed relative to q. A *gblock* (generalized block) of $\mathbf{db}$ relative to q is a maximal (with respect to $\subseteq$) subset $\mathbf{g}$ of $\mathbf{db}$ such that all facts in $\mathbf{g}$ have mode i and agree on their primary-key position (but may disagree on their relation name). Note that a gblock has at most polynomially many repairs (in the size of $\mathbf{db}$).[4] We say that $\mathbf{db}$ is *gpurified relative to q* if for every gblock $\mathbf{g}$ of $\mathbf{db}$, every repair of $\mathbf{g}$ is grelevant for q in $\mathbf{db}$.

Clearly, every gblock is the union of one or more blocks. Two facts of the same gblock have the same primary-key value but can have distinct relation names.

[4]Indeed, since $\mathbf{db}$ is purified relative to q, every gblock of $\mathbf{db}$ contains at most $|q|$ distinct relation names, thus has at most $|\mathbf{db}|^{|q|}$ distinct repairs.

Example 8.5. Let $q = \{R(\underline{x}, y), S(\underline{x}, y)\}$. Let $\mathbf{db} = \{R(\underline{a}, 1), R(\underline{a}, 2), S(\underline{a}, 1), S(\underline{a}, 2)\}$. Then, $\mathbf{db}$ is purified and typed relative to q. All facts of $\mathbf{db}$ together constitute a gblock. The uncertain database $\mathbf{db}$ is not gpurified since $\mathbf{s} = \{R(\underline{a}, 1), S(\underline{a}, 2)\}$ is a repair of the gblock and also a repair of $\mathbf{db}$. However, neither $R(\underline{a}, 1)$ nor $S(\underline{a}, 2)$ is relevant for q in $\mathbf{s}$.

Example 8.6. Let $q = \{R_1(\underline{x}, y), R_2(\underline{x}, z), S(\underline{y, z})\}$, where the signature of S is $[2, 2]$. Let $\mathbf{db}$ be the uncertain database containing the following facts.

R_1	$\underline{x}$	y
	a	1
	a	2

R_2	$\underline{x}$	z
	a	3
	a	4

S	$\underline{y}$	$\underline{z}$
	1	3
	2	4

Then, $\mathbf{db}$ is purified and typed relative to q. All R_1-facts and R_2-facts together constitute a gblock. A repair of this gblock is $\mathbf{s} = \{R_1(\underline{a}, 1), R_2(\underline{a}, 4)\}$. The uncertain database $\mathbf{db}$ is not gpurified. The only repair of $\mathbf{db}$ that extends $\mathbf{s}$ is $\{R_1(\underline{a}, 1), R_2(\underline{a}, 4), S(\underline{1, 3}), S(\underline{2, 4})\}$ (call it $\mathbf{r}$). Neither $R_1(\underline{a}, 1)$ nor $R_2(\underline{a}, 4)$ is relevant for q in $\mathbf{r}$.

The following lemma is similar to Lemma 2.4 and has an easy proof.

LEMMA 8.7. *Let $q \in$ sjfBCQ such that all atoms with mode i in q are simple-key. Let* $\mathbf{db}$ *be an uncertain database that is purified and typed relative to q. It is possible to compute in polynomial time an uncertain database* $\mathbf{db}'$ *that is gpurified relative to q such that every repair of* $\mathbf{db}$ *satisfies q if and only if every repair of* $\mathbf{db}'$ *satisfies q.*

8.4. *k*-Partite Digraph Representation of Database Facts

Let q and $\mathcal{C}$ be as in the statement of Lemma 7.11. Assume that the elementary directed cycle $\mathcal{C}$ in the Markov graph of q is $x_0 \xrightarrow{\mathsf{M}} x_1 \cdots \xrightarrow{\mathsf{M}} x_{k-1} \xrightarrow{\mathsf{M}} x_0$. In what follows, let $\mathsf{dissolve}(\mathcal{C}, q)$ be as in Definition 7.9, with q_0, q_1, $\vec{y}$, u, T, and $U_0, \ldots, U_{k-1}$ as defined there. Moreover, we write $\oplus$ for addition modulo k, and $\ominus$ for subtraction modulo k. For every $i \in \{0, \ldots, k-1\}$, we define X_i as follows:

$$X_i := \mathsf{vars}(\mathsf{C}_q(x_i)).$$

The reduction of Lemma 7.11 will be described under the following simplifying assumptions, which can be made without loss of generality:

- Every uncertain database $\mathbf{db}$ that is input to CERTAINTY(q) is typed, purified, and gpurified relative to q. This assumption is without loss of generality as argued in Section 2 and by Lemmas 2.4 and 8.7; and
- for every $i \in \{0, \ldots, k-1\}$, no atom of $\mathsf{C}_q(x_i)$ contains constants or double occurrences of the same variable. This assumption is without loss of generality by Lemma 7.6.

The reduction will rely on a graph representation of database facts as defined next and illustrated in Figure 6. Let $\mathbf{db}$ be an uncertain database that is input to CERTAINTY(q). Define a k-partite directed graph, denoted $\mathcal{G}(\mathbf{db})$, as follows:

(1) the vertex set of $\mathcal{G}(\mathbf{db})$ is $\bigcup_{i=0}^{k-1} \mathsf{type}(x_i)$; and
(2) there is a directed edge from $a \in \mathsf{type}(x_i)$ to $b \in \mathsf{type}(x_{i\oplus 1})$ if for some valuation θ over $\mathsf{vars}(q)$, we have that $\theta(q) \subseteq \mathbf{db}$ and $\theta(x_i) = a$ and $\theta(x_{i\oplus 1}) = b$. In this case, we say that $\theta[X_i]$ *realizes* the edge (a, b), where $\theta[X_i]$ denotes the restriction of θ on X_i.

Note that distinct valuations can realize the same edge of $\mathcal{G}(\mathbf{db})$ (but if $\mathbf{db}$ is consistent, then every edge in $\mathcal{G}(\mathbf{db})$ is realized at most once).

Example 8.8. Let $q = \{R_1(\underline{x_0}, y_1), R_2(\underline{x_0}, y_2), S^c(\underline{y_1, y_2}, x_1), R_3(\underline{x_0}, y_3), V(\underline{x_1}, x_0)\}$. Then, $x_0 \xrightarrow{\mathsf{M}} x_1$ and $X_0 = \{x_0, y_1, y_2, y_3\}$. Assume an uncertain database **db** containing, among others, the following facts.

R_1	$\underline{x_0}$	y_1
	a	c_1

R_2	$\underline{x_0}$	y_2
	a	c_2
	a	c_3

S^c	$\underline{y_1}$	$\underline{y_2}$	x_1
	c_1	c_2	1
	c_1	c_3	1

R_3	$\underline{x_0}$	y_3
	a	β
	a	γ

The graph $\mathcal{G}(\mathbf{db})$ contains a directed edge $(a, 1)$, which is realized by $\{x_0 \mapsto a, y_1 \mapsto c_1, y_2 \mapsto c_2, y_3 \mapsto \beta\}$. The edge $(a, 1)$ is also realized by $\{x_0 \mapsto a, y_1 \mapsto c_1, y_2 \mapsto c_3, y_3 \mapsto \gamma\}$.

Let $[\![\mathbf{db}]\!]$ be the subset of **db** that contains all facts with mode c. A useful insight is that the edges in $\mathcal{G}(\mathbf{db})$ outgoing from some constant $a \in \mathsf{type}(x_j)$ (for some $j \in \{0, \dots, k-1\}$) are fully determined by $[\![\mathbf{db}]\!]$ and the gblock of **db** containing all facts whose relation name is in $\mathsf{C}_q(x_j)$ and whose primary-key position contains the constant a (call this gblock $\mathbf{g}_a$). In other words, $\mathbf{g}_a$ contains all facts that are of the form $R^i(\underline{a}, \vec{b})$ for some $\vec{b}$ and some relation name R of mode i. As a matter of fact, since **db** is gpurified, for every repair **s** of $\mathbf{g}_a$, there exists a unique constant $b \in \mathsf{type}(x_{j\oplus 1})$ such that

$$\mathbf{s} \cup [\![\mathbf{db}]\!] \models \left(\mathsf{C}_q(x_j) \cup [\![q]\!]\right)_{[x_j, x_{j\oplus 1} \mapsto a, b]},$$

in which case $\mathcal{G}(\mathbf{db})$ will contain a directed edge from a to b. Uniqueness of b follows from $\mathcal{K}(\mathsf{C}_q(x_j) \cup [\![q]\!]) \models x_j \rightarrow x_{j\oplus 1}$ and [43, Lemma 4.3].

8.5. Specification of the Reduction of Lemma 7.11

Under the notations and assumptions of Section 8.4, we now specify the reduction of the Dissolution Lemma. Since **db** is assumed to be gpurified, $\mathcal{G}(\mathbf{db})$ is a vertex-disjoint union of strong components such that no edge leads from one strong component to another strong component (i.e., all strong components are initial).[5] In what follows, let D be a strong component of $\mathcal{G}(\mathbf{db})$. Since $\mathcal{G}(\mathbf{db})$ is k-partite, the length of any cycle in $\mathcal{G}(\mathbf{db})$ must be a multiple of k, that is, must be in $\{k, 2k, 3k, \dots\}$. Let $\mathbf{db}_D$ be the subset of **db** that contains $R(\underline{a}, \vec{b})$ whenever R is of mode i and the constant a is a vertex in D (and $\vec{b}$ is any sequence of constants). Obviously, every block of **db** is either included in $\mathbf{db}_D$ or disjoint with $\mathbf{db}_D$.

Clearly, D must contain a cycle. Among the cycles in D of length exactly k, we now distinguish the cycles that *support* q from those that do not, as defined next. Let this cycle in D be

$$\langle a_0, a_1, \dots, a_{k-1}, a_0 \rangle, \tag{9}$$

where for $i \in \{0, \dots, k-1\}$, $a_i \in \mathsf{type}(x_i)$. For $i \in \{0, \dots, k-1\}$, let Δ_i be the set of all valuations over X_i that realize $(a_i, a_{i\oplus 1})$. We say that the cycle (9) *supports* q if for all $i, j \in \{0, \dots, k-1\}$, for all $\mu_i \in \Delta_i$ and $\mu_j \in \Delta_j$, it is the case that μ_i and μ_j agree on all variables in $X_i \cap X_j$. Note that $X_i \cap X_j$ can be empty. The cycle (9) may not support q, because μ_i and μ_j can disagree on variables in $X_i \cap X_j \cap \mathsf{vars}(\vec{y})$, as illustrated next. Recall from the first paragraph of Section 8.4 that $\vec{y}$ is as in Definition 7.9.

[5]Strong components are defined by Definition 3.8.

Example 8.9. Let $q = \{R(\underline{x_0}, x_1, y), S(\underline{x_1}, x_0, y)\}$. We have that $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_0$. Let **db** be the uncertain database containing the following facts.

R	$\underline{x_0}$	x_1	y
	a	1	α
	a	1	β

S	$\underline{x_1}$	x_0	y
	1	a	α
	1	a	β

The edge set of $\mathcal{G}(\mathbf{db})$ is $\{(a, 1), (1, a)\}$. Both $(a, 1)$ and $(1, a)$ are realized by the valuations $\{x_0 \mapsto a, x_1 \mapsto 1, y \mapsto \alpha\}$ and $\{x_0 \mapsto a, x_1 \mapsto 1, y \mapsto \beta\}$, which disagree on y. Thus, the cycle $\langle a, 1, a\rangle$ does not support q. Despite that the cycle $\langle a, 1, a\rangle$ contains both edges of $\mathcal{G}(\mathbf{db})$, we can still construct a repair that falsifies q by choosing an R-fact and an S-fact that disagree on the position of y, for example, $\{R(\underline{a}, 1, \alpha), S(\underline{1}, a, \beta)\}$.

On the other hand, we can assume without loss of generality that μ_i and μ_j agree on all variables in $X_i \cap X_j \cap \{x_0, \dots, x_{k-1}\}$. In particular, if $x_i \in X_j$, then $\mu_j(x_i) = \mu_i(x_i) = a_i$. To see why this is the case, assume that $x_i \in X_j$, where $i, j \in \{0, \dots, k-1\}$ and $i \neq j$. Then, it must be that $x_j \xrightarrow{\mathsf{M}} x_i$. Two cases can occur:

- if $j = i \ominus 1$, then μ_j realizes the edge $(a_{i\ominus 1}, a_i)$ and $\mu_j(x_i) = a_i$; and
- if $j \neq i \ominus 1$, then $x_j \xrightarrow{\mathsf{M}} x_i \xrightarrow{\mathsf{M}} x_{i\oplus 1} \cdots \xrightarrow{\mathsf{M}} x_{j\ominus 1} \xrightarrow{\mathsf{M}} x_j$ is a shorter Markov cycle.

The second case can be avoided by picking $\mathcal{C}$ to be the shorter cycle, as illustrated by Example 8.10. It can be seen that such choice of $\mathcal{C}$ is without loss of generality. In particular, in Lemma 7.13, if $\mathcal{C}$ was premier, then the shorter cycle will also be premier.

Example 8.10. Let $q = \{R(\underline{x_0}, x_1), S(\underline{x_1}, x_2, x_0), V(\underline{x_2}, x_0)\}$. Then, $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_2 \xrightarrow{\mathsf{M}} x_0$. We have that $X_0 = \{x_0, x_1\}$, $X_1 = \{x_1, x_2, x_0\}$, and $X_2 = \{x_2, x_0\}$. Assume an uncertain database **db** with the following facts.

R	$\underline{x_0}$	x_1
	a	1
	b	1

S	$\underline{x_1}$	x_2	x_0
	1	β	a
	1	β	b

V	$\underline{x_2}$	x_0
	β	a
	β	b

The graph $\mathcal{G}(\mathbf{db})$ contains an elementary directed cycle $\langle a, 1, \beta, a\rangle$. The edge $(a, 1)$ is realized by $\mu_0 = \{x_0 \mapsto a, x_1 \mapsto 1\}$. The edge $(1, \beta)$ is realized, among others, by $\mu_1 = \{x_1 \mapsto 1, x_2 \mapsto \beta, x_0 \mapsto b\}$. Note that μ_0 and μ_1 disagree on x_0. Although it would be easy to deal with this situation in which two valuations disagree on a variable in the Markov cycle, it is even easier to avoid this situation by working with the shorter Markov cycle $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_0$.

If we start with $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_0$, we find that $\mathcal{G}(\mathbf{db})$ contains elementary cycles $\langle a, 1, a\rangle$ and $\langle b, 1, b\rangle$. The edges $(a, 1)$ and $(1, a)$ are realized by the valuations $\{x_0 \mapsto a, x_1 \mapsto 1\}$ and $\{x_0 \mapsto a, x_1 \mapsto 1, x_2 \mapsto \beta\}$, respectively, which agree on x_0 and x_1. Likewise, the edges $(b, 1)$ and $(1, b)$ are realized by $\{x_0 \mapsto b, x_1 \mapsto 1\}$ and $\{x_0 \mapsto b, x_1 \mapsto 1, x_2 \mapsto \beta\}$,respectively, which agree on x_0 and x_1.

We now distinguish two cases that are handled in Sections 8.5.1 and 8.5.2, respectively.

8.5.1. Case: D Contains Either an Elementary Directed Cycle of Size k that Does Not Support q or an Elementary Directed Cycle of Size Strictly Greater than k. We now show how to construct a repair **s** of $\mathbf{db}_D$ such that **s** is not grelevant for q in **db**. Then, by Lemma 8.3, every repair of **db** satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{db}_D$ satisfies q. In this case, the reduction deletes from **db** all facts of $\mathbf{db}_D$.

The construction of **s** proceeds as follows. Pick an elementary cycle in D that has size strictly greater than k or that has size k but does not support q. The cycle picked will be denoted by $\mathcal{E}$ from this point on. It is trivial to build in polynomial time a subgraph of D that contains one outgoing edge for every vertex in D such that $\mathcal{E}$ is the only cycle in the subgraph. Essentially, for every vertex u not in the cycle $\mathcal{E}$, the subgraph will contain a path from u to some vertex in $\mathcal{E}$. Let E be the edge set of this subgraph.

To construct $\mathbf{s}$, for each $j \in \{0, \dots, k-1\}$, for each vertex a in D that belongs to $\mathsf{type}(x_j)$, select some valuation μ that realizes the edge in E outgoing from a and add $\mu(\mathsf{C}_q(x_j))$ to $\mathbf{s}$. If $\mathcal{E}$ has size k, then the valuations μ should be selected such that for some vertices a, b in $\mathcal{E}$, the valuations chosen for a and b disagree on some variable of $\mathsf{vars}(\vec{y})$. It is not hard to see that the set $\mathbf{s}$ so obtained is a repair of $\mathbf{db}_D$ that is not grelevant for q in $\mathbf{db}$.

We illustrate this construction by two examples.

Example 8.11. In Example 8.9, one can choose $\mathbf{s} = \{R(\underline{a}, 1, \alpha), S(\underline{1}, a, \beta)\}$. The treatment of a directed cycle of size strictly greater than k is illustrated by $\mathbf{db}_{03}$ in the example of Section 8.1.

Example 8.12. Let $q = \{R(\underline{x_0}, y_1, y_2), V(\underline{x_1}, y_2), S_1^c(\underline{y_1, y_2}, x_1), S_2^c(\underline{y_2}, x_0)\}$. We have that $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_0$, $X_0 = \{x_0, y_1, y_2\}$, and $X_1 = \{x_1, y_2\}$. Let $\mathbf{db}$ be an uncertain database with the following facts.

R	$\underline{x_0}$	y_1	y_2
	a	1	2
	a	3	4
	a	1	6

V	$\underline{x_1}$	y_2
	γ	2
	γ	4
	β	6

S_1^c	$\underline{y_1}$	$\underline{y_2}$	x_1
	1	2	γ
	3	4	γ
	1	6	β

S_2^c	$\underline{y_2}$	x_0
	2	a
	4	a
	6	a

The following table lists the edges in $\mathcal{G}(\mathbf{db})$ by type along with the valuations that realize each edge.

Edges in $\mathsf{type}(x_0) \times \mathsf{type}(x_1)$

Edge	Realized by
(a, γ)	$\{x_0 \mapsto a, y_1 \mapsto 1, y_2 \mapsto 2\}=\mu_1$
	$\{x_0 \mapsto a, y_1 \mapsto 3, y_2 \mapsto 4\}=\mu_2$
(a, β)	$\{x_0 \mapsto a, y_1 \mapsto 1, y_2 \mapsto 6\}=\mu_3$

Edges in $\mathsf{type}(x_1) \times \mathsf{type}(x_0)$

Edge	Realized by
(γ, a)	$\{x_1 \mapsto \gamma, y_2 \mapsto 2\}=\mu_4$
	$\{x_1 \mapsto \gamma, y_2 \mapsto 4\}=\mu_5$
(β, a)	$\{x_1 \mapsto \beta, y_2 \mapsto 6\}=\mu_6$

Then, $\mathcal{G}(\mathbf{db})$ contains two elementary cycles, $\langle a, \gamma, a\rangle$ and $\langle a, \beta, a\rangle$, both of length 2. The cycle $\langle a, \beta, a\rangle$ supports q. The cycle $\langle a, \gamma, a\rangle$ does not support q because μ_1 and μ_5 disagree on y_2. Therefore, the edges (a, γ) and (γ, a), along with μ_1 and μ_5, will be used in the construction of a consistent set $\mathbf{s}$ that is not grelevant for q in $\mathbf{db}$. For the remaining vertex β, we add the edge (β, a), which is only realized by μ_6. Then, $\mathbf{s}$ contains the R-fact $R(\underline{a}, 1, 2)$ (because of μ_1), and the V-facts $V(\underline{\gamma}, 4)$ and $V(\underline{\beta}, 6)$ (because of μ_5 and μ_6, respectively). In this example, there is only one repair that contains $\mathbf{s}$, and this repair falsifies q.

8.5.2. Case: Every Elementary Directed Cycle in D Has Length k and Supports q. In this case, we will encode each cycle of D as a set of T-facts, as follows. Consider any cycle of the form (9) in D and take the cross product

$$\Delta_0 \times \Delta_2 \times \cdots \times \Delta_{k-1}, \tag{10}$$

which is of polynomial size (in the size of $\mathbf{db}$). Since we are in the case in which any cycle of the form (9) supports q, for every tuple $(\mu_0, \mu_1, \dots, \mu_{k-1})$ in the cross product (10), the set $\mu := \bigcup_{i=0}^{k-1} \mu_i$ is a well-defined valuation over $\{x_0, \dots, x_{k-1}\} \cup \mathsf{vars}(\vec{y})$. In this case, for each such tuple, the reduction adds the following $1 + k$ facts:

$$T(\underline{D}, a_0, \dots, a_{k-1}, \mu(\vec{y})), U_0^c(\underline{a_0}, D), U_1^c(\underline{a_0}, D), \dots, U_{k-1}^c(\underline{a_{k-1}}, D),$$

in which D is used as a constant. Recall that $a_i = \mu(x_i)$ for $i \in \{0, \ldots, k-1\}$. Note that if the sequence $\vec{y}$ is empty, then the reduction will add exactly one T-fact for every cycle of the form (9). Otherwise, the reduction may add multiple T-facts for the same cycle, as illustrated next.

Example 8.13. Let $q = \{R(\underline{x_0}, x_1, y),\ S(\underline{x_1}, x_0)\}$. We have that $x_0 \xrightarrow{\mathsf{M}} x_1 \xrightarrow{\mathsf{M}} x_0$, $X_0 = \{x_0, x_1, y\}$, and $X_1 = \{x_0, x_1\}$. Let **db** be the uncertain database containing the following facts.

R	$\underline{x_0}$	x_1	y
	a	1	α
	a	1	β

S	$\underline{x_1}$	x_0
	1	a

The edge set of $\mathcal{G}(\mathbf{db})$ is $\{(a, 1), (1, a)\}$. The edge $(a, 1)$ is realized by both $\{x_0 \mapsto a, x_1 \mapsto 1, y \mapsto \alpha\}$ and $\{x_0 \mapsto a, x_1 \mapsto 1, y \mapsto \beta\}$. The edge $(1, a)$ is realized only by $\{x_0 \mapsto a, x_1 \mapsto 1\}$. The cycle $\langle a, 1, a\rangle$ in $\mathcal{G}(\mathbf{db})$ supports q. The reduction will add the following T-facts (for some identifier D):

T	$\underline{u}$	x_0	x_1	y
	D	a	1	α
	D	a	1	β

Example 8.14. Take the query q of Example 8.12, with the following uncertain database **db**.

R	$\underline{x_0}$	y_1	y_2
	a	1	2
	a	1	6
	a	3	6

V	$\underline{x_1}$	y_2
	γ	2
	β	6

S_1^c	$\underline{y_1}$	$\underline{y_2}$	x_1
	1	2	γ
	1	6	β
	3	6	β

S_2^c	$\underline{y_2}$	x_0
	2	a
	6	a

Then, $\mathcal{G}(\mathbf{db})$ contains two elementary cycles, $\langle a, \gamma, a\rangle$ and $\langle a, \beta, a\rangle$, both of length 2 and both supporting q. The reduction will add the following T-facts (for some identifier D):

T	$\underline{u}$	x_0	x_1	y_1	y_2
	D	a	γ	1	2
	D	a	β	1	6
	D	a	β	3	6

Each relation U_i^c encodes that each constant in $\mathsf{type}(x_i) \cap \mathsf{adom}(\mathbf{db})$ occurs in a unique strong component of $\mathcal{G}(\mathbf{db})$. The meaning of the T-facts is as follows. Let $V = \{x_0, \ldots, x_{k-1}\} \cup \mathsf{vars}(\vec{y})$. Let Θ_D be the set of all valuations μ over V such that

$$T(\underline{D}, \mu(x_1), \ldots, \mu(x_{k-1}), \mu(\vec{y}))$$

has been added by the reduction. Then, the following hold (recall that $q_0 = \bigcup_{i=0}^{k-1} \mathsf{C}_q(x_i)$):

- for every repair $\mathbf{r}$ of $\mathbf{db}$, there exists $\mu \in \Theta_D$ such that $\mathbf{r} \models \mu(q_0)$; and
- for every $\mu \in \Theta_D$, there exists a repair $\mathbf{r}$ of $\mathbf{db}$ such that
  (a) $\mathbf{r} \models \mu(q_0)$; and
  (b) for each $\mu' \in \Theta_D$, if $\mu' \neq \mu$, then $\mathbf{r} \not\models \mu'(q_0)$.

The cycles in D can be found in polynomial time by solving reachability problems, as explained in [44, Theorem 4] and [23]. The crux is that the number of cycles in $\mathcal{G}(\mathbf{db})$ of length exactly k is polynomially bounded. Any longer cycle consists of an elementary path $\langle a_0, a_1, \ldots, a_{k-1}, a_0'\rangle$ of length k ($a_0 \neq a_0'$), concatenated with an elementary path from a'_0 to a_0 that contains no vertex in $\{a_1, \ldots, a_{k-1}\}$. Note incidentally that the reduction needs to know the existence (or not) of cycles of size strictly greater than k in any strong component D, but the vertices on this cycle need not be remembered.

8.6. Proof of Lemma 7.11

The reduction of Section 8.5, by its construction, results in a database $\mathbf{db}'$ that is as in the following lemma.

LEMMA 8.15. *Let q and $\mathcal{C}$ be as in the statement of Lemma 7.11. Let $q^* = \mathsf{dissolve}(q, \mathcal{C})$, and let the variable u be as in Definition 7.9. Let $\mathbf{db}$ be an uncertain database that is input to $\mathsf{CERTAINTY}(q)$. We can compute in polynomial time an uncertain database $\mathbf{db}'$ that is a legal input to $\mathsf{CERTAINTY}(q^*)$ such that the following hold:*

(1) *for every repair $\mathbf{r}$ of $\mathbf{db}$, there exists a repair $\mathbf{r}'$ of $\mathbf{db}'$ such that for every valuation θ over $\mathsf{vars}(q^*)$, if $\theta(q^*) \subseteq \mathbf{r}'$, then $\theta(q) \subseteq \mathbf{r}$; and*
(2) *for every repair $\mathbf{r}'$ of $\mathbf{db}'$, there exists a repair $\mathbf{r}$ of $\mathbf{db}$ such that for every valuation θ over $\mathsf{vars}(q)$, if $\theta(q) \subseteq \mathbf{r}$, then there exists a constant D such that $\theta_{[u \mapsto D]}(q^*) \subseteq \mathbf{r}'$.*

We can now prove Lemma 7.11.

PROOF OF LEMMA 7.11. Let $\mathbf{db}$ be an uncertain database that is input to $\mathsf{CERTAINTY}(q)$. By Lemma 8.15, we can compute in polynomial time an uncertain database $\mathbf{db}'$ that is a legal input to $\mathsf{CERTAINTY}(q^*)$ such that $\mathbf{db}'$ satisfies conditions 1 and 2 in the statement of Lemma 8.15. It suffices to show that the following are equivalent.

(1) Every repair of $\mathbf{db}$ satisfies q.
(2) Every repair of $\mathbf{db}'$ satisfies q^*.

$\boxed{1 \Rightarrow 2}$ Proof by contraposition. Assume a repair $\mathbf{r}'$ of $\mathbf{db}'$ such that $\mathbf{r}' \not\models q^*$. By item 2 in the statement of Lemma 8.15, we can assume a repair $\mathbf{r}$ of $\mathbf{db}$ such that for every valuation θ over $\mathsf{vars}(q)$, if $\theta(q) \subseteq \mathbf{r}$, then there exists a constant D such that $\theta_{[u \mapsto D]}(q^*) \subseteq \mathbf{r}'$. Obviously, if $\mathbf{r} \models q$, then $\mathbf{r}' \models q^*$, a contradiction. We conclude by contradiction that $\mathbf{r} \not\models q$. $\boxed{2 \Rightarrow 1}$ Proof by contraposition. Assume a repair $\mathbf{r}$ of $\mathbf{db}$ such that $\mathbf{r} \not\models q$. By item 1 in the statement of Lemma 8.15, we can assume a repair $\mathbf{r}'$ of $\mathbf{db}'$ such that for every valuation θ over $\mathsf{vars}(q^*)$, if $\theta(q^*) \subseteq \mathbf{r}'$, then $\theta(q) \subseteq \mathbf{r}$. Obviously, $\mathbf{r}' \not\models q^*$. □

9. RELATED WORK

Theoretical developments. CQA goes back to the seminal work by Arenas, Bertossi, and Chomicki [2]. Fuxman and Miller [17, 18] were the first to focus on CQA under the restrictions that consistency is only with respect to primary keys and that queries are self-join-free conjunctive. The term $\mathsf{CERTAINTY}(q)$ was coined in [41]. A recent and comprehensive survey on $\mathsf{CERTAINTY}(q)$ is [45].

In the past decade, a variety of tools and techniques have been used in the complexity classification task for $\mathsf{CERTAINTY}(q)$ with $q \in \mathsf{sjfBCQ}$. In their pioneering work, Fuxman and Miller [17] introduced the notion of *join graph* (not to be confused with the classical notion of join tree). Later, Wijsen [41] introduced the notion of *attack graph*. Kolaitis and Pema [21] showed that, under some conditions, Minty's algorithm [35] can be used to solve $\mathsf{CERTAINTY}(q)$. Koutris and Suciu [23] introduced the notion of *query graph* and the distinction between consistent and possibly inconsistent relations. All these techniques have limited applicability: join graphs seem too rudimentary to obtain general complexity dichotomies; the earliest notion of attack graph enables characterization of first-order expressibility of $\mathsf{CERTAINTY}(q)$, but only for α-acyclic queries q;

Minty's algorithm has been used to establish a **P-coNP**-complete dichotomy in the complexity of CERTAINTY(q), but only for queries q with exactly two atoms; the framework of Koutris and Suciu has also resulted in a **P-coNP**-complete dichotomy, but only when all primary keys consist of a single attribute. On top of the limited applicability of each individual technique, there is the difficulty that complexity classifications expressed in terms of different techniques cannot be easily compared.

Little is known about CERTAINTY(q) beyond self-join-free conjunctive queries. An interesting recent result by Fontaine [15] goes as follows. Let UCQ be the class of Boolean first-order queries that can be expressed as disjunctions of Boolean conjunctive queries (possibly with constants and self-joins). A daring conjecture is that, for every query q in UCQ, CERTAINTY(q) is either in **P** or **coNP**-complete. Fontaine showed that this conjecture implies Bulatov's dichotomy theorem for conservative CSP [8], the proof of which is highly involved. The complexity of CQA for aggregation queries with respect to violations of functional dependencies has been studied in [4].

The counting variant of CERTAINTY(q), denoted $\sharp$CERTAINTY(q), asks to determine the exact number of repairs that satisfy some Boolean query q. In [32], it was shown that, for every self-join-free Boolean conjunctive query q, the counting problem $\sharp$CERTAINTY(q) is either in **FP** or $\sharp$**P**-complete. For conjunctive queries q with self-joins, the complexity of $\sharp$CERTAINTY(q) has been established under the restriction that all primary keys consist of a single attribute [33].

The work on database repairing has inspired work on inconsistency-tolerant query answering in ontology-based data access [7, 27, 29]. It is common to assume that the ontological theory (usually a TBox in some description logic) is correct, while the database (ABox) may contain erroneous facts that are not consistent with the ontological theory. A repair is defined as a maximal (with respect to set inclusion) subset of the ABox that is consistent with the TBox. Since computing the intersection of query answers over all repairs is generally intractable, alternative query semantics have been introduced. In particular, the *Intersection ABox Repair* (IAR) semantics executes queries on the intersection of all repairs (instead of intersecting query answers). Note that in our setting, given an uncertain database, the intersection of its repairs can be computed in polynomial time by removing all blocks that contain at least two facts. Other approaches for restricting repairs are based on preference orders [14, 37] and user-defined policies [31].

Implemented systems. In the past, the paradigm of consistent query answering, and CERTAINTY(q) in particular, has been implemented in expressive formalisms such as Disjunctive Logic Programming [3, 19, 30] and Binary Integer Programming (BIP) [22]. In these formalisms, it is relatively easy to express exponential-time algorithms for CERTAINTY(q). The drawback is that the efficiency of these algorithms is likely to be far from optimal in the case that the certain answer is computable in polynomial time or expressible in first-order logic. In the latter case, the consistent answer can be computed by a single SQL query using standard database technology, including query optimization. In [6, page 38], the author mentions that logic programs for CQA cannot compete with first-order query rewriting mechanisms when they exist. Likewise, in an experimental comparison of EQUIP [22] and ConQuer [16], the authors of the former system found that BIP never outperformed first-order query rewriting.

The Hippo system [10] implements a polynomial-time algorithm for CQA with respect to denial constraints for quantifier-free first-order queries. Since primary keys are denial constraints, this algorithm can be used in our setting for computing certain answers to self-join-free conjunctive queries in which all variables are free. However, from our discussion in Section 3.3, it follows that such queries also have consistent first-order rewritings because their attack graphs are empty (and thus acyclic) when free variables are treated as constants.

In summary, the practical relevance of our results is that they tell us when computationally expensive formalisms can be avoided in the computation of consistent answers. Moreover, by looking at practical examples, we found that many natural self-join-free conjunctive queries have a consistent first-order rewriting. That is, the "easiest" case is by no means exceptional. For example, as soon as a self-join-free conjunctive query, expressed in SQL, on the example database of Figure 1 satisfies one of the following conditions, then its certain answer can be computed in SQL:

- the FROM clause contains only one table;
- the SELECT clause includes one or both primary keys (i.e., E.EID or D.DNAME); or
- the WHERE clause joins E and D on either E.EID = D.MGR or E.DNAME = D.DNAME (but not on both). In other words, the join is a simple primary-to-foreign key join.

10. CONCLUSION

This article settles a long-standing open question in consistent query answering by solving the complexity classification task for CERTAINTY(q) with $q \in$ sjfBCQ. In particular, we showed that, given $q \in$ sjfBCQ, there exists a procedure that looks at the structure of the attack graph of q and decides whether CERTAINTY(q) is in **FO**, in **P** $\setminus$ **FO**, or **coNP**-complete.

An exciting question is whether our results can be extended beyond self-join-free conjunctive queries to conjunctive queries with self-joins and unions of conjunctive queries.

APPENDIXES

A. PROOFS FOR SECTION 3

A.1. Proof of Lemma 3.3

Proof of Lemma 3.3. Let $q \in$ sjfBCQ. The attack graph of q can be computed by means of the algorithm *QuadAttack* in [43], which runs in quadratic time in the size of q. Moreover, the algorithm *QuadAttack* decides whether the attack graph contains a cycle. Significantly, although *QuadAttack* was originally presented for sjfBCQ queries that are α-acyclic in the sense of [13], nothing in the algorithm actually requires α-acyclicity.

The algorithm can be easily extended to compute for each attack whether the attack is weak or strong, as follows. For every atom F of q, the set $F^{\boxplus,q}$ can be computed in linear time in the size of q (in a way similar to the computation of $F^{+,q}$). The computation of $F^{\boxplus,q}$ also yields an array COUNT of size $|q|$ indicating, for every atom G in q, whether $\mathsf{key}(G) \subseteq F^{\boxplus,q}$. Then, when an attack $F \overset{q}{\rightsquigarrow} G$ is discovered, the array COUNT allows one to determine in constant time whether or not the attack is weak. Finally, by Lemma 3.6, to test whether the attack graph contains a strong cycle, it suffices to test whether the attack graph contains a strong cycle of size two. □

A.2. Proof of Lemma 3.5

We use the following helpful lemma.

Lemma A.1. *Let $q \in$ sjfBCQ. Let $F, G \in q$ such that $F \overset{q}{\rightsquigarrow} G$. Then, for every $x \in F^{+,q} \setminus G^{+,q}$, there exists a sequence $F_0, F_1, \ldots, F_n$ of atoms of q such that $F_0 = F$, $x \in \mathsf{vars}(F_n)$, and, for all $i \in \{0, \ldots, n-1\}$, $\mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1}) \not\subseteq G^{+,q}$.*

PROOF. Consider a maximal sequence

$$\begin{array}{rll} \mathsf{key}(F) = & S_0 & H_1 \\ & S_1 & H_2 \\ & \vdots & \vdots \\ & S_{k-1} & H_k \\ & S_k & \end{array}$$

where
(1) $S_0 \subsetneq S_1 \subsetneq \cdots \subsetneq S_{k-1} \subsetneq S_k$; and
(2) for every $i \in \{1, 2, \ldots, k\}$,
 (a) $H_i \in q \setminus \{F\}$. Thus, $\mathcal{K}(q \setminus \{F\})$ contains the functional dependency $\mathsf{key}(H_i) \rightarrow \mathsf{vars}(H_i)$.
 (b) $\mathsf{key}(H_i) \subseteq S_{i-1}$ and $S_i = S_{i-1} \cup \mathsf{vars}(H_i)$.

Then, $S_k = F^{+,q}$. From $F \overset{q}{\rightsquigarrow} G$, it follows that $G \notin \{H_1, \ldots, H_k\}$. For every $v \in S_k$, define $d(v)$ as the smallest integer i such that $v \in S_i$. Let $x \in F^{+,q} \setminus G^{+,q}$. We define the desired result by induction on $d(x)$.

Basis: $d(x) = 0$. Then, the desired sequence is F.

Step: $d(x) = i$. Thus, $x \in S_i$ and $x \notin S_{i-1}$. Then, $x \notin \mathsf{key}(H_i) \subseteq S_{i-1}$ and $x \in \mathsf{vars}(H_i)$. Since $H_i \neq G$, we have that $\mathsf{key}(H_i) \nsubseteq G^{+,q}$, or else $x \in G^{+,q}$, a contradiction. Therefore, we can assume some variable $y \in \mathsf{key}(H_i) \setminus G^{+,q}$. Since $y \in S_{i-1}$, we have that $d(y) < d(x)$. By the induction hypothesis, there exists a sequence $F_0, F_1, \ldots, F_n$ of atoms of q such that

- $F_0 = F$;
- for all $i \in \{0, \ldots, n-1\}$, $\mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1}) \nsubseteq G^{+,q}$; and
- $y \in \mathsf{vars}(F_n)$.

The desired sequence is $F_0, F_1, \ldots, F_n, H_i$. □

The proof of Lemma 3.5 is given next.

PROOF OF LEMMA 3.5. Assume that $F \overset{q}{\rightsquigarrow} G$, $G \overset{q}{\rightsquigarrow} H$, and $F \overset{q}{\not\rightsquigarrow} H$.

Since $F \overset{q}{\rightsquigarrow} G$, there exists a sequence $F_0, F_1, \ldots, F_n$ of atoms of q such that

- $F_0 = F$ and $F_n = G$; and
- for all $i \in \{0, \ldots, n-1\}$, $\mathsf{vars}(F_i) \cap \mathsf{vars}(F_{i+1}) \nsubseteq F^{+,q}$.

Since $G \overset{q}{\rightsquigarrow} H$, there exists a sequence $G_0, G_1, \ldots, G_m$ of atoms of q such that

- $G_0 = G$ and $G_m = H$; and
- for all $i \in \{0, \ldots, m-1\}$, $\mathsf{vars}(G_i) \cap \mathsf{vars}(G_{i+1}) \nsubseteq G^{+,q}$.

Consider the sequence

$$F_0, F_1, \ldots, F_n, G_1, G_2, \ldots, G_m,$$

where $F_0 = F$, $F_n = G = G_0$, and $G_m = H$. Since $F \overset{q}{\not\rightsquigarrow} H$, we can assume some $j \in \{0, \ldots, m-1\}$ such that $\mathsf{vars}(G_j) \cap \mathsf{vars}(G_{j+1}) \subseteq F^{+,q}$. Since $\mathsf{vars}(G_j) \cap \mathsf{vars}(G_{j+1}) \nsubseteq G^{+,q}$, we can assume some $x \in \mathsf{vars}(G_j) \cap \mathsf{vars}(G_{j+1})$ such that $x \in F^{+,q} \setminus G^{+,q}$.

By Lemma A.1, there exists a sequence $H_0, H_1, \ldots, H_k$ of atoms of q such that

- $H_0 = F$;
- for all $i \in \{0, \ldots, k-1\}$, $\mathsf{vars}(H_i) \cap \mathsf{vars}(H_{i+1}) \nsubseteq G^{+,q}$; and
- $x \in \mathsf{vars}(H_k)$.

Consider the sequence

$$G_0, G_1, \ldots, G_j, H_k, H_{k-1}, \ldots, H_0,$$

where $G_0 = G$ and $H_0 = F$. Every two consecutive atoms in this sequence share a variable not in $G^{+,q}$. In particular, G_j and H_k share the variable x. It follows that $G \overset{q}{\rightsquigarrow} F$. □

A.3. Proof of Lemma 3.6

Proof of Lemma 3.6. The first item is an immediate consequence of Lemma 3.5. In what follows, we show the second item.

We show that if the attack graph of q contains a strong cycle of length n with $n \geq 3$, then it contains a strong cycle of some length m with $m < n$.

Let $H_0 \overset{q}{\rightsquigarrow} H_1 \overset{q}{\rightsquigarrow} H_2 \overset{q}{\rightsquigarrow} \cdots \overset{q}{\rightsquigarrow} H_{n-1} \overset{q}{\rightsquigarrow} H_0$ be a strong cycle of length n ($n \geq 3$) in the attack graph of q, where $i \neq j$ implies $H_i \neq H_j$. Assume without loss of generality that the attack $H_0 \overset{q}{\rightsquigarrow} H_1$ is strong. Thus, $\mathcal{K}(q) \not\models \mathsf{key}(H_0) \rightarrow \mathsf{key}(H_1)$.

We write $i \oplus j$ as shorthand for $(i+j) \mod n$. If $H_1 \overset{q}{\rightsquigarrow} H_{1\oplus 2}$, then $H_0 \overset{q}{\rightsquigarrow} H_1 \overset{q}{\rightsquigarrow} H_{1\oplus 2} \overset{q}{\rightsquigarrow} \cdots \overset{q}{\rightsquigarrow} H_{n-1} \overset{q}{\rightsquigarrow} H_0$ is a strong cycle of length $n-1$, and the desired result holds. Assume next that $H_1 \overset{q}{\not\rightsquigarrow} H_{1\oplus 2}$. By Lemma 3.5, $H_2 \overset{q}{\rightsquigarrow} H_1$. We distinguish two cases.

- Case $H_2 \overset{q}{\rightsquigarrow} H_1$ is a strong attack. Then, $H_1 \overset{q}{\rightsquigarrow} H_2 \overset{q}{\rightsquigarrow} H_1$ is a strong cycle of length $2 < n$.
- Case $H_2 \overset{q}{\rightsquigarrow} H_1$ is a weak attack. If $H_1 \overset{q}{\rightsquigarrow} H_0$, then $H_0 \overset{q}{\rightsquigarrow} H_1 \overset{q}{\rightsquigarrow} H_0$ is a strong cycle of length $2 < n$. Assume next that $H_1 \overset{q}{\not\rightsquigarrow} H_0$. Then, from $H_0 \overset{q}{\rightsquigarrow} H_1 \overset{q}{\rightsquigarrow} H_2$ and Lemma 3.5, it follows that $H_0 \overset{q}{\rightsquigarrow} H_2$. The cycle $H_0 \overset{q}{\rightsquigarrow} H_2 \overset{q}{\rightsquigarrow} H_{2\oplus 1} \overset{q}{\rightsquigarrow} \cdots \overset{q}{\rightsquigarrow} H_{n-1} \overset{q}{\rightsquigarrow} H_0$ has length $n-1$. It suffices to show that the attack $H_0 \overset{q}{\rightsquigarrow} H_2$ is strong. Assume toward a contradiction that the attack $H_0 \overset{q}{\rightsquigarrow} H_2$ is weak. Then, $\mathcal{K}(q) \models \mathsf{key}(H_0) \rightarrow \mathsf{key}(H_2)$. Since $H_2 \overset{q}{\rightsquigarrow} H_1$ is a weak attack, $\mathcal{K}(q) \models \mathsf{key}(H_2) \rightarrow \mathsf{key}(H_1)$. By transitivity, $\mathcal{K}(q) \models \mathsf{key}(H_0) \rightarrow \mathsf{key}(H_1)$, a contradiction. This concludes the proof. □

A.4. Proof of Lemma 3.7

Proof of Lemma 3.7. Let $q' = q_{[x\mapsto a]}$. For every $F \in q'$, there exists a (unique) atom $\widehat{F} \in q$ such that $F = \widehat{F}_{[x\mapsto a]}$. It can be easily shown that, for every $F \in q'$, we have $\widehat{F}^{+,q} \setminus \{x\} \subseteq F^{+,q'}$.

Assume that $F \overset{q'}{\rightsquigarrow} G$. Then, there exists a witness $F_0 \overset{z_1}{\frown} F_1 \overset{z_2}{\frown} F_2 \cdots \overset{z_n}{\frown} F_n$ for $F \overset{q'}{\rightsquigarrow} G$, where $F_0 = F$ and $F_n = G$. It can now be easily seen that $\widehat{F_0} \overset{z_1}{\frown} \widehat{F_1} \overset{z_2}{\frown} \widehat{F_2} \cdots \overset{z_n}{\frown} \widehat{F_n}$ is a witness for $\widehat{F} \overset{q}{\rightsquigarrow} \widehat{G}$. Therefore, if the attack graph of q' is cyclic, then the attack graph of q is cyclic.

The second item in the statement of Lemma 3.7 follows from the observation that, for all $F, G \in q'$, if $\mathcal{K}(q) \models \mathsf{key}(\widehat{F}) \rightarrow \mathsf{key}(\widehat{G})$, then $\mathcal{K}(q') \models \mathsf{key}(F) \rightarrow \mathsf{key}(G)$. □

B. PROOFS FOR SECTION 4

B.1. Proof of Lemma 4.2

Proof of Lemma 4.2. We show a first-order reduction from the problem Undirected Forest Accessibility (UFA) [11] to CERTAINTY(q_0). In UFA, we are given an acyclic undirected graph and nodes u, v. The problem is to determine whether there is a path between u and v. The problem is **L**-complete and remains **L**-complete when the given graph has exactly two connected components. Moreover, we can assume in the reduction that the two connected components each contain at least one edge.

Given an acyclic undirected graph $G = (V, E)$ with exactly two connected components and two nodes u, v, we construct an uncertain database **db** as follows:

(1) for every edge $\{a, b\}$ in E, the uncertain database **db** contains the facts $R_0(\underline{a}, \{a, b\})$, $R_0(\underline{b}, \{a, b\})$, $S_0(\underline{\{a, b\}}, a)$, and $S_0(\underline{\{a, b\}}, b)$, in which $\{a, b\}$ is treated as a constant; and
(2) **db** contains $R_0(\underline{u}, t)$ and $R_0(\underline{v}, t)$, where t is a new value not occurring elsewhere.

Clearly, the computation of **db** from G is in **FO**.

We next show that there exists a path between u and v in G if and only if every repair of **db** satisfies q_0.

Assume first that u, v belong to the same connected component. Let $\mathbf{db}'$ be the uncertain database that is constructed from the connected component not containing u, v. Let $a_0, b_0, a_1, b_1, \ldots, a_{n-1}, b_{n-1}, a_n$ be a sequence of distinct constants such that

(1) $a_0 = a_n$ and for $0 \leq i < j \leq n-1$, $a_i \neq a_j$ and $b_i \neq b_j$; and
(2) for $i \in \{0, \ldots, n-1\}$, $\mathbf{db}'$ contains $R_0(\underline{a_i}, b_i)$ and $S_0(\underline{b_i}, a_{i+1})$.

Since G is acyclic, any such sequence satisfies $n = 1$. An existing algorithm for $\mathsf{CERTAINTY}(q_0)$ [23, 44] will return that every repair of $\mathbf{db}'$ satisfies q_0. Consequently, every repair of **db** satisfies q_0.

For the opposite implication, assume that u and v belong to distinct connected components. By Lemma 2.4, there exists an uncertain database $\mathbf{db}'$ that is purified relative to q_0 such that q_0 is true in every repair of $\mathbf{db}'$ if and only if q_0 is true in every repair of **db**. It is easy to see that if u and v belong to distinct connected components, then this purified uncertain database $\mathbf{db}'$ will be the empty database whose only repair is the empty repair that falsifies q_0. It follows that q_0 is not true in every repair of **db**. □

B.2. Proof of Lemma 4.4

We first show two helpful lemmas.

LEMMA B.1. *Let $q \in \mathsf{sjfBCQ}$. Let $X \subseteq \mathsf{vars}(q)$ and let $G \in q$ be an R-atom such that for every $x \in X$, $G \overset{q}{\not\rightsquigarrow} x$. Let $\mathbf{r}$ be a repair of some database such that $\mathbf{r} \models q$. Let $A \in \mathbf{r}$ be an R-fact that is relevant for q in $\mathbf{r}$. Let B be key-equal to A and $\mathbf{r}_B = (\mathbf{r} \setminus \{A\}) \cup \{B\}$. Then, for every valuation ζ over X, if $\mathbf{r}_B \models \zeta(q)$, then $\mathbf{r} \models \zeta(q)$.*

PROOF. Let ζ be a valuation over X such that $\mathbf{r}_B \models \zeta(q)$. We can assume a valuation ζ^+ over $\mathsf{vars}(q)$ such that $\zeta^+[X] = \zeta[X]$ and $\zeta^+(q) \subseteq \mathbf{r}_B$. Thus, ζ^+ extends ζ to $\mathsf{vars}(q)$. We need to show that $\mathbf{r} \models \zeta(q)$, which is obvious if $B \notin \zeta^+(q)$. Assume next that $B \in \zeta^+(q)$. Since A is relevant for q in $\mathbf{r}$, we can assume a valuation μ over $\mathsf{vars}(q)$ such that $A \in \mu(q) \subseteq \mathbf{r}$. Let $q' = q \setminus \{G\}$. Let $\mathbf{r}' = \mathbf{r}_B \setminus \{B\} = \mathbf{r} \setminus \{A\}$. Since q' contains no R-atom (no self-join), $\zeta^+(q') \subseteq \mathbf{r}'$ and $\mu(q') \subseteq \mathbf{r}'$. Moreover, $\zeta^+[\mathsf{key}(G)] = \mu[\mathsf{key}(G)]$ because A and B are key-equal. From $\mathcal{K}(q') \models \mathsf{key}(G) \rightarrow G^{+,q}$ and [43, Lemma 4.3], it follows that $\zeta^+[G^{+,q}] = \mu[G^{+,q}]$.

Let q_G be the subset of q that contains G as well as every atom $H \in q$ such that $G \overset{q}{\rightsquigarrow} H$. Let $q_X = q \setminus q_G$. Let κ be the valuation over $\mathsf{vars}(q)$ such that, for every $x \in \mathsf{vars}(q)$,

$$\kappa(x) = \begin{cases} \mu(x) & \text{if } x \in \mathsf{vars}(q_G) \\ \zeta^+(x) & \text{if } x \in \mathsf{vars}(q_X). \end{cases}$$

We show that κ is well defined. Assume that $x \in \mathsf{vars}(q_X) \cap \mathsf{vars}(q_G)$. Then, there exist atoms $F' \in q_X$ and $G' \in q_G$ such that $x \in \mathsf{vars}(F') \cap \mathsf{vars}(G')$. Since $G \overset{q}{\not\rightsquigarrow} F'$ and either $G = G'$ or $G \overset{q}{\rightsquigarrow} G'$, it follows that $\mathsf{vars}(F') \cap \mathsf{vars}(G') \subseteq G^{+,q}$. Consequently, $x \in G^{+,q}$. Since $\zeta^{+}[G^{+,q}] = \mu[G^{+,q}]$, it follows that $\mu(x) = \zeta^{+}(x)$.

Obviously, $\kappa(q) \subseteq \mathbf{r}$. Finally, we show that, for every $u \in X$, $\kappa(u) = \zeta(u)$. This is obvious if $u \in X \cap G^{+,q}$. Assume next that $u \in X \setminus G^{+,q}$. Since $G \overset{q}{\not\rightsquigarrow} u$ by the assumption in the statement of Lemma B.1, it must be the case that $u \in \mathsf{vars}(q_X)$; thus, $\kappa(u) = \zeta^{+}(u) = \zeta(u)$. It follows that $\mathbf{r} \models \zeta(q)$. This concludes the proof. □

The following helpful lemma extends [43, Lemma B.1].

LEMMA 12.2. *Let $q \in \mathsf{sjfBCQ}$. Let $F \in q$ such that F has zero indegree in the attack graph of q. Let $\mathbf{r}$ be a repair of some database. Let $A \in \mathbf{r}$ such that A is relevant for q in $\mathbf{r}$.*[6] *Let B be key-equal to A and $\mathbf{r}_B = (\mathbf{r} \setminus \{A\}) \cup \{B\}$. Then, for every valuation ζ over* $\mathsf{key}(F)$*, if $\mathrm{r}_B \models \zeta(q)$, then $\mathbf{r} \models \zeta(q)$.*

PROOF. The proof is obvious if A has the same relation name as F. Assume next that relation names in A and F are distinct. We can assume some atom $G \in q \setminus \{F\}$ such that A has the same relation name as G. Since $G \overset{q}{\not\rightsquigarrow} F$, we have that, for each $x \in \mathsf{key}(F)$, $G \overset{q}{\not\rightsquigarrow} x$. The desired result then follows by Lemma B.1. □

Assume that a sjfBCQ query q contains an R-atom that has no incoming attack in the attack graph of q. Paraphrasing Lemma B.2, if one replaces, in a repair $\mathbf{r}$, some relevant fact A with another fact B that belongs to the same block as A, then every R-fact of $\mathbf{r}$ that was not relevant in $\mathbf{r}$ will remain nonrelevant in $(\mathbf{r} \setminus \{A\}) \cup \{B\}$. Note, however, that the fact B may be nonrelevant in the new repair $(\mathbf{r} \setminus \{A\}) \cup \{B\}$.

The proof of Lemma 4.4 can now be given.

PROOF OF LEMMA 4.4. Let $X = \mathsf{key}(F)$. Let $\mathbf{db}$ be an uncertain database.

$\boxed{2 \Rightarrow 1}$ Trivial. $\boxed{1 \Rightarrow 2}$ Assume that q is true in every repair of $\mathbf{db}$. We can assume a repair $\mathbf{r}$ of $\mathbf{db}$ that is $\preceq_q^X$-frugal. Let $\mathbf{s}$ be any repair of $\mathbf{db}$. Construct a maximal sequence

$$\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_n, \tag{11}$$

where

(1) $\mathbf{r}_0 = \mathbf{r}$; and
(2) for every $i \in \{1, \ldots, n\}$, $\mathbf{r}_i = (\mathbf{r}_{i-1} \setminus \{A\}) \cup \{B\}$ for distinct, key-equal facts A, B such that $A \in \mathbf{r}_{i-1}$, $B \in \mathbf{s}$, and A is relevant for q in $\mathbf{r}_{i-1}$.

The sequence obviously terminates when for every A that is relevant for q in $\mathbf{r}_n$, we have $A \in \mathbf{s}$. It follows that, for every valuation θ over X,

$$\mathbf{r}_n \models \theta(q) \Rightarrow \mathbf{s} \models \theta(q). \tag{12}$$

By Lemma B.2, for every valuation θ over X,

$$\mathbf{r}_n \models \theta(q) \Rightarrow \mathbf{r} \models \theta(q). \tag{13}$$

From Equation (13) and since $\mathbf{r}$ is $\preceq_q^X$-frugal, it follows that, for every valuation θ over X,

$$\mathbf{r}_n \models \theta(q) \iff \mathbf{r} \models \theta(q). \tag{14}$$

[6]Recall from Section 2 that $A \in \mathbf{r}$ is *relevant* for q in $\mathbf{r}$ if $A \in \theta(q) \subseteq \mathbf{r}$ for some valuation θ over $\mathsf{vars}(q)$.

From Equations (14) and (12), it follows that, for every valuation θ over X,

$$\mathbf{r} \models \theta(q) \Rightarrow \mathbf{s} \models \theta(q). \tag{15}$$

Since $\mathbf{r} \models q$, we can assume a valuation μ over X such that $\mathbf{r} \models \mu(q)$. By (15), $\mathbf{s} \models \mu(q)$. Since $\mathbf{s}$ is an arbitrary repair, the desired result follows. □

C. PROOFS FOR SECTIONS 7 AND 8

See the online appendix of this article, which can be accessed in the ACM Digital Library.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, New York, NY.

[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent query answers in inconsistent databases. In *Proceedings of the 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, May 31 - June 2, 1999, Philadelphia, Pennsylvania, Victor Vianu and Christos H. Papadimitriou (Eds.). ACM Press, 68–79. DOI:http://dx.doi.org/10.1145/303976.303983

[3] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 2003a. Answer sets for consistent query answering in inconsistent databases. *Theory and Practice of Logic Programming* 3, 4–5, 393–424. DOI:http://dx.doi.org/10.1017/S1471068403001832

[4] Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy P. Spinrad. 2003b. Scalar aggregation in inconsistent databases. *Theoretical Computer Science* 296, 3, 405–434. DOI:http://dx.doi.org/10.1016/S0304-3975(02)00737-5

[5] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. 1979. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters* 8, 3, 121–123. DOI:http://dx.doi.org/10.1016/0020-0190(79)90002-4

[6] Leopoldo E. Bertossi. 2011. *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, San Francisco, CA. DOI:http://dx.doi.org/10.2200/S00379ED1V01Y201108DTM020

[7] Meghyn Bienvenu, Camille Bourgaux, and François Goasdoué. 2016. Explaining inconsistency-tolerant query answering over description logic knowledge bases. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, February 12-17, 2016, Phoenix, AZ, Dale Schuurmans and Michael P. Wellman (Eds.). AAAI Press, 900–906. http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12025

[8] Andrei A. Bulatov. 2011. Complexity of conservative constraint satisfaction problems. *ACM Transactions on Computational Logic* 12, 4, 24:1–24:66. DOI:http://dx.doi.org/10.1145/1970398.1970400

[9] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Information and Computation* 197, 1–2, 90–121. DOI:http://dx.doi.org/10.1016/j.ic.2004.04.007

[10] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. 2004. Hippo: A system for computing consistent answers to a class of SQL queries. In *EDBT (Lecture Notes in Computer Science)*, Elisa Bertino, Stavros Christodoulakis, Dimitris Plexousakis, Vassilis Christophides, Manolis Koubarakis, Klemens Böhm, and Elena Ferrari (Eds.), Vol. 2992. Springer, Berlin, 841–844. DOI:http://dx.doi.org/10.1007/978-3-540-24741-8_53

[11] Stephen A. Cook and Pierre McKenzie. 1987. Problems complete for deterministic logarithmic space. *Journal of Algorithms* 8, 3, 385–394. DOI:http://dx.doi.org/10.1016/0196-6774(87)90018-6

[12] Alexandre Decan, Fabian Pijcke, and Jef Wijsen. 2012. Certain conjunctive query answering in SQL. In *Proceedings of the Scalable Uncertainty Management - 6th International Conference, SUM 2012*, Marburg, Germany, September 17-19, 2012 *(Lecture Notes in Computer Science)*, Eyke Hüllermeier, Sebastian Link, Thomas Fober, and Bernhard Seeger (Eds.), Vol. 7520. Springer, Berlin, 154–167. DOI:http://dx.doi.org/10.1007/978-3-642-33362-0_12

[13] Ronald Fagin. 1983. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM* 30, 3, 514–550. DOI:http://dx.doi.org/10.1145/2402.322390

[14] Ronald Fagin, Benny Kimelfeld, and Phokion G. Kolaitis. 2015. Dichotomies in the complexity of preferred repairs, In Tova Milo and Diego Calvanese (Eds.). *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM. 3–15. DOI:http://dx.doi.org/10.1145/2745754.2745762

[15] Gaëlle Fontaine. 2013. Why is it hard to obtain a dichotomy for consistent query answering? In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*. IEEE Computer Society, 550–559. DOI:http://dx.doi.org/10.1109/LICS.2013.62

[16] Ariel Fuxman, Elham Fazli, and Renée J. Miller. 2005. ConQuer: Efficient management of inconsistent databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, Fatma Özcan (Ed.). ACM, 155–166. DOI:http://dx.doi.org/10.1145/1066157.1066176

[17] Ariel Fuxman and Renée J. Miller. 2005. First-order query rewriting for inconsistent databases. In *Proceedings of the 10th International Conference on Database Theory (ICDT'05), Edinburgh, UK, January 5-7, 2005, (Lecture Notes in Computer Science)*, Thomas Eiter and Leonid Libkin (Eds.), Vol. 3363. Springer, Berlin, 337–351. DOI:http://dx.doi.org/10.1007/978-3-540-30570-5_23

[18] Ariel Fuxman and Renée J. Miller. 2007. First-order query rewriting for inconsistent databases. *Journal of Computer and System Sciences* 73, 4, 610–635. DOI:http://dx.doi.org/10.1016/j.jcss.2006.10.013

[19] Gianluigi Greco, Sergio Greco, and Ester Zumpano. 2003. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowl.edge and Data Engineering* 15, 6, 1389–1408. DOI:http://dx.doi.org/10.1109/TKDE.2003.1245280

[20] Neil Immerman. 1999. *Descriptive Complexity*. Springer. DOI:http://dx.doi.org/10.1007/978-1-4612-0539-5

[21] Phokion G. Kolaitis and Enela Pema. 2012. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Information Processing Letters* 112, 3, 77–85. DOI:http://dx.doi.org/10.1016/j.ipl.2011.10.018

[22] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. 2013. Efficient querying of inconsistent databases with binary integer programming. *Proceedings of the PLVD Endowment* 6, 6, 397–408. http://www.vldb.org/pvldb/vol6/p397-tan.pdf.

[23] Paraschos Koutris and Dan Suciu. 2014. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). 2014. *Proceedings of the 17th International Conference on Database Theory (ICDT'14), Athens, Greece, March 24-28, 2014*, 165–176. DOI:http://dx.doi.org/10.5441/002/icdt.2014.19

[24] Paraschos Koutris and Jef Wijsen. 2015. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In Tova Milo and Diego Calvanese (Eds.). 2015. *Proceedings of the 34th ACM Symposium on Principles of Database Systems, PODS 2015, Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM. 17–29. DOI:http://dx.doi.org/ 10.1145/2745754.2745769

[25] Paraschos Koutris and Jef Wijsen. 2016. Consistent query answering for primary keys. *SIGMOD Record* 45, 1, 15–22. DOI:http://dx.doi.org/10.1145/2949741.2949746

[26] Richard E. Ladner. 1975. On the structure of polynomial time reducibility. *Journal of the ACM* 22, 1, 155–171. DOI:http://dx.doi.org/10.1145/321864.321877

[27] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. 2015. Inconsistency-tolerant query answering in ontology-based data access. *Journal of Web Semantics* 33, 3–29. DOI:http://dx.doi.org/10.1016/j.websem.2015.04.002

[28] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer. DOI:http://dx.doi.org/10.1007/978-3-662-07003-1

[29] Thomas Lukasiewicz, Maria Vanina Martinez, Andreas Pieris, and Gerardo I. Simari. 2015. From classical to consistent query answering under existential rules. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, TX*. Blai Bonet and Sven Koenig (Eds.). AAAI Press, 1546–1552. http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9817

[30] Mónica Caniupán Marileo and Leopoldo E. Bertossi. 2010. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data and Knowledge Engineering* 69, 6, 545–572. DOI:http://dx.doi.org/10.1016/j.datak.2010.01.005

[31] Maria Vanina Martinez, Francesco Parisi, Andrea Pugliese, Gerardo I. Simari, and V. S. Subrahmanian. 2014. Policy-based inconsistency management in relational databases. *International Journal of Approximate Reasoning* 55, 2, 501–528. DOI:http://dx.doi.org/10.1016/j.ijar.2013.12.004

[32] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *Journal of Computer and System Sciences* 79, 6, 958–983. DOI:http://dx.doi.org/10.1016/ j.jcss.2013.01.011

[33] Dany Maslowski and Jef Wijsen. 2014. Counting database repairs that satisfy conjunctive queries with self-joins, In Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). 2014. *Proceedings of the 17th International Conference on Database Theory (ICDT'14), Athens, Greece, March 24-28, 2014*. 155–164. DOI:http://dx.doi.org/10.5441/002/icdt.2014.18

[34] Tova Milo and Diego Calvanese (Eds.). 2015. *Proceedings of the 34th ACM Symposium on Principles of Database Systems (PODS'15), Melbourne, Victoria, Australia, May 31 - June 4, 2015*. ACM. http://dl.acm.org/citation.cfm?id=2745754

[35] George J. Minty. 1980. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B* 28, 3, 284–304. DOI:http://dx.doi.org/10.1016/0095-8956(80)90074-X

[36] Nicole Schweikardt, Vassilis Christophides, and Vincent Leroy (Eds.). 2014. *Proceedings of the 17th International Conference on Database Theory (ICDT'14), Athens, Greece, March 24-28, 2014*. OpenProceedings.org. http://openproceedings.org/edbticdt2014/ICDT_toc.html.

[37] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Annals of Mathematics and Artificial Intelligence* 64, 2–3, 209–246. DOI:http://dx.doi.org/10.1007/s10472-012-9288-8

[38] Wang-Chiew Tan. 2016. Technical perspective: Attacking the problem of consistent query answering. *SIGMOD Record* 45, 1, 14. DOI:http://dx.doi.org/10.1145/2949741.2949745

[39] Jeffrey D. Ullman. 1988. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, Rockville, MD.

[40] Jef Wijsen. 2009. On the consistent rewriting of conjunctive queries under primary key constraints. *Information Systems* 34, 7, 578–601. DOI:http://dx.doi.org/10.1016/j.is.2009.03.011

[41] Jef Wijsen. 2010a. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *Proceedings of the 29th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'10), June 6-11, 2010, Indianapolis, IN*, Jan Paredaens and Dirk Van Gucht (Eds.). ACM, 179–190. DOI:http://dx.doi.org/10.1145/1807085.1807111

[42] Jef Wijsen. 2010b. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Information Processing Letters* 110, 21, 950–955. DOI:http://dx.doi.org/10.1016/j.ipl.2010.07.021

[43] Jef Wijsen. 2012. Certain conjunctive query answering in first-order logic. *ACM Transactions on Database Systems* 37, 2, 9:1–9:35. DOI:http://dx.doi.org/10.1145/2188349.2188351

[44] Jef Wijsen. 2013. Charting the tractability frontier of certain conjunctive query answering. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, (PODS'13), New York, NY, June 22-27, 2013*, Richard Hull and Wenfei Fan (Eds.). ACM, 189–200. DOI:http://dx.doi.org/10.1145/2463664.2463666

[45] Jef Wijsen. 2014. A survey of the data complexity of consistent query answering under key constraints. In *Proceedings of Foundations of Information and Knowledge Systems - 8th International Symposium (FoIKS'14), Bordeaux, France, March 3-7, 2014. (Lecture Notes in Computer Science)*, Christoph Beierle and Carlo Meghini (Eds.), Vol. 8367. Springer, Berlin, 62–78. DOI:http://dx.doi.org/10.1007/978-3-319-04939-7_2

Received December 2015; revised September 2016; accepted March 2017