Consistent Query Answering for Primary Keys

Paraschos Koutris
University of Wisconsin-Madison
Madison, Wisconsin, USA
paris@cs.wisc.edu

Jef Wijsen
University of Mons
Mons, Belgium
jef.wijsen@umons.ac.be

ABSTRACT

We study the complexity of consistent query answering with respect to primary key violations, for self-join-free conjunctive queries. A repair of a possibly inconsistent database is obtained by selecting a maximal number of tuples without selecting two distinct tuples with the same primary key value. For any Boolean query q, CERTAINTY(q) is the problem that takes a database as input, and asks whether q is true in every repair of the database. The complexity of this problem has been extensively 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. We show that for any self-join-free Boolean conjunctive query q, it can be decided whether or not CERTAINTY(q) is in **FO**. Further, CERTAINTY(q) is either in **P** or **coNP**-complete, and the complexity dichotomy is effective. This settles a research question of practical relevance that has been open for ten years.

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, when we integrate data from heterogeneous sources, or when the original data sources are imprecise or noisy.

The standard method of dealing with inconsistency is *data cleaning*, where the database is first *repaired* to satisfy the integrity constraints, and then we use the clean version to answer queries. However, since we can typically repair a database in many different ways, it is often the case that we have to make arbitrary choices about which data to keep, which means information may be lost. An alternative to data cleaning is *consistent query answering* (CQA), which was first introduced in [1]. In this framework, we answer queries by considering all possible repairs of the inconsistent database, and returning the intersection of the answers on all repairs, which is called the *consistent* (or *certain*) answer.

In this article, we focus on integrity constraints that are *primary key constraints*. Consider the database in Fig. 1, which includes the table E that stores employee information and the table D that stores department information. 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. The CITY column in table E stores the city of birth for each employee.

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

Figure 1: Example of inconsistent database that violates the primary key constraints.

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 Fig. 1, blocks are separated by dashed lines for readability. The reader should notice 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 where we would represent the city with a single uninformative NULL.

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.

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*). 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 the names of employees who were born in London,"* which is encoded in SQL as follows.

The original version of this article was published in PODS 2015.

```
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-join, 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 above 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 non existence of a city of birth other than London, and thereby excludes Blake from the answer. Unfortunately, it is not always possible to obtain certain answers directly in SQL. We show in this paper that for all conjunctive queries without self-joins, CQA with respect to primary keys can be classified into one of three exclusive classes of increasing complexity:

1) For some queries, the certain answer can be expressed in relational calculus (or first-order logic), and hence 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 which 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.

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.

3) For some queries, the certain answer cannot be obtained in polynomial time (unless $\mathbf{P} = \mathbf{NP}$), since it is **coNP**-hard to compute the certain answer. We will see in Section 3 that an example of such 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.

Given a conjunctive query q without self-join, it can be decided which of the three classes it belongs to. Moreover, if the query falls into the first or second class, then we know how to construct an SQL query or a polynomial-time algorithm for computing its certain answer.

Our result provides a complexity classification for CQA with respect to primary keys, when the query ranges over the set of self-join-free conjunctive queries. This complexity classification task has been an open problem since 2005 [8], and culminates a long line of research [8, 10, 12, 20, 22].

In Section 6, we explain why our theoretical results are of interest to practitioners and system builders. In short, CQA has often been implemented by means of expressive and computationally expensive languages, like Disjunctive Logic Programming. The efficiency of these algorithms is likely to be far from optimal in the case that consistent answers can be computed in polynomial time or, even faster, by executing a single SQL query. The practical relevance of our results is that they tell us when computationally expensive formalisms can be avoided. Moreover, it turns out that for many natural queries, consistent answers can be obtained with low complexity.

Organization. Section 2 introduces the data and query model. In Section 3, we define a syntactic tool, called *attack graph*, and use it to state Theorem 1, which is the main result of the article. Sections 4 and 5 zoom in on two important complexity boundaries: Section 4 elaborates on first-order expressibility, and Section 5 on polynomial-time tractability. Section 6 discusses both the theoretical and systems-oriented related work.

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.

Atoms and key-equal facts. Each *relation name* R of arity n, $n \geq 1$, has a unique *primary key* which 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*. 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 Fig. 1 has signature $[4, 1]$ and is simple-key.

If R is a relation name with signature $[n, k]$, then the expression $R(s_1, \ldots, s_n)$ 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*, denoted $R_1(\underline{\vec{a}_1}, \vec{b}_1) \sim R_2(\underline{\vec{a}_2}, \vec{b}_2)$, 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 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. An *uncertain database* is a finite set **db** of facts using only the relation names of a fixed database schema. We refer to databases as "uncertain databases" to emphasize that such databases can violate primary key constraints.

A *block* of **db** is a maximal set of key-equal facts of **db**. The term R-block refers to a block of R-facts, i.e., facts with relation name R. An uncertain database **db** is *consistent* if no two distinct facts are key-equal (i.e., if every block of **db** is a singleton). A *repair* of **db** is a maximal (with respect to set inclusion) consistent subset of **db**.

Boolean conjunctive queries. A *Boolean query* is a mapping q that associates true or false to each uncertain database, such that q is closed under isomorphism [15]. We write $\mathbf{db} \models q$ to denote that q associates true to **db**, in which case **db** is said to *satisfy* q. A *Boolean first-order query* is a Boolean query that can be defined in first-order logic (with equality and constants, but without other built-in predicates). 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. 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 uncertain database **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*. 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 write BCQ for the class of Boolean conjunctive queries, and sjfBCQ for the class of self-join-free Boolean conjunctive queries. If q is a sjfBCQ query with an R-atom, then, by an abuse of notation, we write R to mean the R-atom of q.

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**.

PROBLEM:	$\mathsf{CERTAINTY}(q)$
INPUT:	uncertain database **db**
QUESTION:	Does every repair of **db** satisfy q?

Notice that the Boolean query q is not part of the input. Hence, every Boolean query q gives rise to a new problem. Since the input to $\mathsf{CERTAINTY}(q)$ is an uncertain database, we consider the *data complexity* of the problem. The extension to non-Boolean queries will be discussed in Section 3.

3. ATTACK GRAPHS

The construct of *attack graph* is the main tool for solving the complexity classification task of $\mathsf{CERTAINTY}(q)$. Attack graphs were first introduced in [20] for studying first-order expressibility of $\mathsf{CERTAINTY}(q)$ for acyclic (in the sense of [2]) self-join-free conjunctive queries q.

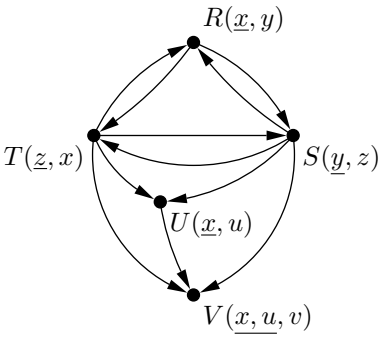


Figure 2: Attack graph of the query in Example 1.

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 $F^{+,q}$ as the following set of variables:

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

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

$$F_0 \overset{z_1}{\frown} F_1 \overset{z_2}{\frown} F_2 \ldots \overset{z_n}{\frown} F_n \tag{1}$$

where

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

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 $F \overset{q}{\rightsquigarrow} G$.

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 1. Let $q = \{R(\underline{x}, y),\ S(\underline{y}, z),\ T(\underline{z}, x),\ U(\underline{x}, u),\ V(\underline{x, u}, v)\}$. We have $R^{+,q} = \{x, u, v\}$. 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 Fig. 2. All attacks are weak.

The above definition of an attack graph is purely syntactic. Semantically, an attack from an R-atom to an S-atom means that there exists an uncertain database **db** such that every repair of **db** satisfies q, and such that two key-equal R-facts join exclusively with two S-facts that are not key-equal. For the query of Example 1, such a database could be $\mathbf{db} = \{R(\underline{1}, a),\ R(\underline{1}, b),\ S(\underline{a}, \alpha),\ S(\underline{b}, \beta), \ldots\}$, in which the two R-facts are key-equal, $R(\underline{1}, a)$ joins exclusively with $S(\underline{a}, \alpha)$, and $R(\underline{1}, b)$ joins exclusively with $S(\underline{b}, \beta)$, and the two S-facts are not key-equal. Therefore, the attack graph of Fig. 2 contains a directed edge from the R-atom to the S-atom.

Equipped with the notion of attack graph, we can now present our result for the complexity classification task of

CERTAINTY(q). **FO** is the descriptive complexity class of decision problems expressible in first-order logic.

THEOREM 1. *For every $q \in$ sjfBCQ,*

1. *if the attack graph of q is acyclic, then* CERTAINTY(q) *is in* **FO**;
2. *if the attack graph of q is cyclic but contains no strong cycle, then* CERTAINTY(q) *is in* **P** *and is* **L**-*hard; and*
3. *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP**-*complete.*

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

Before giving some examples, we explain how to deal with non-Boolean queries, which are common in practice. If $q \in$ sjfBCQ and $\vec{x}$ is a sequence of distinct variables of vars(q), then the *rule* $\vec{x} \leftarrow q$ denotes the conjunctive query that is obtained from q by letting the variables of $\vec{x}$ be free variables. Given an uncertain database **db**, the *certain answer* to this rule is the set of tuples $\vec{a}$ (of the same length as $\vec{x}$), such that $q_{[\vec{x} \mapsto \vec{a}]}$ is satisfied by every repair of **db**.

The attack graph of $\vec{x} \leftarrow q$ is the attack graph of $q_{[\vec{x} \mapsto \vec{c}]}$ for some sequence $\vec{c}$ of constants (of the same length as $\vec{x}$). The results that follow are independent of the choice of $\vec{c}$. This is tantamount to saying that free variables can be treated like constants.

It is easy to show that Theorem 1 extends to rules $\vec{x} \leftarrow q$, where q is always assumed to be self-join-free:

1. if the attack graph is acyclic, then there exists a first-order query that computes the certain answer;
2. if the attack graph is cyclic but contains no strong cycle, then there exists a polynomial-time algorithm (but no first-order query) that computes the certain answer; and
3. if the attack graph constains a strong cycle, then, unless **P** = **NP**, there exists no polynomial-time algorithm that computes the certain answer.

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

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

Treating the free variable d as a constant, we obtain $\mathsf{E}(\underline{m}, n, c_1, d)^{+,q_1} = \{m, b, c_2\}$ and $\mathsf{D}(\underline{d}, b, c_2, m)^{+,q_1} = \{\}$. 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 computed in SQL.

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 $\mathsf{E}(\underline{m}, n, c_1, d)^{+,q_2} = \{m\}$ 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 $\mathsf{E}(\underline{e}, n, c, d)^{+,q_3} = \{e\}$ 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 **P** = **NP**).

Before providing more insights into the proof of Theorem 1, one more definition is needed. 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 1. Let $q \in$ sjfBCQ. Let R be a relation name with signature $[1, 1]$ such that R does not occur in q. For $F \in q$ and $z \in$ 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 2. Clearly, if $F_0 \overset{z_1}{\frown} F_1 \ldots \overset{z_n}{\frown} F_n$ is a witness for $F_0 \overset{q}{\rightsquigarrow} F_n$, then $F_0 \overset{q}{\rightsquigarrow} z_i$ for every $i \in \{1, \ldots, n\}$. Notice also that if $q = \{R(\underline{x}, y)\}$, then the attack graph of q contains no edge, yet $R \overset{q}{\rightsquigarrow} y$.

4. FIRST-ORDER EXPRESSIBILITY

In this section, we elaborate on the first item in the statement of Theorem 1, 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 2. *For every $q \in$ sjfBCQ,* CERTAINTY(q) *is in* **FO** *if and only if the attack graph of q is acyclic.*

To prove the only-if direction, the first step is to show that for $q_0 = \{R_0(\underline{x}, y), S_0(\underline{y}, x)\}$, CERTAINTY($q_0$) is **L**-hard. The query q_0 is in some sense the "simplest" query for which consistent query answering is **L**-hard (an earlier result in [21] showed the weaker result that CERTAINTY(q_0) is not in **FO**). Then, it is shown that for every $q \in$ sjfBCQ, if the attack graph of q is cyclic, there exists a first-order reduction from CERTAINTY(q_0) to CERTAINTY(q), which implies that CERTAINTY(q) is **L**-hard (and thus not in **FO**).

For the if-direction, assume that $q \in$ sjfBCQ such that the attack graph of q is acyclic. Assume $q = \{R_1(\underline{\vec{x}_1}, \vec{y}_1), \ldots, R_n(\underline{\vec{x}_n}, \vec{y}_n)\}$, where the atoms are listed in a topological ordering of the attack graph. Then, it can be shown that CERTAINTY(q) is solved by Algorithm 1, in which $\sim$ denotes "is key-equal to."

Input : Uncertain database **db**
Output: Does every repair of **db** satisfy q?

if $\exists s_1 \in R_1 \quad \forall r_1 \in R_1$ *s.t.* $r_1 \sim s_1$*:*
  $\exists s_2 \in R_2 \quad \forall r_2 \in R_2$ *s.t.* $r_2 \sim s_2$*:*
    $\ddots$
      $\exists s_n \in R_n \quad \forall r_n \in R_n$ *s.t.* $r_n \sim s_n$*:*
        the tuples $r_1, r_2, \ldots, r_n$ together satisfy q
then
  return "yes"
else
  return "no"

Algorithm 1: Algorithm for CERTAINTY(q) for an sjfBCQ query q whose (acyclic) attack graph has topological ordering $R_1, R_2, \ldots, R_n$.

In the **if**-condition of Algorithm 1, every existential quantifier $\exists s_i$ selects an R_i-block, and the subsequent universal quantifier $\forall r_i$ lets r_i range over all facts of that block. At the end, all facts r_i together must satisfy the query. It is easy to encode Algorithm 1 in first-order logic (or in SQL). This implies that if the attack graph of $q \in \mathsf{sjfBCQ}$ is acyclic, we can effectively construct a first-order definition of $\mathsf{CERTAINTY}(q)$. Such a first-order definition is commonly called a *consistent first-order rewriting* for q.

5. POLYNOMIAL-TIME TRACTABILITY

In this section, we outline a polynomial-time algorithm for $\mathsf{CERTAINTY}(q)$ in case that the attack graph of q contains no strong cycles (stated in the second item of Theorem 1). We refer the reader to [13, 14] for the proof of the **coNP**-hard lower bound in case that the attack graph contains a strong cycle.

THEOREM 3. *For every $q \in \mathsf{sjfBCQ}$, if the attack graph of q contains no strong cycle, then $\mathsf{CERTAINTY}(q)$ is in* **P**.

Theorem 3 is proved roughly as follows by syntactic induction. Let $q \in \mathsf{sjfBCQ}$ such that the attack graph of q contains no strong cycle. If the attack graph of q contains an R_1-atom without incoming attacks, then this atom can be processed like the R_1-atom in the first line of Algorithm 1. The more difficult case is if all atoms have incoming attacks in the attack graph of q. In this case, $\mathsf{CERTAINTY}(q)$ can be reduced in polynomial time to some problem $\mathsf{CERTAINTY}(q')$ which is in polynomial time by induction hypothesis. The query q' is obtained from q by a technique called *"dissolution of Markov cycles."*

The proof of Theorem 3 is technically involved, and thus we will only sketch the main ideas. The first important idea is an extension of the data model that allows some syntactic simplifications, which we explain in Section 5.1. The central idea is the notion of *Markov cycle*, which we present in Section 5.2. The dissolution of Markov cycles is illustrated in Section 5.3.

5.1 Relations Known to Be Consistent

We extend the data model by distinguishing between two kinds of relation names: those that can be inconsistent, and those that cannot.

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.

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, as before, 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. However, having relation names with mode c is convenient but not fundamental, since we can simulate relation names with mode c by using relation names with mode i. Indeed, consider any $q \in \mathsf{sjfBCQ}$ and let $R^c(\underline{\vec{x}}, \vec{y})$ be an atom in q. Construct a new query $q' = (q \setminus \{R^c(\underline{\vec{x}}, \vec{y})\}) \cup \{R_1(\underline{\vec{x}}, \vec{y}), R_2(\underline{\vec{x}}, \vec{y})\}$, where R_1, R_2 are two new relation names with mode i and the same signature as R. 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 slight change:

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

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

Our polynomial-time algorithm relies on the notion of saturated query, defined next, which we admit to be technical and not intuitive. Lemma 1 states that $\mathsf{CERTAINTY}(q)$ can be polynomially reduced to $\mathsf{CERTAINTY}(q')$, where q' is saturated and syntactically simplified.

Definition 2. A query $q \in \mathsf{sjfBCQ}$ is said to be *saturated* if whenever $x, z \in \mathsf{vars}(q)$ such that $\mathcal{K}(q) \models x \to z$ and $\mathcal{K}([\![q]\!]) \not\models x \to z$, then there exists an atom $F \in q$ with $\mathcal{K}(q) \models x \to \mathsf{key}(F)$ such that $F \overset{q}{\rightsquigarrow} x$ or $F \overset{q}{\rightsquigarrow} z$.

LEMMA 1. *For each $q \in \mathsf{sjfBCQ}$, there exists a polynomial-time many-one reduction from the problem $\mathsf{CERTAINTY}(q)$ 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' has 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 the attack graph of q contains no strong cycle, then the attack graph of q' contains no strong cycle either.*

5.2 Dissolving Markov Cycles

Our polynomial-time algorithm relies on a new tool called *Markov graph*, which is defined next.

Definition 3. 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

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

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 $x \xrightarrow{\mathsf{M}} y$, if $x \neq y$ and $\mathcal{K}(\mathsf{C}_q(x) \cup [\![q]\!]) \models x \to y$.

The use of the term Markov refers to the intuition that along a path in the Markov graph, each variable functionally determines the next variable on the path, independently of preceding variables. A *Markov cycle* refers to a (directed) cycle in the Markov graph.

Definition 4. A Markov cycle $\mathcal{C}$ is said to be *premier* if there exists a variable $x \in \mathsf{vars}(q)$ such that

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

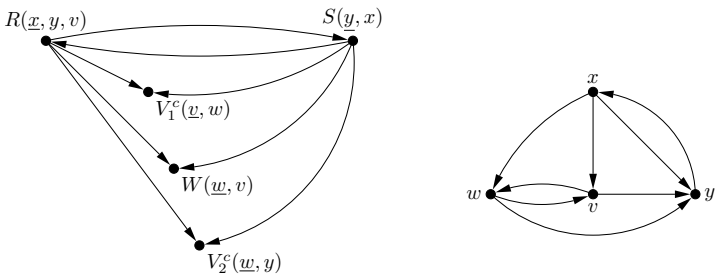


Figure 3: Attack graph (left) and Markov graph (right) of query $\{R(\underline{x}, y, v)$, $S(\underline{y}, x)$, $V_1^c(\underline{v}, w)$, $W(\underline{w}, v)$ $V_2^c(\underline{w}, y)\}$.

Example 3. 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 $\mathsf{C}_q(x) = \{R(\underline{x}, v, y)\}$. Since $\mathcal{K}(\mathsf{C}_q(x) \cup [\![q]\!]) \models x \to \{y, v, w\}$, the Markov graph of q contains directed edges from x to each of y, v, and w.

We have $\mathsf{C}_q(v) = \emptyset$. Since $\mathcal{K}(\mathsf{C}_q(v) \cup [\![q]\!]) \models v \to \{y, w\}$, the Markov graph of q contains directed edges from v to both y and w. The complete Markov graph of q is shown in Fig. 3 (right).

The attack graph of q is shown in Fig. 3 (left). The two 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 Markov cycle containing x or y must be premier. Further, the Markov cycle $\langle v, w, v\rangle$ is also premier, because $\mathcal{K}(q) \models v \to x$ and the Markov graph contains a directed path from x to v.

Let q be like in Definition 3 and assume that the Markov graph of q contains an elementary directed cycle $\mathcal{C}$. Our key result (Lemma 2) 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 $\mathcal{C}$, a notion that is defined next.

Definition 5. 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

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

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

Example 4. Let q be the query of Fig. 3. Let $\mathcal{C}$ be the cycle $\langle x, w, y, x\rangle$ in the Markov graph of q. Using the notation of Definition 5, we have

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

Hence, $\mathsf{dissolve}(\mathcal{C}, q) = \{V_1^c(\underline{v}, w)$, $V_2^c(\underline{w}, y)$, $T(\underline{u}, x, w, y, v)$, $U_1^c(\underline{x}, u)$, $U_2^c(\underline{w}, u)$, $U_3^c(\underline{y}, u)\}$.

LEMMA 2. *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 $\mathsf{CERTAINTY}(q^*)$.*

We will illustrate the reduction of Lemma 2 in Section 5.3. In order to show that dissolving Markov cycles leads to a polynomial-time algorithm, two more results are needed:

- We need to show that the "dissolution" of Markov cycles can be done while keeping the attack graph free of strong cycles (Lemma 3). Surprisingly, this turns out to be true only for Markov cycles that are premier.
- We need to show the existence of premier Markov cycles that can be "dissolved" (Lemma 4).

LEMMA 3. *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 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 4. *Let $q \in \mathsf{sjfBCQ}$ 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 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 is needed in the proof of Theorem 3 which runs by induction on the number of atoms with mode i.

5.3 The Reduction of Lemma 2

We will use an example to illustrate the main ideas behind the reduction of Lemma 2. Let $q \in \mathsf{sjfBCQ}$, and assume 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 $\mathbf{db}$ be an uncertain database. Let $\mathbf{db}_0$ be the subset of $\mathbf{db}$ containing all R-facts, S-facts, and V-facts of $\mathbf{db}$.

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	$\underline{x}$	y
	1	a
	2	b
	2	c
	3	d
	3	e
	4	e
	4	f

S	$\underline{y}$	z
	a	α
	a	κ
	b	β
	c	γ
	d	δ
	e	ϵ
	e	δ
	f	ϕ

V	$\underline{z}$	x	
	α	1	$\}\ \mathbf{db}_{01}$
	κ	1	
	β	2	$\}\ \mathbf{db}_{02}$
	γ	2	
	δ	3	$\}\ \mathbf{db}_{03}$
	ϵ	3	
	δ	4	
	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, both satisfying q_0. For every repair $\mathbf{r}$ of $\mathbf{db}$, we have 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, both satisfying q_0. For every repair $\mathbf{r}$ of $\mathbf{db}$, we have 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 $\mathbf{db}$ satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{db}_{03}$ satisfies q. That is, $\mathbf{db}_{03}$ can henceforth be ignored.

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 table U_x shown below 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	$\underline{x}$	u
	1	01
	2	02

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

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

Let $\mathbf{db}'$ be the database that extends $\mathbf{db}$ with all the facts shown in the tables T, U_x, U_y, and U_z. 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^*.

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

6. RELATED WORK

Theoretical developments. Consistent query answering (CQA) goes back to the seminal work by Arenas, Bertossi, and Chomicki [1]. Fuxman and Miller [8] 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 queries. The term CERTAINTY(q) was coined in [20]. A recent and comprehensive survey on the problem CERTAINTY(q) is [23].

In the past decade, a variety of techniques have been used in the complexity classification task of CERTAINTY(q) for sjfBCQ queries q. In their pioneering work, Fuxman and Miller [8] introduced the notion of *join graph* (not to be confused with the classical notion of join tree). Later, Wijsen [20] introduced the notion of *attack graph*. Kolaitis and Pema [10] applied Minty's algorithm [18]. Koutris and Suciu [12] introduced the notion of *query graph* and the distinction between consistent and possibly inconsistent relations.

Little is known about CERTAINTY(q) beyond self-join-free conjunctive queries. An interesting recent result by Fontaine [6] 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 UCQ query q, CERTAINTY(q) is either in **P** or **coNP**-complete. Fontaine showed that this conjecture implies Bulatov's dichotomy theorem for conservative CSP [4], the proof of which is highly involved (the full paper contains 66 pages).

The counting variant of CERTAINTY(q), which has been denoted ♯CERTAINTY(q), asks to determine the exact number of repairs that satisfy some Boolean query q. In [16], it was shown that for every sjfBCQ query q, the counting problem ♯CERTAINTY(q) is either in **FP** or ♯**P**-complete. For conjunctive queries q with self-joins, the complexity of ♯CERTAINTY(q) has been established under the restriction that all atoms are simple-key [17].

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 [9] and Binary Integer Programming (BIP) [11]. In these formalisms, it is relatively easy to express an exponential-time algorithm for CERTAINTY(q). The drawback is that the efficiency of these algorithms is likely to be far from optimal in 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 [3, 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 [11] and ConQuer [7], the authors of the former system found that BIP never outperformed first-order query rewriting.

The Hippo system [5] 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. Note, however, that such queries have an empty (and hence acyclic) attack graph, in which case our results imply that the consistent answer

can also be computed by a single SQL query.

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 acyclic attack graphs, meaning that the certain answer can be computed by a single SQL query. 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 Fig. 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.

7. CONCLUSION

This paper settles a long-standing open question in consistent query answering, by providing a solution to the complexity classification task of CERTAINTY(q) for sjfBCQ queries q. We show that it is decidable, given q, whether the problem CERTAINTY(q) is in **FO**, in $\mathbf{P} \setminus \mathbf{FO}$, or **coNP**-complete. Moreover, if the problem is in **FO** or in **P**, then we can effectively construct a first-order query or a polynomial-time algorithm for solving it.

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.

Acknowledgments

This work is supported in part by the NSF through NSF grant NSF IIS-1115188.

8. REFERENCES

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

[2] C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[3] L. E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[4] A. A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24, 2011.

[5] J. Chomicki, J. Marcinkowski, and S. Staworko. Hippo: A system for computing consistent answers to a class of SQL queries. In E. Bertino et al., editors, *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 841–844. Springer, 2004.

[6] G. Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? In *LICS*, pages 550–559. IEEE Computer Society, 2013.

[7] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient management of inconsistent databases. In F. Özcan, editor, *SIGMOD Conference*, pages 155–166. ACM, 2005.

[8] A. Fuxman and R. J. Miller. First-order query rewriting for inconsistent databases. In T. Eiter and L. Libkin, editors, *ICDT*, volume 3363 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2005.

[9] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.

[10] P. G. Kolaitis and E. Pema. A dichotomy in the complexity of consistent query answering for queries with two atoms. *Inf. Process. Lett.*, 112(3):77–85, 2012.

[11] P. G. Kolaitis, E. Pema, and W. Tan. Efficient querying of inconsistent databases with binary integer programming. *PVLDB*, 6(6):397–408, 2013.

[12] P. Koutris and D. Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In Schweikardt et al. [19], pages 165–176.

[13] P. Koutris and J. Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In T. Milo and D. Calvanese, editors, *PODS*, pages 17–29. ACM, 2015.

[14] P. Koutris and J. Wijsen. A trichotomy in the data complexity of certain query answering for conjunctive queries. *CoRR*, abs/1501.07864, 2015.

[15] L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

[16] D. Maslowski and J. Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.

[17] D. Maslowski and J. Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In Schweikardt et al. [19], pages 155–164.

[18] G. J. Minty. On maximal independent sets of vertices in claw-free graphs. *J. Comb. Theory, Ser. B*, 28(3):284–304, 1980.

[19] N. Schweikardt, V. Christophides, and V. Leroy, editors. *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014*. OpenProceedings.org, 2014.

[20] J. Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In J. Paredaens and D. V. Gucht, editors, *PODS*, pages 179–190. ACM, 2010.

[21] J. Wijsen. A remark on the complexity of consistent conjunctive query answering under primary key violations. *Inf. Process. Lett.*, 110(21):950–955, 2010.

[22] J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9, 2012.

[23] J. Wijsen. A survey of the data complexity of consistent query answering under key constraints. In C. Beierle and C. Meghini, editors, *FoIKS*, volume 8367 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 2014.