Consistent Query Answering for Primary Keys on Rooted Tree Queries

PARASCHOS KOUTRIS, University of Wisconsin–Madison, USA
XIATING OUYANG, University of Wisconsin–Madison, USA
JEF WIJSEN, University of Mons, Belgium

We study the data complexity of consistent query answering (CQA) on databases that may violate the primary key constraints. A repair is a maximal subset of the database satisfying the primary key constraints. For a Boolean query q, the problem $\mathsf{CERTAINTY}(q)$ takes a database as input, and asks whether or not each repair satisfies q. The computational complexity of $\mathsf{CERTAINTY}(q)$ has been established whenever q is a self-join-free Boolean conjunctive query, or a (not necessarily self-join-free) Boolean path query. In this paper, we take one more step towards a general classification for all Boolean conjunctive queries by considering the class of rooted tree queries. In particular, we show that for every rooted tree query q, $\mathsf{CERTAINTY}(q)$ is in **FO**, **NL**-hard $\cap$ **LFP**, or **coNP**-complete, and it is decidable (in polynomial time), given q, which of the three cases applies. We also extend our classification to larger classes of queries with simple primary keys. Our classification criteria rely on query homomorphisms and our polynomial-time fixpoint algorithm is based on a novel use of context-free grammar (CFG).

CCS Concepts: • **Information systems → Relational database query languages**; • **Theory of computation → Incomplete, inconsistent, and uncertain databases**.

Additional Key Words and Phrases: consistent query answering, complexity classification, homomorphism, context-free gramma

ACM Reference Format:
Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. 2024. Consistent Query Answering for Primary Keys on Rooted Tree Queries. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 76 (May 2024), 26 pages. https://doi.org/10.1145/3651139

1 INTRODUCTION

A relational database is *inconsistent* if it violates one or more integrity constraints that are supposed to be satisfied. Database inconsistency is a common issue when integrating datasets from heterogeneous sources. In this paper, we focus on what are probably the most commonly imposed integrity constraints on relational databases: primary keys. A primary key constraint enforces that no two distinct tuples in the same relation agree on all primary key attributes.

A *repair* of such an inconsistent database instance is naturally defined as a maximal consistent subinstance of the database. Two approaches are then possible. In *data cleaning*, the objective is to single out the "best" repair, which however may not be practically possible. In *consistent query answering* (CQA) [2], instead of cleaning the inconsistent database instance, we attempt to query *every* possible repair of the database and obtain the *consistent* (or *certain*) answers that are returned across all repairs. In computational complexity studies, consistent query answering is commonly

Authors' addresses: Paraschos Koutris, University of Wisconsin–Madison, Madison, USA, paris@cs.wisc.edu; Xiating Ouyang, University of Wisconsin–Madison, Madison, USA, xouyang@cs.wisc.edu; Jef Wijsen, University of Mons, Mons, Belgium, jef.wijsen@umons.ac.be.

ACM 2836-6573/2024/5-ART76
https://doi.org/10.1145/3651139

defined as the following decision problem, for a fixed Boolean query q and fixed primary keys for all relation names occurring in q:

> **PROBLEM** $\mathsf{CERTAINTY}(q)$
> **Input**: A database instance **db**.
> **Question**: Does q evaluate to true on every repair of **db**?

The CQA problem for queries $q(\vec{x})$ with free variables is to find all sequences of constants $\vec{c}$, of the same length as $\vec{x}$, such that $q(\vec{c})$ is true in every repair. We often do not need separate treatment for different constants, in which case we can handle $q(\vec{x})$ as Boolean by treating free variables as if they were constants [17, 27].

The problem $\mathsf{CERTAINTY}(q)$ is obviously in **coNP** for every Boolean first-order query q. It has been extensively studied for q in the class of Boolean conjunctive queries, denoted BCQ. Despite significant research efforts (see Section 2), the following dichotomy conjecture remains notoriously open.

Conjecture 1.1. *For every query q in* BCQ, $\mathsf{CERTAINTY}(q)$ *is either in* **PTIME** *or* **coNP**-*complete.*

An ever stronger conjecture is that the dichotomy of Conjecture 1.1 extends to unions of conjunctive queries. Fontaine [19] showed that this stronger conjecture implies the dichotomy theorem for conservative *Constraint Satisfaction Problems* (CSP) [7, 56].

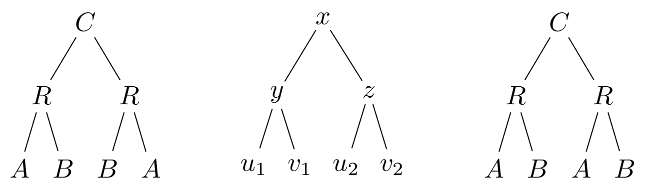
On the other hand, for self-join-free queries q in BCQ, the complexity of $\mathsf{CERTAINTY}(q)$ is well established by the next theorem.

Theorem 1.2 ([40]). *For every self-join-free query q in* BCQ, $\mathsf{CERTAINTY}(q)$ *is in* **FO**, **L**-*complete, or* **coNP**-*complete, and it is decidable in polynomial time in the size of q which of the three cases applies.*

Past research has indicated that the tools for proving Theorem 1.2 largely fall short in dealing with difficulties caused by self-joins. A notable example concerns *path queries*, i.e., queries of the form $\exists x_1 \cdots \exists x_{k+1}(R_1(\underline{x_1}, x_2) \wedge R_2(\underline{x_2}, x_3) \wedge \cdots \wedge R_k(\underline{x_k}, x_{k+1}))$. If a query of this form is self-join-free (i.e., if $R_i \neq R_j$ whenever $i \neq j$), then the "attack graph" tool [40] immediately tells us that $\mathsf{CERTAINTY}(q)$ is in **FO**. However, for path queries q with self-joins, $\mathsf{CERTAINTY}(q)$ exhibits a tetrachotomy between **FO**, **NL**-complete, **PTIME**-complete, and **coNP**-complete [32], and the complexity classification requires sophisticated tools. Note incidentally that self-join-freeness is a simplifying assumption that is also frequent outside CQA (e.g., [1, 5, 20, 21]).

A natural question is to extend the complexity classification for path queries to queries that are syntactically less constrained. In particular, while path queries are restricted to binary relation names, we aim for unrestricted arities, as in practical database systems, which brings us to the construct of tree queries.

A query q in BCQ is a *rooted (ordered) tree query* if it is uniquely (up to a variable renaming) representable by a rooted ordered tree in which each non-leaf vertex is labeled by a relation name, and each leaf vertex is labeled by a unary relation name, a constant, or $\perp$. The query q is read from this tree as follows: each vertex labeled by either a relation name or $\perp$ is first associated with a fresh variable, and each vertex labeled by a constant is associated with that same constant; then, a vertex labeled with relation name R and associated with variable x represents the query atom $R(\underline{x}, y_1, \ldots, y_n)$, where $y_1, \ldots, y_n$ are the symbols (variables or constants) associated with the left-to-right ordered children of the vertex x. The underlined position is the primary key. Note that a vertex labeled with a relation name of arity $n+1$ must have n children. For example, consider the rooted tree in Fig. 1(a) and associate fresh variables to its vertices as depicted in Fig. 1(b). The rooted tree thus represents a query q_1 that contains, among others, the atoms $C(\underline{x}, y, z)$ and $R(\underline{y}, u_1, v_1)$. It



(a) A rooted ordered tree representing q_1.

(b) Each vertex is associated with a fresh variable.

(c) A rooted ordered tree representing q_2.

Fig. 1. The left rooted ordered tree represents (up to a variable renaming) the Boolean conjunctive query q_1 with atoms $C(\underline{x}, y, z)$, $R(\underline{y}, u_1, v_1)$, $A(\underline{u_1})$, $B(\underline{v_1})$, $R(\underline{z}, u_2, v_2)$, $B(\underline{u_2})$, $A(\underline{v_2})$. The right rooted ordered tree represents q_2 with atoms $C(\underline{x}, y, z)$, $R(\underline{y}, u_1, v_1)$, $A(\underline{u_1})$, $B(\underline{v_1})$, $R(\underline{z}, u_2, v_2)$, $A(\underline{u_2})$, $B(\underline{v_2})$.

is easy to see that every path query is a rooted tree query. The class of all rooted tree queries is denoted TreeBCQ. We can now present our main results.

Theorem 1.3. *For every query q in* TreeBCQ, CERTAINTY(q) *is in* **FO**, **NL**-*hard* $\cap$ **LFP**, *or* **coNP**-*complete, and it is decidable in polynomial time in the size of q which of the three cases applies.*

Here **LFP** denotes least fixed point logic as defined in [41, p. 181] (a.k.a. **FO**[**LFP**]), and **NL** denotes the class of problems decidable by a non-deterministic Turing machine using only logarithmic space. The classification criteria implied in Theorem 1.3 are explicitly stated in Theorem 4.5.

It will turn out that subtree homomorphisms play a crucial role in the complexity classification of CERTAINTY(q) for queries q in TreeBCQ. For example, our results show that for the queries q_1 and q_2 represented in, respectively, Fig. 1(a) and (c), CERTAINTY(q_1) is **coNP**-complete, while CERTAINTY(q_2) is in **FO**. The difference occurs because the two ordered subtrees rooted at R are isomorphic in q_2 (A precedes B in both subtrees), but not in q_1. Another novel and useful tool in the complexity classification is a context-free grammar (CFG) that generalizes the NFA for path queries used in [32].

Once Theorem 1.3 is proved, it is natural to generalize rooted tree queries further by allowing queries that can be represented by graphs that are not trees. We thereto define GraphBCQ (Definition 9.1), a subclass of BCQ that extends TreeBCQ. In GraphBCQ queries, two distinct atoms can share a variable occurring at non-primary-key positions, which requires representations by DAGs rather than trees. Moreover, GraphBCQ gives up on the acyclicity requirement that is cooked into TreeBCQ. Significantly, we were able to establish the **FO**-boundary in the set $\{$CERTAINTY$(q) \mid q \in$ GraphBCQ$\}$.

Theorem 1.4. *For every query q in* GraphBCQ, *it is decidable whether or not* CERTAINTY(q) *is in* **FO**; *and when it is, a first-order rewriting can be effectively constructed.*

We have not achieved a fine-grained complexity classification of all problems in $\{$CERTAINTY$(q) \mid q \in$ GraphBCQ$\}$. However, we were able to do so for the set of Berge-acyclic queries in GraphBCQ, denoted Graph$_{\text{Berge}}$BCQ. Recall that a conjunctive query is Berge-acyclic if its incidence graph (i.e., the undirected bipartite graph that connects every variable x to all query atoms in which x occurs) is acyclic.

Theorem 1.5. *For every query q in* Graph$_{\text{Berge}}$BCQ, CERTAINTY(q) *is in* **FO**, **NL**-*hard* $\cap$ **LFP**, *or* **coNP**-*complete, and it is decidable in polynomial time in the size of q which of the three cases applies.*

Since TreeBCQ $\subsetneq$ $\text{Graph}_{\text{Berge}}$BCQ $\subsetneq$ GraphBCQ, Theorem 1.3 is subsumed by Theorem 1.5. We nevertheless provide Theorem 1.3 explicitly, because its proof makes up the main part of this paper. In Section 9.2, we will discuss the challenges in extending Theorem 1.5 beyond $\text{Graph}_{\text{Berge}}$BCQ.

The full version of this paper is available in arXiv [33].

2 RELATED WORK

Inconsistency management has been studied in various database contexts (e.g., graph databases [3, 4], medical databases [25], online databases [26], spatial databases [49]), and under different repair semantics (e.g., [13, 42, 52]). Arenas, Bertossi, and Chomicki initiated Consistent Query Answering (CQA) in 1999 [2]. Twenty years later, their contribution was acknowledged in a *Gems of PODS session* [6]. An overview of complexity classification results in CQA appeared in the *Database Principles* column of SIGMOD Record [55].

The term CERTAINTY(q) was coined in [53] 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 the class of self-join-free Boolean conjunctive queries underwent a series of efforts [22, 30, 34, 35, 38], until it was revealed that the complexity of CERTAINTY(q) for self-join-free conjunctive queries displays a trichotomy between **FO**, **L**-complete, and **coNP**-complete [36, 40]. A few extensions beyond this trichotomy result are known. Under the requirement of self-join-freeness, it remains decidable whether or not CERTAINTY(q) is in **FO** in the presence of negated atoms [37], multiple keys [39], and unary foreign keys [24].

Little is known concerning the complexity classification of the problem CERTAINTY(q) beyond self-join-free conjunctive queries. For the restricted class of Boolean path queries q, the complexity classification of CERTAINTY(q) already exhibits a tetrachotomy between **FO**, **NL**-complete, **PTIME**-complete and **coNP**-complete [32]. Padmanabha et al. [48] recently established a dichotomy between **PTIME** and **coNP**-complete for CERTAINTY(q) when q contains only two atoms allowing self-joins. Figueira et al. [18] have recently discovered a simple fixpoint algorithm that solves CERTAINTY(q) when q is a self-join free conjunctive query or a path query such that CERTAINTY(q) is in **PTIME**. As already discussed in Section 1, relationships have been found between CQA and CSP [19, 43].

The counting variant of the problem CERTAINTY(q), denoted $\sharp$CERTAINTY(q), asks to count the number of repairs that satisfy some Boolean query q. For self-join-free Boolean conjunctive queries, $\sharp$CERTAINTY(q) exhibits a dichotomy between **FP** and $\sharp$**PTIME**-complete [46]. This dichotomy has been shown to extend to queries with self-joins if primary keys are singletons [47], and to functional dependencies [11]. Calautti, Console, and Pieris present in [8] a complexity analysis of these counting problems under many-one logspace reductions and conducted an experimental evaluation of randomized approximation schemes for approximating the percentage of repairs that satisfy a given query [9]. CQA is also studied under different notions of repairs like operational repairs [10, 12] and preferred repairs [29, 50]. CQA has also been studied for queries with aggregation, in both theory and practice [16, 28].

Theoretical research in CQA has stimulated implementations and experiments in prototype systems, using different target languages and engines: SAT [15], ASP [27, 44, 45], BIP [31], SQL [17], logic programming [23], and hypergraph algorithms [14].

3 PRELIMINARIES

We assume disjoint sets of *variables* and *constants*. A *valuation* over a set U of variables is a total mapping θ from U to the set of constants.

Atoms and key-equal facts. Every relation name has a fixed arity, and a fixed set of primary-key positions. We will underline primary-key positions and assume w.l.o.g. that all primary-key positions precede all other positions. An *atom* is then an expression $R(\underline{s_1, \ldots, s_k}, s_{k+1}, \ldots, s_n)$ where each s_i is a variable or a constant for $1 \leq i \leq n$. The sequence $s_1, \ldots, s_k$ is called the *primary key* (of the atom). This primary key is called *simple* if $k = 1$, and *constant-free* if no constant occurs in it. An atom without variables is a *fact*. Two facts are *key-equal* if they use the same relation name and agree on the primary key.

Database instances, 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 instance* (or *database* for short) is a finite set **db** of facts using only the relation names of the schema. We write $\text{adom}(\mathbf{db})$ for the active domain of **db** (i.e., the set of constants that occur in **db**). A *block* of **db** is a maximal set of key-equal facts of **db**. Whenever a database instance **db** is understood, we write $R(\underline{\vec{c}}, *)$ for the block that contains all facts with relation name R and primary-key value $\vec{c}$, where $\vec{c}$ is a sequence of constants. A database instance **db** is *consistent* if it contains no two distinct facts that are key-equal (i.e., if no block of **db** contains more than one fact). A *repair* of **db** is an inclusion-maximal consistent subset of **db**.

Boolean conjunctive queries. 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, representing the first-order sentence $\exists u_1 \cdots \exists u_k (R_1(\underline{\vec{x}_1}, \vec{y}_1) \wedge \cdots \wedge R_n(\underline{\vec{x}_n}, \vec{y}_n))$. We denote $\mathbf{vars}(q) = \{u_1, \ldots, u_k\}$, the set of variables that occur in q, and write $\text{const}(q)$ for the set of constants that occur in q. We write BCQ for the class of Boolean conjunctive queries.

Let q be a query in BCQ. We say that q has a *self-join* if some relation name occurs more than once in q. If q has no self-joins, it is called *self-join-free*. We say that q is *minimal* if it is not equivalent to a query in BCQ with a strictly smaller number of atoms.

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

Rooted relation trees. A *rooted relation tree* is a (directed) rooted ordered tree where each internal vertex is labeled by a relation name, and each leaf vertex is labeled with either a unary relation name, a constant, or $\perp$, such that every two vertices sharing the same label have the same number of children. We denote by τ^u_Δ the subtree rooted at vertex u in τ. Any rooted relation tree τ has a string representation recursively defined as follows: the string representation of a tree with only one vertex is the label of that vertex; otherwise, if the root of τ is labeled R and has the following ordered children $v_1, v_2, \ldots, v_n$, then τ's string representation is $R(s_1, s_2, \ldots, s_n)$, where s_i is the string representation of $\tau^{v_i}_\Delta$. For example, the tree in Fig. 1(a) has string representation $C(R(A, B), R(B, A))$. We will often blur the distinction between rooted relation trees and their string representation.

Rooted tree query and rooted tree sets. A *querification* of a rooted relation tree τ is a total function f with domain τ's vertex set that maps each vertex labeled by a constant to that same constant, and injectively maps all other vertices to variables. Such a querification naturally extends to a mapping $f(\tau)$ of the entire tree: if u is a vertex in τ with label R and children $v_1, v_2, \ldots, v_n$, then $f(\tau)$ contains the atom $R(\underline{f(u)}, f(v_1), f(v_2), \ldots, f(v_n))$. A Boolean conjunctive query is a *rooted tree query* if it is equal to $f(\tau)$ for some querification f of some rooted relation tree τ. If $q = f(\tau)$, we also say that q is *represented* by τ, in which case we often blur the distinction between q and τ. We write $R[x]$ for the unique R-atom in q with primary key variable x. TreeBCQ denotes the class of rooted tree queries. It can be verified that every rooted tree query is minimal.

Every query q in TreeBCQ is represented by a unique rooted relation tree. Conversely, every rooted relation tree represents a query in TreeBCQ that is unique up to a variable renaming. When $f(\tau) = q$, by a slight abuse of terminology, we may use q to refer to τ, and use the query variable x (or the expression $R[x]$) to refer to the vertex u in τ that satisfies $f(u) = x$ and whose label is R. The variable r is the *root variable* of a query q in TreeBCQ if r is the root vertex of q's rooted relation tree. For two distinct vertices x and y, we write $x <_q y$ if the vertex x is an ancestor of y in q, and write $x \parallel_q y$ if neither $x <_q y$ nor $y <_q x$. When x and y have the same label R, we can also write $R[x] <_q R[y]$ and $R[x] \parallel_q R[y]$ instead of $x <_q y$ and $x \parallel_q y$ respectively. For every variable x in a rooted tree query q, we write $q^x_\triangle$ for the subquery of q whose rooted relation tree is the subtree rooted at vertex x in q. A variable x is a leaf variable in q if $q^x_\triangle = \perp$, $q^x_\triangle = c$, or $q^x_\triangle = A$, for some constant c or unary relation name A.

An *instantiation* of a rooted relation tree τ is a total function g from τ's vertex set to constants such that each vertex labeled by a constant c is mapped to c. Such an instantiation naturally extends to a mapping $g(\tau)$ of the entire tree: if u is a vertex in τ with label R and children $v_1, v_2, \ldots, v_n$, then $g(\tau)$ contains the fact $R(\underline{g(u)}, g(v_1), g(v_2), \ldots, g(v_n))$. A subset S of **db** is a *rooted tree set in* **db** *starting in* c if $S = g(\tau)$ for some instantiation g of τ that maps τ's root to c. A case of particular interest is when **db** is consistent, in particular, when **db** is a repair. It can be verified that a rooted tree set in a repair **r** is uniquely determined by a constant c and a rooted tree τ (because only one instantiation is possible); by overloading terminology, τ is also called a rooted tree set in **r** starting in c. For convenience, an empty rooted tree set, denoted by $\perp$, starts in any constant c.

Homomorphism. Let $p, q \in$ BCQ. We write $p \leq_\rightarrow q$ if there exists a homomorphism from p to q, i.e., a mapping $h : \mathsf{vars}(p) \rightarrow \mathsf{vars}(q) \cup \mathsf{const}(q)$ that acts as identity when applied on constants, such that for every atom $R(\underline{\vec{x}}, \vec{y})$ in p, $R(\underline{h(\vec{x})}, h(\vec{y}))$ is an atom of q. For $u \in \mathbf{vars}(p)$ and $v \in \mathbf{vars}(q)$, we write $p \leq_{u \rightarrow v} q$ if there exists a homomorphism h from p to q with $h(u) = v$. It can now be verified that for rooted tree queries p and q, there is a homomorphism h from p to q if and only if there is a label-preserving graph homomorphism from the rooted relation tree of p to that of q (we assume that a leaf vertex with label $\perp$ can map to a vertex with any label). Since rooted relation trees are *ordered* trees, graph homomorphisms must evidently be order-preserving. For example, there is no homomorphism between the trees $R(A, B)$ and $R(B, A)$.

Example 3.1. The following rooted tree query and its rooted relation tree are depicted in Fig. 2:

$$q = \{A(\underline{x_0}, x_1, x_2), R(\underline{x_1}, x_3, x_4), R(\underline{x_2}, x_5, x_6), R(\underline{x_3}, x_7, x_8), U(\underline{x_7}), X(\underline{x_4}, c_1), Y(\underline{x_5}, x_9), Z(\underline{x_6}, c_2, x_{10})\}.$$

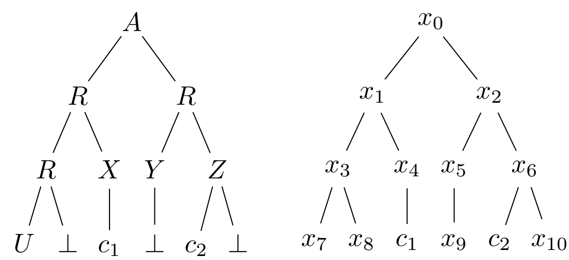
We have:

$$\begin{aligned} q^{x_1}_\triangle &= R(\underline{x_1}, x_3, x_4), R(\underline{x_3}, x_7, x_8), U(\underline{x_7}), X(\underline{x_4}, c_1) \\ &= R(R(U, \perp), X(c_1)), \\ q^{x_2}_\triangle &= R(\underline{x_2}, x_5, x_6), Y(\underline{x_5}, x_9), Z(\underline{x_6}, c_2, x_{10}) \\ &= R(Y(\perp), Z(c_2, \perp)), \\ q^{x_3}_\triangle &= R(\underline{x_3}, x_7, x_8), U(\underline{x_7}) \\ &= R(U, \perp). \end{aligned}$$

In this query q, we have $R[x_1] \parallel_q R[x_2]$, $R[x_1] <_q R[x_3]$, and $R[x_2] \parallel_q R[x_3]$.

4 THE COMPLEXITY CLASSIFICATION

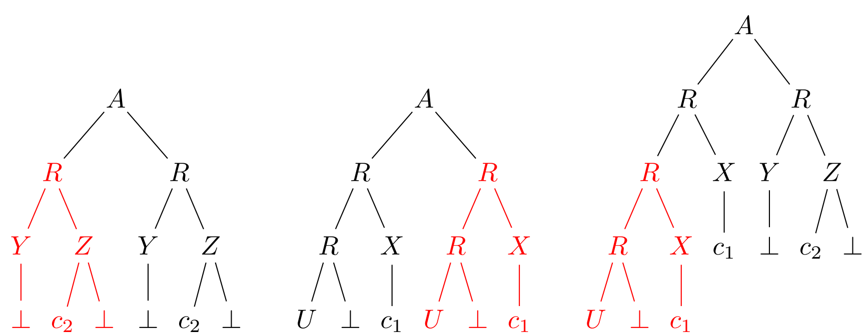
Our classification focuses on rooted tree queries (TreeBCQ). We will extend to $\mathsf{Graph}_{\mathsf{Berge}}\mathsf{BCQ}$ and GraphBCQ in Section 9. The classification of path queries in [32] uses a notion of "rewinding" to deal with self-joins: a path query $u \cdot Rv \cdot Rw$ rewinds to $u \cdot Rv \cdot Rv \cdot Rw$. Very informally, rewinding captures that query atoms with the same relation name can be "confused" with one another (or "rewind" to one another in our terminology) during query evaluation: in $u \cdot Rv \cdot Rw$, once we have evaluated the prefix $u \cdot Rv \cdot R$, the last R can be confused with the first one, in which case we



(a) A rooted relation tree τ

(b) A mapping f from vertices in τ to variables in q

Fig. 2. An example rooted relation tree, where c_1 and c_2 are constants.



(a) $q^{R:x_1 \hookrightarrow x_2}$

(b) $q^{R:x_2 \hookrightarrow x_1}$

(c) $q^{R:x_3 \hookrightarrow x_1}$

Fig. 3. An illustration of rewinding for the query of Fig. 2; the modified subtrees are highlighted in red.

continue with the suffix $Rv \cdot Rw$ (instead of merely Rw). We generalize the notion of rewinding from path queries to rooted tree queries.

Definition 4.1 (Rewinding). Let q be a query in TreeBCQ. Let $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ be two (not necessarily distinct) atoms in q. We define $q^{R:y \hookrightarrow x}$ as the following rooted tree query

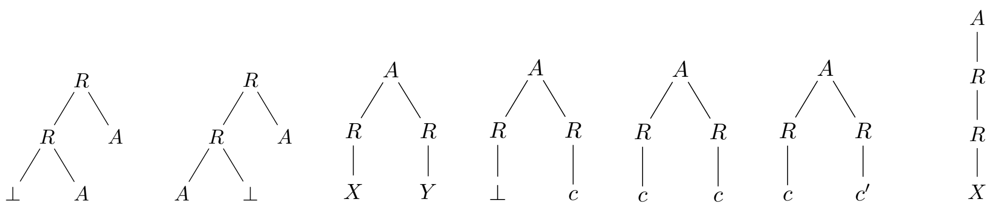
$$q^{R:y \hookrightarrow x} := \left(q \setminus q^{y}_{\triangle}\right) \cup f(q^{x}_{\triangle}),$$

for some isomorphism f that maps x to y (i.e., $f(x) = y$), and maps every other variable in $q^{x}_{\triangle}$ to a fresh variable.

Intuitively, the rooted tree query $q^{R:y \hookrightarrow x}$ can be obtained by replacing $q^{y}_{\triangle}$ with a fresh copy of $q^{x}_{\triangle}$. Fig. 3 presents some rooted tree queries obtained from rewinding on the rooted tree q in Fig. 2.

The classification criteria in [32] uses the notions of factors and prefixes that are specific to words, which can be generalized using homomorphism on rooted tree queries. Consider the following syntactic conditions on a rooted tree query q with root variable r:

- C_2 : for every two atoms $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ in q, either $q \leq_{\rightarrow} q^{R:y \hookrightarrow x}$ or $q \leq_{\rightarrow} q^{R:x \hookrightarrow y}$.
- C_1 : for every two atoms $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ in q, either $q \leq_{r \rightarrow r} q^{R:y \hookrightarrow x}$ or $q \leq_{r \rightarrow r} q^{R:x \hookrightarrow y}$.



(a) C_1 (b) $\mathsf{C}_2, \neg\mathsf{C}_{\text{prefix}}$ (c) $\neg\mathsf{C}_{\text{branch}}$ (d) C_1 (e) C_1 (f) $\neg\mathsf{C}_{\text{branch}}$ (g) $\neg\mathsf{C}_{\text{factor}}$

Fig. 4. Examples of rooted relation trees. Trees annotated with $\neg\mathsf{C}$ violate syntactic condition C, while trees annotated with C satisfy C. For example, the tree in (a) satisfies C_1; and the tree (b) satisfies C_2 but violates $\mathsf{C}_{\text{prefix}}$.

It is easy to see that conditions C_1 and C_2 are decidable in polynomial time in the size of the query. We may restate C_2 and C_1 using more fine-grained syntactic conditions below.

- $\mathsf{C}_{\text{branch}}$: for every two atoms $R[x] \parallel_q R[y]$ in q, either $q^y_\triangle \leq_{y\to x} q^x_\triangle$ or $q^x_\triangle \leq_{x\to y} q^y_\triangle$.
- $\mathsf{C}_{\text{factor}}$: for every two atoms $R[x] <_q R[y]$ in q, we have $q \leq_\to q^{R:y\hookrightarrow x}$.
- $\mathsf{C}_{\text{prefix}}$: for every two atoms $R[x] <_q R[y]$ in q, we have $q \leq_{r\to r} q^{R:y\hookrightarrow x}$.

Lemma 4.2. *For every two atoms $R[x] \parallel_q R[y]$ in a rooted tree query q, we have $q \leq_\to q^{R:y\hookrightarrow x}$ if and only if $q^y_\triangle \leq_{y\to x} q^x_\triangle$.*

For the sake of simplicity, we postpone the proof of Lemma 4.2 to Appendix A. Lemma 4.2 implies the following connections among the syntactic conditions.

Proposition 4.3. $\mathsf{C}_2 = \mathsf{C}_{\text{factor}} \land \mathsf{C}_{\text{branch}}$, $\mathsf{C}_1 = \mathsf{C}_{\text{prefix}} \land \mathsf{C}_{\text{branch}}$.

Example 4.4. Let q be as in Fig. 2. We have that q violates $\mathsf{C}_{\text{branch}}$ (and therefore C_2), since there is no homomorphism from q to neither $q^{R:x_1\hookrightarrow x_2}$ nor $q^{R:x_2\hookrightarrow x_1}$.

Fig. 4 shows some example rooted relation trees annotated with the syntactic conditions they satisfy or violate.

Our main classification result can now be stated.

Theorem 4.5 (Trichotomy Theorem). *For every query q in* TreeBCQ,

- *if q satisfies C_2, then the problem* CERTAINTY(q) *is in* **LFP**; *otherwise it is* **coNP**-*complete; and*
- *if q satisfies C_1, then the problem* CERTAINTY(q) *is in* **FO**; *otherwise it is* **NL**-*hard.*

Let us provide some intuitions behind Theorem 4.5. Both $\mathsf{C}_{\text{prefix}}$ and $\mathsf{C}_{\text{factor}}$ concern the homomorphism from q to the rooted tree query obtained by rewinding from a subtree to its ancestor subtree, which resembles the case on path queries. The condition $\mathsf{C}_{\text{branch}}$ is vacuously satisfied for path queries, but is crucial to the classification of rooted tree queries.

For the complexity lower bound, if q violates $\mathsf{C}_{\text{branch}}$, then CERTAINTY($q$) is **coNP**-hard. Intuitively, this is because if $q^x_\triangle$ and $q^y_\triangle$ are not homomorphically comparable and appear in different branches, then the facts in their common ancestor relation may "choose" which branch to satisfy, which allows us to reduce from SAT in item (1) of Proposition 8.1. For example, consider the query q_1 as in Fig. 1(a) and the example database instance **db** in Fig. 5. It can be shown that there is a repair of **db** that falsifies q_1 if and only if the following CNF formula is satisfiable:

$$\underbrace{(x_1 \lor x_2)}_{C_1} \land \underbrace{(\overline{x_1} \lor \overline{x_2})}_{C_2}.$$

	C	$\underline{1}$	2	3
*		c_1	x_1	z_-
		c_1	x_2	z_-
		c_2	z_+	x_1
*		c_2	z_+	x_2

	R	$\underline{1}$	2	3
		x_1	a	b
*		x_1	b	a
*		x_2	a	b
		x_2	b	a
*		z_+	a	b
*		z_-	b	a

	A	$\underline{1}$
*		a

	B	$\underline{1}$
*		b

Fig. 5. An inconsistent database instance **db** for CERTAINTY(q_1), where q_1 is represented in Fig. 1(a). Blocks are separated by dashed lines. The facts with $*$ form a repair that falsifies q_1, corresponding to a satisfying truth assignment $x_1 = 1$ and $x_2 = 0$.

For the complexity upper bound, if $q^y_\Delta \preceq_{y\to x} q^x_\Delta$, the arguments above fail because the facts in their common ancestor relation cannot "choose" which branch to satisfy anymore: informally, whenever q^x_Δ is satisfied, q^y_Δ will be satisfied due to the homomorphism. This crucial observation from $\mathsf{C}_{\mathsf{branch}}$ also leads to a total preorder on all self-joining atoms, which allows us to deal with self-joining atoms in different branches as if they were on a path.

Definition 4.6 (Relation $\preceq_q$). Let q be a query in TreeBCQ. Let $R[x]$ and $R[y]$ be two atoms in q. We write $R[x] \preceq_q R[y]$ if either $R[x] <_q R[y]$ or $q^y_\Delta \preceq_{y\to x} q^x_\Delta$.

Proposition 4.7. *Let q be a query in* TreeBCQ *satisfying* $\mathsf{C}_{\mathsf{branch}}$. *For every relation name R, the relation $\preceq_q$ is a total preorder on all R-atoms in q.*

Proof Sketch. We first show that every two distinct atoms $R[x]$ and $R[y]$ are comparable by $\preceq_q$. Let $R[x]$ and $R[y]$ be two distinct atoms in q. The claim holds if $R[x] <_q R[y]$ or $R[y] <_q R[x]$. Otherwise, we have $R[x] \parallel_q R[y]$, and since q satisfies $\mathsf{C}_{\mathsf{branch}}$, we have either $q^x_\Delta \preceq_{x\to y} q^y_\Delta$ or $q^y_\Delta \preceq_{y\to x} q^x_\Delta$, as desired. In Appendix A, we show that $\preceq_q$ is transitive. □

The remainder of this paper is organized as follows. Section 5 defines a context-free grammar $\mathsf{CFG}^\clubsuit(q)$ for each $q \in$ TreeBCQ, and the problem $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ that concerns $\mathsf{CFG}^\clubsuit(q)$. Lemma 5.4 concludes the equivalence of CERTAINTY(q) and $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ if q satisfies C_2 (or C_1). In Section 6, we show that $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ is in **LFP** (and in **PTIME**) if q satisfies $\mathsf{C}_{\mathsf{branch}}$. In Sections 7 and 8, we show the upper bounds and lower bounds in Theorem 4.5 respectively. In Section 9, we prove Theorems 1.4 and 1.5.

5 CONTEXT-FREE GRAMMAR

We first generalize NFAs used in the study of path queries [32] to context-free grammars (CFGs).

Definition 5.1 ($\mathsf{CFG}^\clubsuit(q)$). Let q be a query in TreeBCQ with root variable r. We define a context-free grammar $\mathsf{CFG}^\clubsuit(q)$ over the string representations of rooted relation trees for each rooted tree query q. The alphabet Σ of $\mathsf{CFG}^\clubsuit(q)$ contains every relation symbol and constant in q, open/close parentheses, $\perp$ and comma.

Whenever v is a variable or a constant in q, there is a nonterminal symbol S_v. Every symbol in Σ is a terminal symbol. The rules of $\mathsf{CFG}^\clubsuit(q)$ are as follows:

- for each atom $R[y] = R(\underline{y}, y_1, y_2, \ldots, y_n)$ in q, there is a forward production rule

$$S_y \to_q R(S_{y_1}, S_{y_2}, \ldots, S_{y_n}) \tag{1}$$

- whenever $R[x]$ and $R[y]$ are atoms in q such that $R[x] <_q R[y]$, there is a backward production rule

$$S_y \longrightarrow_q S_x \tag{2}$$

- for every leaf variable u whose label L is either $\perp$ or a unary relation name, there is a rule

$$S_u \longrightarrow_q L \tag{3}$$

- for each constant c in q, there is a rule

$$S_c \longrightarrow_q c \tag{4}$$

The starting symbol of $\mathsf{CFG}^{\clubsuit}(q)$ is S_r where r is the root variable of q. A rooted relation tree τ is accepted by $\mathsf{CFG}^{\clubsuit}(q)$, denoted as $\tau \in \mathsf{CFG}^{\clubsuit}(q)$, if the string representation of τ can be derived from S_r, written as $S_r \overset{*}{\longrightarrow}_q \tau$.

Example 5.2. Let q be as in Fig. 2(a) with variables labeled as in Fig. 2(b). The rooted relation tree τ in Fig. 3(c) has string representation $\tau = A(\tau_1, \tau_2)$ where

$$\tau_1 = R(R(R(U, \perp), X(c_1)), X(c_1)),$$
$$\tau_2 = R(Y(\perp), Z(c_2, \perp)).$$

We have $S_{x_2} \overset{*}{\longrightarrow}_q \tau_2$ by applying only forward rewrite rules. We show next $S_{x_1} \overset{*}{\longrightarrow}_q \tau_1$, using the backward rewrite rule $S_{x_3} \longrightarrow_q S_{x_1}$ at some point, highlighted in red:

$$\begin{aligned}
S_{x_1} &\longrightarrow_q R(S_{x_3}, S_{x_4}) \\
&\longrightarrow_q R(S_{x_1}, X(S_{c_1})) \\
&\longrightarrow_q R(R(S_{x_3}, S_{x_4}), X(c_1)) \\
&\longrightarrow_q R(R(R(S_{x_7}, S_{x_8}), X(S_{c_1})), X(c_1)) \\
&\longrightarrow_q R(R(R(U, \perp), X(c_1)), X(c_1)) \\
&= \tau_1.
\end{aligned}$$

Thus $S_{x_0} \longrightarrow_q A(S_{x_1}, S_{x_2}) \overset{*}{\longrightarrow}_q A(\tau_1, \tau_2) = \tau$. Consequently, τ is accepted by $\mathsf{CFG}^{\clubsuit}(q)$.

Recall from Section 3 that a rooted tree set in a repair **r** is uniquely determined by a rooted tree τ and a constant c; such a rooted tree set is said to be accepted by $\mathsf{CFG}^{\clubsuit}(q)$ if $\tau \in \mathsf{CFG}^{\clubsuit}(q)$. For our technical treatment later, we next define modifications of $\mathsf{CFG}^{\clubsuit}(q)$ by changing its starting terminal.

Definition 5.3 (S-$\mathsf{CFG}^{\clubsuit}(q, u)$)*.* For a query q in TreeBCQ and a variable or constant u in q, we define S-$\mathsf{CFG}^{\clubsuit}(q, u)$ as the context-free grammar that accepts a rooted relation tree τ if and only if $S_u \overset{*}{\longrightarrow}_q \tau$.

We now introduce the *certain trace problem.* For each q in TreeBCQ, $\mathsf{CERTAIN_{tr}}(q)$ is defined as the following decision problem:

PROBLEM $\mathsf{CERTAIN_{tr}}(q)$
Input: A database instance **db**.
Question: Is there a constant $c \in \mathrm{adom}(\mathbf{db})$ such that for every repair **r** of **db**, there is a rooted tree set τ in **r** starting in c with $\tau \in \mathsf{CFG}^{\clubsuit}(q)$?

The problems $\mathsf{CERTAINTY}(q)$ and $\mathsf{CERTAIN_{tr}}(q)$ reduce to each other if q satisfies C_2.

Initialization Step: **for every** $c \in \text{adom}(\mathbf{db})$ and leaf variable or constant u in q
add $\langle c, u\rangle$ to B **if** $u = c$ is a constant,
or the label of variable u in q is either $\perp$,
or L with $L(\underline{c}) \in \mathbf{db}$.

Iterative Rule: **for every** $c \in \text{adom}(\mathbf{db})$ and atom $R(\underline{y}, y_1, y_2, \ldots, y_n)$ in q
add $\langle c, y\rangle$ to B **if** the following formula holds:

$$\exists \vec{d} : R(\underline{c}, \vec{d}) \in \mathbf{db} \land \forall \vec{d} : \left(R(\underline{c}, \vec{d}) \in \mathbf{db} \rightarrow \text{fact}(R(\underline{c}, \vec{d}), y)\right),$$

where

$$\text{fact}(R(\underline{c}, \vec{d}), y) = \underbrace{\left(\bigwedge_{1 \le i \le n} \langle d_i, y_i\rangle \in B\right)}_{\text{forward production}} \lor \underbrace{\left(\bigvee_{R[x] <_q R[y]} \text{fact}(R(\underline{c}, \vec{d}), x)\right)}_{\text{backward production}}$$

and $\vec{d} = \langle d_1, d_2, \ldots, d_n\rangle$.

Fig. 6. A fixpoint algorithm for computing a set B, for a fixed rooted tree q.

Lemma 5.4. *Let q be a query in* TreeBCQ *satisfying* C_2. *Let* **db** *be a database instance. Then,* **db** *is a "yes"-instance of* CERTAINTY(q) *if and only if* **db** *is a "yes"-instance of* $\text{CERTAIN}_{\text{tr}}(q)$.

The proof of Lemma 5.4 is deferred to Section 7; it uses results developed in the next section.

6 MEMBERSHIP OF $\text{CERTAIN}_{\text{tr}}(q)$ IN LFP

In this section, we show that the problem $\text{CERTAIN}_{\text{tr}}(q)$ is expressible in **LFP** (and thus in **PTIME**) if q satisfies $\mathsf{C}_{\text{branch}}$. Let **db** be a database instance. Consider the algorithm in Fig. 6, following a dynamic programming fashion. The algorithm iteratively computes a set B of pairs $\langle c, y\rangle$ until it reaches a fixpoint, ensuring that

> whenever $\langle c, y\rangle$ is added to B, then every repair of **db** contains a rooted tree set starting in c that is accepted by $\text{S-CFG}^{\clubsuit}(q, y)$.

Intuitively, this holds true because $\langle c, y\rangle$ is added to B if for every possible fact $f = R(\underline{c}, \vec{d})$ that can be chosen by a repair of **db**, the context-free grammar $\text{S-CFG}^{\clubsuit}(q, y)$ can proceed by firing forward rule with nonterminal S_y that consumes f from the rooted tree set, or by non-deterministically firing some backward rule of the form $S_y \rightarrow_q S_x$.

The formal semantics for each pair $\langle c, y\rangle$ is stated in Lemma 6.1.

Lemma 6.1. *Let q be a query in* TreeBCQ *satisfying* $\mathsf{C}_{\text{branch}}$. *Let* **db** *be a database instance. Let B be the output of the algorithm in Fig. 6. Then for every constant $c \in$* adom(**db**) *and every variable or constant y in q, the following statements are equivalent:*

(1) $\langle c, y\rangle \in B$; and
(2) for every repair **r** *of* **db**, *there exists a rooted tree set τ in* **r** *starting in c such that $\tau \in \text{S-CFG}^{\clubsuit}(q, y)$.*

The crux in the proof of Lemma 6.1 relies on the existence of repairs called *frugal*: to show item (2) of Lemma 6.1, it will be sufficient to show that it holds true for frugal repairs. Frugal repairs also turn out to be useful in proving Lemma 5.4 and offer an alternative perspective to the algorithm, as stated in Corollary 7.5.

6.1 Frugal repairs

We first show that the evaluation result of the predicate "fact" and the membership in B in the algorithm of Fig. 6 propagate along the total preorder $\preceq_q$.

Lemma 6.2. *Let q be a query in* TreeBCQ *satisfying* $\mathsf{C}_{\mathsf{branch}}$*, and* **db** *a database instance. Let $R[x], R[y]$ be two atoms of q. Then for every fact $R(\underline{c}, \vec{d})$ in* **db** *and two atoms $R[x] \preceq_q R[y]$,*

(1) if $\mathsf{fact}(R(\underline{c}, \vec{d}), x)$ is true, then $\mathsf{fact}(R(\underline{c}, \vec{d}), y)$ is true, with fact *as defined in Fig. 6; and*
(2) if $\langle c, x\rangle \in B$, then $\langle c, y\rangle \in B$, where B is the output of the algorithm of Fig. 6.

The technical proof of Lemma 6.2 is deferred to Appendix B.

Definition 6.3 (Frugal Set). Let q be a query in TreeBCQ satisfying $\mathsf{C}_{\mathsf{branch}}$, and **db** a database instance. Let $f = R(\underline{c}, \vec{d})$ be an R-fact in **db**. We define the frugal set of f in **db** with respect to q as

$$\mathsf{FrugalSet}_q(f, \mathbf{db}) = \{R[x] \in q \mid \mathsf{fact}(R(\underline{c}, \vec{d}), x) \text{ is true}\}.$$

Lemma 6.4. *Let q be a query in* TreeBCQ *satisfying* $\mathsf{C}_{\mathsf{branch}}$*, and* **db** *a database instance. For every two key-equal facts f and g in* **db***, the sets $\mathsf{FrugalSet}_q(f, \mathbf{db})$ and $\mathsf{FrugalSet}_q(g, \mathbf{db})$ are comparable by $\subseteq$.*

Proof. Suppose for contradiction that there exist two key-equal facts $f = R(\underline{c}, \vec{d_1})$ and $g = R(\underline{c}, \vec{d_2})$ in **db** such that $R[x] \in \mathsf{FrugalSet}_q(f, \mathbf{db}) \setminus \mathsf{FrugalSet}_q(g, \mathbf{db})$ and $R[y] \in \mathsf{FrugalSet}_q(g, \mathbf{db}) \setminus \mathsf{FrugalSet}_q(f, \mathbf{db})$. By Proposition 4.7, assume without loss of generality that $R[x] \preceq_q R[y]$. Then since $R[x] \in \mathsf{FrugalSet}_q(f, \mathbf{db})$, we have $\mathsf{fact}(R(\underline{c}, \vec{d_1}), x)$ is true, and thus $\mathsf{fact}(R(\underline{c}, \vec{d_1}), y)$ is true by Lemma 6.2, and hence $R[y] \in \mathsf{FrugalSet}_q(f, \mathbf{db})$, a contradiction. A similar contradiction can also be reached if $R[y] \preceq_q R[x]$. This completes the proof. □

Informally, by Lemma 6.4, among all facts of a non-empty block $R(\underline{c}, *)$ in **db**, there is a (not necessarily unique) fact $R(\underline{c}, \vec{d})$ with a $\subseteq$-minimal frugal set in **db**. The repair $\mathbf{r}^*$ of **db** containing all such facts is frugal in the sense that each fact in it satisfies as few R-atoms as possible; and if $\mathbf{r}^*$ contains a rooted tree set τ starting in c accepted by $\mathsf{S\text{-}CFG}^{\clubsuit}(q, y)$, so will every repair of **db**. We now formalize this idea, and then show Lemma 6.6 as an easy consequence.

Definition 6.5 (Frugal repair). Let q be a query in TreeBCQ satisfying $\mathsf{C}_{\mathsf{branch}}$. Let **db** be a database instance. A *frugal repair* $\mathbf{r}^*$ of **db** with respect to q is constructed by picking, from each block $R(\underline{c}, *)$ of **db**, a fact $R(\underline{c}, \vec{d})$ which $\subseteq$-minimizes $\mathsf{FrugalSet}_q(R(\underline{c}, \vec{d}), \mathbf{db})$.

Lemma 6.6. *Let q be a rooted tree query satisfying* $\mathsf{C}_{\mathsf{branch}}$*. Let* **db** *be a database instance. Let $\mathbf{r}^*$ be a frugal repair of* **db** *with respect to q and let $R(\underline{c}, \vec{d}) \in \mathbf{r}^*$. Let $R[u]$ be an atom in q. If $\mathsf{fact}(R(\underline{c}, \vec{d}), u)$ is true, then $\langle c, u\rangle \in B$.*

Proof. Let $R(\underline{c}, \vec{b})$ be an arbitrary fact in the block $R(\underline{c}, *)$ in **db**. By construction of a frugal repair, we have that $\mathsf{FrugalSet}_q(R(\underline{c}, \vec{d}), \mathbf{db}) \subseteq \mathsf{FrugalSet}_q(R(\underline{c}, \vec{b}), \mathbf{db})$. Since $R(\underline{c}, \vec{d}) \in \mathbf{r}^*$ and $\mathsf{fact}(R(\underline{c}, \vec{d}), u)$ is true, we have $R[u] \in \mathsf{FrugalSet}_q(R(\underline{c}, \vec{d}), \mathbf{db})$. Thus, $R[u] \in \mathsf{FrugalSet}_q(R(\underline{c}, \vec{b}), \mathbf{db})$ and $\mathsf{fact}(R(\underline{c}, \vec{b}), u)$ is true. Hence $\langle c, u\rangle \in B$. □

Lemma 6.7 shows a desirable property of frugal repairs.

Lemma 6.7. *Let q be a query in* TreeBCQ *satisfying* $\mathsf{C}_{\mathsf{branch}}$*. Let* **db** *be a database instance. Let $\mathbf{r}^*$ be a frugal repair of* **db** *with respect to q. If there is a rooted tree set τ in $\mathbf{r}^*$ starting in c such that $\tau \in \mathsf{S\text{-}CFG}^{\clubsuit}(q, y)$, then $\langle c, y\rangle \in B$.*

Proof. Let τ be a rooted tree set starting in c in $\mathbf{r}^*$ such that $\tau \in$ S-CFG$^\clubsuit(q, y)$. We recursively define a tree trace $\mathcal{T}$ on nodes of the form (c, x, τ), where $c \in \text{adom}(\mathbf{r}^*)$, x is a variable in q, and τ is a rooted relation tree, as follows:

- the root node of $\mathcal{T}$ is (c, y, τ); and
- whenever (a, u, σ) is a node in $\mathcal{T}$ with a rooted tree set σ starting in a in $\mathbf{r}^*$ for an atom $R(\underline{u}, u_1, u_2, \ldots, u_n)$ in q and fact $R(\underline{a}, b_1, b_2, \ldots, b_n)$ in $\mathbf{r}^*$,
  (i) if S-CFG$^\clubsuit(q, y)$ invokes a forward production rule $S_u \rightarrow_q R(S_{u_1}, S_{u_2}, \ldots, S_{u_n})$, then the node (a, u, σ) has n outgoing R-edges to its children (b_1, u_1, τ_1), (b_2, u_2, τ_2), …, (b_n, u_n, τ_n); or
  (ii) if S-CFG$^\clubsuit(q, y)$ invokes a backward production rule $S_u \rightarrow_q S_v$, then the node (a, u, σ) has a single outgoing ε-edge to its only child (a, v, σ).

The tree trace $\mathcal{T}$ succinctly records the rule productions that witness $\tau \in$ S-CFG$^\clubsuit(q, y)$ in $\mathbf{r}^*$. We use a structural induction to show that for every node (a, u, σ) in $\mathcal{T}$, $\langle a, u \rangle \in B$.

- Basis. Let (a, u, σ) be a leaf node in $\mathcal{T}$. If $\sigma = \perp$, then $\langle a, u \rangle \in B$. If $\sigma = L$ starting in a in $\mathbf{r}^*$ for some unary relation name L, then $L(a)$ is in $\mathbf{db}$ and thus $\langle a, u \rangle \in B$. If $\sigma = c$ for some constant c, since $\tau \in$ S-CFG$^\clubsuit(q, y)$, we must have $u = c = a$ at the leaf, and thus $\langle a, u \rangle = \langle a, a \rangle \in B$. Hence the claim holds for every leaf node (a, u, σ) in $\mathcal{T}$.
- Inductive step. Let (a, u, σ) be a node in $\mathcal{T}$. Assume that for every child node (b, w, σ') of (a, u) in $\mathcal{T}$ (possibly $b = a$), $\langle b, w \rangle \in B$. It suffices to argue that for the atom $R[u] = R(\underline{u}, u_1, u_2, \ldots, u_n)$ in q, $\langle a, u \rangle \in B$.
  (i) Case that (a, u, σ) has child nodes (b_1, u_1, τ_1), (b_2, u_2, τ_2), …, (b_n, u_n, τ_n) in $\mathcal{T}$ with $\sigma = R(\tau_1, \tau_2, \ldots, \tau_n)$. By the inductive hypothesis $\langle b_i, u_i \rangle \in B$ for every $1 \leq i \leq n$, which yields that fact$(R(\underline{a}, \vec{b}), u)$ is true, where $\vec{b} = \langle b_1, b_2, \ldots, b_n \rangle$. Then by Lemma 6.6, $\langle a, u \rangle \in B$.
  (ii) Case that (a, u, σ) has a child node (a, v, σ) in $\mathcal{T}$ connected with an ε-edge. Then there is some atom $R[v]$ with $R[v] <_q R[u]$. By the inductive hypothesis on the child (a, v, σ), $\langle a, v \rangle \in B$. Hence $\langle a, u \rangle \in B$ by Lemma 6.2.

This completes the proof. □

The proof of Lemma 6.1 can now be given.

Proof of Lemma 6.1. $\boxed{2 \Longrightarrow 1}$ Let $\mathbf{r}^*$ be a frugal repair of $\mathbf{db}$ with respect to q. Then there is a rooted tree set τ starting in c in $\mathbf{r}^*$ with $\tau \in$ S-CFG$^\clubsuit(q, y)$. The claim follows by Lemma 6.7.

$\boxed{1 \Longrightarrow 2}$ Assume that $\langle c, y \rangle \in B$. We use induction on k to show that if $\langle c, y \rangle$ is added to B at the k-th iteration, then for every repair $\mathbf{r}$ of $\mathbf{db}$, there exists a rooted tree set τ starting in c in τ with $\tau \in$ S-CFG$^\clubsuit(q, y)$.

- Basis $k = 0$. Then $\langle c, u \rangle$ is added to B for every leaf variable u of q such that either the label of u in q is $\perp$, or a unary relation name L, or $u = c$ is a constant. If the label of u is $\perp$, the empty rooted tree set $\tau = \emptyset$ starting in c with string representation $\perp$ is accepted by S-CFG$^\clubsuit(q, u)$. If the label of u is L, then we must have $L(c) \in \mathbf{db}$, and the rooted tree set $\tau = L$ starting in c is accepted by S-CFG$^\clubsuit(q, u)$. If $u = c$ is a constant, then the rooted tree set $\tau = c$ starting in c is accepted by S-CFG$^\clubsuit(q, c)$.
- Inductive step. Assume that $\langle c, y \rangle$ is added to B in the k-th iteration, and for every tuple $\langle b, x \rangle$ added to B prior to the addition of $\langle c, y \rangle$, any repair of $\mathbf{db}$ contains a rooted tree set $\tau \in$ S-CFG$^\clubsuit(q, x)$ starting in b. Let $\mathbf{r}$ be any repair of $\mathbf{db}$. It suffices to construct a rooted tree set τ in $\mathbf{r}$ starting in c such that $\tau \in$ S-CFG$^\clubsuit(q, y)$. Let $R[y] = R(\underline{y}, y_1, y_2, \ldots, y_n)$. Let $R(\underline{c}, d_1, d_2, \ldots, d_n) \in \mathbf{r}$ and let $\vec{d} = \langle d_1, d_2, \ldots, d_n \rangle$. Since $\langle c, y \rangle \in B$, fact$(R(\underline{c}, \vec{d}), y)$ is true. Consider two cases.

– Case that $\langle d_i, y_i \rangle \in B$ for every $1 \leq i \leq n$. Since each $\langle d_i, y_i \rangle$ was added to B in an iteration $< k$, by the inductive hypothesis, there is a rooted tree set τ_i starting in d_i in $\mathbf{r}$ with $\tau_i \in \text{S-CFG}^{\clubsuit}(q, y_i)$, i.e., $S_{y_i} \xrightarrow{*}_q \tau_i$. Consider the rooted tree set $\tau = \{R(\underline{c}, \vec{d})\} \cup \bigcup_{1 \leq i \leq n} \tau_i$, starting in c in $\mathbf{r}$ with a string representation $\tau = R(\tau_1, \tau_2, \ldots, \tau_n)$. From

$$S_y \to_q R(S_{y_1}, S_{y_2}, \ldots, S_{y_n}) \xrightarrow{*}_q R(\tau_1, \tau_2, \ldots, \tau_n) = \tau,$$

we conclude that $\tau \in \text{S-CFG}^{\clubsuit}(q, y)$.

– Case that $\text{fact}(R(\underline{c}, \vec{d}), x)$ is true for some $R[x] <_q R[y]$. Without loss of generality, we assume that x is the smallest with respect to $<_q$ for the atom $R(\underline{x}, x_1, x_2, \ldots, x_n)$. Hence we must have $\langle d_i, x_i \rangle \in B$ for every $1 \leq i \leq n$, and by the previous case, there exists a rooted tree set τ_i starting in d_i such that $\tau_i \in \text{S-CFG}^{\clubsuit}(q, x_i)$, i.e., $S_{x_i} \xrightarrow{*}_q \tau_i$. Since $R[x] <_q R[y]$, we have

$$S_y \to_q S_x \to_q R(S_{x_1}, S_{x_2}, \ldots, S_{x_n}) \xrightarrow{*}_q R(\tau_1, \tau_2, \ldots, \tau_n) = \tau,$$

and therefore $\tau \in \text{S-CFG}^{\clubsuit}(q, y)$.

The proof is now complete. □

6.2 Expressibility in LFP and FO

Lemma 6.8. *For every query q in* TreeBCQ *that satisfies* $\mathsf{C}_{\text{branch}}$, $\mathsf{CERTAIN}_{\text{tr}}(q)$ *is expressible in* **LFP** *(and thus is in* **PTIME***).*

Proof. Let r be the root variable of q. Our algorithm first computes the set B, and then checks $\exists c : \langle c, r \rangle \in B$. The algorithm is correct by Lemma 6.1. The following query (5) in **LFP** [41] straightforwardly captures the computation of the set B of Fig. 6. Herein, $\alpha(x)$ denotes a first-order query that computes the active domain, and $\bot(u)$ denotes that u is a leaf variable corresponding to a leaf vertex labeled $\bot$. We write "y" for a variable y in $\text{vars}(q)$ that becomes a constant in φ_q. The first and second rows in the definition of $\varphi_q(B, v, z)$ correspond, respectively, to the initialization step and the iterative rule of the algorithm of Fig. 6:

$$\psi_q(s, t) := \left[\mathbf{lfp}_{B,v,z}\varphi_q(B, v, z)\right](s, t), \tag{5}$$

where $\varphi_q(B, v, z) :=$

$$\begin{aligned} &(\alpha(v) \wedge z = v) \vee \left(\bigvee\nolimits_{y \in \text{vars}(q), \bot(y)} (\alpha(v) \wedge z = \text{"y"})\right) \vee \left(\bigvee\nolimits_{L(y) \in q} (L(v) \wedge z = \text{"y"})\right) \vee \\ &\left(\bigvee\nolimits_{R(\underline{y}, y_1, \ldots, y_n) \in q} \left(\begin{array}{l} z = \text{"y"} \wedge \\ \exists w_1 \ldots \exists w_n \left(R(\underline{v}, w_1, \ldots, w_n)\right) \wedge \\ \forall w_1 \ldots \forall w_n \left(R(\underline{v}, w_1, \ldots, w_n) \rightarrow f_{R[y]}(\underline{v}, w_1, \ldots, w_n)\right) \end{array}\right)\right), \end{aligned}$$

and $f_{R[y]}$ is defined as follows:

$$f_{R[y]}(v, w_1, \ldots, w_n) := \left(\bigwedge_{1 \leq i \leq n} B(w_i, \text{"y}_\text{i}\text{"})\right) \vee \left(\bigvee_{R[x] <_q R[y]} f_{R[x]}(v, w_1, \ldots, w_n)\right),$$

in which $f_{R[x]}$ is recursively expanded using the same definition, eventually reaching a vertex labeled R without ancestor labeled R. This concludes the proof. □

We now show that if q satisfies C_1, we can safely remove the recursion from the algorithm in Fig. 6.

Lemma 6.9. *Let q be a rooted tree query satisfying* C_1, *and let* $R[y] = R(\underline{y}, y_1, y_2, \ldots, y_n)$ *be an atom in q. Let* **db** *be a database containing a fact* $R(\underline{c}, \vec{d}) = R(\underline{c}, d_1, d_2, \ldots, d_n)$. *Then,* $\mathsf{fact}(R(\underline{c}, \vec{d}), y)$ *is true if and only if for every atom $T_i[y_i]$ in q,* $\langle d_i, y_i \rangle \in B$.

Proof. $\boxed{\Longleftarrow}$ Immediate by definition of $\mathsf{fact}(R(\underline{c}, \vec{d}), y)$. $\boxed{\Longrightarrow}$ Assume that $\mathsf{fact}(R(\underline{c}, \vec{d}), y)$ is true. Let $R[x]$ be a minimal atom with respect to $<_q$ such that $R[x] <_q R[y]$ and $\mathsf{fact}(R(\underline{c}, \vec{d}), x)$ is true. If such an atom $R[x]$ does not exist, then the claim follows by definition of $\mathsf{fact}(R(\underline{c}, \vec{d}), y)$. Otherwise, since $R[x]$ is minimal with respect to $<_q$, for every atom $T_i[x_i]$ in q, $\langle d_i, x_i \rangle \in B$, where $R(\underline{x}, \vec{x}) = R(\underline{x}, x_1, x_2, \ldots, x_n)$. It suffices to show that $\langle d_i, y_i \rangle \in B$ for every i. From C_1 and $R[x] <_q R[y]$, $q^{y_i}_{\triangle} \leq_{y_i \to x_i} q^{x_i}_{\triangle}$. If both y_i and x_i are variables, let $T_i[y_i]$ be an atom in q. Then there is some atom $T_i[x_i]$ in q with $T_i[x_i] <_q T_i[y_i]$. Since $\langle d_i, x_i \rangle \in B$, by Lemma 6.2, $\langle d_i, y_i \rangle \in B$. If $y_i = x_i = c$ for some constant c, then we have $\langle d_i, y_i \rangle = \langle d_i, x_i \rangle \in B$. □

Lemma 6.10. *For every q in* TreeBCQ *that satisfies* C_1, $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ *is in* **FO** .

Proof. Consider the following variant of the algorithm in Fig. 6, where we simply have

$$\mathsf{fact}(R(\underline{c}, \vec{d}), y) = \bigwedge_{1 \leq i \leq n} \langle d_i, y_i \rangle \in B.$$

The variant algorithm is correct for $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ by Lemma 6.9. Since the size of the query q is fixed, for every constant c and variable y in q, deciding whether $\langle c, y \rangle \in B$ is in **FO** since the algorithm in Fig. 6 can be expanded into a sentence of fixed size. So is our algorithm, which checks $\exists c : \langle c, r \rangle \in B$, where r is the root variable of q. □

7 COMPLEXITY UPPER BOUNDS

In this section, we prove the upper bound results in Theorem 4.5. First, we shall prove Lemma 5.4.

Lemma 7.1. *Let q be a rooted tree query. Then q satisfies* C_{factor} *if and only if* $q \leq_{\to} \tau$ *for every* $\tau \in \mathsf{CFG}^{\clubsuit}(q)$.

Proof. Consider two directions.

$\boxed{\Longleftarrow}$ Let $R[x]$ and $R[y]$ be two atoms in q with $R[x] <_q R[y]$. It suffices to show that $q^{R:y \hookrightarrow x} \in \mathsf{CFG}^{\clubsuit}(q)$. Indeed, there is an execution of $S_r(q^{R:y \hookrightarrow x})$ that follows exactly $S_r(q)$, until it invokes $S_y(q^x_{\triangle})$, instead of $S_y(q^y_{\triangle})$ in $S_r(q)$. Note that $S_y \to_q S_x \xrightarrow{*}_q q^x_{\triangle}$. Thus $S_r \xrightarrow{*}_q q^{R:y \hookrightarrow x}$, concluding that $q^{R:y \hookrightarrow x} \in \mathsf{CFG}^{\clubsuit}(q)$.

$\boxed{\Longrightarrow}$ Let $\tau \in \mathsf{CFG}^{\clubsuit}(q)$ with $S_r \xrightarrow{*}_q \tau$. We use an induction on the number k of backward transitions in $S_r \xrightarrow{*}_q \tau$ to show that $q \leq_{\to} \tau$.

- Basis $k = 0$. We have $\tau = q$, and the claim follows.
- Inductive step $k \to k + 1$. Assume that if $S_r \xrightarrow{*}_q \sigma$ uses k backward transitions, then $q \leq_{\to} \sigma$. Let $\tau \in \mathsf{CFG}^{\clubsuit}(q)$ such that $S_r \xrightarrow{*}_q \tau$ uses $k+1$ backward transitions. Let σ be a subtree of τ such that the execution of $S_r(\sigma)$ invokes exactly 1 backward transition $S_y \to_q S_x \xrightarrow{*}_q \sigma$. Hence $\sigma = q^x_{\triangle}$. Consider the rooted tree τ^*, obtained by replacing $\sigma = q^x_{\triangle}$ with $\sigma^* = q^y_{\triangle}$. We have $\tau^* \in \mathsf{CFG}^{\clubsuit}(q)$, since $S_r \xrightarrow{*}_q \tau$ would invoke $S_y \xrightarrow{*}_q \sigma^*$ and use exactly k backward transitions. By the inductive hypothesis, there is a homomorphism h from q to τ^*. If $h(q) \cap \sigma^* = \emptyset$, then $h(q)$ is still present in τ, and thus $q \leq_{\to} \tau$. Otherwise, assume that the homomorphism h maps $q^z_{\triangle}$ in q to σ^*. Hence $R[x] <_q R[y] <_q R[z]$. Since q satisfies C_{factor}, there is a homomorphism g from q to $q^{R:z \hookrightarrow x}$, and thus a homomorphism from q to τ.

The proof is now complete. □

The following definition is helpful in our exposition.

Definition 7.2. Let q be a rooted tree query. Let **db** be a database. For each repair $\mathbf{r}$ of **db**, we define $\mathsf{start}(q, \mathbf{r})$ as the set containing all (and only) constants $c \in \mathsf{adom}(\mathbf{r})$ such that there is a rooted tree set τ in $\mathbf{r}$ starting in c with $\tau \in \mathsf{CFG}^{\clubsuit}(q)$.

The problem $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$ essentially asks whether there is some constant c such that for every repair $\mathbf{r}$ of **db**, $c \in \mathsf{start}(q, r)$. Surprisingly, the frugal repair $\mathbf{r}^*$ of **db** minimizes $\mathsf{start}(q, \cdot)$ across all repairs of **db**.

Lemma 7.3. *Let q be a rooted tree query satisfying $\mathsf{C}_{\mathsf{branch}}$. Let* **db** *be a database. Let $\mathbf{r}^*$ be a frugal repair of* **db**. *Then for every repair $\mathbf{r}$ of* **db**, $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$.

Proof. Let B be the output of the algorithm in Fig. 6. Let $\mathbf{r}^*$ be a frugal repair of **db**. Let $\mathbf{r}$ be any repair of **db**. We show that $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$. Let r be the root variable of q. Assume that $c \in \mathsf{start}(q, \mathbf{r}^*)$. Then there exists a rooted tree set τ starting in c in $\mathbf{r}^*$ with $\tau \in \mathsf{CFG}^{\clubsuit}(q) = \mathsf{S\text{-}CFG}^{\clubsuit}(q, r)$. By Lemma 6.7, we have $\langle c, r\rangle \in B$. By Lemma 6.1, there exists a rooted tree set τ' starting in c in $\mathbf{r}$ with $\tau' \in \mathsf{S\text{-}CFG}^{\clubsuit}(q, r) = \mathsf{CFG}^{\clubsuit}(q)$. Thus $c \in \mathsf{start}(q, \mathbf{r})$. □

The proof of Lemma 5.4 can now be given.

Proof of Lemma 5.4. $\boxed{\Longrightarrow}$ Let **db** be a "yes"-instance of $\mathsf{CERTAINTY}(q)$. Let $\mathbf{r}^*$ be a frugal repair of **db**. Since $\mathbf{r}^*$ satisfies q, there is a rooted tree set starting in c that is isomorphic to q in $\mathbf{r}^*$. Since $q \in \mathsf{CFG}^{\clubsuit}(q)$, we have $c \in \mathsf{start}(q, \mathbf{r}^*)$. By Lemma 7.3, for every repair $\mathbf{r}$ of **db**, $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$. It follows that $c \in \mathsf{start}(q, \mathbf{r})$ for every repair $\mathbf{r}$ of **db**. $\boxed{\Longleftarrow}$ Let $\mathbf{r}$ be any repair of **db**. By the hypothesis that **db** is a "yes"-instance of $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$, there is some constant $c \in \mathsf{start}(q, \mathbf{r})$. Let τ be a rooted tree set in $\mathbf{r}$ starting in c with $\tau \in \mathsf{CFG}^{\clubsuit}(q)$. Since q satisfies C_2 by the hypothesis of the current lemma, it follows by Lemma 7.1 that $q \leq_{\rightarrow} \tau$. Consequently, $\mathbf{r}$ satisfies q. □

The upper bounds in Theorem 4.5 thus follow.

Proposition 7.4. *For every q in* TreeBCQ,

(1) if q satisfies C_2, then $\mathsf{CERTAINTY}(q)$ is in **LFP**; *and*
(2) if q satisfies C_1, then $\mathsf{CERTAINTY}(q)$ is in **FO**.

Proof. Immediate from Lemmas 5.4, 6.8, and 6.10 by noting that C_1 implies C_2. □

Interestingly, for each query q in TreeBCQ satisfying C_2, "checking the frugal repair is all you need". A repair with this property is known as a "universal repair" in [51].

Corollary 7.5. *Let q be a query in* TreeBCQ *that satisfies C_2, and let* **db** *be a database instance. Let $\mathbf{r}^*$ be a frugal repair of* **db** *with respect to q. Then,* **db** *is a "yes"-instance of $\mathsf{CERTAINTY}(q)$ if and only if $\mathbf{r}^*$ satisfies q.*

Proof. $\boxed{\Longrightarrow}$ Straightforward. $\boxed{\Longleftarrow}$ Assume that $\mathbf{r}^*$ satisfies q. Let r be the root variable of q. Hence there is a constant c in **db** such that there exists a rooted relation tree τ in $\mathbf{r}^*$ that is isomorphic to q and accepted by $\mathsf{S\text{-}CFG}^{\clubsuit}(q, r)$. Then by Lemma 6.7, $\langle c, r\rangle \in B$, where B is the output of the algorithm in Fig. 6. Hence **db** is a "yes"-instance for $\mathsf{CERTAIN}_{\mathsf{tr}}(q)$, and by Lemma 5.4, a "yes"-instance of $\mathsf{CERTAINTY}(q)$. □

8 COMPLEXITY LOWER BOUNDS

In this section, we present the hardness results in Theorem 4.5. The following proposition is proved in Appendix C through reductions from Monotone SAT and REACHABILITY.

PROPOSITION 8.1. *For every q in* TreeBCQ,

(1) if q violates C_2, *then* CERTAINTY(q) *is* **coNP**-*hard; and*

(2) if q violates C_1, *then* CERTAINTY(q) *is* **NL**-*hard.*

9 EXTENDING THE TRICHOTOMY

In this section, we extend the complexity classification for rooted tree queries to larger classes of Boolean conjunctive queries. We postpone most proofs to Appendix D.

9.1 From TreeBCQ to GraphBCQ

We define GraphBCQ, a subclass of BCQ that extends TreeBCQ.

*Definition 9.1 (*GraphBCQ*).* GraphBCQ is the class of Boolean conjunctive queries q satisfying the following conditions:

(1) every atom in q is of the form $R(\underline{x}, y_1, \ldots, y_n)$ where x is a variable and $y_1, \ldots, y_n$ are symbols (variables or constants) such that no variable occurs twice in the atom; and

(2) if $R(\underline{x}, y_1, \ldots, y_n)$ and $S(\underline{u}, v_1, \ldots, v_m)$ are distinct atoms of q, then $x \neq u$. Note that R and S need not be distinct.

For a query q in BCQ, we define $\mathcal{G}(q)$ as the undirected graph whose vertices are the atoms of q; two atoms are adjacent if they have a variable in common. The connected components of q are the connected components of $\mathcal{G}(q)$. Note that queries in GraphBCQ, unlike TreeBCQ, can have more than one connected component. The following lemma implies that the complexity of CERTAINTY(q) is equal to the highest complexity of CERTAINTY(q') over every connected component q' of q. The proof of Lemma 9.2 is in Appendix C of [33].

LEMMA 9.2. *Let q be a minimal query in* BCQ *with connected components $q_1, q_2, \ldots, q_n$. Then:*

(1) for every $1 \leq i \leq n$, there exists a first-order reduction from the problem CERTAINTY(q_i) *to* CERTAINTY(q)*; and*

(2) for every database instance **db**, **db** *is a "yes"-instance of the problem* CERTAINTY(q) *if and only if for every $1 \leq i \leq n$,* **db** *is a "yes"-instance of* CERTAINTY(q_i).

PROPOSITION 9.3. *If q is a connected minimal conjunctive query in* GraphBCQ \ TreeBCQ, *then* CERTAINTY(q) *is* **L**-*hard (and not in* **FO**)*; if q is also Berge-acyclic, then* CERTAINTY(q) *is* **coNP**-*hard.*

We can now give the proof of Theorems 1.4 and 1.5.

PROOF OF THEOREMS 1.4 AND 1.5. Let q be a query in GraphBCQ. Then the minimal query q^* of q is also in GraphBCQ. If every connected component of q^* is in TreeBCQ and satisfies C_1, then CERTAINTY(q) is in **FO**. Otherwise, there exists some connected component q' of q^* that is either not in TreeBCQ, or violates C_1, and CERTAINTY(q) is **L**-hard or **NL**-hard by Lemma 9.2, Proposition 9.3, and Theorem 4.5. Assume that q is also Berge-acyclic. If some connected component q' of q^* is not in TreeBCQ, then CERTAINTY(q) is **coNP**-complete; or otherwise, CERTAINTY(q) exhibits a trichotomy by Theorem 4.5. □

Lemma 9.4 (adapted from [54]) is essential to the proof of Proposition 9.3, but is of independent interest. Given a query q in BCQ, a *self-join-free version of* q, denoted q^{sjf}, is any self-join-free

Boolean conjunctive query obtained from q by (only) renaming relation names. For example, a self-join-free version of $\{R(\underline{x}, y), R(\underline{y}, x)\}$ is $\{R(\underline{x}, y), S(\underline{y}, x)\}$.

Lemma 9.4 (Bridging Lemma). *Let q be a minimal query in* BCQ *that contains no two distinct atoms $R_1(\underline{\vec{x}_1}, \vec{y}_1)$ and $R_2(\underline{\vec{x}_2}, \vec{y}_2)$ such that $R_1 = R_2$ and $\vec{x}_1 = \vec{x}_2$. Then, there is a first-order reduction from* CERTAINTY(q^{sjf}) *to* CERTAINTY(q).

The use of the Bridging Lemma is illustrated by Example 9.5.

Example 9.5. For $q_1 = \{R(\underline{x}, y, z), R(\underline{z}, x, y)\}$, we have ${q_1}^{\mathsf{sjf}} = \{R_1(\underline{x}, y, z), R_2(\underline{z}, x, y)\}$. By Theorem 1.2 [35], CERTAINTY$({q_1}^{\mathsf{sjf}})$ is **L**-complete, and thus CERTAINTY(q_1) is **L**-hard by Lemma 9.4.

For $q_2 = \{R(\underline{x}, z), R(\underline{y}, z)\}$, we have ${q_2}^{\mathsf{sjf}} = \{R_1(\underline{x}, z), R_2(\underline{y}, z)\}$. Although by Theorem 1.2 [35], CERTAINTY$({q_2}^{\mathsf{sjf}})$ is **coNP**-complete, CERTAINTY(q_2) is in **FO** because $q_2 \equiv q_2'$ where $q_2' = \{R(\underline{x}, z)\}$. Lemma 9.4 does not apply here because q_2 is not minimal.

9.2 Open Challenges

So far, we have established the **FO**-boundary of CERTAINTY(q) for all queries q in GraphBCQ, and a fine-grained complexity classification for all Berge-acyclic queries in GraphBCQ, which include all rooted tree queries. We briefly discuss the remaining syntactic restrictions.

The complexity classification of CERTAINTY(q) for queries q in GraphBCQ that are not Berge-acyclic is likely to impose new challenges. In particular, Figueira et al. [18] showed that for q_1 in Example 9.5 (that is not Berge-acyclic), the complement of CERTAINTY(q_1) is complete for Bipartite Matching under LOGSPACE-reductions.

The restriction imposed by GraphBCQ that every variable occurs at most once at a primary-key position allows for an elegant graph representation. We found that dropping this requirement imposes serious challenges. The following Proposition 9.6 hints at the difficulty of having to "correlate two rooted tree branches" that share the same primary-key variable.

Proposition 9.6. *Consider the following queries:*

- $q_1 = \{R(\underline{u}, x_1), R(\underline{x_1}, x_2), S(\underline{u}, y_1), S(\underline{y_1}, y_2)\}$;
- $q_2 = q_1 \cup \{X(\underline{x_2}, x_3)\}$; *and*
- $q_3 = q_1 \cup \{X(\underline{x_2}, x_3), Y(\underline{y_2}, y_3)\}$.

Then we have CERTAINTY(q_1) *is in* **FO**, CERTAINTY(q_2) *is in* **NL***-hard* $\cap$ **LFP**, *and* CERTAINTY(q_3) *is* **coNP***-complete.*

The proof of Proposition 9.6 is in Appendix C of [33]. The restrictions that no atom contains repeated variables, and that no constant occurs at a primary-key position ease the technical treatment, but it is likely that they can be dropped at the price of some technical involvement. On the other hand, all our techniques fundamentally rely on the restriction that primary keys are simple.

10 CONCLUSION

We established a fine-grained complexity classification of the problem CERTAINTY(q) for all rooted tree queries q. We then lifted our complexity classification to a larger class of queries. A notorious open problem in consistent query answering is Conjecture 1.1, which conjectures that for every query q in BCQ, CERTAINTY(q) is either in **PTIME** or **coNP**-complete. Despite our progress, this problem remains open even under the restriction that all primary keys are simple.

Acknowledgements. The authors thank the anonymous reviewers for their constructive feedback and comments. This work is supported by the National Science Foundation under grant IIS-1910014 and the Anthony C. Klug NCR Fellowship.

REFERENCES

[1] Marcelo Arenas, Pablo Barceló, and Mikaël Monet. 2021. The Complexity of Counting Problems Over Incomplete Databases. *ACM Trans. Comput. Log.* 22, 4 (2021), 21:1–21:52.

[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. ACM Press, 68–79.

[3] Pablo Barceló and Gaëlle Fontaine. 2015. On the Data Complexity of Consistent Query Answering over Graph Databases. In *ICDT (LIPIcs, Vol. 31)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 380–397.

[4] Pablo Barceló and Gaëlle Fontaine. 2017. On the data complexity of consistent query answering over graph databases. *J. Comput. Syst. Sci.* 88 (2017), 164–194.

[5] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *PODS*. ACM, 303–318.

[6] Leopoldo E. Bertossi. 2019. Database Repairs and Consistent Query Answering: Origins and Further Developments. In *PODS*. ACM, 48–58.

[7] Andrei A. Bulatov. 2011. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.* 12, 4 (2011), 24:1–24:66.

[8] Marco Calautti, Marco Console, and Andreas Pieris. 2019. Counting Database Repairs under Primary Keys Revisited. In *PODS*. ACM, 104–118.

[9] Marco Calautti, Marco Console, and Andreas Pieris. 2021. Benchmarking Approximate Consistent Query Answering. In *PODS*. ACM, 233–246.

[10] Marco Calautti, Leonid Libkin, and Andreas Pieris. 2018. An Operational Approach to Consistent Query Answering. In *PODS*. ACM, 239–251.

[11] Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider. 2022. Counting Database Repairs Entailing a Query: The Case of Functional Dependencies. In *PODS*. ACM, 403–412.

[12] Marco Calautti, Ester Livshits, Andreas Pieris, and Markus Schneider. 2022. Uniform Operational Consistent Query Answering. In *PODS*. ACM, 393–402.

[13] Jan Chomicki and Jerzy Marcinkowski. 2005. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.* 197, 1-2 (2005), 90–121. https://doi.org/10.1016/j.ic.2004.04.007

[14] Jan Chomicki, Jerzy Marcinkowski, and Slawomir Staworko. 2004. Hippo: A System for Computing Consistent Answers to a Class of SQL Queries. In *EDBT (Lecture Notes in Computer Science, Vol. 2992)*. Springer, 841–844.

[15] Akhil A. Dixit and Phokion G. Kolaitis. 2019. A SAT-Based System for Consistent Query Answering. In *SAT (Lecture Notes in Computer Science, Vol. 11628)*. Springer, 117–135.

[16] Akhil A. Dixit and Phokion G. Kolaitis. 2021. CAvSAT: Answering Aggregation Queries over Inconsistent Databases via SAT Solving. In *SIGMOD Conference*. ACM, 2701–2705.

[17] Zhiwei Fan, Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. 2023. LinCQA: Faster Consistent Query Answering with Linear Time Guarantees. *Proc. ACM Manag. Data* 1, 1 (2023), 38:1–38:25.

[18] Diego Figueira, Anantha Padmanabha, Luc Segoufin, and Cristina Sirangelo. 2023. A Simple Algorithm for Consistent Query Answering Under Primary Keys. In *ICDT (LIPIcs, Vol. 255)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24:1–24:18.

[19] Gaëlle Fontaine. 2015. Why Is It Hard to Obtain a Dichotomy for Consistent Query Answering? *ACM Trans. Comput. Log.* 16, 1 (2015), 7:1–7:24.

[20] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2015. The Complexity of Resilience and Responsibility for Self-Join-Free Conjunctive Queries. *Proc. VLDB Endow.* 9, 3 (2015), 180–191.

[21] Cibele Freire, Wolfgang Gatterbauer, Neil Immerman, and Alexandra Meliou. 2020. New Results for the Complexity of Resilience for Binary Conjunctive Queries with Self-Joins. In *PODS*. ACM, 271–284.

[22] Ariel Fuxman and Renée J. Miller. 2007. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73, 4 (2007), 610–635.

[23] Gianluigi Greco, Sergio Greco, and Ester Zumpano. 2003. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Trans. Knowl. Data Eng.* 15, 6 (2003), 1389–1408.

[24] Miika Hannula and Jef Wijsen. 2022. A Dichotomy in Consistent Query Answering for Primary Keys and Unary Foreign Keys. In *PODS*. ACM, 437–449.

[25] Lara A Kahale, Assem M Khamis, Batoul Diab, Yaping Chang, Luciane Cruz Lopes, Arnav Agarwal, Ling Li, Reem A Mustafa, Serge Koujanian, Reem Waziry, et al. 2020. Meta-Analyses Proved Inconsistent in How Missing Data Were Handled Across Their Included Primary Trials: A Methodological Survey. *Clinical Epidemiology* 12 (2020), 527–535.

[26] Yannis Katsis, Alin Deutsch, Yannis Papakonstantinou, and Vasilis Vassalos. 2010. Inconsistency resolution in online databases. In *ICDE*. IEEE Computer Society, 1205–1208.

[27] Aziz Amezian El Khalfioui, Jonathan Joertz, Dorian Labeeuw, Gaëtan Staquet, and Jef Wijsen. 2020. Optimization of Answer Set Programs for Consistent Query Answering by Means of First-Order Rewriting. In *CIKM*. ACM, 25–34.

[28] Aziz Amezian El Khalfioui and Jef Wijsen. 2023. Consistent Query Answering for Primary Keys and Conjunctive Queries with Counting. In *ICDT (LIPIcs, Vol. 255)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 23:1–23:19.

[29] Benny Kimelfeld, Ester Livshits, and Liat Peterfreund. 2020. Counting and enumerating preferred database repairs. *Theor. Comput. Sci.* 837 (2020), 115–157.

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

[31] Phokion G. Kolaitis, Enela Pema, and Wang-Chiew Tan. 2013. Efficient Querying of Inconsistent Databases with Binary Integer Programming. *Proc. VLDB Endow.* 6, 6 (2013), 397–408.

[32] Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. 2021. Consistent Query Answering for Primary Keys on Path Queries. In *PODS*. ACM, 215–232.

[33] Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. 2023. Consistent Query Answering for Primary Keys on Rooted Tree Queries. *CoRR* abs/2310.19642 (2023). https://doi.org/10.48550/ARXIV.2310.19642 arXiv:2310.19642

[34] Paraschos Koutris and Dan Suciu. 2014. A Dichotomy on the Complexity of Consistent Query Answering for Atoms with Simple Keys. In *ICDT*. OpenProceedings.org, 165–176.

[35] Paraschos Koutris and Jef Wijsen. 2015. The Data Complexity of Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. In *PODS*. ACM, 17–29.

[36] Paraschos Koutris and Jef Wijsen. 2017. Consistent Query Answering for Self-Join-Free Conjunctive Queries Under Primary Key Constraints. *ACM Trans. Database Syst.* 42, 2 (2017), 9:1–9:45.

[37] Paraschos Koutris and Jef Wijsen. 2018. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *PODS*. ACM, 209–224.

[38] Paraschos Koutris and Jef Wijsen. 2019. Consistent Query Answering for Primary Keys in Logspace. In *ICDT (LIPIcs, Vol. 127)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 23:1–23:19.

[39] Paraschos Koutris and Jef Wijsen. 2020. First-Order Rewritability in Consistent Query Answering with Respect to Multiple Keys. In *PODS*. ACM, 113–129.

[40] Paraschos Koutris and Jef Wijsen. 2021. Consistent Query Answering for Primary Keys in Datalog. *Theory Comput. Syst.* 65, 1 (2021), 122–178.

[41] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

[42] Andrei Lopatenko and Leopoldo E. Bertossi. 2007. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *ICDT*, Vol. 4353. Springer, 179–193.

[43] Carsten Lutz and Frank Wolter. 2015. On the Relationship between Consistent Query Answering and Constraint Satisfaction Problems. In *ICDT (LIPIcs, Vol. 31)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 363–379.

[44] Marco Manna, Francesco Ricca, and Giorgio Terracina. 2015. Taming primary key violations to query large inconsistent data via ASP. *Theory Pract. Log. Program.* 15, 4-5 (2015), 696–710.

[45] Mónica Caniupán Marileo and Leopoldo E. Bertossi. 2010. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data Knowl. Eng.* 69, 6 (2010), 545–572.

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

[47] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*. OpenProceedings.org, 155–164.

[48] Anantha Padmanabha, Luc Segoufin, and Cristina Sirangelo. 2023. A dichotomy in the complexity of consistent query answering for two atom queries with self-join. *CoRR* abs/2309.12059 (2023). https://doi.org/10.48550/ARXIV.2309.12059 arXiv:2309.12059

[49] M. Andrea Rodríguez, Leopoldo E. Bertossi, and Mónica Caniupán Marileo. 2013. Consistent query answering under spatial semantic constraints. *Inf. Syst.* 38, 2 (2013), 244–263.

[50] Slawek Staworko, Jan Chomicki, and Jerzy Marcinkowski. 2012. Prioritized repairing and consistent query answering in relational databases. *Ann. Math. Artif. Intell.* 64, 2-3 (2012), 209–246.

[51] Balder ten Cate, Gaëlle Fontaine, and Phokion G. Kolaitis. 2012. On the data complexity of consistent query answering. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, Alin Deutsch (Ed.). ACM, 22–33. https://doi.org/10.1145/2274576.2274580

[52] Jef Wijsen. 2005. Database repairing using updates. *ACM Trans. Database Syst.* 30, 3 (2005), 722–768.

[53] Jef Wijsen. 2010. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*. ACM, 179–190.

[54] Jef Wijsen. 2019. Corrigendum to "Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins". *CoRR* abs/1903.12469 (2019).

[55] Jef Wijsen. 2019. Foundations of Query Answering on Inconsistent Databases. *SIGMOD Rec.* 48, 3 (2019), 6–16.

[56] Dmitriy Zhuk. 2020. A Proof of the CSP Dichotomy Conjecture. *J. ACM* 67, 5 (2020), 30:1–30:78. https://doi.org/10.1145/3402029

A MISSING PROOFS IN SECTION 4

Proof of Lemma 4.2. We denote

$$p = q^{R:y \hookrightarrow x} = (q \setminus q^y_\triangle) \cup f(q^x_\triangle),$$

for some isomorphism f that maps every variable in $q^x_\triangle$ to a fresh variable, except for x, for which we have $f(x) = y$.

Assume first that $q^y_\triangle \preceq_{y\to x} q^x_\triangle$, witnessed by the homomorphism h with $h(y) = x$. Let $g : \mathbf{vars}(q) \to \mathbf{vars}(p)$ be the mapping such that $g(z) = z$ if $z \in \mathbf{vars}(q \setminus q^y_\triangle)$, and $g(z) = f(h(z))$ otherwise. It is easily seen that g is a homomorphism from q to p.

Conversely, assume there is a homomorphism $h : \mathbf{vars}(q) \to \mathbf{vars}(p)$ from q to p. Hence

$$h(q) = h(q \setminus q^y_\triangle) \cup h(q^y_\triangle) \subseteq (q \setminus q^y_\triangle) \cup f(q^x_\triangle).$$

Note that q is minimal, i.e., there is no automorphism α such that $\alpha(q) \subsetneq q$. If $h(y) = y$, since q is minimal, we have $h(q \setminus q^y_\triangle) = q \setminus q^y_\triangle$, and we have $h(q^y_\triangle) \subseteq f(q^x_\triangle)$. Thus $q^y_\triangle \preceq_{y\to x} q^x_\triangle$, witnessed by the homomorphism $g = f^{-1} \circ h$ with $g(y) = f^{-1}(h(y)) = f^{-1}(y) = x$, as desired.

Suppose for contradiction that $h(y) \neq y$. In this case, we have $h(q \setminus q^y_\triangle) \cap f(q^x_\triangle) \neq \emptyset$. There are three possible cases, each of which leads to a contradiction, as shown next.

Case that $h(y) <_p y = f(x)$. Then h maps the unique path of nodes from r to y in q to the unique path from $h(r)$ to $h(y)$ in p, and hence both paths have the same length. However, since $h(y) <_p y$ and either $r = h(r)$ or $r <_p h(r)$, the path from $h(r)$ to $h(y)$ in p is strictly shorter than the path from r to y in q, a contradiction.

Case that $h(y) \parallel_p y = f(x)$. Let $y_0 = y$, and for each $i \geq 1$, let $y_i = h(y_{i-1})$. In particular, $h(y_0) = y_1$. We argue that variables $y_0, y_1, \ldots$ are all distinct, thereby reaching a contradiction to the finite size of q. We define a left sibling of some variable u in q as a variable that precedes u in the depth-first, left-to-right order of q. Assume that y_1 is a left sibling of y_0 in q (the case where y_1 is a right sibling of y_0 is symmetrical) : for the greatest common ancestor y^* of y_1 and y_0, there is an atom $R(\underline{y^*}, .., y_\ell, .., y_r)$ such that y_ℓ and y_r are ancestors of, respectively, y_1 and y_0. Note that y_1 appears in both p and q and its subtree is not affected by the rewinding operation since $y_1 \parallel_p y_0$. Since y_1 is a left sibling of y_0 and that the children of rooted trees are ordered, $h(y_1)$ is a left sibling of $h(y_0)$, that is y_2 is a left sibling of y_1 in q, and this process continues. Since each y_{i+1} is a left sibling of y_i, the variables need to be distinct, or otherwise some y_{j+1} is a right sibling of y_j, a contradiction.

Case that $y = f(x) <_p h(y)$. Since $R[x] \parallel_q R[y]$, let $T[z]$ be the greatest common ancestor of $R[x]$ and $R[y]$ in q and let u and v be variables in $T[z]$ such that $u <_q x$ and $v <_q y$ and $u \parallel_q v$. Hence, z appears in both q and p. Since $y <_p h(y)$ and $y \neq h(y)$, we have $z <_p h(z)$ and $h(z) \neq z$, by a size argument. We have $|q^u_\triangle| + |q^v_\triangle| + 1 \leq |q^z_\triangle| \leq |p^{h(z)}_\triangle|$, because the homomorphism maps $q^z_\triangle$ to the subtree of p, rooted at $h(z)$. We show that $v <_q h(z)$. Since $z <_q h(z)$ and v is the immediate child of z, we can have either $v <_q h(z)$ or $v \parallel_q h(z)$. Suppose for contradiction that $v \parallel_q h(z)$, then $h(z) \notin \{u, v\}$. Then, $p^{h(z)}_\triangle = q^{h(z)}_\triangle$ since the rewinding leaves $q^{h(z)}_\triangle$ intact. But that implies $h(q^z_\triangle) \subseteq q^{h(z)}_\triangle$ with $z <_q h(z)$, a contradiction. It follows $|p^{h(z)}_\triangle| \leq |p^v_\triangle| \leq |q^v_\triangle| - |q^y_\triangle| + |q^x_\triangle|$, where the second inequality follows by construction of rewinding that replaces $q^y_\triangle$ with $q^x_\triangle$. Putting everything together, we obtain $0 \leq |q^u_\triangle| - |q^x_\triangle| \leq -|q^y_\triangle| - 1 < 0$, a contradiction. □

Proof of Transitivity in Proposition 4.7. We show show that $\preceq_q$ is transitive. Assume $R[x] \preceq_q R[y]$ and $R[y] \preceq_q R[z]$. We distinguish four cases.

- Case that $R[x] <_q R[y]$ and $R[y] <_q R[z]$. Then we have $R[x] <_q R[z]$, as desired.
- Case that $q^y_\triangle \preceq_{y\to x} q^x_\triangle$ and $q^z_\triangle \preceq_{z\to y} q^y_\triangle$. Then we have $q^z_\triangle \preceq_{z\to x} q^x_\triangle$, as desired.

- Case that $R[x] <_q R[y]$ and $q^z_\triangle \leq_{z\to y} q^y_\triangle$. The claim follows if $R[x] <_q R[z]$. Suppose for contradiction that $R[z] <_q R[x]$. Then $R[z] <_q R[y]$, and $q^z_\triangle$ contains more atoms than $q^y_\triangle$. However, we have $q^z_\triangle \leq_{z\to y} q^y_\triangle$, a contradiction. It then must be that $R[x] \parallel_q R[z]$. Suppose for contradiction that $q^x_\triangle \leq_{x\to z} q^z_\triangle$. Then we have $q^x_\triangle \leq_{x\to y} q^y_\triangle$, but $R[x] <_q R[y]$, a contradiction. Since q satisfies $\mathsf{C}_{\mathsf{branch}}$, we have $q^z_\triangle \leq_{z\to x} q^x_\triangle$, as desired.
- Case that $q^y_\triangle \leq_{y\to x} q^x_\triangle$ and $R[y] <_q R[z]$. The claim follows if $R[x] <_q R[z]$. Suppose for contradiction that $R[z] <_q R[x]$. Then $R[y] <_q R[x]$, and $q^y_\triangle$ contains more atoms than $q^x_\triangle$. However, we have $q^y_\triangle \leq_{y\to x} q^x_\triangle$, a contradiction. It then must be that $R[x] \parallel_q R[z]$. Suppose for contradiction that $q^x_\triangle \leq_{x\to z} q^z_\triangle$. Then we have $q^y_\triangle \leq_{y\to z} q^z_\triangle$, but $R[y] <_q R[z]$, a contradiction. Since q satisfies $\mathsf{C}_{\mathsf{branch}}$, it follows that $q^z_\triangle \leq_{z\to x} q^x_\triangle$.

This concludes the proof. □

B MISSING PROOFS IN SECTION 6

We first show that the formula in Fig. 6 propagates on root homomorphisms.

Lemma B.1. *Let q be a rooted tree query satisfying $\mathsf{C}_{\mathsf{branch}}$ and $\mathbf{db}$ a database instance. Then for constants $c, d_1, d_2, \dots, d_n \in \mathrm{adom}(\mathbf{db})$ where $\vec{d} = \langle d_1, d_2, \dots, d_n\rangle$ and any two atoms $R[x]$ and $R[y]$ with $q^y_\triangle \leq_{y\to x} q^x_\triangle$, the following statements hold:*

(1) if $\mathrm{fact}(R(\underline{c}, \vec{d}), x)$ is true, then $\mathrm{fact}(R(\underline{c}, \vec{d}), y)$ is true; and
(2) if $\langle c, x\rangle \in B$, then $\langle c, y\rangle \in B$.

Proof. We show both (1) and (2) by an induction on the height k of the atom $R[y]$ in q.

- Basis $k = 0$. In this case, y is a leaf variable of q and (1) holds vacuously. Assume that the label of y is L, then there is an atom $L(\underline{y})$ in q. Then there must be an atom $L(\underline{x})$ in q. From $\langle c, x\rangle \in B$, we have $L(\underline{c}) \in \mathbf{db}$, and thus $\langle c, y\rangle \in B$ by the initialization step.
- Inductive step. Assume that both (1) and (2) holds if the height of $q^y_\triangle$ is less than k. Consider the case where the height of $q^y_\triangle$ is k.

  First we show (1) holds. Assume that $\mathrm{fact}(R(\underline{c}, \vec{d}), x)$ holds. Let $R[x] = R(\underline{x}, x_1, x_2, \dots, x_n)$ and $R[y] = R(\underline{y}, y_1, y_2, \dots, y_n)$. Consider two cases.
  - Case (I) that the following formula is true:

$$\bigwedge_{1\leq i\leq n} \langle d_i, x_i\rangle \in B. \tag{6}$$

    To show $\mathrm{fact}(R(\underline{c}, \vec{d}), y)$ holds, it suffices to show $\bigwedge_{1\leq i\leq n}\langle d_i, y_i\rangle \in B$. Consider any y_i. If y_i is a constant or a leaf variable with label $\perp$, then $\langle d_i, y_i\rangle \in B$ by the initialization step. Otherwise, there is an atom $T[y_i]$ in q. Since $q^y_\triangle \leq_{y\to x} q^x_\triangle$, there is some atom $T[x_i]$ in q such that $q^{y_i}_\triangle \leq_{y_i\to x_i} q^{x_i}_\triangle$ and $\langle d_i, x_i\rangle \in B$, by Equation (6). Since the height of $T[y_i]$ is less than k, by the inductive hypothesis for (2), we have $\langle d_i, y_i\rangle \in B$.
  - Case (II) that there is some atom $R[u] <_q R[x]$ such that $\mathrm{fact}(R(\underline{c}, \vec{d}), u)$ is true.

    If $R[u] <_q R[y]$, then $\mathrm{fact}(R(\underline{c}, \vec{d}), y)$ holds, as desired. Assume from here on that u is not an ancestor of y in q. Then, we must have $R[u] \parallel_q R[y]$. Indeed, if not, we would have $R[y] <_q R[u] <_q R[x]$, but $q^y_\triangle \leq_{y\to x} q^x_\triangle$, a contradiction.

    We argue that $q^y_\triangle \leq_{y\to u} q^u_\triangle$. If not, then by $\mathsf{C}_{\mathsf{branch}}$, we have $q^u_\triangle \leq_{u\to y} q^y_\triangle \leq_{y\to x} q^x_\triangle$, but $R[u] <_q R[x]$, a contradiction.

    Note that we just established $\mathrm{fact}(R(\underline{c}, \vec{d}), u)$ is true and $q^y_\triangle \leq_{y\to u} q^u_\triangle$ for $R[u] <_q R[x]$. If Case (I) holds when $\mathrm{fact}(R(\underline{c}, \vec{d}), u)$ is true, then $\mathrm{fact}(R(\underline{c}, \vec{d}), y)$ is true, as desired. Otherwise, by the previous argument in Case (II), either $\text{fact}(R(\underline{c}, \vec{d}), y)$ is true as desired, or there is another atom $R[w]$ such that $R[w] <_q R[u] <_q R[x]$ and $q^y_\Delta \leq_{y\to w} q^w_\Delta$. Since there are only finitely many R-atoms in q, this process must terminate and show that $\text{fact}(R(\underline{c}, \vec{d}), y)$ is true.

For (2), assume that $\langle c, x\rangle \in B$. For every fact $R(\underline{c}, \vec{d})$ in the block $R(\underline{c}, *)$ of **db**, $\text{fact}(R(\underline{c}, \vec{d}), x)$ holds. By (1), $\text{fact}(R(\underline{c}, \vec{d}), y)$ holds for every fact $R(\underline{c}, \vec{d})$ in the block $R(\underline{c}, *)$ of **db**. Hence, $\langle c, y\rangle \in B$.

The proof is now complete. □

Proof of Lemma 6.2. The lemma follows from Lemma B.1 if $q^y_\Delta \leq_{y\to x} q^x_\Delta$. Assume that $R[x] <_q R[y]$, and both (1) and (2) are straightforward by definition of $\text{fact}(R(\underline{c}, \vec{d}), y)$. □

C MISSING PROOFS IN SECTION 8

We define a *canonical copy* of a query q as a set of facts $\mu(q)$, where μ maps each variable in q to a unique constant. The following notation will be central in all our reductions. For a query q, variables x_i in q and distinct constants c_i, we denote

$$\langle q, [x_1, x_2, \dots, x_n \to c_1, c_2, \dots, c_n]\rangle$$

as the canonical copy $\mu(q)$, where $\mu(z) = c_i$ if $z = x_i$, and $\mu(z)$ is a fresh distinct constant otherwise.

Lemma C.1. CERTAINTY(q) *is* **coNP**-*hard for each* q *in* TreeBCQ *that violates* C_2.

Proof of Lemma C.1. Since q violates C_2, there exist two atoms $R(\underline{p}, \dots)$ and $R(\underline{n}, \dots)$ in q such that there is no homomorphism from q to neither $q^{R:p\hookrightarrow n}$ nor $q^{R:n\hookrightarrow p}$.

Consider now the root atom $A(\underline{r}, \dots)$. It must be that $r \neq p$, since otherwise, there would be a homomorphism from q to $q^{R:n\hookrightarrow p}$, a contradiction. Similarly, we have that $r \neq n$. Hence, the root atom is distinct from $R(\underline{p}, \dots)$ and $R(\underline{n}, \dots)$. We also have that $r <_q p$ and $r <_q n$.

We present a reduction from MonotoneSAT: Given a monotone CNF formula φ, i.e., each clause in φ contains either only positive literals or only negative literals, does φ have a satisfying assignment?

Let φ be a monotone CNF formula. We construct an instance **db** for CERTAINTY(q) as follows.

- for each variable z in φ, we introduce the facts $\langle q^p_\Delta, [p \to z]\rangle$ and $\langle q^n_\Delta, [n \to z]\rangle$;
- for each positive literal z in clause C, we introduce the facts $\langle q \setminus q^p_\Delta, [r, p \to C, z]\rangle$;
- for each negative literal z in clause $\overline{C}$, we introduce the facts $\langle q \setminus q^n_\Delta, [r, n \to \overline{C}, z]\rangle$;

Observe that the instance **db** has two types of inconsistent blocks. For relation A, we have a block for each positive or negative clause, where the primary key position is the clause. For relation R, for every variable z we have a block of size two, which corresponds to choosing a true/false assignment for z. All the other relations are consistent.

Additionally, for a positive literal $z \in C$, the set of facts $\langle q^p_\Delta, [p \to z]\rangle \cup \langle q \setminus q^p_\Delta, [r, p \to C, z]\rangle$ makes q true; similarly for a negative literal $z \in \overline{C}$, the facts $\langle q^n_\Delta, [n \to z]\rangle \cup \langle q \setminus q^n_\Delta, [n, p \to \overline{C}, z]\rangle$ makes q true. Note also that $\langle q^n_\Delta, [n \to z]\rangle \cup \langle q \setminus q^p_\Delta, [r, p \to C, z]\rangle$ is a canonical copy of $q^{R:p\hookrightarrow n}$ (and hence cannot satisfy q), while $\langle q^p_\Delta, [p \to z]\rangle \cup \langle q \setminus q^n_\Delta, [r, n \to \overline{C}, z]\rangle$ is a canonical copy of $q^{R:n\hookrightarrow p}$ (which also cannot satisfy q).

Now we argue that φ has a satisfying assignment χ if and only if **db** has a repair **r** that does not satisfy q.

$\boxed{\Longrightarrow}$ Assume that φ has a satisfying assignment χ. Consider the repair **r** of **db** constructed as follows:

- for each variable z, if $\chi(z) = \text{true}$, then **r** picks $\langle q^n_\Delta, [n \to z]\rangle$, otherwise **r** picks $\langle q^p_\Delta, [p \to z]\rangle$;

- for each positive clause C, $\mathbf{r}$ picks $\langle q \setminus q^p_\triangle, [r, p \to C, z] \rangle$ where z is a positive literal in C with $\chi(z) = \text{true}$; and
- for each negative clause $\overline{C}$, $\mathbf{r}$ picks $\langle q \setminus q^n_\triangle, [r, n \to \overline{C}, \overline{z}] \rangle$ where $\overline{z}$ is a negative literal in $\overline{C}$ with $\chi(z) = \text{false}$.

We show that $\mathbf{r}$ does not satisfy q. Indeed, for each positive clause C, there is a literal $z \in C$ with $\chi(z) = \text{true}$, and thus $\langle q \setminus q^p_\triangle, [r, p \to C, z] \rangle \subseteq \mathbf{r}$. However, we have $\langle q^n_\triangle, [n \to z] \rangle \subseteq \mathbf{r}$, and thus q is not satisfied. Similarly, for each negative clause $\overline{C}$, there is a literal $\overline{z} \in \overline{C}$ with $\chi(z) = \text{false}$, and thus $\langle q \setminus q^n_\triangle, [n, p \to \overline{C}, z] \rangle \subseteq \mathbf{r}$. However, we have $\langle q^p_\triangle, [p \to z] \rangle \subseteq \mathbf{r}$ and hence this part also cannot satisfy q. Hence $\mathbf{r}$ does not satisfy q.

$\boxed{\Longleftarrow}$ Now assume that **db** has a repair $\mathbf{r}$ that does not satisfy q. Consider the assignment χ such that $\chi(z) = \text{true}$ if $\langle q^n_\triangle, [n \to z] \rangle \subseteq \mathbf{r}$, and $\chi(z) = \text{false}$ otherwise. We argue that χ is a satisfying assignment for φ. For each positive clause C, there exists some $z \in C$ such that $\langle q \setminus q^p_\triangle, [r, p \to C, z] \rangle \subseteq \mathbf{r}$. Since $\mathbf{r}$ does not satisfy q, it must be that $\langle q^p_\triangle, [p \to z] \rangle \nsubseteq \mathbf{r}$ and thus $\langle q^n_\triangle, [n \to z] \rangle \subseteq \mathbf{r}$. By construction, z is true and the clause C is satisfied. Similarly, the negative clauses are all satisfied by the assignment. □

Lemma C.2. *Let q be a rooted tree query. If there exist two distinct atoms $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ such that $x <_q y$ and there is no root homomorphism from $q^y_\triangle$ to $q^x_\triangle$ (i.e., it does not hold that $q^y_\triangle \leq_{y \to x} q^x_\triangle$), then* CERTAINTY$(q)$ *is* **NL***-hard.*

Proof. The two following assumptions are without loss of generality: (i) there is no atom $R(\underline{z}, \dots)$ such that $z \notin \{x, y\}$, $x <_q z <_q y$ (we then say that $R[x]$ and $R[y]$ are consecutive), and (ii) for any $y <_q z$, $z \neq y$, we have $q^z_\triangle \leq_{z \to y} q^y_\triangle$. Indeed, we can pick $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ to be the pair of consecutive R-atoms that violates the root homomorphism condition and occurs lowest in the rooted tree. Such a pair must always exists, since the root homomorphism property is transitive, i.e., if $q^y_\triangle \leq_{y \to z} q^z_\triangle$ and $q^z_\triangle \leq_{z \to x} q^x_\triangle$, then we also have that $q^y_\triangle \leq_{y \to x} q^x_\triangle$.

We present a reduction from the complement of the REACHABILITY problem, which is **NL**-hard: Given a directed acyclic graph $G = (V, E)$ and $s, t \in V$, is there a directed path from s to t in G?

We construct an instance **db** for CERTAINTY(q) as follows. First, we introduce two new constants s' and t'. Then:

- for each $u \in V \cup \{s'\}$, introduce $\langle q \setminus q^x_\triangle, [x \to u] \rangle$;
- for every edge $(u, v) \in E \cup \{(s', s), (t, t')\}$, introduce $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \to u, v] \rangle$;
- for every vertex $u \in V$, introduce $\langle q^y_\triangle, [y \to u] \rangle$.

Note that the above construction guarantees that only R has inconsistent blocks.

We now argue that there is a directed path $(u_1, u_2, \dots, u_k)$ with $(u_i, u_{i+1}) \in E$, $u_1 = s$ and $u_k = t$ in G if and only if there is a repair of **db** that does not satisfy q.

$\boxed{\Longrightarrow}$ Assume that there exists a directed path $(u_1, u_2, \dots, u_k)$ with $(u_i, u_{i+1}) \in E$, $u_1 = s$ and $u_k = t$ in G. Denote $u_0 = s'$ and $u_{k+1} = t'$. Let $\mathbf{r}$ be the repair that picks $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \to u_i, u_{i+1}] \rangle$ for every $1 \leq i \leq k-1$, and picks $\langle q^y_\triangle, [y \to u] \rangle$ for any other vertex u. Suppose for contradiction that $\mathbf{r}$ satisfies q with a valuation θ. By a simple size argument, it is not possible that $\theta(q) \subseteq \langle q^y_\triangle, [y \to u] \rangle$ for any $u \notin V$ since the size does not fit.

We argue that we must have $\theta(x) = u_i$ and $\theta(y) = u_{i+1}$ for some $0 \leq i < k$. If $\theta(x) = u_i \in \{u_0, u_1, \dots, u_k\}$, then we must have $\theta(y) = u_{i+1}$ since $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \to u, v] \rangle$ is a canonical copy. Suppose for contradiction that $\theta(x) \notin \{u_0, u_1, \dots, u_k\}$. It is not possible that $\theta(x) = u_{k+1} = t'$ since by construction, there is no rooted tree set rooted at t'. Note that there is no atom $R(\underline{z}, \dots)$ such that $z \notin x, y, x <_q z <_q y$. Hence $\theta(x)$ cannot fall on the path connecting any u_i and u_{i+1}, and $\theta(q^x_\triangle)$ must be contained in some $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \to u_i, u_{i+1}] \rangle$. Then, there must be an atom $R(\underline{z}, \dots)$ such that (i) $x <_q z$, (ii) $z \parallel_q y$, and (iii) $\theta(q^x_\triangle)$ is contained in $\langle q^z_\triangle, [z \rightarrow \theta(x)]\rangle$, which, by a simple size argument, can be seen to be impossible.

By construction, there is a canonical copy of $q^y_\triangle$ rooted at u_{i+1}. If this canonical copy is contained in $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \rightarrow u_{i+1}, u_{i+2}]\rangle$, then there is a root homomorphism from $q^y_\triangle$ to $q^x_\triangle \setminus q^y_\triangle$, and so from $q^y_\triangle$ to $q^x_\triangle$, a contradiction. Otherwise, there exists some atom $R(\underline{z}, \dots)$ such that (i) $y <_q z$ and (ii) $q^y_\triangle \setminus q^z_\triangle$ has a root homomorphism to $q^x_\triangle \setminus q^y_\triangle$. Recall now that by our initial assumption, we must have that $q^z_\triangle \leq_{z \rightarrow y} q^y_\triangle$. This implies that we can now generate a root homomorphism from $q^y_\triangle$ to $q^x_\triangle$, a contradiction.

$\boxed{\Longleftarrow}$ Assume that there is no directed path from s to t in G. Consider any repair $\mathbf{r}$ of $\mathbf{db}$. Since G is acyclic, there exists a maximal sequence $u_0, u_1, \dots, u_k$ with $k \geq 1$ such that $u_0 = s'$, $u_1 = s$, $\langle q^x_\triangle \setminus q^y_\triangle, [x, y \rightarrow u_i, u_{i+1}]\rangle \subseteq \mathbf{r}$ for $0 \leq i < k$ and $\langle q^y_\triangle, [y \rightarrow u_k]\rangle \subseteq \mathbf{r}$. Then, the following set of facts satisfies q:

$$\langle q \setminus q^x_\triangle, [x \rightarrow u_{k-1}]\rangle \cup \langle q^x_\triangle \setminus q^y_\triangle, [x, y \rightarrow u_{k-1}, u_k]\rangle \cup \langle q^y_\triangle, [y \rightarrow u_k]\rangle.$$

This shows that CERTAINTY(q) is **NL**-hard since **NL** is closed under complement. □

Lemma C.3. *CERTAINTY(q) is* **NL**-*hard for each q in* TreeBCQ *that violates* C_1.

Proof of Lemma C.3. Assume that q violates C_1. Then there exist two distinct atoms $R(\underline{x}, \dots)$ and $R(\underline{y}, \dots)$ in q such that there is no root homomorphism from $q^y_\triangle$ to $q^x_\triangle$ or from $q^x_\triangle$ to $q^y_\triangle$. If $x \parallel_q y$, Lemma 4.2 implies that C_2 is also violated, so CERTAINTY(q) is **coNP**-hard by Lemma C.1. Otherwise, CERTAINTY(q) is **NL**-hard by Lemma C.2. □

Proof of Proposition 8.1. Immediate from Lemmas C.1 and C.3. □

D MISSING PROOFS IN SECTION 9

Proof of Bridging Lemma. Assume that, in moving from q to q^{sjf}, occurrences of a same relation name R in q are renamed in $R_1, R_2, \dots, R_m$, where m is the number of occurrences of R in q. Let f be a mapping from facts to facts such that for every atom $R_i(x_1, \dots, x_n) \in q^{\mathsf{sjf}}$, for every R_i-fact $A := R_i(a_1, \dots, a_n)$, $f(A) := R(\langle a_1, x_1\rangle, \dots, \langle a_n, x_n\rangle)$. Notice that f maps R_i-facts to R-facts. Here, every couple $\langle a_i, x_i\rangle$ denotes a constant such that $\langle a_i, x_i\rangle = \langle a_j, x_j\rangle$ if and only if both $a_i = a_j$ and $x_i = x_j$. Moreover, if c is a constant, then $\langle c, c\rangle := c$. Since no two distinct atoms of q agree on both their relation name and primary key, it will be the case that for all facts A and B, $A \sim B$ if and only if $f(A) \sim f(B)$, where $\sim$ denotes "is key-equal-to."

We extend the function f in the natural way to databases $\mathbf{db}$ that use only relation names from q^{sjf}: $f(\mathbf{db}) := \{f(A) \mid A \in \mathbf{db}\}$. Clearly, $f(\mathbf{db})$ can be computed in **FO**. Let $\mathbf{db}$ be a set of facts with relation names in q^{sjf}. It can be easily seen that $|\mathsf{rset}(\mathbf{db})| = |\mathsf{rset}(f(\mathbf{db}))|$ and $\mathsf{rset}(f(\mathbf{db})) = \{f(\mathbf{r}) \mid \mathbf{r} \in \mathsf{rset}(\mathbf{db})\}$, where $\mathsf{rset}(\mathbf{db})$ is the set of repair of $\mathbf{db}$. Let $\mathbf{r}$ be an arbitrary repair of $\mathbf{db}$. It suffices to show that

$$\mathbf{r} \models q^{\mathsf{sjf}} \iff f(\mathbf{r}) \models q.$$

For the implication $\Longrightarrow$, assume that $\mathbf{r} \models q^{\mathsf{sjf}}$. We can assume a valuation θ over $\mathbf{vars}(q^{\mathsf{sjf}})$ such that $\theta(q^{\mathsf{sjf}}) \subseteq \mathbf{r}$. Let μ be the valuation such that for every variable $x \in \mathbf{vars}(q^{\mathsf{sjf}})$, $\mu(x) = \langle \theta(x), x\rangle$. By our construction of q^{sjf} and f, it will be the case that $\mu(q) \subseteq f(\mathbf{r})$, thus $f(\mathbf{r}) \models q$.

For the implication $\Longleftarrow$, assume that $f(\mathbf{r}) \models q$. We can assume a valuation θ over $\mathbf{vars}(q)$ such that $\theta(q) \subseteq f(\mathbf{r})$. Notice that if c is a constant in q, then it must be the case that $\theta(c) = \langle c, c\rangle := c$. We define θ_L as the substitution that maps every variable x in $\mathbf{vars}(q)$ to the first coordinate of $\theta(x)$; and θ_R maps every x to the second coordinate of $\theta(x)$. It is convenient to think of L and R as references to the Left and the Right coordinates, respectively. Thus, by definition, $\theta(x) = \langle \theta_L(x), \theta_R(x)\rangle$.

By inspecting the right-hand coordinates of couples $\langle a_i, x_i\rangle$ in $f(\mathbf{r})$, it can be easily seen that $\theta(q) \subseteq f(\mathbf{r})$ implies $\theta_R(q) \subseteq q$. Since the query q is minimal, it follows that $\theta_R(q) = q$, i.e., θ_R is an automorphism. Since the inverse of an automorphism is an automorphism, ${\theta_R}^{-1}$ is an automorphism as well. Note that θ_R will be the identity on constants that appear in q. We now define $\mu := \theta_L \circ {\theta_R}^{-1}$ (i.e., μ is the composed function θ_L after the inverse of θ_R), and show that $\mu(q^{\mathsf{sjf}}) \subseteq \mathbf{r}$, which implies the desired result that $\mathbf{r} \models q^{\mathsf{sjf}}$. To this extent, let $R_i(x_1, \dots, x_n)$ be an arbitrary atom of q^{sjf}. It suffices to show $R_i(\mu(x_1), \dots, \mu(x_n)) \in \mathbf{r}$, which can be proved as follows. From $R_i(x_1, \dots, x_n) \in q^{\mathsf{sjf}}$, it follows $R(x_1, \dots, x_n) \in q$. Thus, since ${\theta_R}^{-1}$ is an automorphism, $R(\ {\theta_R}^{-1}(x_1),\ \dots,\ {\theta_R}^{-1}(x_n)\ ) \in q$. Since $\theta(q) \subseteq f(\mathbf{r})$, $R(\ \theta\left({\theta_R}^{-1}(x_1)\right),\ \dots,\ \theta\left({\theta_R}^{-1}(x_n)\right)\ ) \in f(\mathbf{r})$. Since, for every symbol s, $\theta(s) = \langle\theta_L(s), \theta_R(s)\rangle$ and $\theta_R\left({\theta_R}^{-1}(s)\right) = s$, we obtain $R(\ \langle\theta_L({\theta_R}^{-1}(x_1)), x_1\rangle,\ \dots,\ \langle\theta_L({\theta_R}^{-1}(x_n)), x_n\rangle\ ) \in f(\mathbf{r})$. That is, by our definition of μ, $R(\ \langle\mu(x_1), x_1\rangle,\ \dots,\ \langle\mu(x_n), x_n\rangle\ ) \in f(\mathbf{r})$. From this, it is correct to conclude that $R_i(\mu(x_1), \dots, \mu(x_n)) \in \mathbf{r}$. This concludes the proof. □

Attacks. Let q be a self-join-free Boolean CQ. For every atom $F \in q$, we define $F^{+,q}$ as the set of all variables in q that are functionally determined by $\mathsf{key}(F)$ with respect to all functional dependencies of the form $\mathsf{key}(G) \rightarrow \mathsf{vars}(G)$ with $G \in q \setminus \{F\}$. Following [36], the *attack graph* of q is a directed graph whose vertices are the atoms of q. There is a directed edge, called *attack*, from F to G ($F \neq G$), written $F \overset{q}{\rightsquigarrow} G$, if there exists a path between F and G in $\mathcal{G}(q)$ such that every two adjacent atoms share a variable not in $F^{+,q}$. The attack is called *weak* if every variable in $\mathsf{key}(G)$ is functionally determined by $\mathsf{key}(F)$ with respect to all functional dependencies of the form $\mathsf{key}(H) \rightarrow \mathsf{vars}(H)$ with $H \in q$; otherwise it is called *strong*.

We can now prove the proposition.

Proof of Proposition 9.3. Let q be a connected minimal query in $\mathsf{GraphBCQ}$.

Assume that q is not a rooted tree query. Then, q contains two atoms $R(\underline{x}, \dots, z, \dots)$ and $S(\underline{y}, \dots, z, \dots)$ with $x \neq y$ (and possibly $R = S$). Consider now q^{sjf}, and let R_0 and S_0 be the corresponding atoms of R and S in q^{sjf}. It is easily verified that $R_0^{+,q^{\mathsf{sjf}}} = \{x\}$ and $S_0^{+,q^{\mathsf{sjf}}} = \{y\}$, with neither set containing the shared variable z. Hence, $R_0 \overset{q^{\mathsf{sjf}}}{\rightsquigarrow} S_0$ and $S_0 \overset{q^{\mathsf{sjf}}}{\rightsquigarrow} R_0$. By [36, Theorem 3.2], $\mathsf{CERTAINTY}(q^{\mathsf{sjf}})$ is $\mathbf{L}$-hard (due to this cycle in the attack graph of q^{sjf}), and so is $\mathsf{CERTAINTY}(q)$ by Lemma 9.4.

Next we additionally assume that q is Berge-acyclic, that is, $q \in \mathsf{Graph_{Berge}BCQ}$. It is easily verified that q^{sjf} also belongs to $\mathsf{Graph_{Berge}BCQ}$. Let Σ_q be the set of functional dependencies containing $x \rightarrow y$ whenever $x, y \in \mathsf{vars}(q)$ such that y occurs in an atom of q with primary key x. Assume for the sake of a contradiction that $\Sigma_q \models x \rightarrow y$ and $\Sigma_q \models y \rightarrow x$. Then, there exist atoms $R_0, R_1, \dots, R_n$ and $S_0, S_1, \dots, S_m$ and variables $x_0, x_1, x_2, \dots, x_{n+1}, y_0, y_1, \dots, y_{m+1}$ in q^{sjf} where $x_0 = x$, $x_{n+1} = y$, $y_0 = y$, $y_{m+1} = x$ such that q^{sjf} contains atoms $R_i(\underline{x_i}, \dots, x_{i+1}, \dots)$ for every $0 \leq i \leq n$, and $S_i(\underline{y_i}, \dots, y_{i+1}, \dots)$ for every $0 \leq i \leq m$. Then,

$$\left(x_0, R_0, x_1, R_1, \dots, R_n, x_{n+1}(= y = y_0), S_0, y_1, S_1, \dots, S_m, y_{m+1}(= x = x_0)\right)$$

is a Berge-cycle in q^{sjf}, contradicting that q^{sjf} is Berge-acyclic. We conclude by contradiction that at least one of $x \rightarrow y$ or $y \rightarrow x$ is not implied by Σ_q. Consequently, among the mutual attacks between R_0 and S_0 in q^{sjf}, there is at least one that is strong. By [36, Theorem 3.2], $\mathsf{CERTAINTY}(q^{\mathsf{sjf}})$ is $\mathbf{coNP}$-hard (due to this strong cycle in the attack graph of q^{sjf}), and so is $\mathsf{CERTAINTY}(q)$ by Lemma 9.4. □

Received June 2023; revised August 2023; accepted September 2023