


Consistent Query Answering for Primary Keys and Conjunctive Queries with Counting

Aziz Amezian El Khalfioui ✉

Research fellow FNRS
University of Mons, Mons, Belgium

Jef Wijsen ✉ 

University of Mons, Mons, Belgium

Abstract

The problem of consistent query answering for primary keys and self-join-free conjunctive queries has been intensively studied in recent years and is by now well understood. In this paper, we study an extension of this problem with counting. The queries we consider count how many times each value occurs in a designated (possibly composite) column of an answer to a full conjunctive query. In a setting of database repairs, we adopt the semantics of [Arenas et al., ICDT 2001] which computes tight lower and upper bounds on these counts, where the bounds are taken over all repairs. Ariel Fuxman defined in his PhD thesis a syntactic class of queries, called *Cforest*, for which this computation can be done by executing two first-order queries (one for lower bounds, and one for upper bounds) followed by simple counting steps. We use the term “parsimonious counting” for this computation. A natural question is whether *Cforest* contains all self-join-free conjunctive queries that admit parsimonious counting. We answer this question negatively. We define a new syntactic class of queries, called *Cparsimony*, and prove that it contains all (and only) self-join-free conjunctive queries that admit parsimonious counting.

2012 ACM Subject Classification Information systems → Relational database query languages; Theory of computation → Incomplete, inconsistent, and uncertain databases; Theory of computation → Logic and databases

Keywords and phrases Consistent query answering, primary key, conjunctive query, aggregation, counting

Digital Object Identifier 10.4230/LIPIcs.ICDT.2023.23

Related Version *Extended Version*: <https://doi.org/10.48550/arXiv.2211.04134> [25]

1 Introduction

The problem of consistent query answering (CQA) [2, 4, 5, 37] with respect to primary keys is by now well understood for self-join-free conjunctive queries: a dichotomy between tractable and intractable queries has been established, and it is known which queries have a consistent first-order rewriting [29, 32]. It remains a largely open question to extend these complexity results to queries with aggregation. In this paper, we look at a simple form of aggregation: counting the number of times each (possibly composite) value occurs in the answer to a conjunctive query. Although this problem has been studied since the early years of CQA [18], a fine-grained characterization of its complexity remains open.

Formally, let q be a full (i.e., quantifier-free) self-join-free conjunctive query. We define a counting query as follows. We designate a tuple \vec{z} of distinct variables of q , called the *grouping variables*, and let \vec{w} be a tuple of the variables in q that are not in \vec{z} . The variables of q , which are all free, are made explicit by denoting q as $q(\vec{z}, \vec{w})$. We are interested in a query that, on a given database instance \mathbf{db} , returns all tuples (\vec{c}, i) with \vec{c} a tuple of



© Aziz Amezian El Khalfioui and Jef Wijsen;
licensed under Creative Commons License CC-BY 4.0
26th International Conference on Database Theory (ICDT 2023).

Editors: Floris Geerts and Brecht Vandevoort; Article No. 23; pp. 23:1–23:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

constants, of the same arity as \vec{z} , and with i a positive integer that is the number of distinct tuples \vec{d} , of the same arity as \vec{w} , satisfying $(\vec{c}, \vec{d}) \in q(\mathbf{db})$. This counting query will be denoted $\mathbf{cnt}(q, \vec{z})$.

For example, consider the database schema of Fig. 1, which is intended to store the unique gender and department of each employee, and the unique building of each department. Ignore for now that the database instance of Fig. 1 is inconsistent (as it stores two departments for Anny, and two buildings for IT). Let $q_0(x, y, z) = E(x, \text{'F'}, y) \wedge D(y, z)$, where x, y, z are variables and 'F' denotes a constant. Then, on a consistent database instance, $\mathbf{cnt}(q_0, z)$ would return the number of female employees working in each building. In SQL, $\mathbf{cnt}(q_0, z)$ can be encoded as follows:

```
SELECT  Building, COUNT(*) AS CNT
FROM    E, D
WHERE   E.Dept = D.Dept AND Gender = 'F'
GROUP BY Building
```

On the database instance of Fig. 1, this query will return $(A, 4)$ and $(B, 3)$. These answers are however not meaningful because they suffer from double-counting due to inconsistencies. We describe next a more meaningful semantics that was introduced in [2].

First, following [1], we define a *repair* of a database instance as a maximal subinstance that satisfies all primary-key constraints. In this paper, we consider no other constraints than primary keys. Then, following the approach of [2], more meaningful answers are obtained by returning, for every value \vec{c} for the grouping variables \vec{z} , tight lower and upper bounds on the corresponding counts over all repairs. This new query is denoted by $\mathbf{cqacnt}(q, \vec{z})$. Thus, an answer $(\vec{c}, [m, n])$ to this new query means that on every repair, our original query $\mathbf{cnt}(q, \vec{z})$ returns a tuple (\vec{c}, i) with $m \leq i \leq n$, and, moreover, these bounds m and n are tight. By tight, we mean that for every $j \in \{m, n\}$, there is a repair on which $\mathbf{cnt}(q, \vec{z})$ returns (\vec{c}, j) .

For example, the database instance of Fig. 1 has four repairs, because there are two choices for Anny's department, and two choices for the building of the IT department. Note that in Fig. 1, *blocks* of conflicting tuples are separated by dashed lines. The query $\mathbf{cnt}(q_0, z)$ returns different answers on each repair: there are two repairs where the answer is $\{(A, 3), (B, 1)\}$; there is one repair where the answer is $\{(A, 1), (B, 3)\}$; and there is one repair where the answer is $\{(A, 2), (B, 2)\}$. The latter set of answers, for example, is obtained in the repair that assigns Anny to department HR, and IT to building B. The query $\mathbf{cqacnt}(q_0, z)$ would thus return $\{(A, [1, 3]), (B, [1, 3])\}$.

In this paper, we are concerned about the complexity of computing $\mathbf{cqacnt}(q, \vec{z})$. In general, there exist self-join-free conjunctive queries q such that, for some choice of the grouping variables \vec{z} , $\mathbf{cqacnt}(q, \vec{z})$ cannot be solved in polynomial time (under standard complexity assumptions). This follows from earlier research showing that there are self-join-free conjunctive queries $q'(\vec{z})$ for which the following problem is coNP-complete: given \vec{c} and \mathbf{db} , determine whether $q'(\vec{c})$ is true in every repair of \mathbf{db} . The latter problem obviously reduces to counting: $q'(\vec{c})$ is true in every repair of \mathbf{db} if and only if $\mathbf{cqacnt}(q, \vec{z})$ returns $(\vec{c}, [m, n])$ on \mathbf{db} for some $m \geq 1$, where q is the full query obtained from q' by dropping quantification.

In his PhD thesis [18], Fuxman showed that for some q and \vec{z} , the answer to $\mathbf{cqacnt}(q, \vec{z})$ can be computed by executing first-order queries followed by simple counting steps. To illustrate his approach, consider the following query in SQL:

```
SELECT  Building, COUNT(DISTINCT Emp) AS CNT
FROM    E, D
WHERE   E.Dept = D.Dept AND Gender = 'F'
GROUP BY Building
```

<i>E</i>	<u><i>Emp</i></u>	<i>Gender</i>	<u><i>Dept</i></u>	<i>D</i>	<u><i>Dept</i></u>	<u><i>Building</i></u>
	Suzy	F	HR		HR	A
	Anny	F	HR		IT	A
	Anny	F	IT		IT	B
	Grety	F	IT		MIS	B
	Lucy	F	MIS			

■ **Figure 1** Example database. Primary keys are underlined.

On our example database of Fig. 1, this query returns $\{(A, 3), (B, 3)\}$. We observe that the returned counts match the upper bounds previously found for $\mathbf{cqacnt}(q_0, z)$. Importantly, it can be shown that this is not by accident: on *every* database instance, the latter SQL query will return the correct upper bounds for $\mathbf{cqacnt}(q_0, z)$. Note that the latter query uses `COUNT(DISTINCT Emp)`, which means that duplicates are removed, which is a standard practice in relational algebra.

We now explain how to obtain the lower bounds for our example query. To this end, consider the following query:

```
SELECT Building, Emp
FROM   E, D
WHERE  E.Dept = D.Dept AND Gender = 'F'
```

Following [1], we define the *consistent answer* to such a query as the intersection of the query answers on all repairs. For our example database, the consistent answer is the following table, which we call C :

<i>C</i>	<u><i>Emp</i></u>	<u><i>Building</i></u>
	Suzy	A
	Lucy	B

Note that Anny does not occur in the consistent answer, because $(Anny, A)$ is false in some repair, and so is $(Anny, B)$. From [29], it follows that computing the consistent answers to the latter SQL query is in FO (i.e., the class of problems that can be solved by a first-order query), using a technique known as *consistent first-order rewriting*. The lower bounds $\{(A, 1), (B, 1)\}$ are now found by executing the following query on C (and, again, this is not by accident):

```
SELECT  Building, COUNT(DISTINCT Emp) AS CNT
FROM    C
GROUP BY Building
```

Since C can be expressed in SQL, we can actually construct a single SQL query that computes the lower bounds in $\mathbf{cqacnt}(q_0, z)$.

In general, if $q(\vec{z}, \vec{w})$ is a full self-join-free conjunctive query for which $\mathbf{cqacnt}(q, \vec{z})$ can be computed as previously described, then we will say that the query obtained from q by existentially binding the variables in \vec{w} (i.e., by binding the variables that are not grouping variables) admits *parsimonious counting*. Thus, our example showed that $\exists x \exists y E(x, 'F', y) \wedge D(y, z)$ admits parsimonious counting. A formal definition of parsimonious counting will be given later on (Definition 8). In this introduction, we content ourselves by saying that parsimonious counting, if possible, computes $\mathbf{cqacnt}(q, \vec{z})$ by executing two first-order queries (one for lower bounds, and one for upper bounds), followed by simple counting steps.

The main contribution of our paper can now be described. In his doctoral dissertation [18], Fuxman defined a class of self-join-free conjunctive queries, called *Cforest*, and showed the following.

► **Theorem 1** ([18]). *Every query in Cforest admits parsimonious counting.*

The class Cforest has been used in several studies on consistent query answering. It was an open question whether Cforest contains *all* self-join-free conjunctive queries that admit parsimonious counting. We will answer this question negatively in Section 8. More fundamentally, we introduce a new syntactic class, called Cparsimony, which includes Cforest and contains *all* (and only) self-join-free conjunctive queries that admit parsimonious counting. That is, we prove the following theorem.

► **Theorem 2 (Main theorem)**. *For every self-join-free conjunctive query q , it holds that q admits parsimonious counting if and only if q is in Cparsimony.*

Moreover, a new and simpler proof for Theorem 1 will follow in Section 8.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 introduces preliminary constructs and notations. Section 4 introduces the semantic notion of parsimonious counting. Section 5 introduces our new syntactic class of queries, called Cparsimony, which restricts self-join-free conjunctive queries. Section 6 shows that every query in Cparsimony admits parsimonious counting, and Section 7 shows that Cparsimony contains every self-join-free conjunctive query that admits parsimonious counting. Section 8 shows that Cforest is strictly included in Cparsimony, and provides a new proof for Theorem 1. Section 9 concludes the paper. Several helping lemmas and proofs are available in [25].

2 Related Work

Consistent query answering (CQA) started by a seminal paper in 1999 co-authored by Arenas, Bertossi, and Chomicki [1], who introduced the notions of repair and consistent answer. Two years later, the same authors introduced the *range semantics* (with lower and upper bounds) for queries with aggregation [2, 3][4, Chapter 5], which has been commonly adopted ever since. In particular, it was adopted in the PhD thesis [18] of Fuxman, who provided Theorem 1 (albeit using different terminology) and its proof, and used this result in the implementation of the ConQuer system [19]. ConQuer aims at computations in first-order logic with counting (coined “parsimonious counting” in the current paper), which can be encoded in SQL. This is different from AggCAvSAT [15], a recent system by Dixit and Kolaitis, which uses powerful SAT solvers for computing range semantics, and thus can solve queries that are beyond the computational power of ConQuer. Aggregation queries were also studied in the context of CQA in [6].

Consistent query answering for self-join-free conjunctive queries q and primary keys has been intensively studied. Its decision variant, which was coined CERTAINTY(q) in 2010 [36], asks whether a Boolean query q is true in every repair of a given database instance. A systematic study of its complexity for self-join-free conjunctive queries had started already in 2005 [21], and was eventually solved in two journal articles by Koutris and Wijsen [29, 32], as follows: for every self-join-free Boolean conjunctive query q , CERTAINTY(q) is either in FO, L-complete, or coNP-complete, and it is decidable, given q , which case applies. This complexity classification extends to non-Boolean queries by treating free variables as constants. Other extensions beyond this trichotomy deal with foreign keys [23], more than one key per relation [31], negated atoms [30], or restricted self-joins [28]. For unions of conjunctive queries q , Fontaine [17] established interesting relationships between CERTAINTY(q) and Bulatov’s dichotomy theorem for conservative CSP [7].

The counting variant of CERTAINTY(q), denoted #CERTAINTY(q), asks to count the number of repairs that satisfy some Boolean query q . This counting problem is fundamentally different from the range semantics in the current paper. For self-join-free conjunctive queries,

\sharp CERTAINTY(q) exhibits a dichotomy between FP and \sharp P-complete under polynomial-time Turing reductions [33]. This dichotomy has been shown to extend to queries with self-joins if primary keys are singletons [34], and to functional dependencies [11]. Calautti, Console, and Pieris present in [8] a complexity analysis of these counting problems under weaker reductions, in particular, under many-one logspace reductions. The same authors have conducted an experimental evaluation of randomized approximation schemes for approximating the percentage of repairs that satisfy a given query [9]. Other approaches to making CQA more meaningful and/or tractable include operational repairs [10, 12] and preferred repairs [26, 35].

Recent overviews of two decades of theoretical research in CQA are [5, 37]. It is worthwhile to note that theoretical research in CERTAINTY(q) has stimulated implementations and experiments in prototype systems [14, 16, 19, 20, 24, 27].

3 Preliminaries

We assume that every relation name R is associated with an *arity*, which is a positive integer. We assume that all *primary-key positions* precede all *non-primary-key positions*. We say that R has *signature* $[n, k]$ if R has arity n and primary-key positions $1, \dots, k$.

If R has signature $[n, k]$ and s_1, \dots, s_n are variables or constants, then $R(s_1, \dots, s_n)$ is an *R-atom* (or simply *atom*), which will often be denoted as $R(\underline{s_1, \dots, s_k}, \overline{s_{k+1}, \dots, s_n})$ to distinguish between primary-key and non-primary-key positions. Two atoms $R_1(\underline{\vec{s}_1}, \overline{\vec{t}_1})$ and $R_2(\underline{\vec{s}_2}, \overline{\vec{t}_2})$ are said to be *key-equal* if $R_1 = R_2$ and $\vec{s}_1 = \vec{s}_2$. A *fact* is an atom in which no variable occurs. A *database instance* (or simply *database*) is a finite set of facts. A database instance \mathbf{db} is *consistent* if it does not contain two distinct key-equal facts. A *repair* of \mathbf{db} is a \subseteq -maximal consistent subset of \mathbf{db} .

If \vec{s} is a tuple of variables or constants, then $|\vec{s}|$ denotes the arity of \vec{s} , and $\text{vars}(\vec{s})$ denotes the set of variables occurring in \vec{s} . By an abuse of notation, if we use a tuple \vec{z} of variables at places where a set of variables is expected, we mean $\text{vars}(\vec{z})$. For an atom $F = R(\underline{\vec{s}}, \overline{\vec{t}})$, we define $\text{Key}(F) := \text{vars}(\vec{s})$, $\text{notKey}(F) := \text{vars}(\vec{t}) \setminus \text{vars}(\vec{s})$, and $\text{vars}(F) := \text{vars}(\vec{s}) \cup \text{vars}(\vec{t})$. For example, if $F = R(\underline{c, x, x, y}, \overline{y, z, c})$, then $\text{Key}(F) = \{x, y\}$ and $\text{notKey}(F) = \{z\}$, where c is a constant.

Conjunctive Queries. A conjunctive query q is a first-order formula of the form:

$$\exists \vec{w} (R_1(\underline{\vec{x}_1}, \overline{\vec{y}_1}) \wedge \dots \wedge R_n(\underline{\vec{x}_n}, \overline{\vec{y}_n})), \quad (1)$$

where the variables of \vec{w} are *bound*, and the other variables are *free*. Such a query is also denoted by $q(\vec{z})$ with \vec{z} a tuple composed of the free variables. We write $\text{vars}(q)$ for the set of variables that occur in q , and can assume $\text{vars}(q) = \text{vars}(\vec{w}) \cup \text{vars}(\vec{z})$ without loss of generality. We say that q is *full* if all variables of $\text{vars}(q)$ are free. We say that q is *self-join-free* if $i \neq j$ implies $R_i \neq R_j$. The quantifier-free part $R_1(\underline{\vec{x}_1}, \overline{\vec{y}_1}) \wedge \dots \wedge R_n(\underline{\vec{x}_n}, \overline{\vec{y}_n})$ of q is denoted $\text{body}(q)$. By slightly overloading notation, we also use $\text{body}(q)$ for the set $\{R_1(\underline{\vec{x}_1}, \overline{\vec{y}_1}), \dots, R_n(\underline{\vec{x}_n}, \overline{\vec{y}_n})\}$. We write $\text{free}(q)$ for the set of free variables in q .

If a self-join-free conjunctive query q is understood, and we use a relation name R at places where an atom is expected, then we mean the unique R -atom of q . If \vec{c} is a tuple of constants of arity $|\vec{z}|$ and \mathbf{db} a database instance, then $\mathbf{db} \models q(\vec{c})$ denotes that $q(\vec{c})$ is true in \mathbf{db} using standard first-order semantics. If $\mathbf{db} \models q(\vec{c})$, we also write $\vec{c} \in q(\mathbf{db})$, and we say that \vec{c} is an *answer* to q on \mathbf{db} .

We now introduce operators for turning bound variables into free variables, or vice versa, and for instantiating free variables.

Making bound variables free. Let q be a conjunctive query with $\text{free}(q) = \vec{z}$. Let \vec{x} be a tuple of (not necessarily all) bound variables in q (hence $\vec{x} \cap \vec{z} = \emptyset$). We write $\#_{\vec{x}}[q]$ for the conjunctive query q' such that $\text{free}(q') = \vec{z} \cup \vec{x}$ and $\text{body}(q') = \text{body}(q)$. Informally, $\#_{\vec{x}}[q]$ is obtained from q by omitting the quantification $\exists \vec{x}$. For example, if $q(z) = \exists x \exists y R(x, y) \wedge R(y, z)$, then $\#_{\vec{x}}[q] = \exists y R(x, y) \wedge R(y, z)$.

Binding free variables. Let q be a conjunctive query, and \vec{x} a tuple of (not necessarily all) free variables of q . Then $\exists \vec{x}[q]$ denotes the query with the same body as q , but whose set of free variables is $\text{free}(q) \setminus \vec{x}$.

Instantiating free variables. Let q be a conjunctive query, and \vec{z} a tuple of distinct free variables of q . Let \vec{c} be a tuple of constants of arity $|\vec{z}|$. Then $q_{[\vec{z} \rightarrow \vec{c}]}$ is the query obtained from q by replacing, for every $i \in \{1, 2, \dots, |\vec{z}|\}$, each occurrence of the i th variable in \vec{z} by the i th constant in \vec{c} .

Consistent Query Answering. Let $q(\vec{z})$ be a conjunctive query. We write $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$ if for every repair \mathbf{r} of \mathbf{db} , we have $\mathbf{r} \models q(\vec{c})$. If $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$, we also say that \vec{c} is a *consistent answer* to q on \mathbf{db} . A *consistent first-order rewriting* of $q(\vec{z})$ is a first-order formula $\varphi(\vec{z})$ such that for every database instance \mathbf{db} and every tuple \vec{c} of constants of arity $|\vec{z}|$, we have $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$ if and only if $\mathbf{db} \models \varphi(\vec{c})$. Note incidentally that the set of integrity constraints is always implicitly understood to be the primary keys associated with the relation names that occur in the query.

Query Graph. The *query graph* of a conjunctive query $q(\vec{z})$ is an undirected graph whose vertices are the bound variables of q . There is an edge between x and y if $x \neq y$ and x, y occur together in some atom of $\text{body}(q)$.

Attack Graph. The following is a straightforward extension of attack graphs [29] to deal with free variables.

Let $q(\vec{z})$ be a self-join-free conjunctive query. If S is a subset of $\text{body}(q)$, then $q \setminus S$ denotes the query obtained from q by removing from q all atoms in S . Every variable of $q \setminus S$ that is free in q remains free in $q \setminus S$; and every variable of $q \setminus S$ that is bound in q remains bound in $q \setminus S$.

We define $\mathcal{K}(q)$ as the set of functional dependencies that contains $\emptyset \rightarrow \text{free}(q)$ and contains, for every atom F in q , the functional dependency $\text{Key}(F) \rightarrow \text{vars}(F)$. Note that since $\mathcal{K}(q)$ contains $\emptyset \rightarrow \text{free}(q)$, we have that $\mathcal{K}(q) \models \emptyset \rightarrow y$ if and only if $\mathcal{K}(q) \models \text{free}(q) \rightarrow y$, for each $y \in \text{vars}(q)$. If F is an atom of q , then $F^{+,q}$ is the set that contains every variable $y \in \text{vars}(q)$ such that either $y \in \text{free}(q)$ or $\mathcal{K}(q \setminus \{F\}) \models \text{Key}(F) \rightarrow y$ (or both).

It is known that in the study of consistent query answering for self-join-free conjunctive queries, we often do not need a special treatment of free variables, because computing consistent answers to $q(\vec{z})$ has the same time complexity as the decision problem $\text{CERTAINTY}(q(\vec{z})_{[\vec{z} \rightarrow \vec{c}]})$ with \vec{c} a sequence of pairwise distinct fresh constants. The addition of functional dependencies $\emptyset \rightarrow \text{free}(q)$ has the same effect as treating variables in $\text{free}(q)$ as constants. In the following example, we omit curly braces and commas when denoting sets of variables. For example, $\{z_1, z_2\}$ is denoted $z_1 z_2$.

► **Example 3.** Let $q = \exists u \exists v \exists x \exists y R(u, x) \wedge S(x, z_1, y) \wedge T(y, v, z_2) \wedge U(y, u)$. We have $\text{free}(q) = z_1 z_2$. Then, $q \setminus \{T\}$ ¹ is the query $\exists u \exists v \exists x \exists y R(u, x) \wedge S(x, z_1, y) \wedge U(y, u)$, whose only free variable is z_1 . Note incidentally that since v does not occur in the latter query, the

¹ Recall that we use T as a shorthand for the T -atom of q .

quantification $\exists v$ can be dropped. We have $\mathcal{K}(q \setminus \{T\}) = \{\emptyset \rightarrow z_1, u \rightarrow x, xz_1 \rightarrow y, y \rightarrow u\}$. Note that $\emptyset \rightarrow z_1$ belongs to the latter set because z_1 is free in $q \setminus \{T\}$. The set of variables that are functionally dependent on $\text{Key}(T)$ relative to $\mathcal{K}(q \setminus \{T\})$ is xyz_1 . Finally, we obtain $T^{+,q} = xyz_1z_2$. Note that the latter set contains the variable z_2 that is free in q . \square

We say that an atom F of q *attacks* a variable x occurring in q , denoted $F \overset{q}{\rightsquigarrow} x$, if there exists a sequence $\langle x_1, x_2, \dots, x_n \rangle$ of bound variables of q ($n \geq 1$) such that:

1. if two variables are adjacent in the sequence, then they occur together in some atom of q ;
2. $x_1 \in \text{notKey}(F)$ and $x_n = x$; and
3. for every $\ell \in \{1, \dots, n\}$, $x_\ell \notin F^{+,q}$.

Such a sequence will be called a *witness* of $F \overset{q}{\rightsquigarrow} x$. We say that an atom F of q *attacks* another atom G of q , denoted $F \overset{q}{\rightsquigarrow} G$, if $F \neq G$ and F attacks some variable of $\text{vars}(G)$. It is now easily verified that if F attacks G , then F also attacks a variable in $\text{Key}(G)$. A variable or atom that is not attacked, is called *unattacked* (where q is understood from the context). 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 if $F \overset{q}{\rightsquigarrow} G$. A directed edge in the attack graph is called an *attack*. Koutris and Wijsen [29] showed the following.

► **Theorem 4** ([29]). *A self-join-free conjunctive query $q(\vec{z})$ has a consistent first-order rewriting if and only if its attack graph is acyclic.*

An attack from F to G is *weak* if $\mathcal{K}(q) \models \text{Key}(F) \rightarrow \text{Key}(G)$; otherwise it is strong. By a *component* of an attack graph, we always mean a maximal weakly connected component.

Let q be a self-join-free conjunctive query. Whenever the relationship $\mathcal{K}(q) \models Z \rightarrow w$ holds true, then there exists a sequential proof of it, as defined next.

Sequential Proof. Let $q(\vec{z})$ be a self-join-free conjunctive query, and $Z \subseteq \text{vars}(q)$. Let $\langle F_1, F_2, \dots, F_n \rangle$ be a (possibly empty) sequence of atoms in $\text{body}(q)$ such that for every $i \in \{1, \dots, n\}$, $\text{Key}(F_i) \subseteq \text{free}(q) \cup Z \cup \left(\bigcup_{j=1}^{i-1} \text{vars}(F_j) \right)$. Such a sequence is called a *sequential proof* of $\mathcal{K}(q) \models Z \rightarrow w$, for every $w \in \text{free}(q) \cup Z \cup \left(\bigcup_{j=1}^n \text{vars}(F_j) \right)$. A sequential proof of $\mathcal{K}(q) \models Z \rightarrow w$ is called *minimal* if $\langle F_1, \dots, F_{n-1} \rangle$ is not a sequential proof of $\mathcal{K}(q) \models Z \rightarrow w$.

4 Parsimonious Counting

Consider a conjunctive query $q(\vec{z}) = \exists \vec{w} B$, with B a quantifier-free conjunction of atoms (called the *body*). We introduce a query that takes a database instance \mathbf{db} as input and returns, for every tuple $\vec{c} \in q(\mathbf{db})$, the number of valuations for \vec{w} that make the query true.

► **Definition 5** ($\text{cnt}(q, \vec{z})$). *Let $q(\vec{z}, \vec{w})$ be a full conjunctive query, in which notation it is understood that \vec{z} and \vec{w} are disjoint, duplicate-free tuples of variables. $\text{cnt}(q, \vec{z})$ is the query that takes as input a database instance \mathbf{db} and returns every tuple (\vec{c}, i) for which the following hold:*

1. \vec{c} a tuple of constants of arity $|\vec{z}|$; and
2. i is a positive integer such that i is the number of distinct tuples \vec{d} , of arity $|\vec{w}|$, satisfying $\mathbf{db} \models q(\vec{c}, \vec{d})$.

A maximal set of answers to $q(\mathbf{db})$ that agree on \vec{z} will also be called a \vec{z} -group (where q and \mathbf{db} are implicitly understood). Thus, $\text{cnt}(q, \vec{z})$ counts the number of tuples in each \vec{z} -group.

The following definition introduces range consistent query answers as introduced in [2].

► **Definition 6** ($\mathbf{cqacnt}(q, \vec{z})$). Let $q(\vec{z}, \vec{w})$ be a full conjunctive query, in which notation it is understood that \vec{z} and \vec{w} are disjoint, duplicate-free tuples of variables. $\mathbf{cqacnt}(q, \vec{z})$ is the query that takes as input a database instance \mathbf{db} and returns every tuple $(\vec{c}, [m, n])$ for which the following hold:

1. for every repair \mathbf{r} of \mathbf{db} , there exists \vec{d} such that $\mathbf{r} \models q(\vec{c}, \vec{d})$;
2. there is a repair of \mathbf{db} on which $\mathbf{cnt}(q, \vec{z})$ returns (\vec{c}, m) ;
3. there is a repair of \mathbf{db} on which $\mathbf{cnt}(q, \vec{z})$ returns (\vec{c}, n) ; and
4. if $\mathbf{cnt}(q, \vec{z})$ returns (\vec{c}, i) on some repair of \mathbf{db} , then $m \leq i \leq n$.

If $(\vec{c}, [m, n])$ is an answer to $\mathbf{cqacnt}(q, \vec{z})$ on \mathbf{db} , then we will say that it is a range-consistent answer. Note that if $(\vec{c}, [m, n])$ is a range-consistent answer, then, by definition, \vec{c} is a consistent answer to $q(\vec{z})$, hence $m \geq 1$.

The following proposition states that computing $\mathbf{cqacnt}(q(\vec{z}, \vec{w}), \vec{z})$ can be NP-hard, even if the query $\exists \vec{w} [q]$ has a consistent first-order rewriting.

► **Proposition 7.** *There exists a self-join-free conjunctive query $q(\vec{z})$ that has a consistent first-order rewriting such that $\mathbf{cqacnt}(\text{body}(q), \vec{z})$ is NP-hard to compute.*

Proof sketch. In 3-DIMENSIONAL MATCHING (3DM), we are given a set $M \subseteq A_1 \times A_2 \times A_3$, where A_1, A_2, A_3 are disjoint sets having the same number n of elements. We are asked whether M contains a matching, that is, a subset $M' \subseteq M$ such that $|M'| = n$ and no two elements of M' agree in any coordinate. The problem 3DM is NP-complete [22].

Consider the query $q(z) = \exists x_1 \exists x_2 \exists x_3 \exists y Z(\underline{z}) \wedge \bigwedge_{i=1}^3 (R_i(x_i, y) \wedge S_i(x_i, y))$. The edge-set of q 's attack graph is empty. Therefore, q 's attack graph is acyclic. By Theorem 4, $q(z)$ has a consistent first-order rewriting. Let $M \subseteq A_1 \times A_2 \times A_3$ be an instance of 3DM. Let \mathbf{db}_M be the database instance that contains $Z(\underline{c})$ and includes, for every $a_1 a_2 a_3$ in M , $\bigcup_{i=1}^3 \{R_i(a_i, a_1 a_2 a_3), S_i(a_i, a_1 a_2 a_3)\}$. Moreover, \mathbf{db}_M includes $\bigcup_{i=1}^3 \{R_i(\perp_i, \top), S_i(\perp_i, \top)\}$, where $\perp_1, \perp_2, \perp_3, \top$ are fresh constants not in $A_1 \cup A_2 \cup A_3$. Clearly, \mathbf{db}_M is first-order computable from M . It can now be verified that M has a matching if and only if for some ℓ , $\mathbf{cqacnt}(\text{body}(q), z)$ returns $(c, [\ell, n + 1])$ on \mathbf{db}_M . ◀

Note that the foregoing proof can be easily adapted from 3DM to 2DM. That is, the query $q(z) = \exists x_1 \exists x_2 \exists y Z(\underline{z}) \wedge \bigwedge_{i=1}^2 (R_i(x_i, y) \wedge S_i(x_i, y))$ has a consistent first-order rewriting, but computing $\mathbf{cqacnt}(\text{body}(q), z)$ is as hard as 2DM.

We now introduce the semantic notion of *parsimonious counting*, which was illustrated by the running example in Section 1. Informally, for a query $q(\vec{z})$ that admits parsimonious counting, it will be the case that on every database instance \mathbf{db} , the answers to $\mathbf{cqacnt}(\text{body}(q), \vec{z})$ can be computed by a first-order query followed by a simple counting step.

► **Definition 8** (Parsimonious counting). Let q be a conjunctive query with $\text{free}(q) = \vec{z}$.² Let \vec{x} be a (possibly empty) sequence of distinct bound variables of $q(\vec{z})$. We say that q admits parsimonious counting on \vec{x} if the following hold (let $q'(\vec{z}, \vec{x}) = \# \vec{x} [q]$):

- (A) $q(\vec{z})$ has a consistent first-order rewriting;
- (B) $q'(\vec{z}, \vec{x})$ has a consistent first-order rewriting (call it $\varphi(\vec{z}, \vec{x})$); and
- (C) for every database instance \mathbf{db} , the following conditions (Ca) and (Cb) are equivalent:

² We will commonly write $q(\vec{z})$ to make explicit that $\text{free}(q) = \vec{z}$.

- (a) $(\vec{c}, [m, n])$ is an answer to $\mathbf{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{db} ;
- (b) $m \geq 1$ and both the following hold:
 - (i) m is the number of distinct tuples \vec{d} , of arity \vec{x} , such that $\mathbf{db} \models \varphi(\vec{c}, \vec{d})$; and
 - (ii) n is the number of distinct tuples \vec{d} such that $\mathbf{db} \models q'(\vec{c}, \vec{d})$.

We say that q admits parsimonious counting if it admits parsimonious counting on some sequence \vec{x} of bound variables.

Significantly, since Definition 8 contains a condition that must hold for every database instance \mathbf{db} , it does not give us an efficient procedure for deciding whether a given self-join-free query $q(\vec{z})$ admits parsimonious counting.

We now give some examples. From the proof of Proposition 7 and the paragraph after that proof, it follows that under standard complexity assumptions, for $k \geq 2$,

$$q_k(z) := \exists x_1 \cdots \exists x_k \exists y Z(z) \wedge \bigwedge_{i=1}^k (R_i(x_i, y) \wedge S_i(x_i, y))$$

does not admit parsimonious counting, even though $q_k(z)$ has a consistent first-order rewriting. The following example shows a query $q(z)$ that does not admit parsimonious counting, but for which $\mathbf{cqacnt}(\text{body}(q), z)$ can be computed in first-order logic with a counting step that is slightly more involved than what is allowed in parsimonious counting.

► **Example 9.** Let $q(z) = \exists x \exists y R(\underline{z}, x) \wedge S(x, y)$ and $q^*(z, x, y) = R(\underline{z}, x) \wedge S(x, y)$. We first argue that $q(z)$ does not admit parsimonious counting. Let \mathbf{db} be the following database instance:

R	\underline{z} x	S	x y
	c_1 a		a d
	c_2 a		a e
	c_2 b		b f

It can be verified that on this database instance, $\mathbf{cqacnt}(q^*, z)$ must return $(c_1, [2, 2])$ and $(c_2, [1, 2])$. We next show the answer to q^* on \mathbf{db} :

$q^*(\mathbf{db})$	z x y
	c_1 a d
	c_1 a e
	c_2 a d
	c_2 a e
	c_2 b f

The correct upper bound of 2 in $(c_2, [1, 2])$ could only be obtained by counting, within the c_2 -group, the number of distinct $\langle x \rangle$ -values. However, such a counting would conclude an incorrect upper bound of 1 for the c_1 -group. It is now correct to conclude that $q(z)$ does not admit parsimonious counting.

The lower and upper bounds can be obtained from $q^*(\mathbf{db})$ by a counting step that is only slightly more involved than what is allowed in parsimonious counting. First, construct the following relation where $\tilde{R}(c_j, v \mid n)$ means that $\mathbf{cnt}(q^*, z)$ returns (c_j, n) on a repair that contains $R(c_j, v)$.

\tilde{R}	\underline{z} x	
	c_1 a	2
	c_2 a	2
	c_2 b	1

These counts can be obtained from $q^*(\mathbf{db})$ by counting the number of distinct y -values within each zx -group. Next, the lower and upper bounds are obtained as the minimal and maximal counts within each z -group.

Note incidentally that for $q_0(z) := \exists x \exists y R(\underline{z}, x) \wedge T(\underline{z}, x) \wedge S(\underline{x}, y)$, which is obtained from $q(z)$ by adding $T(\underline{z}, x)$, we have that q_0 admits parsimonious counting. The change occurs because if $\mathbf{db} \models_{\text{cqa}} q_0(c)$, then there exists a unique value a such that $\mathbf{db} \models \forall x (R(\underline{c}, x) \rightarrow x = a)$ and $\mathbf{db} \models \forall x (T(\underline{c}, x) \rightarrow x = a)$. That is, the only blocks that can contribute to $\text{cqcant}(\text{body}(q_0), z)$ have cardinality 1. This means that range semantics reduces to counting on a consistent database instance. \lrcorner

5 The Class Cparsimony

The notion of parsimonious counting is a semantic property defined for conjunctive queries. A natural question is to syntactically characterize the class of conjunctive queries that admit parsimonious counting. In this paper, we will answer this question under the restriction that queries are self-join-free. This is the best we can currently hope for, because consistent query answering for primary keys and conjunctive queries with self-joins is a notorious open problem for which no tools are known (e.g., attack graphs are not helpful in the presence of self-joins). We now define our new syntactic class Cparsimony, which uses the following notion of *frozen variable*.

► **Definition 10** (Frozen variable). *Let $q(\vec{z})$ be a self-join-free conjunctive query. We say that a bound variable y of $q(\vec{z})$ is frozen in q if there exists a sequential proof of $\mathcal{K}(q) \models \emptyset \rightarrow y$ such that $F \not\stackrel{q}{\rightsquigarrow} y$ for every atom F that occurs in the sequential proof. We write $\text{frozen}(q)$ for the set of all bound variables of $\text{vars}(q)$ that are frozen in q . A bound variable that is not frozen in q is called nonfrozen in q .*

► **Example 11.** Let $q(z) = \exists x R(\underline{z}, x) \wedge S(\underline{z}, x)$. We have $R \not\stackrel{q}{\rightsquigarrow} x$. Therefore, $\langle R(\underline{z}, x) \rangle$ is a sequential proof of $\mathcal{K}(q) \models \emptyset \rightarrow x$ that uses no atom attacking x . Hence, x is frozen. Note here that z is free, hence $\mathcal{K}(q) \models \emptyset \rightarrow z$ by definition. \lrcorner

► **Definition 12** (The class Cparsimony). *We define Cparsimony as the set of self-join-free conjunctive queries $q(\vec{z})$ satisfying the following conditions:*

- (I) *the attack graph of $q(\vec{z})$ is acyclic and contains no strong attacks; and*
- (II) *there is a tuple \vec{x} of bound variables of $q(\vec{z})$ such that:*
 - (1) *every component³ of $q(\vec{z})$'s attack graph contains an unattacked atom R such that $\mathcal{K}(q) \models \vec{x} \rightarrow \text{Key}(R)$; and*
 - (2) *for every atom R in $\text{body}(q(\vec{z}))$, every (possibly empty) path in the query graph of $q(\vec{z})$ between a variable of $\text{notKey}(R)$ and a variable of \vec{x} uses a variable in $\text{Key}(R) \cup \text{frozen}(q)$.*

We will say that such an \vec{x} is an id-set for $q(\vec{z})$. We will say that an id-set \vec{x} is minimal if any sequence obtained from \vec{x} by omitting one or more variables is no longer an id-set.

Informally, id-sets \vec{x} will play the role of \vec{x} in Definition 8: they identify the values that have to be counted within each \vec{z} -group to obtain range-consistent answers.

³ Whenever we use the term component, we mean a maximal weakly connected component.



■ **Figure 2** Attack graph (left) and query graph (right) of $q(z) = \exists x \exists y_1 \exists y_2 \exists y_3 \exists v \exists w R(x, y_1) \wedge S(x, y_2) \wedge T(y_1, y_2, y_3, z) \wedge P(v, w)$.

We now illustrate Definition 12 by some examples. Then Proposition 17 implies that every query $q(\vec{z})$ in Cparsimony has a unique minimal id-set that can be easily constructed from $q(\vec{z})$'s attack graph. Finally, Proposition 18 settles the complexity of checking membership in Cparsimony .

► **Example 13.** In the paragraph following the proof of Proposition 7, we introduced the query $q(z) = \exists x_1 \exists x_2 \exists y Z(z) \wedge \bigwedge_{i=1}^2 (R_i(x_i, y) \wedge S_i(x_i, y))$. The edge-set of $q(z)$'s attack graph is empty. No variable is frozen. According to condition III1 in Definition 12, every id-set (if any) must contain x_1 . However, no id-set can contain x_1 , because for the atom $R_2(x_2, y)$, the edge $\{y, x_1\}$ in the query graph is a path between a variable of $\text{notKey}(R_2)$ and x_1 that uses no variable of $\text{Key}(R_2)$. We conclude that $q(z)$ is not in Cparsimony . ◻

► **Example 14.** The query $q(z) = \exists x \exists y \exists v R(x, y) \wedge S(y, v) \wedge T(v, y) \wedge P_1(z, y) \wedge P_2(z, y)$ belongs to Cparsimony . The attack graph of $q(z)$ has a single attack from S to T . The query graph of $q(z)$ has two undirected edges: $\{x, y\}$ and $\{y, v\}$. The variable y is frozen, because $\langle P_1(z, y) \rangle$ is a sequential proof of $\mathcal{K}(q) \models \emptyset \rightarrow y$ (note here that z is free), and $P_1 \not\stackrel{q}{\rightarrow} y$.

It can be verified that $\langle x \rangle$ is an id-set. Note that $\langle y, x \rangle$ is a path in the query graph between $y \in \text{notKey}(T)$ and x that uses no variable of $\text{Key}(T) = \{v\}$. However, that path uses the frozen variable y . ◻

► **Example 15.** Let $q(z) = \exists x \exists y_1 \exists y_2 \exists y_3 \exists v \exists w R(x, y_1) \wedge S(x, y_2) \wedge T(y_1, y_2, y_3, z) \wedge P(v, w)$. The attack graph and the query graph of $q(z)$ are shown in Fig. 2. We now argue that $q(z)$ is in Cparsimony . First, the attack graph of q is acyclic and contains no strong attacks. We next argue that xv is an id-set for q . The attack graph of $q(z)$ has two components. Condition III1 in Definition 12 is obviously satisfied for $\vec{x} = xv$ since $\mathcal{K}(q) \models xv \rightarrow v$ and $\mathcal{K}(q) \models xv \rightarrow x$. It is easily verified that condition II2 is also verified. In particular, for the atom $T(y_1, y_2, y_3, z)$, every path between y_3 and x uses either y_1 or y_2 . ◻

► **Example 16.** Let $q(z) = \exists x \exists y R_1(x, y, z) \wedge R_2(x, y) \wedge S_1(y, x) \wedge S_2(y, x)$. The attack graph of $q(z)$ contains no edges and, thus, is acyclic and has four components. It can be verified that no variable is frozen. We claim that $q(z)$ is not in Cparsimony , because it has no id-set. Indeed, from condition III1 in Definition 12, it follows that every id-set must contain either x or y (or both). For the atom $S_1(y, x)$, the empty path is a path between a variable in $\text{notKey}(S_1)$ to x that uses no variable in $\text{Key}(S_1)$. It follows by condition II2 that no id-set can contain x . From $R_2(x, y)$, by similar reasoning, we conclude that no id-set can contain y . It follows that $q(z)$ has no id-set. ◻

23:12 CQA for Primary Keys and CQs with Counting

► **Proposition 17.** *Let $q(\vec{z})$ be a query in Cparsimony, and let \vec{x} be a minimal id-set for it. Let $N = \bigcup\{\text{notKey}(R) \mid R \in q\}$. Let V be a \subseteq -minimal subset of $\text{vars}(q)$ that includes, for every unattacked atom R of q , every bound variable of $\text{Key}(R) \setminus N$. Then,*

1. $V = \text{vars}(\vec{x})$; and
2. whenever R, S are unattacked atoms that are weakly connected in $q(\vec{z})$'s attack graph, $\text{Key}(R) \cap \vec{x} = \text{Key}(S) \cap \vec{x}$.

► **Proposition 18.** *The following decision problem is in quadratic time: Given a self-join-free conjunctive query $q(\vec{z})$, decide whether or not $q(\vec{z})$ belongs to Cparsimony.*

6 The Class Cparsimony Admits Parsimonious Counting

In this section, we show the if-direction of Theorem 2, which is the following theorem.

► **Theorem 19.** *Every self-join-free conjunctive query in Cparsimony admits parsimonious counting.*

We use a number of helping lemmas and constructs. The following lemma says that if \vec{x} is an id-set of a query $q(\vec{z})$ in Cparsimony, then for a consistent database \mathbf{db} , the answers to $\text{cnt}(\text{body}(q), \vec{z})$ can be obtained by counting the number of distinct \vec{x} -values within each \vec{z} -group, while variables not in $\vec{x} \cdot \vec{z}$ can be ignored.

► **Lemma 20.** *Let $q(\vec{z}, \vec{x}, \vec{w})$ be a full self-join free conjunctive query, in which notation it is understood that \vec{z} , \vec{x} and \vec{w} are disjoint, duplicate-free tuples of variables. Assume that the query $\exists \vec{x} \vec{w} [q]$ belongs to Cparsimony and that \vec{x} is an id-set for it. Let \mathbf{db} be a consistent database instance. For all tuples \vec{a} and \vec{b} of constants, of arities $|\vec{z}|$ and $|\vec{x}|$ respectively, for all tuples \vec{c}_1 and \vec{c}_2 of arity $|\vec{w}|$, if $\mathbf{db} \models q(\vec{a}, \vec{b}, \vec{c}_1)$ and $\mathbf{db} \models q(\vec{a}, \vec{b}, \vec{c}_2)$, then $\vec{c}_1 = \vec{c}_2$.*

We now present the notion of *optimistic repair*, which was originally introduced by Fuxman [18]. Informally, a repair \mathbf{r} of a database \mathbf{db} is an optimistic repair with respect to a conjunctive query $q(\vec{z})$ if every tuple that is an answer to $q(\vec{z})$ on \mathbf{db} is also an answer to $q(\vec{z})$ on \mathbf{r} . The converse obviously holds true because conjunctive queries are monotone and repairs are subsets of the original database instance.

► **Definition 21 (Optimistic repair).** *Let $q(\vec{x})$ be a conjunctive query. Let \mathbf{db} be a database instance. We say that a repair \mathbf{r} of \mathbf{db} is an optimistic repair with respect to $q(\vec{x})$ if for every tuple \vec{a} of constants, of arity $|\vec{x}|$, $\mathbf{db} \models q(\vec{a})$ implies $\mathbf{r} \models q(\vec{a})$ (the converse implication is obviously true).*

The following lemma gives a sufficient condition for the existence of optimistic repairs.

► **Lemma 22.** *Let $q(\vec{z})$ be a self-join free conjunctive query in Cparsimony, and let \vec{x} be a minimal id-set for it. Let $q'(\vec{z}, \vec{x})$ be the query $\exists \vec{x} [q]$. Let \mathbf{db} be a database instance, and \vec{c} a tuple of constants, of arity $|\vec{z}|$, such that $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$. Then, \mathbf{db} has an optimistic repair with respect to $q'_{[\vec{z} \rightarrow \vec{c}]}$.*

We now present the notion of *pessimistic repair*, also borrowed from [18]. Informally, a repair of a database \mathbf{db} is a pessimistic repair with respect to a conjunctive query $q(\vec{z})$ if every answer to $q(\vec{z})$ on \mathbf{r} is a consistent answer to $q(\vec{z})$ on \mathbf{db} . The converse trivially holds true.

► **Definition 23 (Pessimistic repair).** *Let $q(\vec{x})$ be a conjunctive query. Let \mathbf{db} be a database instance. We say that a repair \mathbf{r} of \mathbf{db} is a pessimistic repair with respect to $q(\vec{x})$ if for every tuple \vec{a} of constants, of arity $|\vec{x}|$, if $\mathbf{r} \models q(\vec{a})$, then $\mathbf{db} \models_{\text{cqa}} q(\vec{a})$.*

The following lemma gives a sufficient condition for the existence of pessimistic repairs.

► **Lemma 24.** *Let $q(\vec{z})$ be a self-join free conjunctive query in Cparsimony, and let \vec{x} be a minimal id-set for it. Let \mathbf{db} be a database instance, and \vec{c} a tuple of constants, of arity $|\vec{z}|$, such that $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$. Then, \mathbf{db} has a pessimistic repair with respect to $q'_{[\vec{z} \rightarrow \vec{c}]}$.*

The following example illustrates the preceding constructs and lemmas.

► **Example 25.** Let $q(z) = \exists x \exists y \exists v R(\underline{x}, y) \wedge S(\underline{y}, v, z) \wedge T(\underline{y}, v)$. Let \mathbf{db} be the following database instance:

R	\underline{x} y	S	\underline{y} v z	T	\underline{y} v
$\frac{a_1}{a_2}$	$\frac{b_1}{b_2}$	$\frac{b_1}{b_2}$	$\frac{c_1}{c_2}$ g_1	$\frac{b_1}{b_2}$	$\frac{c_1}{c_2}$
$\frac{a_3}{a_4}$	$\frac{b_2}{b_3}$	$\frac{b_2}{b_3}$	$\frac{c_2}{c_3}$ g_2	$\frac{b_3}{b_3}$	$\frac{c_2}{c_3}$

Clearly, \mathbf{db} has two repairs, which are $\mathbf{r}_1 := \mathbf{db} \setminus \{S(\underline{b}_2, \underline{b}_2, g_2)\}$ and $\mathbf{r}_2 := \mathbf{db} \setminus \{S(\underline{b}_2, \underline{b}_2, g_1)\}$.

We first determine the answers to $\mathbf{cqa}(\text{body}(q), z)$ on \mathbf{db} in a naive way without using parsimonious counting, but by enumerating repairs. To this end, let $q^*(z, x, y, v) = R(\underline{x}, y) \wedge S(\underline{y}, v, z) \wedge T(\underline{y}, v)$. We have:

$$q^*(\mathbf{r}_1) = \{(g_1, a_1, b_1, c_1), (g_1, a_2, b_2, c_2), (g_1, a_3, b_2, c_2), (g_2, a_4, b_3, c_3)\}$$

$$q^*(\mathbf{r}_2) = \{(g_1, a_1, b_1, c_1), (g_2, a_2, b_2, c_2), (g_2, a_3, b_2, c_2), (g_2, a_4, b_3, c_3)\}$$

The value g_1 occurs in 3 tuples of $q^*(\mathbf{r}_1)$, and in one tuple of $q^*(\mathbf{r}_2)$. On the other hand, g_2 occurs in one tuple of $q^*(\mathbf{r}_1)$, and in 3 tuples of $q^*(\mathbf{r}_2)$. It follows that $(g_1, [1, 3])$ and $(g_2, [1, 3])$ are the answers to $\mathbf{cqa}(\text{body}(q), z)$ on \mathbf{db} .

It can be verified that $q(z) \in \text{Cparsimony}$ with an id-set $\vec{x} = \langle x \rangle$. We next compute $\mathbf{cqa}(\text{body}(q), z)$ on \mathbf{db} by means of parsimonious counting. To this end, let $q'(z, x) = \#x[q]$, and let $\varphi(z, x)$ be a consistent first-order rewriting for $q'(z, x)$. If we execute these queries on \mathbf{db} , we obtain:⁴

$$q'(\mathbf{db}) = \{(g_1, a_1), (g_1, a_2), (g_1, a_3), (g_2, a_2), (g_2, a_3), (g_2, a_4)\}$$

$$\varphi(\mathbf{db}) = \{(g_1, a_1), (g_2, a_4)\}$$

As stated in Theorem 19, the set $q'(\mathbf{db})$ yields the upper bound 3 for g_1 and g_2 , and the set $\varphi(\mathbf{db})$ yields the lower bound 1 for g_1 and g_2 . It is important to understand that parsimonious counting obtains these bounds directly on \mathbf{db} , without computing any repair.

We elaborate this example further to illustrate the constructs of optimistic and pessimistic repairs. We have:

$$q'(\mathbf{r}_1) = \{(g_1, a_1), (g_1, a_2), (g_1, a_3), (g_2, a_4)\}$$

$$q'(\mathbf{r}_2) = \{(g_1, a_1), (g_2, a_2), (g_2, a_3), (g_2, a_4)\}$$

Note that the consistent answer to $q'(z, x)$ on \mathbf{db} (i.e., the set $\varphi(\mathbf{db})$ used previously) is equal to $q'(\mathbf{r}_1) \cap q'(\mathbf{r}_2) = \{(g_1, a_1), (g_2, a_4)\}$. We see that \mathbf{r}_1 is an optimistic repair with respect to $q'(z, x)_{[z \rightarrow g_1]}$, and a pessimistic repair with respect to $q'(z, x)_{[z \rightarrow g_2]}$. On the other hand, \mathbf{r}_2 is an optimistic repair with respect to $q'(z, x)_{[z \rightarrow g_2]}$, and a pessimistic repair with respect to $q'(z, x)_{[z \rightarrow g_1]}$. \square

⁴ $\varphi(\mathbf{db})$ is a shorthand for the set of all tuples (c, d) such that $\mathbf{db} \models \varphi(c, d)$.

Proof of Theorem 19. Let $q(\vec{z}) \in \text{Cparsimony}$. We have to prove that $q(\vec{z})$ admits parsimonious counting. Since $q(\vec{z}) \in \text{Cparsimony}$, we can assume an id-set \vec{x} for $q(\vec{z})$. It suffices to show that conditions A, B, and C in Definition 8 are satisfied for this choice of \vec{x} . As in Definition 8, let $q'(\vec{z}, \vec{x}) = \# \vec{x} [q]$.

Since $q(\vec{z})$ is in **Cparsimony**, it has an acyclic attack graph. It follows from Theorem 4 that $q(\vec{z})$ has a consistent first-order rewriting. Thus, condition A in Definition 8 is satisfied. It is known [29] that the attack graph of $q'(\vec{z}, \vec{x})$ is a subgraph of the attack graph of $q(\vec{z})$. Informally, no new attacks are introduced when bound variables are made free. It follows that $q'(\vec{z}, \vec{x})$ has an acyclic attack graph, and therefore, by Theorem 4, a consistent first-order rewriting. Thus, condition B in Definition 8 is satisfied. In the remainder of the proof, we show that condition C in Definition 8 is satisfied. To this end, let \mathbf{db} be an arbitrary database instance.

Let \vec{c} be a tuple of constants such that $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$. Let D be the active domain of \mathbf{db} . Let f be a function that maps every subset \mathbf{s} of \mathbf{db} to the cardinality of the set $\{\vec{a} \in D^{|\vec{x}|} \mid \mathbf{s} \models q'(\vec{c}, \vec{a})\}$. Clearly, for every repair \mathbf{r} of \mathbf{db} , we have $\mathbf{r} \subseteq \mathbf{db}$ and hence, since conjunctive queries are monotone, $f(\mathbf{r}) \leq f(\mathbf{db})$. Moreover, since repairs are consistent, it follows by Lemma 20 that for every repair \mathbf{r} of \mathbf{db} , if (\vec{c}, i) is an answer to the query $\text{cnt}(\text{body}(q), \vec{z})$ on \mathbf{r} , then $i = f(\mathbf{r})$.

By Lemma 22, we can assume an optimistic repair \mathbf{o} of \mathbf{db} with respect to $q'(\vec{z}, \vec{x})_{[\vec{z} \rightarrow \vec{c}]}$. By Definition 21 of optimistic repair, for every tuple \vec{a} of constants, of arity $|\vec{x}|$, we have $\mathbf{o} \models q'(\vec{c}, \vec{a})$ if and only if $\mathbf{db} \models q'(\vec{c}, \vec{a})$. It follows $f(\mathbf{o}) = f(\mathbf{db})$. Consequently, for every repair \mathbf{r} of \mathbf{db} , $f(\mathbf{r}) \leq f(\mathbf{o})$. It follows that for some lower bound m , we have that $(\vec{c}, [m, f(\mathbf{db})])$ is an answer to $\text{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{db} .

By Lemma 24, we can assume a pessimistic repair \mathbf{p} of \mathbf{db} with respect to $q'(\vec{z}, \vec{x})_{[\vec{z} \rightarrow \vec{c}]}$. Let $\varphi(\vec{z}, \vec{x})$ be a consistent first-order rewriting of $q'(\vec{z}, \vec{x})$. By Definition 23 of pessimistic repair, the following hold:

- $\mathbf{p} \models q(\vec{c}, \vec{a})$ if and only if $\mathbf{db} \models \varphi(\vec{c}, \vec{a})$. Therefore, $f(\mathbf{p})$ is the cardinality of the set $S := \{\vec{a} \in D^{|\vec{x}|} \mid \mathbf{db} \models \varphi(\vec{c}, \vec{a})\}$.
- for every repair \mathbf{r} of \mathbf{db} , $f(\mathbf{p}) \leq f(\mathbf{r})$.

It follows that there is an upper bound n such that that $(\vec{c}, [|S|, n])$ is an answer to $\text{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{db} . Putting everything together, we obtain that $(\vec{c}, [|S|, f(\mathbf{db})])$ is an answer to $\text{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{db} . From this, it is correct to conclude that condition C in Definition 8 is satisfied. This concludes the proof. \blacktriangleleft

7 Completeness of the Class Cparsimony

In this section, we show the only-if-direction of Theorem 2, which is the following theorem.

► **Theorem 26.** *Every self-join-free conjunctive query that admits parsimonious counting belongs to Cparsimony.*

The following three lemmas state some properties of queries $q(\vec{z})$ that admit parsimonious counting on some \vec{x} .

► **Lemma 27.** *Let $q(\vec{z})$ be a self-join-free conjunctive query. If $q(\vec{z})$ admits parsimonious counting, then the attack graph of $q(\vec{z})$ is acyclic.*

► **Lemma 28.** *Let $q(\vec{z})$ be a self-join-free conjunctive query. Let \vec{x} be a (possibly empty) sequence of bound variables of $q(\vec{z})$. If $q(\vec{z})$ admits parsimonious counting on \vec{x} , then the attack graph of $q'(\vec{z}, \vec{x})$ has no strong attack.*

► **Lemma 29.** *Let $q(\vec{z})$ be self-join-free conjunctive query whose attack graph is acyclic. Let \vec{x} be a (possibly empty) sequence of bound variables of $q(\vec{z})$. If $q(\vec{z})$ admits parsimonious counting on \vec{x} , then \vec{x} satisfies condition II1 in Definition 12.*

The following two lemmas, and their corollary, concern condition II2 in Definition 12.

► **Lemma 30.** *Let $q(\vec{z})$ be a self-join-free conjunctive query. Let \vec{x} be a (possibly empty) sequence of bound variables of $q(\vec{z})$, and let $q'(\vec{z}, \vec{x}) = \# \vec{x} [q]$. Let \vec{c} a tuple of constants of arity $|\vec{z}|$. If $q(\vec{z})$ admits parsimonious counting on \vec{x} , then for every database instance \mathbf{db} , if $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$, then \mathbf{db} has an optimistic repair with respect to $q'_{[\vec{z} \rightarrow \vec{c}]}$.*

Proof. Assume that $q(\vec{z})$ admits parsimonious counting on \vec{x} . Let \mathbf{db} be a database instance such that $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$. Let $(\vec{c}, [m, n])$ be an answer to $\mathbf{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{db} . Define

$$\mathcal{D} := \{\vec{d} \in D^{|\vec{x}|} \mid \mathbf{db} \models q'(\vec{c}, \vec{d})\}, \quad (2)$$

where D be the active domain of \mathbf{db} . By our hypothesis that $q(\vec{z})$ admits parsimonious counting on \vec{x} , it follows by condition C in Definition 8 that

$$n = |\mathcal{D}|. \quad (3)$$

By Definition 6, we can assume a repair \mathbf{r} of \mathbf{db} such that (\vec{c}, n) is an answer to $\mathbf{cnt}(\text{body}(q), \vec{z})$ on \mathbf{r} . Since \mathbf{r} is consistent, we have that $(\vec{c}, [n, n])$ is an answer to $\mathbf{cqacnt}(\text{body}(q), \vec{z})$ on \mathbf{r} . Define

$$\mathcal{R} := \{\vec{d} \in D^{|\vec{x}|} \mid \mathbf{r} \models q'(\vec{c}, \vec{d})\}. \quad (4)$$

By our hypothesis that $q(\vec{z})$ admits parsimonious counting on \vec{x} , it follows by condition C in Definition 8 that

$$n = |\mathcal{R}|. \quad (5)$$

Since conjunctive queries are monotone and $\mathbf{r} \subseteq \mathbf{db}$, it follows $\mathcal{R} \subseteq \mathcal{D}$. Since $|\mathcal{R}| = |\mathcal{D}|$ by (3) and (5), it follows $\mathcal{R} = \mathcal{D}$. From $\mathcal{D} \subseteq \mathcal{R}$, it follows that \mathbf{r} is an optimistic repair with respect to $q'_{[\vec{z} \rightarrow \vec{c}]}$. ◀

► **Lemma 31.** *Let $q(\vec{z})$, \vec{x} , $q'(\vec{z}, \vec{x})$, and \vec{c} be as in the statement of Lemma 30. Assume that \vec{x} violates condition II2 in Definition 12. Then, there exists a database \mathbf{db} such that $\mathbf{db} \models_{\text{cqa}} q(\vec{c})$, but \mathbf{db} has no optimistic repair with respect to $q'_{[\vec{z} \rightarrow \vec{c}]}$.*

► **Corollary 32.** *Let $q(\vec{z})$ be a self-join-free conjunctive query. Let \vec{x} be a sequence of distinct bound variables of $q(\vec{z})$, and let $q'(\vec{z}, \vec{x}) = \# \vec{x} [q]$. If $q(\vec{z})$ admits parsimonious counting on \vec{x} , then \vec{x} satisfies condition II2 in Definition 12.*

Proof. Immediately from Lemmas 30 and 31. ◀

Finally, we need the following result.

► **Lemma 33.** *Let $q(\vec{z})$ be a self-join-free conjunctive query. Let \vec{x} be a sequence of distinct bound variables of $q(\vec{z})$. Let $q'(\vec{z}, \vec{x}) = \# \vec{x} [q]$. Assume that \vec{x} satisfies condition II2 in Definition 12. If the attack graph of $q(\vec{z})$ has a strong attack from an atom R to an atom S , then the attack graph of $q'(\vec{z}, \vec{x})$ has a strong attack from R to S .*

Before giving a proof of Theorem 26, we illustrate the preceding results with an example.

23:16 CQA for Primary Keys and CQs with Counting

► **Example 34.** Let $q(z) = \exists x \exists y R(\underline{x}, z, y) \wedge S(\underline{y}, x) \wedge T(\underline{y}, x)$. We will argue that $q(z)$ is not in Cparsimony , and then illustrate that it does not admit parsimonious counting.

The only edges in the attack graph of q are (R, S) and (R, T) . Assume for the sake of contradiction that $q \in \text{Cparsimony}$. Then, following Proposition 17, the minimal id-set of $q(\vec{z})$ is $\langle \rangle$. However, since $\mathcal{K}(q(z)) \equiv \{x \rightarrow y, y \rightarrow x, \emptyset \rightarrow z\}$, condition III1 in Definition 12 is violated for $\vec{x} = \langle \rangle$. We conclude by contradiction that $q \notin \text{Cparsimony}$.

We now argue, without using Theorem 26, that $q(z)$ does not admit parsimonious counting. Conditions A and B in Definition 8 of parsimonious counting are satisfied for every choice of \vec{x} in $\{\langle \rangle, \langle x \rangle, \langle y \rangle, \langle x, y \rangle\}$. However, we will show that condition C is not satisfied. To this end, let \vec{x} be a sequence of bound variables of $q(z)$. Let $q'(z, \vec{x}) = \# \vec{x}[q]$. First, suppose that $\vec{x} \in \{\langle x \rangle, \langle x, y \rangle\}$. Consider the following database instance \mathbf{db} :

R	\underline{x}	z	y	S	\underline{y}	x	T	\underline{y}	x
	a	d	e		e	a		e	a
	b	d	e		e	b		e	b
	c	d	f		f	c		f	c

We have that $(d, [1, 2])$ is an answer to $\mathbf{cqacnt}(\text{body}(q), z)$, but it can be easily verified that $|q'(\mathbf{db})| = 3$, which is distinct from the upper bound 2.

Assume next that $\vec{x} \in \{\langle y \rangle, \langle x, y \rangle\}$. Consider the following database instance \mathbf{db} :

R	\underline{x}	z	y	S	\underline{y}	x	T	\underline{y}	x
	a	d	e		e	a		e	a
	a	d	f		f	a		f	a
	b	d	g		g	b		g	b

Now we have that $(d, [2, 2])$ is an answer to $\mathbf{cqacnt}(\text{body}(q), z)$, but $|q'(\mathbf{db})| = 3$.

The only remaining case to be considered is $\vec{x} = \langle \rangle$. In that case $q' = q$. Consider the following database instance \mathbf{db} :

R	\underline{x}	z	y	S	\underline{y}	x	T	\underline{y}	x
	a	d	e		e	a		e	a
	b	d	f		f	b		f	b

Since \mathbf{db} is a consistent database instance, the only repair of \mathbf{db} is \mathbf{db} itself. We have that $(d, [2, 2])$ is an answer to $\mathbf{cqacnt}(\text{body}(q), z)$ on \mathbf{db} . It can be easily verified that $|q'(\mathbf{db})| = 1$, which is distinct from the upper bound 2.

Finally, we claim (without proof) that 2-DIMENSIONAL MATCHING (2DM) can be first-order reduced to computing $\mathbf{cqacnt}(\text{body}(q), z)$. Therefore, since 2DM is NL-hard [13], $q(z)$ cannot admit parsimonious counting under standard complexity assumptions. ◻

Proof of Theorem 26. Assume that $q(\vec{z})$ admits parsimonious counting. Then, $q(\vec{z})$ has a tuple \vec{x} of bound variables such that for the query $q'(\vec{z}, \vec{x}) := \# \vec{x}[q]$, the conditions A, B, and C in Definition 8 are satisfied. From conditions A and B, it follows by Theorem 4 that $q(\vec{z})$ and $q'(\vec{z}, \vec{x})$ have acyclic attack graphs. By Lemma 29, condition III1 in Definition 12 is satisfied for \vec{x} . By Corollary 32, condition II2 in Definition 12 is satisfied by \vec{x} . By Lemma 28, the attack graph of $q'(\vec{z}, \vec{x})$ has no strong attack. By Lemma 33, it is now correct to conclude that the attack graph of $q(\vec{z})$ has no strong attack either, and thus condition I in Definition 12 is satisfied. Since we have shown that $q(\vec{z})$ satisfies all conditions in Definition 12, we conclude $q(\vec{z}) \in \text{Cparsimony}$. ◀

8 Comparison with the Class Cforest

In this section, we introduce Cforest and show $\text{Cforest} \subsetneq \text{Cparsimony}$ without making use of Theorem 1. Theorem 1 then follows by Theorem 19.

► **Definition 35** (Cforest). *Let $q(\vec{z})$ be a self-join-free conjunctive query. The Fuxman graph of q is a directed graph whose vertices are the atoms of q . There is a directed edge from an atom R to an atom S if $R \neq S$ and $\text{notKey}(R)$ contains a bound variable that also occurs in S . The class Cforest contains all (and only) self-join free conjunctive queries $q(\vec{z})$ whose Fuxman graph is a directed forest satisfying, for every directed edge from R to S , $\text{Key}(S) \setminus \text{free}(q) \subseteq \text{notKey}(R)$.*

► **Theorem 36.** *Cforest is a strict subset of Cparsimony.*

9 Conclusion and Future Work

In his PhD thesis, Fuxman [18] defined a syntactically restricted class of self-join-free conjunctive queries, called Cforest, and showed that for every query in Cforest, consistent answers are first-order computable, and range-consistent answers are computable in first-order logic followed by a simple aggregation step. Our notion of “parsimonious counting” captures the latter computation for counting. Later, Koutris and Wijsen [29] syntactically characterized the class of *all* self-join-free conjunctive queries with a consistent first-order rewriting, which strictly includes Cforest. However, it remained an open problem to syntactically characterize the class of *all* self-join-free conjunctive queries that admit parsimonious counting. In this paper, we determined the latter class, named it Cparsimony, and showed that it strictly includes Cforest.

We now list some open problems for future research. In Definition 8 of parsimonious counting, we required that $q(\vec{z})$ has a consistent first-order rewriting. It is known [32] that there are self-join-free conjunctive queries, without consistent first-order rewriting, that have a consistent rewriting in Datalog. We could relax Definition 8 by requiring the existence of a consistent rewriting in Datalog, rather than in first-order logic. It is an open question to syntactically characterize the self-join-free conjunctive queries that admit parsimonious counting under such a relaxed definition.

Another open question is to characterize the complexity of $\mathbf{cqacnt}(q(\vec{z}, \vec{w}), \vec{z})$ for every full self-join-free conjunctive query q and choice of free variables \vec{z} . It is easily verified that the complexity of computing the answers to $\mathbf{cqacnt}(q(\vec{z}, \vec{w}), \vec{z})$ is higher than computing the consistent answers to $q'(\vec{z}) := \exists \vec{w} [q]$ (because of the lower bound in range semantics). It remains an open question to characterize this complexity if $q'(\vec{z})$ is not in Cparsimony, even if it has a consistent first-order rewriting.

The notion of parsimonious counting does not require conjunctive queries to be self-join-free. An ambitious open problem is to syntactically characterize the class of all (i.e., not necessarily self-join-free) conjunctive queries that admit parsimonious counting. This problem is largely open, because it is already a notorious open problem to syntactically characterize the class of conjunctive queries that have a consistent first-order rewriting.

Another open question is to extend the results in the current paper to other aggregation operators than COUNT, including MAX, MIN, SUM, and AVG.

References

- 1 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.
- 2 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Scalar aggregation in FD-inconsistent databases. In *ICDT*, volume 1973 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2001.
- 3 Marcelo Arenas, Leopoldo E. Bertossi, Jan Chomicki, Xin He, Vijay Raghavan, and Jeremy P. Spinrad. Scalar aggregation in inconsistent databases. *Theor. Comput. Sci.*, 296(3):405–434, 2003.
- 4 Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
- 5 Leopoldo E. Bertossi. Database repairs and consistent query answering: Origins and further developments. In *PODS*, pages 48–58. ACM, 2019.
- 6 Leopoldo E. Bertossi, Loreto Bravo, Enrico Franconi, and Andrei Lopatenko. The complexity and approximation of fixing numerical attributes in databases under integrity constraints. *Inf. Syst.*, 33(4-5):407–434, 2008.
- 7 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011.
- 8 Marco Calautti, Marco Console, and Andreas Pieris. Counting database repairs under primary keys revisited. In *PODS*, pages 104–118. ACM, 2019.
- 9 Marco Calautti, Marco Console, and Andreas Pieris. Benchmarking approximate consistent query answering. In *PODS*, pages 233–246. ACM, 2021.
- 10 Marco Calautti, Leonid Libkin, and Andreas Pieris. An operational approach to consistent query answering. In *PODS*, pages 239–251. ACM, 2018.
- 11 Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider. Counting database repairs entailing a query: The case of functional dependencies. In *PODS*, pages 403–412. ACM, 2022.
- 12 Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider. Uniform operational consistent query answering. In *PODS*, pages 393–402. ACM, 2022.
- 13 Ashok K. Chandra, Larry J. Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM J. Comput.*, 13(2):423–439, 1984.
- 14 Akhil A. Dixit and Phokion G. Kolaitis. A SAT-based system for consistent query answering. In *SAT*, volume 11628 of *Lecture Notes in Computer Science*, pages 117–135. Springer, 2019.
- 15 Akhil A. Dixit and Phokion G. Kolaitis. Consistent answers of aggregation queries via SAT. In *ICDE*, pages 924–937. IEEE, 2022.
- 16 Zhiwei Fan, Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. LinCQA: Faster consistent query answering with linear time guarantees. *CoRR*, abs/2208.12339, 2022. [arXiv:2208.12339](https://arxiv.org/abs/2208.12339).
- 17 Gaëlle Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015.
- 18 Ariel Fuxman. *Efficient query processing over inconsistent databases*. PhD thesis, University of Toronto, 2007.
- 19 Ariel Fuxman, Elham Fazli, and Renée J. Miller. ConQuer: Efficient management of inconsistent databases. In *SIGMOD Conference*, pages 155–166. ACM, 2005.
- 20 Ariel Fuxman, Diego Fuxman, and Renée J. Miller. ConQuer: A system for efficient querying over inconsistent databases. In *VLDB*, pages 1354–1357. ACM, 2005.
- 21 Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. In *ICDT*, volume 3363 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 2005.
- 22 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- 23 Miika Hannula and Jef Wijsen. A dichotomy in consistent query answering for primary keys and unary foreign keys. In *PODS*, pages 437–449. ACM, 2022.

- 24 Aziz Amezian El Khalfioui, Jonathan Joertz, Dorian Labeeuw, Gaëtan Staquet, and Jef Wijsen. Optimization of answer set programs for consistent query answering by means of first-order rewriting. In *CIKM*, pages 25–34. ACM, 2020.
- 25 Aziz Amezian El Khalfioui and Jef Wijsen. Consistent query answering for primary keys and conjunctive queries with counting. *CoRR*, abs/2211.04134, 2022. [arXiv:2211.04134](https://arxiv.org/abs/2211.04134).
- 26 Benny Kimelfeld, Ester Livshits, and Liat Peterfreund. Counting and enumerating preferred database repairs. *Theor. Comput. Sci.*, 837:115–157, 2020.
- 27 Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient querying of inconsistent databases with binary integer programming. *Proc. VLDB Endow.*, 6(6):397–408, 2013.
- 28 Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. Consistent query answering for primary keys on path queries. In *PODS*, pages 215–232. ACM, 2021.
- 29 Paraschos Koutris and Jef Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.
- 30 Paraschos Koutris and Jef Wijsen. Consistent query answering for primary keys and conjunctive queries with negated atoms. In *PODS*, pages 209–224. ACM, 2018.
- 31 Paraschos Koutris and Jef Wijsen. First-order rewritability in consistent query answering with respect to multiple keys. In *PODS*, pages 113–129. ACM, 2020.
- 32 Paraschos Koutris and Jef Wijsen. Consistent query answering for primary keys in datalog. *Theory Comput. Syst.*, 65(1):122–178, 2021.
- 33 Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013.
- 34 Dany Maslowski and Jef Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In *ICDT*, pages 155–164. OpenProceedings.org, 2014.
- 35 Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.*, 64(2-3):209–246, 2012.
- 36 Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*, pages 179–190. ACM, 2010.
- 37 Jef Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3):6–16, 2019.