On Counting Database Repairs

Dany Maslowski
Université de Mons (UMONS)
20 Place du Parc
7000 Mons, Belgium
dany.maslowski@umons.ac.be

Jef Wijsen
Université de Mons (UMONS)
20 Place du Parc
7000 Mons, Belgium
jef.wijsen@umons.ac.be

ABSTRACT
An uncertain database $\text{db}$ is defined as a database in which
distinct tuples of the same relation can agree on their primary key. A repair is obtained by selecting a maximal num-
ber of tuples without ever selecting two distinct tuples of the
same relation that agree on their primary key. Obviously,
the number of possible repairs can be exponential in the size
of the database.

Given a Boolean query $q$, certain (or consistent) query an-
swering concerns the problem to decide whether $q$ evaluates
to true on every repair. In this article, we study a counting
variant of consistent query answering. For a fixed Boolean
query $q$, we define $\exists \text{CERTAINTY}(q)$ as the following count-
ing problem: Given an uncertain database $\text{db}$, how many repairs of $\text{db}$ satisfy $q$? Our main result is that conjunc-
tive queries $q$ without self-join exhibit a complexity dichotomy:
$\exists \text{CERTAINTY}(q)$ is in $\mathcal{P}$ or $\exists \mathcal{P}$-complete.

Categories and Subject Descriptors
H.2.3 [Database Management]: Languages—query lan-
guages; H.2.4 [Database Management]: Systems—rela-
tional databases

General Terms
Theory, Algorithms

Keywords
Conjunctive queries, consistent query answering, primary
key, probabilistic databases

1. INTRODUCTION
Uncertainty arises in many database applications. It can be
modeled in a clean and elegant way by relations that
violate their primary key constraint. We use the term un-
certain database for databases that allow primary key viola-
tions. Such uncertainty is not necessarily a bad thing. In
planning databases, for example, primary key violations can
represent different alternatives. In the following conference
planning database, where primary keys are underlined, the
exact town of LID 2016 is still uncertain (it can be Mons,
Gent, or Paris).

<table>
<thead>
<tr>
<th>R</th>
<th>Conf</th>
<th>Year</th>
<th>Town</th>
</tr>
</thead>
<tbody>
<tr>
<td>LID</td>
<td>2016</td>
<td>Mons</td>
<td></td>
</tr>
<tr>
<td>LID</td>
<td>2016</td>
<td>Gent</td>
<td></td>
</tr>
<tr>
<td>LID</td>
<td>2016</td>
<td>Paris</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T</th>
<th>Country</th>
<th>Cont</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belgium</td>
<td>Europe</td>
<td></td>
</tr>
<tr>
<td>France</td>
<td>Europe</td>
<td></td>
</tr>
<tr>
<td>Tunisia</td>
<td>Africa</td>
<td></td>
</tr>
</tbody>
</table>

Uncertainty also arises as an inconvenient but inescapable
consequence of data integration and data exchange. The rela-
tion $S$ in the preceding example combines data from two
different sources, each providing a different country for the
city of Tunis.

Uncertainty by primary key violations gives rise to (expo-
entially many) “possible worlds,” which we will call repairs:
every repair is obtained by selecting a maximal number of
tuples from each relation without ever selecting two distinct
tuples that agree on their primary key.

In previous works [14, 15, 16], we have studied the deci-
sion problem $\exists \text{CERTAINTY}(q)$ for a fixed Boolean conjunc-
tive query $q$. This problem takes as input an uncertain database
and asks whether $q$ evaluates to true on every repair. For
example, our example database has six repairs (there are
three choices for the city of LID 2016, and two choices for
the country of Tunis), each satisfying the Boolean conjunc-
tive query

$$\exists x \exists y \exists z (R(\text{LID}', x, y) \land S(y, z) \land T(z, 'Europe'))$$

stating that LID will be organized in Europe in some year.
On the other hand, not all repairs satisfy the query

$$q_1 = \exists x \exists y \exists z (R(\text{LID}', x, y) \land S(y, 'Belgium'))$$

stating that LID will be organized in Belgium in some year.
A natural follow-up question here is: “How many repairs
satisfy $q_1$?”. We may be interested to know, for example,
whether a query is true in more than half of the repairs.

The query $q_1$ is satisfied by four repairs (out of six) of our
database.

This leads to a counting variant of $\exists \text{CERTAINTY}(q)$, de-
noted by $\exists \mathcal{P} \text{CERTAINTY}(q)$, which is the following problem:
Given an uncertain database $\text{db}$, determine the number...
of repairs that satisfy $q$. Note that throughout this article, all complexity results concern data complexity, that is, database schemas and queries are fixed, and the complexity is in terms of varying database size. The main result of this article is the following dichotomy theorem (self-join-free means that no relation name occurs more than once in $q$):

**Theorem 1.** For every Boolean self-join-free conjunctive query $q$, at least one of the following holds:

- $\exists\text{CERTAINTY}(q)$ is in $P$; or
- $\exists\text{CERTAINTY}(q)$ is $\exists P$-complete under polynomial-time Turing reductions.

Recall that the class $\exists P$ contains the counting variant of problems in $NP$. By Toda's theorem [13], every problem in the polynomial-time hierarchy can be solved in polynomial time given an oracle that solves a $\exists P$-complete problem. Thus, $\exists P$-completeness suggests a higher level of intractability than $NP$-completeness, insofar decision problems and counting problems can be compared.

It is known that the decision problem $\text{CERTAINTY}(q)$ is coNP-complete for the conjunctive query

$$q = \exists x \exists y \exists z (R(x,z) \land S(y,z)).$$

An intriguing open question is whether a theory allows to attach probabilities to tuples, as illustrated next.

**Theorem 2.** There exists a Boolean self-join-free conjunctive query $q$ such that:

- $\text{CERTAINTY}(q)$ is first-order expressible (and hence in the low complexity class AC$^0$); and
- $\exists\text{CERTAINTY}(q)$ is $\exists P$-complete under polynomial-time Turing reductions.

Dalvi et al. recently obtained a dichotomy like that of Theorem 1 for probabilistic databases [7, 5]. Our proof techniques are inspired by that work and we use, to the extent it is possible, terminology and notation that is reminiscent of this earlier work. Nevertheless, the probabilistic database model differs from our uncertain database model in several fundamental respects, so that complexity results in either model do not generally carry over to the other.

Probabilistic databases, unlike our uncertain databases, attach probabilities to tuples, as illustrated next.

<table>
<thead>
<tr>
<th>SP</th>
<th>Town</th>
<th>Country</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tunis</td>
<td>France</td>
<td>0.16</td>
<td></td>
</tr>
<tr>
<td>Tunis</td>
<td>Tunisia</td>
<td>0.60</td>
<td></td>
</tr>
</tbody>
</table>

This probabilistic relation models three possible worlds, each with its own probability. The three possible worlds and their probabilities are:

1. the singleton \{SP(‘Tunis’, ‘France’)\} whose probability is 0.16;
2. the singleton \{SP(‘Tunis’, ‘Tunisia’)\} with probability 0.60; and
3. the empty relation with probability $0.24 = 1 - 0.16 - 0.60$.

Our uncertain database model has no explicit probabilities. It contains some implicit probabilities, to the extent that if a relation contains a block of $n$ tuples that agree on the primary key, then every tuple of that block has probability $1/n$ to be chosen in a repair. As a consequence, deleting or inserting tuples in a block will affect the probability of the other tuples. For example, if we delete the tuple $S(‘Tunis’, ‘France’)\} from the relation 3 shown on the previous page, then the other tuple $S(‘Tunis’, ‘Tunisia’)\} becomes automatically certain. In the probabilistic data model, on the other hand, deleting $SP(‘Tunis’, ‘France’)\} \setminus 0.16)$ will not change the probability of the remaining possible world \{SP(‘Tunis’, ‘Tunisia’)\} (it stays at 0.60), while the probability of the empty possible world will increase by 0.16. To conclude, uncertain and probabilistic databases both capture uncertainty, but they differ in some fundamental respects.

The remainder of this article is organized as follows. Section 2 formally defines our data model and the problem of interest. We focus on Boolean conjunctive queries in which each relation name is used at most once. Section 3 discusses interest. We focus on Boolean conjunctive queries in which each relation name is used at most once. Section 3 discusses interest.

### 2. PRELIMINARIES

#### 2.1 Basic Notions

**Data Model.**

We define $N = \{0, 1, 2, \ldots \}$. 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 $\{1, 2, \ldots , k\}$. Elements of the primary key are called primary-key positions, while $k + 1, k + 2, \ldots , n$ are non-primary-key positions. For all positive integers $n, k$ such that $1 \leq k \leq n$, we assume denumerably many relation names with signature $[n, k]$.

We assume a denumerable set dom of constants, disjoint from a denumerable set var of variables. If $R$ is a relation name of signature $[n, k]$, and $s_1, \ldots , s_n$ are variables or constants, then $R(s_1, \ldots , s_k, s_{k+1}, \ldots , s_n)$ is an $R$-goal (or simply goal if $R$ is understood). Notice that primary key positions are underlined. An $R$-fact (or simply fact) is a goal in which no variable occurs. Two $R$-facts $g$ and $h$ are key-equal if they agree on all primary-key positions. Every fact is key-equal to itself.

If the signature of relation name $R$ is $[n, n]$ for some $n$, then $R$ is called all-key. If $R$ is all-key, then every $R$-goal is also called all-key.

A database $db$ is a finite set of facts. Such database may violate primary keys, and so capture uncertainty. Intuitively, the term database can be understood as “uncertain database.” Given a database $db$, we write $\text{dom}(db)$ for the set of constants that occur in $db$.

---

1. This notion is unrelated to the notion of safety that guarantees domain independence in relational calculus [1, page 75].
A database $\text{db}$ is called consistent if it contains no two distinct, key-equal facts. A repair of a database $\text{db}$ is a maximal subset of $\text{db}$ that is consistent. If $q$ is a fact of $\text{db}$, then $\text{block}(q, \text{db})$ is the subset of $\text{db}$ containing each fact that is key-equal to $q$. The cardinality of this set is denoted by $\sharp\text{block}(q, \text{db})$. The sets $\text{block}(g, \text{db})$ are also called blocks.

Intuitively, repairs are obtained by choosing exactly one fact from each block.

**Queries.**

A Boolean conjunctive query $q$ is a finite set of goals. A Boolean conjunctive query $q = \{g_1, g_2, \ldots, g_n\}$ represents the formula $\exists x_1 \ldots \exists x_m (g_1 \land g_2 \land \ldots \land g_n)$, where $x_1, \ldots, x_m$ are all variables occurring in $q$. Such query is self-join-free if it contains no two distinct goals with the same relation name. Thus, every relation name occurs at most once in a self-join-free query. The class of self-join-free Boolean conjunctive queries is denoted by $\text{SJCQ}$.

Let $V$ be a finite set of variables. A valuation over $V$ is a mapping $\theta : \text{vars} \cup \text{dom} \rightarrow \text{vars} \cup \text{dom}$ such that for every $x \in V$, $\theta(x) \in \text{dom}$, and for every $s \notin V$, $\theta(s) = s$. Valuations extend to goals and queries in the straightforward way.

If $\vec{s}$ is a sequence of variables and constants, then $\text{Vars}(\vec{s})$ denotes the set of variables that occur in $\vec{s}$. If $q$ is a goal, then $\text{Vars}(q)$ denotes the set of variables that occur in $q$, and $\text{KVars}(q)$ denotes the subset of $\text{Vars}(q)$ containing each variable that occurs at a primary-key position. If $q$ is a query, then $\text{Vars}(q)$ denotes the set of variables that occur in $q$.

If $q$ is a conjunctive query, $x \in \text{Vars}(q)$, and $a \in \text{dom}$, then $q_{x \rightarrow a}$ is the query obtained from $q$ by replacing all occurrences of $x$ with $a$. Likewise, if $q$ is a variable, then $q_{x \rightarrow y}$ is the query obtained from $q$ by replacing all occurrences of $x$ with $y$.

A database $\text{db}$ is said to satisfy Boolean conjunctive query $q$, denoted $\text{db} \models q$, if there exists a valuation $\theta$ over $\text{Vars}(q)$ such that $\theta(q) \subseteq \text{db}$.

**Counting Repairs.**

For some fixed $q \in \text{SJCQ}$, $\text{CERTAINTY}(q)$ is the following problem: Given a database $\text{db}$, decide whether every repair of $\text{db}$ satisfies $q$. The counting variant, denoted $\sharp\text{CERTAINTY}(q)$, is the following problem: Given a database $\text{db}$, compute the number of repairs of $\text{db}$ that satisfy $q$.

Let $\text{db}$ be a database and $q$ a Boolean query. We write $\text{rset}(\text{db})$ for the set of repairs of $\text{db}$, and $\text{rset}(\text{db}, q)$ for the subset of $\text{rset}(\text{db})$ containing each repair that satisfies $q$. The cardinality of these sets are denoted by $\sharp\text{rset}(\text{db})$ and $\sharp\text{rset}(\text{db}, q)$, respectively. We define $\text{frac}(\text{db}, q) = \frac{|\text{rset}(\text{db}, q)|}{|\text{rset}(\text{db})|}$, the fraction of repairs satisfying $q$. Thus, for a fixed query $q \in \text{SJCQ}$, $\sharp\text{CERTAINTY}(q)$ is the problem that takes as input a database $\text{db}$ and asks to determine $\sharp\text{rset}(\text{db}, q)$.

**3. RELATED WORK**

The results in this article can be viewed from two angles: consistent query answering and probabilistic databases.

Consistent query answering goes back to [3] and deals with the problem of computing answers to queries on databases that violate integrity constraints. Repairs are defined as consistent databases that are maximally similar, according to some similarity measure, to the original database. A query answer is then certain (or consistent) if it is an answer to the query on every repair. Our framework can be categorized as consistent query answering to Boolean queries on uncertain relations, where the only constraints are primary keys. In [4], it is illustrated that an uncertain relation with $2^n$ tuples can have $2^n$ repairs. Fuxman and Miller [8] were the first to focus on computing consistent answers to conjunctive queries on uncertain relations. Their results have been extended and improved in recent works [14, 15, 16]. These works on primary key violations have focused on the question whether a query is true in every repair; in the current article, we ask to determine the exact number of repairs in which the query is true.

Counting the fraction of repairs that satisfy a query is also studied by Greco et al. in [9]. The constraints in that work are functional dependencies, and the repairs are obtained by updates. Greco et al. present an approach for computing approximate probabilistic answers in polynomial time. We, on the other hand, characterize queries for which exact fractions can be obtained in polynomial time.

Recently, there has been a growing interest in probabilistic data models [2, 10, 6]. From the probabilistic database angle, our uncertain databases are a restricted case of block-independent-disjoint probabilistic databases [6, 5], where the restriction is that facts within a block follow a uniform distribution. Every repair is a possible world, and all these worlds have the same probability.

Our proof of Theorem 1 parallels the proof of Theorem 3.6 in [5], which shows a $\text{P}$ versus $\text{P}$-hard dichotomy in the case of probabilistic databases. Nevertheless, our proof needs some extra technicality because our simple data model does not allow all constructions that are possible in the probabilistic data model. For example, a proof by construction in the probabilistic data model may introduce a tuple with probability $2/3$, or a tuple with very small probability, say $1/(m+1)$, where $m$ is the database size. Such construction is impossible (and hence needs a workaround) in our setting; we are limited to probabilities $1, \frac{1}{2}, \frac{1}{3}, \frac{2}{3}$. Moreover, since probability $1/n$ can only be obtained by a block
with \( n \) facts, the smallest possible probability in a database with \( m \) facts is \( 1/m \).

4. TRACTABLE COUNTING

In this section, we define a syntactically restricted class of SJFCQ queries, called safe queries. We show that for every safe SJFCQ query \( q \), the problem \( \sharp CERTAINTY(q) \) is in \( P \). The outline is as follows: Lemmas 2–6 first provide arithmetic expressions for \( \mathrm{frac}(\db, q) \) under particular syntactic conditions on \( q \); these lemmas are then combined in an algorithm that defines the safe queries.

The following lemma implies that we can count in polynomial time the number of repairs that contain a given fact.

**Lemma 2** (SE0a). Let \( q \) be a fact. Let \( q = \{g\} \), an SJFCQ query. Then, for every database \( \db \),

\[
\frac{\db}{q} = \begin{cases} 
0 & \text{if } g \notin \db \\
1/\#\text{block}(g, \db) & \text{if } g \in \db 
\end{cases}
\]

Lemma 3 deals with queries that are either satisfied or falsified by all repairs. An example of such query is \( q_0 = \{R(x, x, y, z)\} \). Trivially, a database \( \db \) satisfies \( q_0 \) if and only if \( \db \) contains an \( R \)-fact of the form \( R(a, b, c, \ldots) \), for some constants \( a, b, c \). Moreover, if \( \db \) contains an atom of this form, then every repair of \( \db \) will contain an atom of this form, and thus satisfy \( q_0 \). Conversely, if \( \db \) contains no atom of this form, then no repair of \( \db \) will satisfy \( q_0 \).

**Definition 1.** Let \( q \) be an SJFCQ query. A variable \( x \in Vars(q) \) is called a liaison variable if \( x \) has at least two occurrences in \( q \).

A variable \( y \in Vars(q) \) is called an orphan variable if \( y \) occurs only once in \( q \) and the only occurrence of \( y \) is at a non-primary-key position.

The complex part of an SJFCQ query \( q \), denoted \( [q] \), contains every goal \( g \in q \) such that some non-primary-key position in \( g \) contains a liaison variable or a constant.

**Example 1.** Let \( q = \{R(x, y), S(y, z), T(y, u, a)\} \). Then, \( y \) is a liaison variable because \( y \) occurs more than once in \( q \). The variables \( u \) and \( z \) are orphan, because they occur only once in \( q \) at a non-primary-key position. The variable \( x \) is neither liaison nor orphan.

The complex part of a query \( q \) can be empty, as is the case for \( q = \{R(x, y), S(y, u, T(y, w))\} \), in which \( R \) is all-key. Lemma 3 implies that if the complex part of an SJFCQ query \( q \) is empty, then \( \sharp CERTAINTY(q) \) is tractable.

**Lemma 3** (SE0b). Let \( q \) be an SJFCQ query. If \( [q] = \emptyset \), then for every database \( \db \),

\[
\frac{\db}{q} = \begin{cases} 
0 & \text{if } \db \not\models q \\
1 & \text{if } \db \models q 
\end{cases}
\]

**Proof.** Straightforward. \( \square \)

Lemma 4 deals with queries \( q \) that can be partitioned into two subqueries, say \( q_1 \) and \( q_2 \), such that \( q_1 \) and \( q_2 \) have no variables in common. The lemma implies that if \( \sharp CERTAINTY(q_1) \) and \( \sharp CERTAINTY(q_2) \) are both tractable, then so is \( \sharp CERTAINTY(q) \).

Figure 2: Tree representation of an execution of IsSafe. Vertices are labeled by queries. Edge labels indicate the rules that, applied on the parent, result in the children. Since all leaf vertices are true, the query at the root node is safe.

**Lemma 4** (SE1). Let \( q, q_1, q_2 \) be SJFCQ queries such that \( q = q_1 \cup q_2 \), \( q_1 \cap q_2 = \emptyset \), and \( Vars(q_1) \cap Vars(q_2) = \emptyset \). Then, for every database \( \db \),

\[
\frac{\db}{q} = \frac{\db}{q_1} \times \frac{\db}{q_2}
\]

**Lemma 5** (SE2). Let \( q \) be an SJFCQ query such that for some variable \( x \), \( x \in \bigcap_{a \in [0]} KVars(g) \). Then, for every database \( \db \),

\[
\frac{\db}{q} = 1 - \left( \prod_{a \in \text{dom}(\db)} \left( 1 - \frac{\db}{q_{x \rightarrow a}} \right) \right)
\]

Lemma 6 treats queries with a goal \( g \) such that all primary-key positions in \( g \) are occupied by constants, and at least one non-primary-key position is occupied by a variable.

**Lemma 6** (SE3). Let \( q \) be an SJFCQ query such that for some goal \( g \in q \), for some variable \( x \), \( KVars(g) = \emptyset \) and \( x \in Vars(g) \). Then, for every database \( \db \),

\[
\frac{\db}{q} = \sum_{a \in \text{dom}(\db)} \frac{\db}{q_{z \rightarrow a}}
\]

Lemmas 2–6 are now bundled in a recursive algorithm, called IsSafe, which takes as input an SJFCQ query \( q \) and returns \( \text{true} \) if for every database \( \db \), \( \frac{\db}{q} \) can be obtained by recursive application of Lemmas 2–6; otherwise IsSafe returns \( \text{false} \). Algorithm IsSafe is shown in Figure 1 (left). To simplify the notation, we write \( A \uplus B \) for the
Algorithm $IsSafe(q)$
Input: SJFCQ query $q$
Output: Boolean in \{true, false\}

1. (* SE0a *)
2. if $q = \{g\}$ with $Vars(g) = \emptyset$
   then return true
3. (* SE0b *)
4. if $[q] = \emptyset$
   then return true
5. (* SE1 *)
6. if $q = q_1 \uplus q_2$ and $q_1 \neq \emptyset \neq q_2$
   and $Vars(q_1) \cap Vars(q_2) = \emptyset$
   then return $(IsSafe(q_1) \land IsSafe(q_2))$
7. (* SE2 *)
8. if $[q] \neq \emptyset$ and $\exists x \forall y \in [q] (x \in KVars(g))$
   then return $IsSafe(q_{x\rightarrow a})$
9. (* a is any fixed constant *)
10. (* SE3 *)
11. if $\exists x \exists y \in q(KVars(g) = \emptyset, x \in Vars(g))$
   then return $IsSafe(q_{x\rightarrow a})$
12. (* Otherwise *)
13. if none of the above
14. then return false

Algorithm $Eval(db, q)$
Input: uncertain database $db$, safe SJFCQ query $q$
Output: $rfrac(db, q)$

1. (* Evaluation SE0a *)
2. if $q = \{g\}$ with $Vars(g) = \emptyset$
   then return 1/block($g, db$)
3. else return 0
4. (* Evaluation SE0b *)
5. if $[q] = \emptyset$
   then return 1
6. else return 0
7. (* Evaluation SE1 *)
8. if $q = q_1 \uplus q_2$
   and $q_1 \neq \emptyset \neq q_2$
   and $Vars(q_1) \cap Vars(q_2) = \emptyset$
   then return $Eval(db, q_1) \times Eval(db, q_2)$
9. (* Evaluation SE2 *)
10. if $[q] \neq \emptyset$ and $\exists x \forall y \in [q] (x \in KVars(g))$
11. then return $1 - \left(\prod_{a \in adom(db)} (1 - Eval(db, q_{x\rightarrow a}))\right)$
12. (* Evaluation SE3 *)
13. if $\exists x \exists y \in q(KVars(g) = \emptyset, x \in Vars(g))$
14. then return $\sum_{a \in adom(db)} Eval(db, q_{x\rightarrow a})$
15. (* Otherwise *)
16. if none of the above
17. then return false

Figure 1: Algorithms $IsSafe$ and $Eval$

![Figure 3: Tree representation of an execution of $IsSafe$. Since some leaf vertex is false, the query at the root node is unsafe.](image)
at most $m$ applications of SE2 or SE3, and at most $|q| - 1$ applications of SE1. Every application of SE2 or SE3 makes $N$ direct recursive calls of Eval, and every application of SE1 makes 2 direct recursive calls, each with an argument of query of strictly smaller size. It follows that Eval($db$, $q$) makes at most $O(|q| \cdot N^m)$ recursive calls in total, which is polynomially bounded since $q$ is assumed fixed. The basis of the recursion either determines $\frac{glock(q, db)}{\text{or evaluates a conjunctive query, which takes only polynomial time.}}$

5. INTRINSICATABILITY RESULTS

In this section, we show that for every unsafe SJFCQ query $q$, it is the case that $\not\exists\text{CERTAINTY}(q)$ is in $\mathbb{P}$-complete. The overall outline is as follows. We present a number of rewrite rules that allow to rewrite a query $q$ into a query $q'$ such that the complexity of $\not\exists\text{CERTAINTY}(q')$ is not higher than the complexity of $\not\exists\text{CERTAINTY}(q)$. Then, we show that every unsafe query $q$ can be rewritten into a query $q'$ such that $\not\exists\text{CERTAINTY}(q')$ is $\mathbb{P}$-hard—hence $\not\exists\text{CERTAINTY}(q)$ is $\mathbb{P}$-hard as well. Finally, it is easy to show that $\not\exists\text{CERTAINTY}(q)$ is in $\mathbb{P}$ for every SJFCQ query $q$.

We briefly recall different types of reductions that can exist between two counting problems $\exists A$ and $\exists B$. For a more detailed overview, see [11, 12].

- A polynomial-time many-one counting reduction from $\exists A$ to $\exists B$ consists of two polynomial-time computable functions: (1) a function $\sigma$ mapping instances $x$ of $\exists A$ to instances $\sigma(x)$ of $\exists B$, and (2) a function $f : N \rightarrow N$, such that for every instance $x$ of $\exists A$, if the natural number $N$ is the answer to $\sigma(x)$, then $f(N)$ is the answer to $x$. Such a counting reduction is called parsimonious if the function $f$ is the identity function.

- There exists a polynomial-time Turing reduction from $\exists A$ to $\exists B$ if there is a polynomial-time algorithm that solves instances of $\exists A$ given an oracle that solves instances of $\exists B$. This is also written as $\exists A \leq_{\text{P}} \exists B$.

Notice that the existence of a many-one counting reduction implies the existence of a Turing reduction, but the inverse may not be true.

5.1 Rewrite Rules

We define ten rewrite rules (R1, ..., R10) that can be applied on SJFCQ queries. These rules are given in Figure 4. The first rule R1, for example, states that we can rewrite a query $q$ into $q_{x\rightarrow a}$ if $x$ occurs at a primary-key position of some goal $g$ of $q$. A concrete example is $\{R(x, y), S(y, x)\} \triangleright \{R(a, y), S(y, a)\}$. The rewrite rule R5 refers to a rewrite relation $\triangleright$ that is defined on atoms.

Importantly, we will show in Lemma 7 that none of the rewrite rules increases complexity, in the sense that if $q \triangleright q'$, then $\not\exists\text{CERTAINTY}(q') \subseteq \mathbb{P}^{\not\exists\text{CERTAINTY}(q)}$.

Thus, if $\not\exists\text{CERTAINTY}(q)$ is in $\mathbb{P}$, then $\not\exists\text{CERTAINTY}(q')$ is also in $\mathbb{P}$. Inversely, if $\not\exists\text{CERTAINTY}(q')$ is $\mathbb{P}$-hard, then $\not\exists\text{CERTAINTY}(q)$ is $\mathbb{P}$-hard.

Definition 3. Let $R_2$ be a relation name of signature $[n, k]$. Let $R_0$ be a relation name of signature $[n + 1, k]$. Let $R_1$ be a relation name of signature $[n + 1, k + 1]$. The rewrite relation $\triangleright$ is defined on goals as follows:

- $R_0(\tilde{g}, \tilde{r}, x, \tilde{u}) \triangleright R_2(\tilde{g}, \tilde{r}, \tilde{u})$ if $x$ has two or more occurrences in $R_0(\tilde{g}, \tilde{r}, x, \tilde{u})$.
- $R_0(\tilde{g}, \tilde{r}, a, \tilde{u}) \triangleright R_2(\tilde{g}, \tilde{r}, \tilde{u})$ if $a \in \text{dom}$.
- $R_1(\tilde{r}, x, \tilde{s}, \tilde{t}) \triangleright R_2(\tilde{r}, \tilde{s}, \tilde{t})$ if $x$ is a variable that occurs in $\tilde{r}$ or $\tilde{s}$.
- $R_1(\tilde{r}, a, \tilde{s}, \tilde{t}) \triangleright R_2(\tilde{r}, \tilde{s}, \tilde{t})$ if $a \in \text{dom}$ and $\tilde{r}$ or $\tilde{s}$ are not both empty.

The rewrite relation $\triangleright$ is defined on SJFCQ queries by the rules R1-R10 in Figure 4. We write $q \triangleright q'$ if for some $n \geq 1$, there exist queries $q_1, \ldots, q_n$ such that $q \triangleright q_1 \triangleright \ldots \triangleright q_n = q'$. We write $q \not\triangleright q'$ if it is not the case that $q \triangleright q'$.

Example 2. Take query $q_0 = \{U(\tilde{u}, \tilde{w}), R(x, y, z), S(y, y), T(z, z)\}$ shown at the root in Figure 3. We can rewrite this query as follows.

$$q_0 \triangleright q_1 = \{R(x, y, z), S(y, y), T(z, z)\}$$ (by R2)

$$q_1 \triangleright q_2 = \{R(x, y, z), S(y, y)\}$$ (by R4)

$$q_2 \triangleright q_3 = \{R'(x, y), S(y, y)\}$$ (by R8)

$$q_3 \triangleright q_4 = \{R'(x, y), S(y, a)\}$$ (by R10)

$$q_4 \triangleright q_5 = \{R'(a, y), S(y, a)\}$$ (by R1)

$$q_5 \triangleright q_6 = \{R'(a, a), S(a, a)\}$$ (by R11)

$$q_6 \triangleright q_7 = \{R'(a, a)\}$$ (by R2)

$$q_7 \triangleright q_8 = \{R'(a)\}$$ (by R5)

$$q_8 \triangleright q_9 = \emptyset$$ (by R4)

Notice that relation names $R$, $R'$, and $R''$ have distinct signatures.

Lemma 7. Let $q$ and $q'$ be SJFCQ queries. If $q \triangleright q'$, then there exists a polynomial-time many-one counting reduction from $\not\exists\text{CERTAINTY}(q')$ to $\not\exists\text{CERTAINTY}(q)$.

The following lemma implies that $\triangleright$ defines a strict partial order with no infinite chains $q_0 \triangleright q_1 \triangleright q_2 \ldots$.

Lemma 8. Let $q$ be an SJFCQ query, and let $Q = \{q' \in \text{SJFCQ} \mid q \not\triangleright q'\}$. Then,

- $q \not\triangleright q$ (that is, $\triangleright$ is irreflexive);
- the set $Q$ is finite; and
- if $q$ is unsafe, then $Q$ contains a safe query.

5.2 Final Queries

Take the rewriting sequence in Example 2. The initial query $q_0$ is unsafe (cf. Figure 3). The final query $q_9$ is obviously safe. Thus, at some point, we have rewritten an unsafe query into a safe one. This happened in the step $q_8 \triangleright q_9$ (by R1), because $q_8$ is unsafe and $q_9$ is safe. A closer examination reveals that the query $q_1$ has the following special property: the query is unsafe, and any application of any rewrite rule yields a safe query. For example, we also have $q_8 \triangleright \{R'(x, y), S(y, a)\}$ (by R7), and the latter query is safe. Queries that satisfy this property are called final.

Definition 4. An SJFCQ query $q$ is called final if

- $q$ is unsafe; and
- for every SJFCQ query $q'$, if $q \not\triangleright q'$, then $q'$ is safe.
Fix constant $a \in \text{dom}$.

\[
\begin{align*}
R1. \quad q & \quad \triangleright \quad q_{x \to a} & \quad \text{if } \exists y \in q(x \in \text{KVars}(g)) \\
R2. \quad q_1 \uplus q_2 & \quad \triangleright \quad q_1 & \quad \text{if } q_1 \neq \emptyset \neq q_2 \text{ and } \text{Vars}(q_1) \cap \text{Vars}(q_2) = \emptyset \\
R3. \quad q & \quad \triangleright \quad q_{y \to x} & \quad \text{if } \exists y \in q(x, y \in \text{KVars}(g)) \\
R4. \quad q \uplus \{g\} & \quad \triangleright \quad q & \quad \text{if } \text{Vars}(q) \cap \text{Vars}(g) \subseteq \text{KVars}(g) \\
R5. \quad q \uplus \{g_1\} & \quad \triangleright \quad q \uplus \{g_2\} & \quad \text{if } g_1 > g_2 \text{ and the relation name in } g_2 \text{ does not occur in } q \\
R6. \quad q & \quad \triangleright \quad q_{x \to a} & \quad \text{if } x \text{ is a liaison variable of } q \\
R7. \quad q & \quad \triangleright \quad q_{y \to x} & \quad \text{if } \exists y \in q(x, y \in \text{Vars}(g)) \text{ and one of } x \text{ or } y \text{ (or both) is a liaison variable of } q \text{ that occurs at a non-primary-key position in } q \\
R8. \quad q \uplus \{R_1(\vec{s}, \vec{t}, x, \vec{u})\} & \quad \triangleright \quad q \uplus \{R_2(\vec{s}, \vec{t}, \vec{u})\} & \quad \text{if } x \text{ occurs only once in } q \uplus \{R_1(\vec{s}, \vec{t}, x, \vec{u})\} \text{ and } R_2 \text{ does not occur in } q \\
R9. \quad q & \quad \triangleright \quad q' & \quad \text{if some constant } c \text{ with } c \neq a \text{ occurs in } q \text{ and } q' \text{ is obtained from } q \text{ by replacing all constants with } a \quad \text{if true} \\
R10. \quad q \uplus \{R(x, x)\} & \quad \triangleright \quad q \uplus \{R(x, a)\} & \quad \text{if } 0
\end{align*}
\]

Figure 4: Ten rewrite rules.

The following lemma states that for every final query $q$, \text{\texttt{CERTAINTY}}$(q)$ is $\text{\texttt{2P}}$-hard. The proof is technically involved and will be further explained in Section 6.

**Lemma 9.** For every final SJFCQ query $q$, it is the case that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard under polynomial-time Turing reductions.

Finally, we show that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard for unsafe SJFCQ queries $q$.

**Theorem 4.** For every unsafe SJFCQ query $q$, it is the case that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard under polynomial-time Turing reductions.

**Proof.** Let $q$ be an unsafe SJFCQ query. By Lemma 8, every maximal sequence $q_0 \uplus q_1 \uplus q_2 \uplus \ldots$ is finite and ends with a safe query. It follows that there must be a final query $q_0$ such that $q \uplus q_0$ or $q = q_0$. By Lemma 9, $\text{\texttt{CERTAINTY}}(q_0)$ is $\text{\texttt{2P}}$-hard under polynomial time Turing reductions. By Lemma 7 and since polynomial-time many-one counting reductions compose, there exists a polynomial-time many-one counting reduction from $\text{\texttt{CERTAINTY}}(q_0)$ to $\text{\texttt{CERTAINTY}}(q)$. It follows that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard under polynomial time Turing reductions.

The proof of our main dichotomy theorem follows.

**Proof Theorem 1.** Every SJFCQ query $q$ is safe or not safe. If $q$ is safe, then $\text{\texttt{CERTAINTY}}(q)$ is in $\text{\texttt{P}}$ by Theorem 3. If $q$ is not safe, then $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard by Theorem 4.

Finally, we show that $\text{\texttt{CERTAINTY}}(q)$ is in $\text{\texttt{P}}$ for SJFCQ queries $q$. The associated decision problem is to decide, given a database $db$, whether there exists a repair of $db$ that satisfies $q$. The latter decision problem is in $\text{\texttt{NP}}$, because a polynomial-time verifiable certificate of polynomial size is a repair of $db$ that satisfies $q$.

**6. FINAL QUERIES ARE HARD**

In this section, we outline the proof of Lemma 9. We first define a class $\mathcal{F}$ of SJFCQ queries and prove in Lemma 10 that for every query $q \in \mathcal{F}$, it is the case that the problem $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard. After that, we show in Lemma 11 that each final SJFCQ query $q$ belongs to $\mathcal{F}$. It follows that if $q$ is a final SJFCQ query, then $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard. The proofs of Lemmas 10 and 11 are technically involved and use a number of helping lemmas. All the details are available in a separate appendix. Finally, we give the proof of Theorem 2 announced in Section 1.

**Definition 5.** We define $\mathcal{F}$ as the subset of SJFCQ containing each query that has one of the following forms:

- $\{R(x, y), S(x, a), T(y, a)\}$
- $\{R_1(x, y), \ldots, R_k(x, y), S(y, a)\}$ for some $k \geq 1$; or
- $\{R_1(x, y), \ldots, R_k(x, y), S_1(x, x), \ldots, S_l(y, x)\}$ for some $k \geq 1$ and $l \geq 1$.

The queries in $\mathcal{F}$ are slightly different from the final queries found by Dalvi et al. for probabilistic databases [7, 5]. To be precise, Dalvi et al.’s final queries can be obtained from $\mathcal{F}$ by removing all positions at which the constant $a$ occurs. In our framework of uncertain databases, queries $\{R(x, y), S(x), T(y)\}$ and $\{R(x, y), S(y)\}$ are safe, and hence cannot be final.

**Lemma 10.** For every SJFCQ query $q \in \mathcal{F}$, it is the case that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard under polynomial-time Turing reductions.

**Lemma 11.** Every final SJFCQ query belongs to $\mathcal{F}$.

We can now give the proof of Lemma 9

**Proof Lemma 9.** Let $q$ be a final SJFCQ query. By Lemma 11, $q \in \mathcal{F}$. By Lemma 10, $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard under polynomial-time Turing reductions.

Finally, we give the proof of Theorem 2 announced in Section 1. This theorem states that for the same SJFCQ query $q$, moving from the decision problem $\text{\texttt{CERTAINTY}}(q)$ to its counting variant $\text{\texttt{CERTAINTY}}(q)$ can result in a dramatic increase in computational complexity, from $\text{\texttt{AC}}^0$ to $\text{\texttt{2P}}$-complete.

**Proof Theorem 2.** Let $q = \{R(x, y), S(y, a)\}$. From Theorem 3 in [15], it follows that $\text{\texttt{CERTAINTY}}(q)$ is first-order expressible, and hence in $\text{\texttt{AC}}^0$. By Lemma 10, it is the case that $\text{\texttt{CERTAINTY}}(q)$ is $\text{\texttt{2P}}$-hard.
7. CONCLUSION

For databases that are inconsistent with respect to primary key constraints, we moved from the yes-no question “Do all repairs satisfy Boolean conjunctive query q?” toward the counting question “How many repairs satisfy q?” Our study has revealed a significant complexity dichotomy (Theorem 1). Moreover, we showed that moving from yes-no to counting can dramatically increase complexity (from $\text{AC}^0$ to $\sharp\text{P}$-hard).

Several questions are open for future research:

- Is there a comparable dichotomy theorem for the decision problem $\text{CERTAINTY}(q)$? That is, is it true that for every $\text{SJFCQ}$ query $q$, $\text{CERTAINTY}(q)$ is in $\text{P}$ or coNP-complete? Such dichotomy has been pursued by Fuxman and Miller [8]; see also [16]. It remains an intriguing open problem. Notice that, by Theorem 2, the dichotomy of Theorem 1 does not give us a dichotomy for $\text{CERTAINTY}(q)$.

- Can our results be extended to richer query languages, like unions of conjunctive queries or conjunctive queries with self-joins?

8. REFERENCES