Consistent Query Answering for Primary Keys in Logspace

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

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

Abstract

We study the complexity of consistent query answering on databases that may violate primary key constraints. A repair of such a database is any consistent database that can be obtained by deleting a minimal set of tuples. For every Boolean query q, CERTAINTY(q) is the problem that takes a database as input and asks whether q evaluates to true on every repair. In [Koutris and Wijsen, ACM TODS, 2017], the authors show that for every self-join-free Boolean conjunctive query q, the problem CERTAINTY(q) is either in **P** or **coNP**-complete, and it is decidable which of the two cases applies. In this paper, we sharpen this result by showing that for every self-join-free Boolean conjunctive query q, the problem CERTAINTY(q) is either expressible in symmetric stratified Datalog (with some aggregation operator) or **coNP**-complete. Since symmetric stratified Datalog is in **L**, we thus obtain a complexity-theoretic dichotomy between **L** and **coNP**-complete. Another new finding of practical importance is that CERTAINTY(q) is on the logspace side of the dichotomy for queries q where all join conditions express foreign-to-primary key matches, which is undoubtedly the most common type of join condition.

2012 ACM Subject Classification Information systems → Relational database model; Information systems → Inconsistent data; Information systems → Incomplete data; Information systems → Integrity checking

Keywords and phrases conjunctive queries, consistent query answering, Datalog, primary keys

Digital Object Identifier 10.4230/LIPIcs.ICDT.2019.23

Related Version A full version of this paper is available on arXiv [22], `https://arxiv.org/abs/1810.03386`.

1 Motivation

Consistent query answering (CQA) with respect to primary key constraints is the following problem. Given a database **db** that may violate its primary key constraints, define a repair as any consistent database that can be obtained by deleting a minimal set of tuples from **db**. For every Boolean query q, the problem CERTAINTY(q) takes a database as input and asks whether q evaluates to true on every repair of **db**. In this paper, we focus on CERTAINTY(q) for queries q in the class sjfBCQ, the class of self-join-free Boolean conjunctive queries. For all Boolean first-order queries q, CERTAINTY(q) is in **coNP** and can thus be solved by expressive formalisms like answer set programming [27] and binary integer programming [18]. These solutions, however, are likely to be inefficient when CERTAINTY(q) also belongs to a lower complexity class. In particular, given a query q in sjfBCQ, it is decidable [20] whether CERTAINTY(q) is in the low complexity class **FO**. Moreover, if CERTAINTY(q) is in **FO**, then it is possible to construct a first-order query for solving CERTAINTY(q), which is also called a *consistent first-order rewriting for* q. This construction is detailed in [20, Section 5] and has already been implemented [30].

In [20], the authors also show that for every query q in sjfBCQ, the problem CERTAINTY(q) is either in **P** or **coNP**-complete, and it is decidable (in polynomial time in the size of q) which of the two cases applies. The authors show how to construct a polynomial-time algorithm for CERTAINTY(q) when it does not lie on the **coNP**-hard side of the dichotomy. Unfortunately, unlike for consistent first-order rewritings, this construction is complex and does not tell us what language would be appropriate for implementing CERTAINTY(q) when it is in $\mathbf{P} \setminus \mathbf{FO}$. In this paper, we improve this situation: we show that if CERTAINTY(q) is in **P**, then it can be implemented in symmetric stratified Datalog, which has deterministic logspace data complexity [10]. We thus sharpen the complexity dichotomy of [20] as follows: for every query q in sjfBCQ, CERTAINTY(q) is either in **L** or **coNP**-complete. It is significant that Datalog is used as a target language, because this allows using optimized Datalog engines for solving CERTAINTY(q) whenever the problem lies on the logspace side of the dichotomy. Rewriting into Datalog is generally considered a desirable outcome when consistent first-order rewritings do not exist (see, e.g., [5, page 193]). It is also worth noting that the SQL:1999 standard introduced linear recursion into SQL, which has been implemented in varying ways in existing DBMSs [31]. Since the Datalog programs in this paper use only linear recursion, they may be partially or fully implementable in these DBMSs.

Throughout this paper, we use the term *consistent database* to refer to a database that satisfies all primary-key constraints, while the term *database* refers to both consistent and inconsistent databases. This is unlike most database textbooks, which tend to say that databases must always be consistent. The following definition introduces the main focus of this paper; the complexity dichotomy of Theorem 2 is the main result of this paper.

▶ **Definition 1.** *Let q be a Boolean query. Let $\mathcal{L}$ be some logic. A* consistent $\mathcal{L}$ rewriting for q *is a Boolean query P in $\mathcal{L}$ such that for every database* **db**, *P is true in* **db** *if and only if q is true in every repair of* **db**. *If q has a consistent $\mathcal{L}$ rewriting, then we say that* CERTAINTY(q) *is expressible in $\mathcal{L}$.*

▶ **Theorem 2.** *For every self-join-free Boolean conjunctive query q,* CERTAINTY(q) *is either* **coNP***-complete or expressible in SymStratDatalog*$^{\text{min}}$ *(and thus in* **L***).*

The language *SymStratDatalog*$^{\text{min}}$ will be defined in Section 3; informally, the superscript min means that the language allows selecting a minimum (with respect to some total order) from a finite set of values. Since CERTAINTY(q) is **L**-complete for some queries $q \in$ sjfBCQ, the logspace upper bound in Theorem 2 is tight. The proof of Theorem 2 relies on novel constructs and insights developed in this paper. Compared to [20], significant new contributions are the notion of *garbage set* and the helping Lemmas 11 and 22.

Our second significant result in this paper focuses on consistent query answering for foreign-to-primary key joins. In Section 9, we define a subclass of sjfBCQ that captures foreign-to-primary key joins, which is undoubtedly the most common type of join. We show that CERTAINTY(q) lies on the logspace side of the dichotomy for *all* queries q in this class. Thus, for the most common type of joins and primary key constraints, CQA is highly tractable, a result that goes against a widely spread belief that CQA would be impractical because of its high computational complexity.

Organization Section 2 discusses related work. Section 3 defines our theoretical framework, including the notion of *attack graph*. To guide the reader through the technical development, Section 4 provides a high-level outline of where we are heading in this paper, including examples of the different graphs used. Section 5 introduces a special subclass of sjfBCQ, called *saturated queries*, and shows that each problem CERTAINTY(q) can be first-order reduced to

some CERTAINTY(q') where q' is saturated. Section 6 introduces the notion of M-graph, a graph at the schema-level, and its data-level instantiation, called $\rightsquigarrow$-graph. An important result, Lemma 11, relates cycles in attack graphs to cycles in M-graphs, for saturated queries only. Section 7 introduces the notion of garbage set for a subquery. Informally, garbage sets contain facts that can never make the subquery hold true, and thus can be removed from the database without changing the answer to CERTAINTY(q). Section 8 focuses on cycles in the M-graph of a query, and shows that garbage sets for such cycles can be computed and removed in symmetric stratified Datalog. At the end of Section 8, we have all ingredients for the proof of our main theorem. Finally, Section 9 shows that foreign-to-primary key joins fall on the logspace side of the dichotomy. The proofs of lemmas and theorems appear in [22].

2 Related Work

Consistent query answering (CQA) starts from the seminal work by Arenas, Bertossi, and Chomicki [2], and is the topic of a monograph by Bertossi [7]. The term CERTAINTY(q) was coined in [33] to refer to CQA for Boolean queries q on databases that violate primary keys, one per relation, which are fixed by q's schema. The complexity classification of CERTAINTY(q) for all $q \in$ sjfBCQ started with the ICDT 2005 paper of Fuxman and Miller [13, 14], and has attracted much research since then. These previous works (see [35] for a survey) were generalized by [19, 20], where it was shown that the set {CERTAINTY(q) | $q \in$ sjfBCQ} exhibits a **P-coNP**-complete dichotomy. Furthermore, it was shown that membership of CERTAINTY(q) in **FO** is decidable for queries q in sjfBCQ. The current paper culminates this line of research by showing that the dichotomy is actually between **L** and **coNP**-complete, and – even stronger – between expressibility in symmetric stratified Datalog (with some aggregation operator) and **coNP**-complete.

The complexity of CERTAINTY(q) for self-join-free conjunctive queries with negated atoms was studied in [21]. Little is known about CERTAINTY(q) beyond self-join-free conjunctive queries. For UCQ (i.e., unions of conjunctive queries, possibly with self-joins), Fontaine [12] showed that a **P-coNP**-complete dichotomy in the set {CERTAINTY(q) | q is a Boolean query in UCQ} implies Bulatov's dichotomy theorem for conservative CSP [9]. This relationship between CQA and CSP was further explored in [26]. The complexity of CQA for aggregation queries with respect to violations of functional dependencies has been studied in [3].

The counting variant of CERTAINTY(q), which is called #CERTAINTY(q), asks to determine the number of repairs that satisfy some Boolean query q. In [28], the authors show a **FP**-#**P**-complete dichotomy in {#CERTAINTY(q) | $q \in$ sjfBCQ}. For conjunctive queries q with self-joins, the complexity of #CERTAINTY(q) has been established for the case that all primary keys consist of a single attribute [29]. In recent years, CQA has also been studied beyond the setting of relational databases, in ontology-based knowledge bases [8, 23] and in graph databases [6].

3 Preliminaries

We assume an infinite total order (**dom**, $\leq$) of *constants*. We assume a set of *variables* disjoint with **dom**. If $\vec{x}$ is a sequence containing variables and constants, then vars($\vec{x}$) denotes the set of variables that occur in $\vec{x}$. A *valuation* over a set U of variables is a total mapping θ from U to **dom**. At several places, it is implicitly understood that such a valuation θ is extended to be the identity on constants and on variables not in U. If $V \subseteq U$, then $\theta[V]$ denotes the restriction of θ to V. If θ is a valuation over a set U of variables, x is a variable (possibly $x \notin U$), and a is a constant, then $\theta_{[x \mapsto a]}$ is the valuation over $U \cup \{x\}$ such that $\theta_{[x \mapsto a]}(x) = a$ and for every variable y such that $y \neq x$, $\theta_{[x \mapsto a]}(y) = \theta(y)$.

Atoms and key-equal facts Each *relation name* R of arity n, $n \geq 1$, has a unique *primary key* which is a set $\{1, 2, \ldots, k\}$ where $1 \leq k \leq n$. We say that R has *signature* $[n, k]$ if R has arity n and primary key $\{1, 2, \ldots, k\}$. 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 $[n, k]$. Every relation name has a unique *mode*, which is a value in $\{\mathsf{c}, \mathsf{i}\}$. Informally, relation names of mode c will be used for consistent relations, while relations that may be inconsistent will have a relation name of mode i. We often write R^{c} to make clear that R is a relation name of mode c. Relation names of mode c will be a convenient tool in the theoretical development, but they also constitute a useful modeling primitive that can be put at the disposal of end-users with domain knowledge [16].

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

We will use letters F, G, H for atoms. For an atom $F = R(\underline{\vec{x}}, \vec{y})$, we denote by $\mathsf{key}(F)$ the set of variables that occur in $\vec{x}$, and by $\mathsf{vars}(F)$ the set of variables that occur in F, that is, $\mathsf{key}(F) = \mathsf{vars}(\vec{x})$ and $\mathsf{vars}(F) = \mathsf{vars}(\vec{x}) \cup \mathsf{vars}(\vec{y})$. We sometimes blur the distinction between relation names and atoms. For example, if F is an atom, then the term F-fact refers to a fact with the same relation name as F.

Databases, blocks, and repairs A *database schema* is a finite set of relation names. All constructs that follow are defined relative to a fixed database schema. A *database* is a finite set $\mathbf{db}$ of facts using only the relation names of the schema such that for every relation name R of mode c, no two distinct R-facts of $\mathbf{db}$ are key-equal.

A *relation* of $\mathbf{db}$ is a maximal set of facts in $\mathbf{db}$ that all share the same relation name. A *block* of $\mathbf{db}$ is a maximal set of key-equal facts of $\mathbf{db}$. A block of R-facts is also called an R-block. If A is a fact of $\mathbf{db}$, then $\mathsf{block}(A, \mathbf{db})$ denotes the block of $\mathbf{db}$ that contains A. If $A = R(\underline{\vec{a}}, \vec{b})$, then $\mathsf{block}(A, \mathbf{db})$ is also denoted by $R(\underline{\vec{a}}, \vec{*})$. A database $\mathbf{db}$ is *consistent* if no two distinct facts of $\mathbf{db}$ are key-equal (i.e., if no block of $\mathbf{db}$ contains more than one fact). A *repair* of $\mathbf{db}$ is a maximal (with respect to set inclusion) consistent subset of $\mathbf{db}$. We write $\mathsf{rset}(\mathbf{db})$ for the set of repairs of $\mathbf{db}$.

Boolean conjunctive queries A *Boolean query* is a mapping q that associates a Boolean (true or false) to each database, such that q is closed under isomorphism [24]. We write $\mathbf{db} \models q$ to denote that q associates true to $\mathbf{db}$, in which case $\mathbf{db}$ is said to *satisfy* q. A Boolean query q can be viewed as a decision problem that takes a database as input and asks whether $\mathbf{db}$ satisfies q. In this paper, the complexity class $\mathbf{FO}$ stands for the set of Boolean queries that can be defined in first-order logic with equality and constant symbols (which are interpreted as themselves), but without other built-in predicates or function symbols.

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

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

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

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

Let θ be a valuation over some set X of variables. For every Boolean conjunctive query q, we write $\theta(q)$ for the query obtained from q by replacing all occurrences of each $x \in X \cap \mathsf{vars}(q)$ with $\theta(x)$; variables in $\mathsf{vars}(q) \setminus X$ remain unaffected (i.e., θ is understood to be the identity on variables not in X).

Atoms of mode c The *mode* of an atom is the mode of its relation name (a value in $\{\mathsf{c}, \mathsf{i}\}$). If q is a query in sjfBCQ, then q^{cons} is the set of all atoms of q that are of mode c.

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

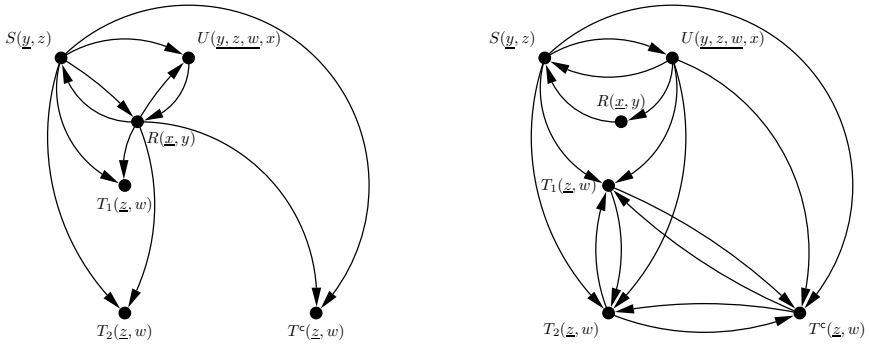
Consistent query answering Let q be a query in sjfBCQ. We define $\mathsf{CERTAINTY}(q)$ as the decision problem that takes as input a database $\mathbf{db}$, and asks whether every repair of $\mathbf{db}$ satisfies q.

The genre of a fact Let q be a query in sjfBCQ. For every fact A whose relation name occurs in q, we denote by $\mathsf{genre}_q(A)$ the (unique) atom of q that has the same relation name as A. From here on, if $\mathbf{db}$ is a database that is given as an input to $\mathsf{CERTAINTY}(q)$, we will assume that each relation name of each fact in $\mathbf{db}$ also occurs in q. Therefore, for every $A \in \mathbf{db}$, $\mathsf{genre}_q(A)$ is well defined. Of course, this assumption is harmless.

Attack graph Let q be a query in sjfBCQ. We define $\mathcal{K}(q)$ as the following set of functional dependencies: $\mathcal{K}(q) := \{\mathsf{key}(F) \to \mathsf{vars}(F) \mid F \in q\}$. For every atom $F \in q$, we define $F^{+,q}$ as the set of all variables $x \in \mathsf{vars}(q)$ satisfying $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) \models \mathsf{key}(F) \to x$. Informally, the term $\mathcal{K}(q^{\mathsf{cons}})$ is the set of all functional dependencies that arise in atoms of mode c. Clearly, if F has mode c, then $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) = \mathcal{K}(q)$; and if F has mode i, then $\mathcal{K}(q \setminus \{F\}) \cup \mathcal{K}(q^{\mathsf{cons}}) = \mathcal{K}(q \setminus \{F\})$. The *attack graph* of q is a directed graph whose vertices are the atoms of q. There is a directed edge from F to G ($F \neq G$), denoted $F \overset{q}{\rightsquigarrow} G$, if there exists a sequence

$$F_0 \overset{x_1}{\frown} F_1 \overset{x_2}{\frown} F_2 \cdots \overset{x_\ell}{\frown} F_\ell \tag{1}$$

such that $F_0 = F$, $F_\ell = G$, and for each $i \in \{1, \ldots, \ell\}$, F_i is an atom of q and x_i is a variable satisfying $x_i \in (\mathsf{vars}(F_{i-1}) \cap \mathsf{vars}(F_i)) \setminus F^{+,q}$. The sequence (1) is also called a *witness* for $F \overset{q}{\rightsquigarrow} G$. An edge $F \overset{q}{\rightsquigarrow} G$ is also called an *attack from* F *to* G; we also say that F *attacks* G. Informally, an attack from an atom $R(\vec{x}, \vec{y})$ to an atom $S(\vec{u}, \vec{w})$ indicates that, given a valuation over $\mathsf{vars}(\vec{x})$, the values for $\vec{u}$ that make the query true depend on the values chosen for $\vec{y}$.



Attack Graph. M-Graph.

Figure 1 Attack graph *(left)* and M-graph *(right)* of the same query $q_1 = \{R(\underline{x}, y), S(\underline{y}, z), U(\underline{y, z, w}, x), T_1(\underline{z}, w), T_2(\underline{z}, w), T^{\mathsf{c}}(\underline{z}, w)\}$. It can be verified that all attacks are weak and that the query is saturated. The attack graph has an initial strong component containing three atoms (R, S, and U). As predicted by Lemma 11, the subgraph of the M-graph induced by $\{R, S, U\}$ is cyclic.

An attack on a variable $x \in \mathsf{vars}(q)$ is defined as follows: $F \overset{q}{\rightsquigarrow} x$ if $F \overset{q \cup \{N(\underline{x})\}}{\rightsquigarrow} N(\underline{x})$ where N is a fresh relation name of signature $[1, 1]$. Informally, x is attacked in q if $N(\underline{x})$ has an incoming attack in the attack graph of $q \cup \{N(\underline{x})\}$.

▶ **Example 3.** Let $q_1 = \{R(\underline{x}, y), S(\underline{y}, z), U(\underline{y, z, w}, x), T_1(\underline{z}, w), T_2(\underline{z}, w), T^{\mathsf{c}}(\underline{z}, w)\}$. Using relation names for atoms, we have $R^{+,q_1} = \{x\}$. A witness for $R \overset{q_1}{\rightsquigarrow} U$ is $R \overset{y}{\frown} U$. The attack graph of q_1 is shown in Fig. 1.

An attack $F \overset{q}{\rightsquigarrow} G$ is *weak* if $\mathcal{K}(q) \models \mathsf{key}(F) \to \mathsf{key}(G)$; otherwise it is *strong*. A cycle in the attack graph is *strong* if at least one attack in the cycle is strong. It has been proved [20, Lemma 3.6] that if the attack graph contains a strong cycle, then it contains a strong cycle of length 2. The main result in [20] can now be stated.

▶ **Theorem 4** ([20])**.** *For every query q in* sjfBCQ*,*

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

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

Sequential proof Let q be a query in sjfBCQ. Let $Z \to w$ be a functional dependency for q with a singleton right-hand side (where set delimiters { and } are omitted). A *sequential proof for* $Z \to w$ is a (possibly empty) sequence $F_1, F_1, \ldots, F_\ell$ of atoms in q such that for every $i \in \{1, \ldots, \ell\}$, $\mathsf{key}(F_i) \subseteq Z \cup \left(\bigcup_{j=1}^{i-1} \mathsf{vars}(F_j)\right)$ and $w \in Z \cup (\bigcup_{j=1}^{\ell} \mathsf{vars}(F_j))$. Clearly, if $w \in \mathsf{vars}(F_k)$ for some $k < \ell$, then such a sequential proof can be shortened by omitting the atoms $F_{k+1}, \ldots, F_\ell$. Sequential proofs mimic the computation of a closure of a set of attributes with respect to a set of functional dependencies; see, e.g., [1, p. 165].

Notions from graph theory We adopt some terminology from [4]. A directed graph is *strongly connected* if there is a directed path from any vertex to any other. The maximal strongly connected subgraphs of a graph are vertex-disjoint and are called its *strong components*. If S_1 and S_2 are strong components such that an edge leads from a vertex in S_1 to a vertex in S_2, then S_1 is a *predecessor* of S_2 and S_2 is a *successor* of S_1. A strong component is called *initial* if it has no predecessor. For a directed graph, we define the length of a directed path as the number of edges it contains. A directed path or cycle without repeated vertices is called *elementary*. If G is a graph, then $V(G)$ denotes the vertex set of G, and $E(G)$ denotes the edge set of G.

Symmetric stratified Datalog We assume that the reader is familiar with the syntax and semantics of stratified Datalog. A stratified Datalog program is *linear* if in the body of each rule there is at most one occurrence of an IDB predicate of the same stratum (but there may be arbitrarily many occurrences of IDB predicates from lower strata). Assume that some stratum of a linear stratified Datalog program contains a recursive rule

$$L_0 \leftarrow L_1, L_2, \ldots, L_m, \neg L_{m+1}, \ldots, \neg L_n$$

such that L_1 is an IDB predicate of the same stratum. Then, since the program is linear, each predicate among $L_2, \ldots, L_n$ is either an EDB predicate or an IDB predicate of a lower stratum. Such a rule has a *symmetric rule*:

$$L_1 \leftarrow L_0, L_2, \ldots, L_m, \neg L_{m+1}, \ldots, \neg L_n.$$

A stratified Datalog program is *symmetric* if it is linear and the symmetric of any recursive rule is also a rule of the program.

It is known (see, for example, [15, Proposition 3.3.72]) that linear stratified Datalog is equivalent to Transitive Closure Logic. The data complexity of linear stratified Datalog is in **NL** (and is complete for **NL**). A symmetric Datalog program can be evaluated in logarithmic space [10] and cannot express directed reachability [11].

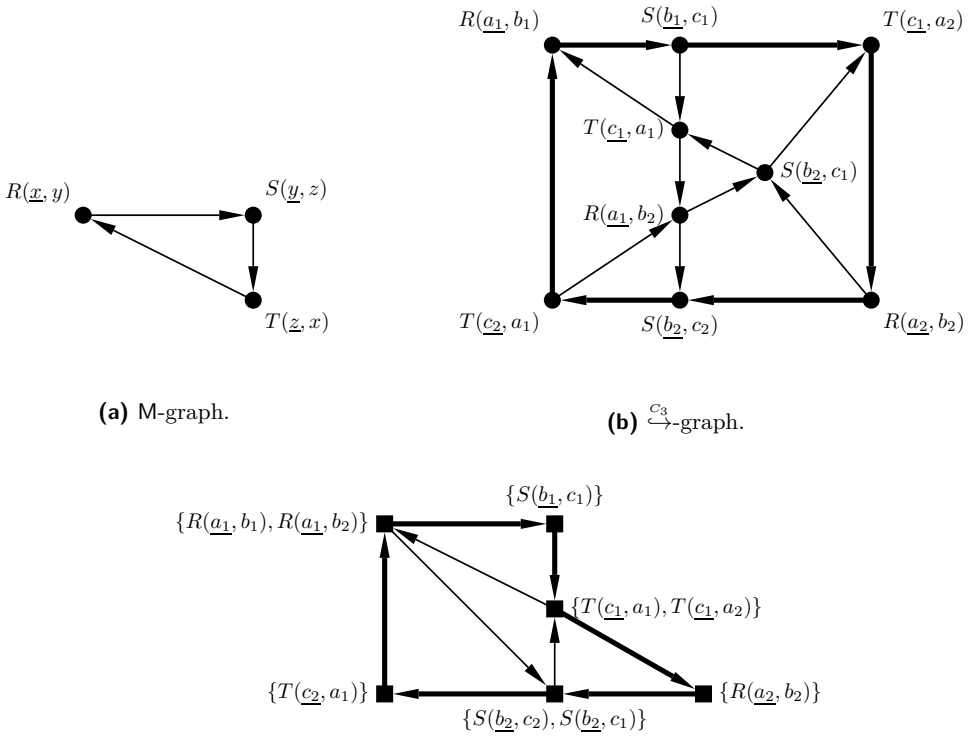
We will assume that given a (extensional or intentional) predicate P of some arity 2ℓ, we can express the following query (let $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$, $\vec{y} = \langle y_1, \ldots, y_\ell \rangle$, and $\vec{z} = \langle z_1, \ldots, z_\ell \rangle$):

$$\{\vec{x}, \vec{y} \mid P(\vec{x}, \vec{y}) \wedge \forall z_1 \cdots \forall z_\ell \, (P(\vec{x}, \vec{z}) \rightarrow \vec{y} \leq_\ell \vec{z})\}, \tag{2}$$

where $\leq_\ell$ is a total order on $\mathbf{dom}^\ell$. Informally, the above query groups by the ℓ leftmost positions, and, within each group, takes the smallest (with respect to $\leq_\ell$) value for the remaining positions. Such a query will be useful in Section 8.3, where P encodes an equivalence relation on a finite subset of $\mathbf{dom}^\ell$, and the query (2) allows us to deterministically choose a representative in each equivalence class. The order $\leq_\ell$ can be first-order defined as the lexicographical order on $\mathbf{dom}^\ell$ induced by the linear order on $\mathbf{dom}$. For example, for $\ell = 2$, the lexicographical order is defined as $(y_1, y_2) \leq_2 (z_1, z_2)$ if $y_1 < z_1 \vee ((y_1 = z_1) \wedge (y_2 \leq z_2))$. Nevertheless, our results do not depend on how the order $\leq_\ell$ is defined. Moreover, all queries in our study will be order-invariant in the sense defined in [17]. The order is only needed in the proof of Lemma 25 to pick, in a deterministic way, an identifier from a set of candidate identifiers. In Datalog, we use the following convenient syntax for (2):

$$\mathsf{Answer}(\vec{x}, \min(\vec{y})) \leftarrow P(\vec{x}, \vec{y}).$$

Such a rule will always be non-recursive. Most significantly, if we extend a logspace fragment of stratified Datalog with queries of the form (2), the extended fragment will also be in



(a) M-graph.

(b) $\overset{C_3}{\hookrightarrow}$-graph.

(c) Block-quotient graph.

Figure 2 Examples of three different graphs used in this paper: M-graph, $\hookrightarrow$-graph, block-quotient graph.

logspace. Therefore, assuming queries of the form (2) is harmless for our complexity-theoretic purposes. We use *SymStratDatalog* for symmetric stratified Datalog, and *SymStratDatalog*$^{\min}$ for symmetric stratified Datalog that allows queries of the form (2). The need for the min operator will occur in the proof of Lemma 25.

4 The Main Theorem and an Informal Guide of its Proof

In this paper, we prove the following main result.

▶ **Theorem 5** (Main Theorem). *For every query q in* sjfBCQ*,*

- *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP***-complete; and*
- *if the attack graph of q contains no strong cycle, then* CERTAINTY(q) *is expressible in* *SymStratDatalog*$^{\min}$ *(and is thus in* **L***).*

The above result is stronger than Theorem 2, since it provides an effective criterion for the dichotomy between **coNP**-completeness and expressibility in symmetric stratified Datalog.

Before we delve into the proof in the next sections, we start with a guided tour that introduces our approach in an informal way. The focus of this paper is a deterministic logspace algorithm for CERTAINTY(q) whenever CERTAINTY(q) is in **P** but not in **FO**

(assuming $\mathbf{P} \neq \mathbf{coNP}$). In what follows, by a logspace algorithm, we will always mean a deterministic logspace algorithm. An exemplar query is $C_3 := \{R(\underline{x}, y), S(\underline{y}, z), T(\underline{z}, x)\}$, which can be thought of as a cycle of length 3. For the purpose of this example, let q be a query in sjfBCQ that includes C_3 as a subquery (i.e., $C_3 \subseteq q$).

An important novel notion in this paper is the M-graph of a query (see Section 6). The M-graph of C_3 is shown in Fig. 2a. Informally, a directed edge from an atom F to an atom G, denoted $F \xrightarrow{\mathsf{M}} G$, means that every variable that occurs in the primary key of G occurs also in F. In Fig. 2a, we have $T(\underline{z}, x) \xrightarrow{\mathsf{M}} R(\underline{x}, y)$, because R's primary key (i.e., x) occurs in the T-atom; there is no edge from $R(\underline{x}, y)$ to $T(\underline{z}, x)$ because z does not occur in the R-atom. Intuitively, one can think of edges in the M-graph as foreign-to-primary key joins. In what follows, we focus on cycles in the M-graph, called M-cycles. As we will see later on, such M-cycles will occur whenever the attack graph of a query is cyclic but contains no strong attack cycles.

Figure 2b shows an *instantiation* of the M-graph, called $\overset{C_3}{\hookrightarrow}$-graph (see Definitions 12 and 18), whose vertices are obtained by replacing variables with constants in $R(\underline{x}, y)$, $S(\underline{y}, z)$, or $T(\underline{z}, x)$. We write $A \overset{C_3}{\hookrightarrow} B$ to denote an edge from fact A to fact B. Each triangle in the $\overset{C_3}{\hookrightarrow}$-graph of Fig. 2b instantiates the query C_3; for example, the inner triangle is equal to $\theta(C_3)$ where θ is the valuation such that $\theta(xyz) = a_1b_2c_1$. We call such a triangle a 1-embedding (see Definition 18). Significantly, some edges are not part of any triangle. For example, the edge $S(\underline{b_1}, c_1) \overset{C_3}{\hookrightarrow} T(\underline{c_1}, a_2)$ is not in a triangle, but is present because the primary key of $T(\underline{c_1}, a_2)$ occurs in $S(\underline{b_1}, c_1)$.

Let **db** be a database that is input to CERTAINTY(q) such that **db** contains (but is not limited to) all facts of Fig. 2b. Since C_3 is a subquery of q, **db** will typically contain other facts with relation names in $q \setminus C_3$. Furthermore, **db** can contain R-facts, S-facts, and T-facts not shown in Fig. 2b. Then, **db** has at least $2^3 = 8$ repairs, because Fig. 2b shows two R-facts with primary key a_1, two S-facts with primary key b_2, and two T-facts with primary key c_1. Consider now the outermost elementary cycle (in thick lines) of length 6, i.e., the cycle using the vertices in $\mathbf{r} := \{R(\underline{a_1}, b_1), S(\underline{b_1}, c_1), T(\underline{c_1}, a_2), R(\underline{a_2}, b_2), S(\underline{b_2}, c_2), T(\underline{c_2}, a_1)\}$, which will be called a 2-embedding in Definition 18 (or an n-embedding with $n = 2$). One can verify that **r** does not contain distinct key-equal facts and does not satisfy C_3 (because the subgraph induced by **r** has no triangle). Let **o** be the database that contains **r** as well as all facts of **db** that are key-equal to some fact in **r**. A crucial observation is that if $\mathbf{db} \setminus \mathbf{o}$ has a repair that falsifies q, then so has **db** (the converse is trivially true). Indeed, if **s** is a repair of $\mathbf{db} \setminus \mathbf{o}$ that falsifies q, then $\mathbf{s} \cup \mathbf{r}$ is a repair of **db** that falsifies q. Intuitively, we can add **r** to **s** without creating a triangle in the $\overset{C_3}{\hookrightarrow}$-graph (i.e., without making C_3 true, and thus without making q true), because the facts in **r** form a cycle on their own and contain no outgoing $\overset{C_3}{\hookrightarrow}$-edges to facts in **s**. In Section 7, the set **o** will be called a *garbage set:* its facts can be thrown away without changing the answer to CERTAINTY(q). Note that the $\overset{C_3}{\hookrightarrow}$-graph of Fig. 2b contains other elementary cycles of length 6, which, however, contain distinct key-equal facts: for example, the cycle with vertices $R(\underline{a_1}, b_1)$, $S(\underline{b_1}, c_1)$, $T(\underline{c_1}, a_1)$, $R(\underline{a_1}, b_2)$, $S(\underline{b_2}, c_2)$, $T(\underline{c_2}, a_1)$ contains both $R(\underline{a_1}, b_1)$ and $R(\underline{a_1}, b_2)$.

Garbage sets thus arise from cycles in the $\overset{C_3}{\hookrightarrow}$-graph that *(i)* do not contain distinct key-equal facts, and *(ii)* are not triangles satisfying C_3. To find such cycles, we construct the quotient graph of the $\overset{C_3}{\hookrightarrow}$-graph with respect to the equivalence relation "is key-equal to." Since the equivalence classes with respect to "is key-equal to" are the *blocks* of the database, we call this graph the *block-quotient graph* (Definition 23). The block-quotient graph for our example is shown in Fig. 2c. The vertices are database blocks; there is an edge from

block $\mathbf{b}_1$ to $\mathbf{b}_2$ if the $\overset{C_3}{\hookrightarrow}$-graph contains an edge from some fact in $\mathbf{b}_1$ to some fact in $\mathbf{b}_2$. The block-quotient graph contains exactly one elementary directed cycle of length 6 (thick lines); this cycle obviously corresponds to the outermost cycle of length 6 in the $\overset{C_3}{\hookrightarrow}$-graph. A core result (Lemma 22) of this article is a logspace algorithm for finding elementary cycles in the block-quotient graph whose lengths are strict multiples of the length of the underlying M-cycle. In our example, since the M-cycle of C_3 has length 3, we are looking for cycles in the block-quotient graph of lengths $6, 9, 12, \ldots$ Note here that, since the $\overset{C_3}{\hookrightarrow}$-graph is tripartite, the length of any cycle in it must be a multiple of 3. Our algorithm can be encoded in symmetric stratified Datalog. This core algorithm is then extended to compute garbage sets (Lemma 24) for M-cycles.

In our example, C_3 is a subquery of q. In general, M-cycles will be subqueries of larger queries. The facts that belong to the garbage set for an M-cycle can be removed, but the other facts must be maintained for computations on the remaining part of the query, and are stored in a new schema that replaces the relations in the M-cycle with a single relation (see Section 8.3). In our example, this new relation has attributes for x, y, and z, and stores all triangles that are outside the garbage set for C_3.

We can now sketch our approach for dealing with queries q such that CERTAINTY(q) is in $\mathbf{P} \setminus \mathbf{FO}$. Lemma 11 tells us that such a query q will have an M-cycle involving two or more atoms of mode i. The garbage set of this M-cycle is then computed, and the facts not in the garbage set will be stored in a single new relation of mode i that replaces the M-cycle. In this way, CERTAINTY(q) is reduced to a new problem CERTAINTY(q'), where q' contains less atoms of mode i than q. Lemma 25 shows that this new problem will be in $\mathbf{P}$, and that our reduction can be expressed in symmetric stratified Datalog. We can repeat this reduction until we arrive at a query q'' such that CERTAINTY(q'') is in $\mathbf{FO}$.

To conclude this guided tour, we point out the role of atoms of mode c in the computation of the M-graph, which was not illustrated by our running example. In the M-graph of Fig. 1 *(right)*, we have $S(\underline{y}, z) \xrightarrow{\mathsf{M}} U(\underline{y, z, w}, x)$, even though w does not occur in the S-atom. The explanation is that the query also contains the consistent relation $T^{\mathsf{c}}(\underline{z}, w)$, which maps each z-value to a unique w-value. So even though w does not occur as such in $S(\underline{y}, z)$, it is nevertheless uniquely determined by z. It is thus important to identify all relations of mode c, which is the topic of the next section.

5 Saturated Queries

In this section, we show that we can safely extend a query q with new consistent relations. To achieve this, we need to identify a particular type of functional dependencies for q, which are called *internal*. Internal functional dependencies are used to define *saturated* queries.

▶ **Definition 6.** *Let q be a query in* sjfBCQ. *Let $Z \to w$ be a functional dependency for q. We say that $Z \to w$ is* internal to q *if the following two conditions are satisfied:*

1. *there exists a sequential proof for $\mathcal{K}(q) \models Z \to w$ such that no atom in the sequential proof attacks a variable in $Z \cup \{w\}$; and*
2. *for some $F \in q$, $Z \subseteq \mathsf{vars}(F)$.*

We say that q is saturated *if for every functional dependency σ that is internal to q, we have $\mathcal{K}(q^{\mathsf{cons}}) \models \sigma$.*

▶ **Example 7.** Assume $q = \{S_1(\underline{z}, u), S_2(\underline{u}, w), R_1(\underline{z}, u'), R_2(\underline{u'}, w), T_1(\underline{u}, v), T_2(\underline{v}, w)\}$. By using relation names as a shorthand for atoms, we have that $\langle S_1, S_2 \rangle$ is a sequential proof for $\mathcal{K}(q) \models z \to w$ in which neither S_1 nor S_2 attacks z or w, Indeed, S_1 attacks neither z nor w because $z, w \in {S_1}^{+,q}$. S_2 attacks no variable because $\mathsf{vars}(S_2) \subseteq {S_2}^{+,q}$. It follows that the functional dependency $z \to w$ is internal to q.

The next key lemma shows that we can assume without loss of generality that every internal functional dependency $Z \to w$ is satisfied, i.e., that every Z-value is mapped to a unique w-value. Therefore, whenever $Z \to w$ is internal, we can safely extend q with a new consistent relation $N^{\mathsf{c}}(\underline{Z}, w)$ that materializes the mapping from Z-values to w-values. Continuing the above example, we would extend q by adding a fresh atom $N^{\mathsf{c}}(\underline{z}, w)$.

▶ **Lemma 8.** *For every query q in* sjfBCQ*, it is possible to compute a query q' in* sjfBCQ *with the following properties:*

1. *there exists a first-order reduction from* CERTAINTY(q) *to* CERTAINTY(q')*;*
2. *if the attack graph of q contains no strong cycle, then the attack graph of* CERTAINTY(q') *contains no strong cycle; and*
3. *q' is saturated.*

6 M-Graphs and $\hookrightarrow$-Graphs

In this section, we introduce the M-*graph* of a query q in sjfBCQ, which is a generalization of the notion of Markov-graph introduced in [20] (hence the use of the letter M). An important new result, Lemma 11, expresses a relationship between attack graphs and M-graphs. Finally, we define $\hookrightarrow$-graphs, which can be regarded as data-level instantiations of M-graphs.

▶ **Definition 9.** *Let q be a query in* sjfBCQ *(which need not be saturated). The* M-graph *of q is a directed graph whose vertices are the atoms of q. There is a directed edge from F to G ($F \neq G$), denoted $F \xrightarrow{\mathsf{M}} G$, if $\mathcal{K}(q^{\mathsf{cons}}) \models \mathsf{vars}(F) \to \mathsf{key}(G)$. A cycle in the* M-*graph is called an* M-cycle*.*

Note that if all relation names in q have mode i, then $F \xrightarrow{\mathsf{M}} G$ implies $\mathsf{key}(G) \subseteq \mathsf{vars}(F)$. M-Graphs are technically easier to deal with than the Markov-graphs [20] on which they are inspired. In fact, Markov-graphs were in [20] only defined for queries containing no atoms of mode i with a composite primary key. Therefore, atoms with composite primary keys had first to be massaged into the form required by Markov graphs. This drawback is resolved by the new notion of M-graph.

▶ **Example 10.** The notion of M-graph is illustrated by Fig. 1. We have $\mathcal{K}({q_1}^{\mathsf{cons}}) = \{z \to w\}$. Since $\mathcal{K}({q_1}^{\mathsf{cons}}) \models \mathsf{vars}(S) \to \mathsf{key}(U)$, the M-graph has a directed edge from S to U.

The following lemma tells us about how the existence of M-cycles goes hand in hand with weak attack cycles. This relationship is important because, on the one hand, our logspace algorithm for CERTAINTY(q), with $q \in$ sjfBCQ, is centered on the existence of M-cycles, and, on the other, it must apply whenever all cycles in q's attack graph are weak. The notion of saturated query is also needed here, because the lemma fails for queries that are not saturated. The lemma generalizes Lemma 7.13 in [20]. Note that the lemma only considers strong components of the attack graph that are initial, which will be sufficient for our purposes.

▶ **Lemma 11.** *Let q be a query in* sjfBCQ *such that q is saturated and the attack graph of q contains no strong cycle. Let $\mathcal{S}$ be an initial strong component in the attack graph of q with $|\mathcal{S}| \geq 2$. Then, the* M-*graph of q contains a cycle all of whose atoms belong to $\mathcal{S}$.*

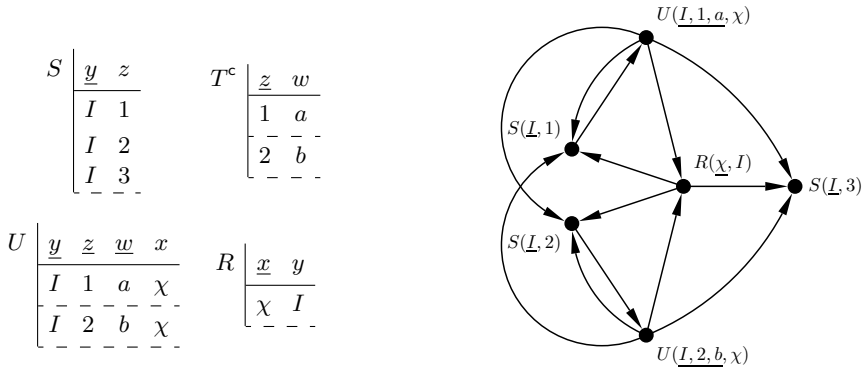


Figure 3 *Left:* Database that is input to CERTAINTY(q_1) for the query q_1 in Fig. 1. The relations for T_1 and T_2, which are identical to the relation for T^{c}, have been omitted. *Right:* The $\rightsquigarrow$-graph from which, for readability reasons, T_1-facts, T_2-facts, and T^{c}-facts have been omitted.

Given a query q, every database that instantiates the schema of q naturally gives rise to an instantiation of the $\xrightarrow{\mathsf{M}}$-edges in q's M-graph, in a way that is captured by the following definition.

▶ **Definition 12.** *The following notions are defined relative to a query q in* sjfBCQ *and a database* **db**. *The* $\rightsquigarrow$-graph *of* **db** *is a directed graph whose vertices are the atoms of* **db**. *There is a directed edge from A to B, denoted $A \rightsquigarrow B$, if there exists a valuation θ over* vars(q) *and an edge $F \xrightarrow{\mathsf{M}} G$ in the* M*-graph of q such that $\theta(q) \subseteq$ **db**, $A = \theta(F)$, and $B \sim \theta(G)$. A cycle in the $\rightsquigarrow$-graph is also called a $\rightsquigarrow$-cycle. In spoken language, the $\rightsquigarrow$-graph may be called the* instantiated M-graph.

The notion of $\rightsquigarrow$-graph is illustrated by Fig. 3. The following lemma states that if the $\rightsquigarrow$-graph of a database **db** has a directed edge from some fact A to some G-fact B, then A has outgoing edges to all the facts of block(B, **db**), and to no other G-facts.

▶ **Lemma 13.** *Let $q \in$ sjfBCQ and let* **db** *be a database. Let $A, B \in$ **db** and $F, G \in q$.*
1. *if $A \rightsquigarrow B$, then $A \rightsquigarrow B'$ for all $B' \in$ block(B, **db**);*
2. *if $A \rightsquigarrow B$ and $A \rightsquigarrow B'$ and $\mathsf{genre}_q(B) = \mathsf{genre}_q(B')$, then $B \sim B'$.*

7 Garbage Sets

Let **db** be a database that is an input to CERTAINTY(q) with $q \in$ sjfBCQ. In this section, we show that it is generally possible to downsize **db** by deleting blocks from it without changing the answer to CERTAINTY(q). That is, if the downsized database has a repair falsifying q, then so does the original database (the converse holds trivially true). Intuitively, the deleted blocks can be considered as "garbage" for the problem CERTAINTY(q).

▶ **Definition 14.** *The following definition is relative to a fixed query q in* sjfBCQ. *Let $q_0 \subseteq q$. Let* **db** *be a database. We say that a subset* **o** *of* **db** *is a* garbage set for q_0 in **db** *if the following conditions are satisfied:*
1. *for every $A \in$ **o**, we have that* $\mathsf{genre}_q(A) \in q_0$ *and* block(A, **db**) $\subseteq$ **o**; *and*
2. *there exists a repair* **r** *of* **o** *such that for every valuation θ over* vars(q), *if $\theta(q) \subseteq$ (**db** $\setminus$ **o**)$\cup$**r**, then $\theta(q_0) \cap$ **r** $= \emptyset$ (and thus $\theta(q_0) \cap$ **o** $= \emptyset$).*

The first condition in the above definition says that the relation names of facts in $\mathbf{o}$ must occur in q_0, and that every block of $\mathbf{db}$ is either included in or disjoint with $\mathbf{o}$. The second condition captures the crux of the definition and was illustrated in Section 4.

We now show a number of useful properties of garbage sets that are quite intuitive. In particular, by Lemma 15, there exists a unique maximum (with respect to $\subseteq$) garbage set for q_0 in $\mathbf{db}$, which will be called *the maximum garbage set for q_0 in $\mathbf{db}$*.

▶ **Lemma 15.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. If $\mathbf{o}_1$ and $\mathbf{o}_2$ are garbage sets for q_0 in $\mathbf{db}$, then $\mathbf{o}_1 \cup \mathbf{o}_2$ is a garbage set for q_0 in $\mathbf{db}$.*

▶ **Lemma 16.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a garbage set for q_0 in $\mathbf{db}$. Then, every repair of $\mathbf{db}$ satisfies q if and only if every repair of $\mathbf{db} \setminus \mathbf{o}$ satisfies q (i.e., $\mathbf{db}$ and $\mathbf{db} \setminus \mathbf{o}$ agree on their answer to* CERTAINTY(q)*).*

▶ **Lemma 17.** *Let q be a query in* sjfBCQ*, and let $q_0 \subseteq q$. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a garbage set for q_0 in $\mathbf{db}$. Then, every garbage set for q_0 in $\mathbf{db} \setminus \mathbf{o}$ is empty if and only if $\mathbf{o}$ is the maximal garbage set for q_0 in $\mathbf{db}$.*

8 Garbage Sets for M-Cycles

In this section, we bring together notions of the two preceding sections. We focus on queries q in sjfBCQ whose M-graph has a cycle C. From here on, if C is an elementary cycle in the M-graph of some query q in sjfBCQ, then the subset of q that contains all (and only) the atoms of C, is also denoted by C.

Section 8.1 shows a procedural characterization of the maximal garbage set for C. Section 8.2 shows that the maximal garbage set for C can be computed in symmetric stratified Datalog. Finally, Section 8.3 shows a reduction, expressible in symmetric stratified Datalog, that replaces C with a single atom.

8.1 Characterizing Garbage Sets for M-Cycles

We define how a given M-cycle C of length k can be instantiated by cycles in the $\rightsquigarrow$-graph, called *embeddings*, whose lengths are multiples of k.

▶ **Definition 18.** *Let q be a query in* sjfBCQ*. Let $\mathbf{db}$ be a database. Let C be an elementary directed cycle in the* M*-graph of q. The cycle C naturally induces a subgraph of the $\rightsquigarrow$-graph, as follows: the vertex set of the subgraph contains all (and only) the facts A of $\mathbf{db}$ such that* $\mathsf{genre}_q(A)$ *is an atom in C; there is a directed edge from A to B, denoted $A \overset{C}{\rightsquigarrow} B$, if $A \rightsquigarrow B$ and the cycle C contains a directed edge from* $\mathsf{genre}_q(A)$ *to* $\mathsf{genre}_q(B)$*.*

Let k be the length of C. Obviously, the length of every $\overset{C}{\rightsquigarrow}$-cycle must be a multiple of k. Let n be a positive integer. An n-embedding of C in $\mathbf{db}$ *(or simply* embedding *if the value n is not important) is an elementary $\overset{C}{\rightsquigarrow}$-cycle of length nk containing no two distinct key-equal facts. A* 1*-embedding of C in $\mathbf{db}$ is said to be* relevant *if there exists a valuation θ over* $\mathsf{vars}(q)$ *such that $\theta(q) \subseteq \mathbf{db}$ and $\theta(q)$ contains every fact of the* 1*-embedding; otherwise the* 1*-embedding is said to be* irrelevant*.*

Let C and q be as in Definition 18, and let $\mathbf{db}$ be a database. There exists an intimate relationship between garbage sets for C in $\mathbf{db}$ and different sorts of embeddings.

- Let $A \in \mathbf{db}$ such that $\mathsf{genre}_q(A)$ belongs to C. If A belongs to some relevant 1-embedding of C in $\mathbf{db}$, then A will have an outgoing edge in the $\overset{C}{\rightsquigarrow}$-graph. If A does not belong to some relevant 1-embedding of C in $\mathbf{db}$, then A will have no outgoing edge in the $\overset{C}{\rightsquigarrow}$-graph, and $\mathsf{block}(A, \mathbf{db})$ is a garbage set for C in $\mathbf{db}$ by Definition 14 (choose $\mathbf{o} = \mathsf{block}(A, \mathbf{db})$ and $\mathbf{r} = \{A\}$).

- Every irrelevant 1-embedding of C in $\mathbf{db}$ gives rise to a garbage set. To illustrate this case, let $C = \{R(\underline{x}, y, z),\ S(\underline{y}, x, z)\}$. Assume that $R(\underline{a}, b, 1) \overset{C}{\hookrightarrow} S(\underline{b}, a, 2) \overset{C}{\hookrightarrow} R(\underline{a}, b, 1)$ is a 1-embedding of C in $\mathbf{db}$. This 1-embedding is irrelevant, because $1 \neq 2$. It can be easily seen that $R(\underline{a}, *, *) \cup S(\underline{b}, *, *)$ is a garbage set for q in $\mathbf{db}$.
- Every n-embedding of C in $\mathbf{db}$ with $n \geq 2$ gives rise to a garbage set. This was illustrated in Section 4 by means of the outermost cycle of length 6 in Fig. 2b, which is a 2-embedding of $\{R(\underline{x}, y),\ S(\underline{y}, z),\ T(\underline{z}, x)\}$.

These observations lead to the following lemma which provides a procedural characterization of the maximal garbage set for C in a given database.

▶ **Lemma 19.** *Let q be a query in* sjfBCQ*. Let $C = F_0 \xrightarrow{\mathsf{M}} F_1 \xrightarrow{\mathsf{M}} \cdots \xrightarrow{\mathsf{M}} F_{k-1} \xrightarrow{\mathsf{M}} F_0$ be an elementary cycle of length k ($k \geq 2$) in the* M*-graph of q. Let $\mathbf{db}$ be a database. Let $\mathbf{o}$ be a minimal (with respect to $\subseteq$) subset of $\mathbf{db}$ satisfying the following conditions:*

1. *the set $\mathbf{o}$ contains every fact A of $\mathbf{db}$ with $\mathsf{genre}_q(A) \in \{F_0, \ldots, F_{k-1}\}$ such that A has zero outdegree in the $\overset{C}{\hookrightarrow}$-graph;*
2. *the set $\mathbf{o}$ contains every fact that belongs to some irrelevant 1-embedding of C in $\mathbf{db}$;*
3. *the set $\mathbf{o}$ contains every fact that belongs to some n-embedding of C in $\mathbf{db}$ with $n \geq 2$;*
4. Recursive condition: *if $\mathbf{o}$ contains some fact of a relevant 1-embedding of C in $\mathbf{db}$, then $\mathbf{o}$ contains every fact of that 1-embedding; and*
5. Closure under "is key-equal to": *if $\mathbf{o}$ contains some fact A, then $\mathbf{o}$ includes $\mathsf{block}(A, \mathbf{db})$.*

Then, $\mathbf{o}$ is the maximal garbage set for C in $\mathbf{db}$.

▶ **Corollary 20.** *Let C be an elementary cycle in the* M*-graph of a query q in* sjfBCQ*. Let $\mathcal{S}$ be a strong component in the $\overset{C}{\hookrightarrow}$-graph of a database $\mathbf{db}$. If some fact of $\mathcal{S}$ belongs to the maximal garbage set for C in $\mathbf{db}$, then every fact of $\mathcal{S}$ belongs to the maximal garbage set for C in $\mathbf{db}$.*

8.2 Computing Garbage Sets for M-Cycles

In this section, we translate Lemma 19 into a Datalog program that computes, in deterministic logspace, the maximal garbage set for an M-cycle C. The main computational challenge lies in condition 3 of Lemma 19, which adds to the maximal garbage set all facts belonging to some n-embedding with $n \geq 2$, where the value of n is not upper bounded. Such n-embeddings can obviously be computed in nondeterministic logspace by using directed reachability in the $\overset{C}{\hookrightarrow}$-graph. This section shows a trick that allows doing the computation by using only *undirected* reachability, which, by the use of Reingold's algorithm [32], will lead to an algorithm that runs in deterministic logspace.

By Corollary 20, instead of searching for n-embeddings, $n \geq 2$, it suffices to search for strong components of the $\overset{C}{\hookrightarrow}$-graph containing such n-embeddings. These strong components can be recognized by a first-order reduction to the following problem, called $\mathsf{LONGCYCLE}(k)$, which is in logspace by Lemma 22.

▶ **Definition 21.** *A* k-circle-layered graph *[25] is a k-partite directed graph $G = (V, E)$ where edges only exist between adjacent partitions. More formally, the vertices of G can be partitioned into k groups such that $V = V_0 \cup V_1 \cup \cdots \cup V_{k-1}$ and $V_i \cap V_j = \emptyset$ if $i \neq j$. The only edges from a partition V_i go to the partition $V_{(i+1) \bmod k}$.*

For every positive integer k, $\mathsf{LONGCYCLE}(k)$ is the following problem.

Problem *$\mathsf{LONGCYCLE}(k)$*

Instance *A connected k-circle-layered graph $G = (V, E)$ such that every edge of E belongs to a directed cycle of length k.*

Question *Does G have an elementary directed cycle of length at least $2k$?*

▶ **Lemma 22.** *For every positive integer k, LONGCYCLE(k) is in* **L** *and can be expressed in SymStratDatalog.*

Proof (Sketch). Let $G = (V, E)$ be an instance of LONGCYCLE(k). A cycle of length k in G is called a *k-cycle*. Let $\widehat{G}$ be the undirected graph whose vertices are the k-cycles of G; there is an undirected edge between two vertices if their k-cycles have an element in common. The full proof in [22] shows that G has an elementary directed cycle of length $\geq 2k$ if and only if one of the following conditions is satisfied:

- for some n such that $2 \leq n \leq 2k-3$, G has an elementary directed cycle of length nk; or
- $\widehat{G}$ has a chordless undirected cycle (i.e., a cycle without cycle chord) of length $\geq 2k$.

The first condition can be tested in **FO**; the second condition can be reduced to an undirected connectivity problem, which is in logspace [32] and can be expressed in *SymStratDatalog*. ◀

To use Lemma 22, we take a detour via the quotient graph of the $\overset{C}{\hookrightarrow}$-graph relative to the equivalence relation "is key-equal to."

▶ **Definition 23.** *Let q be a query in* sjfBCQ*. Let* **db** *be a database. Let C be an elementary directed cycle of length $k \geq 2$ in the* M*-graph of q. The* block-quotient graph *is the quotient graph of the $\overset{C}{\hookrightarrow}$-graph of* **db** *with respect to the equivalence relation $\sim$.* [1]

The block-quotient graph of a database can obviously be constructed in **FO**. The strong components of the $\overset{C}{\hookrightarrow}$-graph that contain some n-embedding of C, $n \geq 2$, can then be recognized in logspace by executing the algorithm for LONGCYCLE(k) on the block-quotient graph.

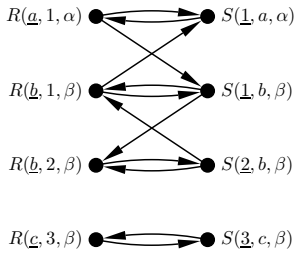
▶ **Lemma 24.** *Let q be a query in* sjfBCQ*. Let C be an elementary cycle of length k ($k \geq 2$) in the* M*-graph of q. There exists a program in SymStratDatalog that takes a database* **db** *as input and returns, as output, the maximal garbage set for C in* **db**.

8.3 Elimination of M-Cycles

Given a database **db**, the Datalog program of Lemma 24 allows us to compute the maximal garbage set **o** for C in **db**. The $\overset{C}{\hookrightarrow}$-graph of $\mathbf{db}' := \mathbf{db} \setminus \mathbf{o}$ will be a set of strong components, all initial, each of which is a collection of relevant 1-embeddings of C in $\mathbf{db}'$. The following Lemma 25 introduces a reduction that encodes this $\overset{C}{\hookrightarrow}$-graph by means of a fresh atom $T(\underline{u}, \vec{w})$, where $\mathsf{vars}(\vec{w}) = \mathsf{vars}(C)$ and u is a fresh variable. Whenever $\theta(q) \subseteq \mathbf{db}'$ for some valuation θ over $\mathsf{vars}(q)$, the reduction will add to the database a fact $T(\underline{cid}, \theta(\vec{w}))$ where *cid* is an identifier for the strong component (in the $\overset{C}{\hookrightarrow}$-graph) that contains $\theta(C)$. The construction is illustrated by Fig. 4. The following lemma captures this reduction and states that it *(i)* is expressible in $\mathit{SymStratDatalog}^{\min}$, and *(ii)* does not result in an increase of computational complexity.

▶ **Lemma 25.** *Let q be a query in* sjfBCQ*. Let $C = F_0 \xrightarrow{\mathsf{M}} F_1 \xrightarrow{\mathsf{M}} \cdots \xrightarrow{\mathsf{M}} F_{k-1} \xrightarrow{\mathsf{M}} F_0$ with $k \geq 2$ be an elementary cycle in the* M*-graph of q. Let u be a variable such that $u \notin \mathsf{vars}(q)$. Let T be an atom with a fresh relation name such that $\mathsf{key}(T) = \{u\}$ and $\mathsf{vars}(T) = \mathsf{vars}(C) \cup \{u\}$. Let p be a set containing, for every $i \in \{1, \ldots, k\}$, an atom N_i of mode* c *with a fresh relation name such that $\mathsf{key}(N_i) = \mathsf{key}(F_i)$ and $\mathsf{vars}(N_i) = \mathsf{key}(F_i) \cup \{u\}$. Then,*

[1] The quotient graph of a directed graph $G = (V, E)$ with respect to an equivalence relation $\equiv$ on V is a directed graph whose vertices are the equivalence classes of $\equiv$; there is a directed edge from class A to class B if E has a directed edge from some vertex in A to some vertex in B.



T

u	x	y	z
a	a	1	α
a	b	1	β
a	b	2	β
c	c	3	β

N_1^{c}

x	u
a	a
b	a
c	c

N_2^{c}

y	u
1	a
2	a
3	c

Figure 4 *Left:* Two strong components in the $\overset{C}{\hookrightarrow}$-graph of a database for an M-cycle $R(\underline{x}, y, z) \xrightarrow{\mathsf{M}} S(\underline{y}, x, z) \xrightarrow{\mathsf{M}} R(\underline{x}, y, z)$. The maximal garbage set is empty. *Right:* Encoding of the relevant 1-embeddings in each strong component. The u-values a and c are used to identify the strong components, and are chosen as the smallest x-values in each strong component.

1. *there exists a reduction from* CERTAINTY(q) *to* CERTAINTY$((q \setminus C) \cup \{T\} \cup p)$ *that is expressible in SymStratDatalog*$^{\min}$*; and*
2. *if the attack graph of q contains no strong cycle and some initial strong component of the attack graph contains every atom of $\{F_0, F_1, \ldots, F_{k-1}\}$, then the attack graph of $(q \setminus C) \cup \{T\} \cup p$ contains no strong cycle either.*

Proof (Crux). The crux in the proof of the first item is the deterministic choice of u-values for T-blocks. In Fig. 4, for example, the T-block encoding the top strong component uses $u = a$, and the T-block encoding the bottom strong component uses $u = c$. These u-values are the smallest x-values in the strong components, which can be obtained by the query (2) introduced in Section 3. In the example, we assumed $a = \min\{a, b\}$ and $c = \min\{c\}$. ◀

The proof of the main theorem, Theorem 5, is now fairly straightforward and is given in full detail in [22]. Informally, let q be a saturated query in sjfBCQ such that the attack graph of q has no strong attack cycles. If q contains an atom of mode i without incoming attacks, then this atom is rewritten in first-order logic, in the form defined by [34, Definition 8.3]; otherwise some M-cycle, which exists by Lemma 11, is eliminated in the way previously described in this section. In either case, the remaining smaller query will have a consistent *SymStratDatalog*$^{\min}$ rewriting.

9 Joins on Primary Keys

It is common that the join condition in a join of two tables expresses a foreign-to-primary key match, i.e., the columns (called the foreign key) of one table reference the primary key of another table. In our setting, we have primary keys but no foreign keys. Nevertheless, foreign keys can often be inferred from the query. For example, in the following query, the variable d in Movies references the primary key of Directors:

$$\{\mathsf{Movies}(\underline{m}, t, \text{'1963'}, d), \mathsf{Directors}(\underline{d}, \text{'Hitchcock'}, b)\}.$$

Given relation schemas Movies(M#, Title, Year, Director) and Directors(D#, Name, BirthYear), this query asks whether there exists a movie released in 1963 and directed by Hitchcock.

The *key-join property* that we define below captures this common type of join. Informally, a query has the key-join property if whenever two atoms have a variable in common, then their set of shared variables is either equal to the set of primary-key variables of one of the atoms, or contains all primary-key variables of both atoms.

▶ **Definition 26.** *We say that a query q in* sjfBCQ *has the* key-join property *if for all $F, G \in q$, either* $\mathsf{vars}(F) \cap \mathsf{vars}(G) \in \{\emptyset, \mathsf{key}(F), \mathsf{key}(G)\}$ *or* $\mathsf{vars}(F) \cap \mathsf{vars}(G) \supseteq \mathsf{key}(F) \cup \mathsf{key}(G)$.

Theorem 27 shows that if some query q in sjfBCQ has the key-join property, then CERTAINTY(q) falls on the logspace side of the dichotomy of Theorem 5.

▶ **Theorem 27.** *For every query q in* sjfBCQ *that has the key-join property,* CERTAINTY(q) *is expressible in SymStratDatalog*$^{\min}$ *(and is thus in* **L***).*

It is worth noting that many of the queries covered by Theorem 27 have an acyclic attack graph as well, and thus even have a consistent first-order rewriting.

10 Conclusion

The main result of this paper is a theorem stating that for every query q in sjfBCQ (i.e., the class of self-join-free Boolean conjunctive queries), CERTAINTY(q) is **coNP**-complete or expressible in *SymStratDatalog*$^{\min}$ (and thus in **L**). Since there exist queries $q \in$ sjfBCQ such that CERTAINTY(q) is **L**-complete, the logspace upper bound in Theorem 2 is tight. The theorem thus culminates a long line of research that started with the ICDT 2005 paper of Fuxman and Miller [13]. The outcome of this research is the following theorem.

▶ **Theorem 28.** *For every self-join-free Boolean conjunctive query q,*

- *if the attack graph of q is acyclic, then* CERTAINTY(q) *is in* **FO***;*
- *if the attack graph of q is cyclic but contains no strong cycle, then* CERTAINTY(q) *is* **L***-complete and expressible in SymStratDatalog*$^{\min}$*; and*
- *if the attack graph of q contains a strong cycle, then* CERTAINTY(q) *is* **coNP***-complete.*

An intriguing open problem is to extend these complexity results to Boolean conjunctive queries with self-joins and to UCQ. Progress in the latter problem may deepen our understanding of relationships between CQA and CSP, which were first discovered in [12].

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. URL: http://webdam.inria.fr/Alice/.

2 Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM PODS*, pages 68–79, 1999. doi:10.1145/303976.303983.

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. doi:10.1016/S0304-3975(02)00737-5.

4 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A Linear-Time Algorithm for Testing the Truth of Certain Quantified Boolean Formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

5 Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. URL: http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97.

6 Pablo Barceló and Gaëlle Fontaine. On the data complexity of consistent query answering over graph databases. *J. Comput. Syst. Sci.*, 88:164–194, 2017. doi:10.1016/j.jcss.2017.03.015.

7 Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. doi:10.2200/S00379ED1V01Y201108DTM020.

8 Meghyn Bienvenu and Camille Bourgaux. Inconsistency-Tolerant Querying of Description Logic Knowledge Bases. In Jeff Z. Pan, Diego Calvanese, Thomas Eiter, Ian Horrocks, Michael Kifer, Fangzhen Lin, and Yuting Zhao, editors, *Reasoning Web: Logical Foundation of Knowledge Graph Construction and Query Answering - 12th International Summer School 2016, Aberdeen, UK, September 5-9, 2016, Tutorial Lectures*, volume 9885 of *Lecture Notes in Computer Science*, pages 156–202. Springer, 2016. doi:10.1007/978-3-319-49493-7_5.

9 Andrei A. Bulatov. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.*, 12(4):24:1–24:66, 2011. doi:10.1145/1970398.1970400.

10 László Egri, Benoit Larose, and Pascal Tesson. Symmetric Datalog and Constraint Satisfaction Problems in Logspace. In *LICS*, pages 193–202, 2007. doi:10.1109/LICS.2007.47.

11 László Egri, Benoit Larose, and Pascal Tesson. Directed st-Connectivity Is Not Expressible in Symmetric Datalog. In *ICALP*, pages 172–183, 2008. doi:10.1007/978-3-540-70583-3_15.

12 Gaëlle Fontaine. Why is it Hard to Obtain a Dichotomy for Consistent Query Answering? In *LICS*, pages 550–559, 2013. doi:10.1109/LICS.2013.62.

13 Ariel Fuxman and Renée J. Miller. First-Order Query Rewriting for Inconsistent Databases. In *ICDT*, pages 337–351, 2005. doi:10.1007/978-3-540-30570-5_23.

14 Ariel Fuxman and Renée J. Miller. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.*, 73(4):610–635, 2007. doi:10.1016/j.jcss.2006.10.013.

15 Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2007. doi:10.1007/3-540-68804-8.

16 Sergio Greco, Fabian Pijcke, and Jef Wijsen. Certain Query Answering in Partially Consistent Databases. *PVLDB*, 7(5):353–364, 2014. URL: http://www.vldb.org/pvldb/vol7/p353-greco.pdf, doi:10.14778/2732269.2732272.

17 Martin Grohe and Thomas Schwentick. Locality of order-invariant first-order formulas. *ACM Trans. Comput. Log.*, 1(1):112–130, 2000. doi:10.1145/343369.343386.

18 Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. Efficient Querying of Inconsistent Databases with Binary Integer Programming. *PVLDB*, 6(6):397–408, 2013. URL: http://www.vldb.org/pvldb/vol6/p397-tan.pdf, doi:10.14778/2536336.2536341.

19 Paraschos Koutris and Jef Wijsen. The Data Complexity of Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. In *PODS*, pages 17–29, 2015. doi:10.1145/2745754.2745769.

20 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. doi:10.1145/3068334.

21 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *PODS*, pages 209–224, 2018. doi:10.1145/3196959.3196982.

22 Paraschos Koutris and Jef Wijsen. Consistent Query Answering for Primary Keys in Logspace. *CoRR*, abs/1810.03386, 2018. arXiv:1810.03386.

23 Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.*, 33:3–29, 2015. doi:10.1016/j.websem.2015.04.002.

24 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

25 Andrea Lincoln, Virginia Vassilevska Williams, and R. Ryan Williams. Tight Hardness for Shortest Cycles and Paths in Sparse Graphs. In *ACM-SIAM SODA*, pages 1236–1252, 2018. doi:10.1137/1.9781611975031.80.

26 Carsten Lutz and Frank Wolter. On the Relationship between Consistent Query Answering and Constraint Satisfaction Problems. In *ICDT*, pages 363–379, 2015. doi:10.4230/LIPIcs.ICDT.2015.363.

27 Mónica Caniupán Marileo and Leopoldo E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.*, 69(6):545–572, 2010. doi:10.1016/j.datak.2010.01.005.

28 Dany Maslowski and Jef Wijsen. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.*, 79(6):958–983, 2013. doi:10.1016/j.jcss.2013.01.011.

29 Dany Maslowski and Jef Wijsen. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*, pages 155–164, 2014. doi:10.5441/002/icdt.2014.18.

30 Fabian Pijcke. *Theoretical and Practical Methods for Consistent Query Answering in the Relational Data Model.* PhD thesis, University of Mons, 2018.

31 Piotr Przymus, Aleksandra Boniewicz, Marta Burzanska, and Krzysztof Stencel. Recursive Query Facilities in Relational Databases: A Survey. In *FGIT*, pages 89–99, 2010. doi:10.1007/978-3-642-17622-7_10.

32 Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008. doi:10.1145/1391289.1391291.

33 Jef Wijsen. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*, pages 179–190, 2010. doi:10.1145/1807085.1807111.

34 Jef Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9:1–9:35, 2012. doi:10.1145/2188349.2188351.

35 Jef Wijsen. A Survey of the Data Complexity of Consistent Query Answering under Key Constraints. In *FoIKS*, pages 62–78, 2014. doi:10.1007/978-3-319-04939-7_2.