Consistent Query Answering for Primary Keys on Path Queries

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

Xiating Ouyang
University of Wisconsin–Madison
WI, USA
xouyang@cs.wisc.edu

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

ABSTRACT

We study the data complexity of consistent query answering (CQA) on databases that may violate the primary key constraints. A repair is a maximal consistent subset of the database. For a Boolean query q, the problem CERTAINTY(q) takes a database as input, and asks whether or not each repair satisfies the query q. It is known that for any self-join-free Boolean conjunctive query q, CERTAINTY(q) is in **FO**, **L**-complete, or **coNP**-complete. In particular, CERTAINTY(q) is in **FO** for any self-join-free Boolean path query q. In this paper, we show that if self-joins are allowed, then the complexity of CERTAINTY(q) for Boolean path queries q exhibits a tetrachotomy between **FO**, **NL**-complete, **PTIME**-complete, and **coNP**-complete. Moreover, it is decidable, in polynomial time in the size of the query q, which of the four cases applies.

CCS CONCEPTS

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

KEYWORDS

conjunctive queries; consistent query answering; database repairing; first-order rewriting; keys

ACM Reference Format:
Paraschos Koutris, Xiating Ouyang, and Jef Wijsen. 2021. Consistent Query Answering for Primary Keys on Path Queries. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3452021.3458334

1 INTRODUCTION

Primary keys are probably the most common integrity constraints in relational database systems. Although databases should ideally satisfy their integrity constraints, data integration is today frequently cited as a cause for primary key violations, for example, when a same client is stored with different birthdays in two data sources. A *repair* of such an inconsistent database instance is then naturally defined as a maximal consistent subinstance. 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 change the notion of query answer: the *consistent* (or *certain*) *answer* is defined as the intersection of the query answers over all (exponentially many) repairs. In computational complexity studies, consistent query answering is commonly defined as the data complexity of the following decision problem, for a fixed Boolean query q:

Problem: CERTAINTY(q)
Input: A database instance **db**.
Question: Does q evaluate to true on every repair of **db**?

For every first-order query q, the problem CERTAINTY(q) is obviously in **coNP**. However, despite significant research efforts (see Section 9), a fine-grained complexity classification is still largely open. A prominent open conjecture is the following.

Conjecture 1.1. *For each Boolean conjunctive query q, the problem* CERTAINTY(q) *is either in* **PTIME** *or* **coNP**-*complete.*

On the other hand, for the smaller class of self-join-free Boolean conjunctive queries, the complexity landscape is by now well understood, as summarized by the following theorem.

Theorem 1.2 ([26]). *For each self-join-free Boolean conjunctive query q,* CERTAINTY(q) *is in* **FO**, **L**-*complete, or* **coNP**-*complete, and it is decidable which of the three cases applies.*

Abandoning the restriction of self-join-freeness turns out to be a major challenge. The difficulty of self-joins is caused by the obvious observation that a single database fact can be used to satisfy more than one atom of a conjunctive query, as illustrated by Example 1.3. Self-joins happen to significantly change the complexity landscape laid down in Theorem 1.2; this is illustrated by Example 1.4. Self-join-freeness is a simplifying assumption that is also used outside CQA (e.g., [3, 10, 11]).

Example 1.3. Take the self-join $q_1 = \exists x \exists y(R(\underline{x}, y) \wedge R(\underline{y}, x))$ and its self-join-free counterpart $q_2 = \exists x \exists y(R(\underline{x}, y) \wedge S(\underline{y}, x))$. Consider the inconsistent database instance **db** in Figure 1. We have that **db** is a "no"-instance of CERTAINTY(q_2), because q_2 is not satisfied by the repair $\{R(\underline{a}, a), R(\underline{b}, b), S(\underline{a}, b), S(\underline{b}, a)\}$. However, **db** is a "yes"-instance of CERTAINTY(q_1). This is because every repair that contains $R(\underline{a}, a)$ or $R(\underline{b}, b)$ will satisfy q_1, while a repair that contains neither of these facts must contain $R(\underline{a}, b)$ and $R(\underline{b}, a)$, which together also satisfy q_1. □

Example 1.4. Take the self-join $q_1 = \exists x \exists y \exists z(R(\underline{x}, z) \wedge R(\underline{y}, z))$ and its self-join-free counterpart $q_2 = \exists x \exists y \exists z(R(\underline{x}, z) \wedge S(\underline{y}, z))$. CERTAINTY($q_2$) is known to be **coNP**-complete, whereas it is easily verified that CERTAINTY(q_1) is in **FO**, by observing that a database instance is a "yes"-instance of CERTAINTY(q_1) if and only if it satisfies $\exists x \exists y(R(\underline{x}, y))$. □

PODS '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8381-3/21/06…$15.00
https://doi.org/10.1145/3452021.3458334

R	1	2
	a	a
	a	b
	b	a
	b	b

S	1	2
	a	a
	a	b
	b	a
	b	b

Figure 1: An inconsistent database instance db.

This paper makes a contribution to the complexity classification of CERTAINTY(q) for conjunctive queries, possibly with self-joins, of the form

$$\exists x_1 \cdots \exists x_{k+1}(R_1(\underline{x_1}, x_2) \wedge R_2(\underline{x_2}, x_3) \wedge \cdots \wedge R_k(\underline{x_k}, x_{k+1})),$$

which we call *path queries*. As will become apparent in our technical treatment, the classification of path queries is already very challenging, even though it is only a first step towards Conjecture 1.1, which is currently beyond reach. If all R_is are distinct (i.e., if there are no self-joins), then CERTAINTY(q) is known to be in **FO** for path queries q. However, when self-joins are allowed, the complexity landscape of CERTAINTY(q) for path queries exhibits a tetrachotomy, as stated by the following main result of our paper.

Theorem 1.5 (Tetrachotomy Theorem). *For each Boolean path query q,* CERTAINTY(q) *is in* **FO**, **NL**-*complete,* **PTIME**-*complete, or* **coNP**-*complete, and it is decidable in polynomial time in the size of q which of the four cases applies.*

Comparing Theorem 1.2 and Theorem 1.5, it is striking that there are path queries q for which CERTAINTY(q) is **NL**-complete or **PTIME**-complete, whereas these complexity classes do not occur for self-join-free queries (under standard complexity assumptions). So even for the restricted class of path queries, allowing self-joins immediately results in a more varied complexity landscape.

Let us provide some intuitions behind Theorem 1.5 by means of examples. Path queries use only binary relation names. A database instance **db** with binary facts can be viewed as a directed edge-colored graph: a fact $R(\underline{a}, b)$ is a directed edge from a to b with color R. A repair of **db** is obtained by choosing, for each vertex, precisely one outgoing edge among all outgoing edges of the same color. We will use the shorthand $q = RR$ to denote the path query $q = \exists x \exists y \exists z(R(\underline{x}, y) \wedge R(\underline{y}, z))$.

In general, path queries can be represented by words over the alphabet of relation names. Throughout this paper, relation names are in uppercase letters, while lowercase letters u, v, w stand for (possibly empty) words. An important operation on words is dubbed *rewinding*: if a word has a factor of the form RvR, then rewinding refers to the operation that replaces this factor with $RvRvR$. That is, rewinding the factor RvR in the word $uRvRw$ yields the longer word $uRvRvRw$. For short, we also say that $uRvRw$ *rewinds to* the word $u{\cdot}Rv{\cdot}\underline{Rv}{\cdot}Rw$, where we used concatenation ($\cdot$) and underlining for clarity. For example, $TWITTER$ rewinds to $TWI{\cdot}\underline{TWI}{\cdot}TTER$, but also to $TWIT{\cdot}\underline{TWIT}{\cdot}TER$ and to $TWI{\cdot}T{\cdot}\underline{T}{\cdot}TER$.

Let $q_1 = RR$. It is easily verified that a database instance is a "yes"-instance of CERTAINTY(q_1) if and only if it satisfies the following first-order formula:

$$\varphi = \exists x(\exists y R(\underline{x}, y) \wedge \forall y(R(\underline{x}, y) \rightarrow \exists z R(\underline{y}, z))).$$

Informally, every repair contains an R-path of length 2 if and only if there exists some vertex x such that every repair contains a path of length 2 starting in x.

Let $q_2 = RRX$, and consider the database instance in Figure 2. Since the only conflicting facts are $R(\underline{1}, 2)$ and $R(\underline{1}, 3)$, this database instance has two repairs. Both repairs satisfy RRX, but unlike the previous example, there is no vertex x such that every repair has a path colored RRX that starts in x. Indeed, in one repair, such path starts in 0; in the other repair it starts in 1. For reasons that will become apparent in our theoretical development, it is significant that both repairs have paths that start in 0 and are colored by a word in the regular language defined by $RR(R)^*X$. This is exactly the language that contains RRX and is closed under the rewinding operation. In general, it can be verified with some effort that a database instance is a "yes"-instance of CERTAINTY(q_2) if and only if it contains some vertex x such that every repair has a path that starts in x and is colored by a word in the regular language defined by $RR(R)^*X$. The latter condition can be tested in **PTIME** (and even in **NL**).

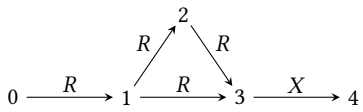


Figure 2: An example database instance db for $q_2 = RRX$.

The situation is still different for $q_3 = ARRX$, for which it will be shown that CERTAINTY(q_3) is **coNP**-complete. Unlike our previous example, repeated rewinding of $ARRX$ into words of the language $ARR(R)^*X$ is not helpful. For example, in the database instance of Figure 3, every repair has a path that starts in 0 and is colored with a word in the language defined by $ARR(R)^*X$. However, the repair that contains $R(\underline{a}, c)$ does not satisfy q_3. Unlike Figure 2, the "bifurcation" in Figure 3 can be used as a gadget for showing **coNP**-completeness in Section 7.

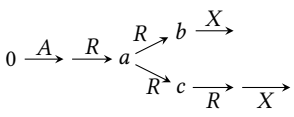


Figure 3: An example database instance db for $q_3 = ARRX$.

Organization. Section 2 introduces the preliminaries. In Section 3, the statement of Theorem 3.2 gives the syntactic conditions for deciding the complexity of CERTAINTY(q) for path queries q. To prove this theorem, we view the rewinding operator from the perspectives of regular expressions and automata, which are presented in Sections 4 and 5 respectively. Sections 6 and 7 present, respectively, complexity upper bounds and lower bounds of our classification. In Section 8, we extend our classification result to path queries with constants. Section 9 discusses related work, and Section 10 concludes this paper.

2 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. We consider only 2-ary *relation names*, where the first position is called the *primary key*. If R is a relation name, and s, t are variables or constants, then $R(\underline{s}, t)$ is an *atom*. 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* is a finite set **db** of facts using only the relation names of the schema. We write adom(**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 $R(\underline{c}, *)$ for the block that contains all facts with relation name R and primary-key value c. 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 $q = \{R_1(\underline{x_1}, y_1), \ldots, R_n(\underline{x_n}, y_n)\}$ of atoms. We denote by vars(q) the set of variables that occur in q. The set q represents the first-order sentence

$$\exists u_1 \cdots \exists u_k (R_1(\underline{x_1}, y_1) \wedge \cdots \wedge R_n(\underline{x_n}, y_n)),$$

where $\{u_1, \ldots, u_k\} = \text{vars}(q)$.

We say that a Boolean conjunctive query q has a *self-join* if some relation name occurs more than once in q. A conjunctive query without self-joins is called *self-join-free*.

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

Path queries. A *path query* is a Boolean conjunctive query without constants of the following form:

$$q = \{R_1(\underline{x_1}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_k(\underline{x_k}, x_{k+1})\},$$

where $x_1, x_2, \ldots, x_{k+1}$ are distinct variables, and $R_1, R_2, \ldots, R_k$ are (not necessarily distinct) relation names. It will often be convenient to denote this query as a *word* $R_1R_2 \cdots R_k$ over the alphabet of relation names. This "word" representation is obviously lossless up to a variable renaming. Importantly, path queries may have self-joins, i.e., a relation name may occur multiple times. Path queries containing constants will be discussed in Section 8. The treatment of constants is significant, because it allows moving from Boolean to non-Boolean queries, by using that free variables behave like constants.

3 THE COMPLEXITY CLASSIFICATION

We define syntactic conditions C_1, C_2, and C_3 for path queries q. Let R be any relation name in q, and let u, v, and w be (possibly empty) words over the alphabet of relation names of q.

C_1: Whenever $q = uRvRw$, q is a prefix of $uRvRvRw$.

C_2: Whenever $q = uRvRw$, q is a factor of $uRvRvRw$; and whenever $q = uRv_1Rv_2Rw$ for consecutive occurrences of R, $v_1 = v_2$ or Rw is a prefix of Rv_1.

C_3: Whenever $q = uRvRw$, q is a factor of $uRvRvRw$.

It is instructive to think of these conditions in terms of the rewinding operator introduced in Section 1: C_1 is tantamount to saying that q is a prefix of every word to which q rewinds; and C_3 says that q is a factor of every word to which q rewinds. These conditions can be checked in polynomial time in the length of the word q. The following result has an easy proof.

Proposition 3.1. *Let q be a path query. If q satisfies C_1, then q satisfies C_2; and if q satisfies C_2, then q satisfies C_3.*

The main part of this paper comprises a proof of the following theorem, which refines the statement of Theorem 1.5 by adding syntactic conditions. The theorem is illustrated by Example 3.3.

Theorem 3.2. *For every path query q, the following complexity upper bounds obtain:*

- *if q satisfies C_1, then* CERTAINTY(q) *is in* **FO**;
- *if q satisfies C_2, then* CERTAINTY(q) *is in* **NL**; *and*
- *if q satisfies C_3, then* CERTAINTY(q) *is in* **PTIME**.

Moreover, for every path query q, the following complexity lower bounds obtain:

- *if q violates C_1, then* CERTAINTY(q) *is* **NL**-*hard;*
- *if q violates C_2, then* CERTAINTY(q) *is* **PTIME**-*hard; and*
- *if q violates C_3, then* CERTAINTY(q) *is* **coNP**-*complete.*

Example 3.3. The query $q_1 = RXRX$ rewinds to (and only to) $RX \cdot \underline{RX} \cdot RX$ and $R \cdot XR \cdot \underline{XR} \cdot X$, which both contain q_1 as a prefix. It is correct to conclude that CERTAINTY(q_1) is in **FO**.

The query $q_2 = RXRY$ rewinds only to $RX \cdot \underline{RX} \cdot RY$, which contains q_2 as a factor, but not as a prefix. Therefore, q_2 satisfies C_3, but violates C_1. Since q_2 vacuously satisfies C_2 (because no relation name occurs three times in q_2), it is correct to conclude that CERTAINTY(q_2) is **NL**-complete.

The query $q_3 = RXRYRY$ rewinds to $RX \cdot \underline{RX} \cdot RYRY$, to $RXRY \cdot \underline{RXRY} \cdot RY$, and to $RX \cdot RY \cdot \underline{RY} \cdot RY = RXR \cdot YR \cdot \underline{YR} \cdot Y$. Since these words contain q_3 as a factor, but not always as a prefix, we have that q_3 satisfies C_3 but violates C_1. It can be verified that q_3 violates C_2 by writing it as follows:

$$q_3 = \underbrace{\varepsilon}_{u} \underbrace{RX}_{Rv_1} \underbrace{RY}_{Rv_2} \underbrace{RY}_{Rw}$$

We have $X = v_1 \neq v_2 = Y$, but $Rw = RY$ is not a prefix of $Rv_1 = RX$. Thus, CERTAINTY(q_3) is **PTIME**-complete.

Finally, the path query $q_4 = RXRXRYRY$ rewinds, among others, to $RX \cdot RXRY \cdot \underline{RXRY} \cdot RY$, which does not contain q_4 as a factor. It is correct to conclude that CERTAINTY(q_4) is **coNP**-complete. □

4 REGEXES FOR C_1, C_2, AND C_3

In this section, we show that the conditions C_1, C_2, and C_3 can be captured by regular expressions (or regexes) on path queries, which will be used in the proof of Theorem 3.2. Since these results are within the field of *combinatorics of words*, we will use the term *word* rather than *path query*.

Definition 4.1. We define four properties $\mathcal{B}_1$, $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, $\mathcal{B}_3$ that a word q can possess:

$\mathcal{B}_1$: For some integer $k \geq 0$, there are words v, w such that vw is self-join-free and q is a prefix of $w\,(v)^k$.

$\mathcal{B}_{2a}$: For some integers $j, k \geq 0$, there are words u, v, w such that uvw is self-join-free and q is a factor of $(u)^j\, w\,(v)^k$.

$\mathcal{B}_{2b}$: For some integer $k \geq 0$, there are words u, v, w such that uvw is self-join-free and q is a factor of $(uv)^k\, wv$.

$\mathcal{B}_3$: For some integer $k \geq 0$, there are words u, v, w such that uvw is self-join-free and q is a factor of $uw\,(uv)^k$. □

We can identify each condition among C_1, C_2, C_3, $\mathcal{B}_1$, $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, $\mathcal{B}_3$ with the set of all words satisfying this condition. Note then that $\mathcal{B}_1 \subseteq \mathcal{B}_{2a} \cap \mathcal{B}_3$. The results in the remainder of this section can be summarized as follows:

- $C_1 = \mathcal{B}_1$ (Lemma 4.2)
- $C_2 = \mathcal{B}_{2a} \cup \mathcal{B}_{2b}$ (Lemma 4.5)
- $C_3 = \mathcal{B}_{2a} \cup \mathcal{B}_{2b} \cup \mathcal{B}_3$ (Lemma 4.3)

Moreover, Lemma 4.5 characterizes $C_3 \setminus C_2$.

Lemma 4.2. *For every word q, the following are equivalent:*

(1) q satisfies C_1; and
(2) q satisfies $\mathcal{B}_1$.

Lemma 4.3. *For every word q, the following are equivalent:*

(1) q satisfies C_3; and
(2) q satisfies $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, or $\mathcal{B}_3$.

Definition 4.4 (First and last symbol). For a nonempty word u, we write $\text{first}(u)$ and $\text{last}(u)$ for, respectively, the first and the last symbol of u. □

Lemma 4.5. *Let q be a word that satisfies C_3. Then, the following three statements are equivalent:*

(1) q violates C_2;
(2) q violates both $\mathcal{B}_{2a}$ and $\mathcal{B}_{2b}$; and
(3) there are words u, v, w with $u \neq \varepsilon$ and uvw self-join-free such that either
  (a) $v \neq \varepsilon$ and $\text{last}(u) \cdot wuvu \cdot \text{first}(v)$ is a factor of q; or
  (b) $v = \varepsilon$, $w \neq \varepsilon$, and $\text{last}(u) \cdot w\,(u)^2 \cdot \text{first}(u)$ is a factor of q.

The shortest word of the form (3a) in the preceding lemma is *RRSRS* (let $u = R$, $v = S$, and $w = \varepsilon$), and the shortest word of the form (3b) is *RSRRR* (let $u = R$, $v = \varepsilon$, and $w = S$). Note that since each of C_2, $\mathcal{B}_{2a}$, and $\mathcal{B}_{2b}$ implies C_3, it is correct to conclude that the equivalence between the first two items in Lemma 4.5 does not need the hypothesis that q must satisfy C_3.

5 AUTOMATON-BASED PERSPECTIVE

In this section, we prove an important lemma, Lemma 5.9, which will be used for proving the complexity upper bounds in Theorem 3.2.

5.1 From Path Queries to Finite Automata

We can view a path query q as a word where the alphabet is the set of relation names. We now associate each path query q with a nondeterministic finite automaton (NFA), denoted $\text{NFA}(q)$.

Definition 5.1 ($\text{NFA}(q)$). Every word q gives rise to a nondeterministic finite automaton (NFA) with ε-moves, denoted $\text{NFA}(q)$, as follows.

States: The set of states is the set of prefixes of q. We include the empty word ε in the prefixes of q.

Forward transitions: If u and uR are states, then there is a transition with label R from state u to state uR. These transitions are said to be *forward*.

Backward transitions: If uR and wR are states such that $|u| < |w|$ (and therefore uR is a prefix of w), then there is a transition with label ε from state wR to state uR. These transitions are said to be *backward*, and capture the operation dubbed rewinding.

Initial and accepting states: The initial state is ε and the only accepting state is q. □

Figure 4 shows the automaton $\text{NFA}(RXRRR)$. Informally, the forward transitions capture the automaton that would accept the word *RXRRR*, while the backward transitions capture the existence of self-joins that allow an application of the rewind operator. We now take an alternative route for defining the language accepted by $\text{NFA}(q)$, which straightforwardly results in Lemma 5.3. Then, Lemma 5.4 gives alternative ways for expressing C_1 and C_3.

Definition 5.2. Let q be a path query, represented as a word over the alphabet of relation names. We define the language $\mathcal{L}^{\curvearrowright}(q)$ as the smallest set of words such that

(a) q belongs to $\mathcal{L}^{\curvearrowright}(q)$; and
(b) *Rewinding:* if $uRvRw$ is in $\mathcal{L}^{\curvearrowright}(q)$ for some relation name R and (possibly empty) words u, v and w, then $uRvRvRw$ is also in $\mathcal{L}^{\curvearrowright}(q)$. □

That is, $\mathcal{L}^{\curvearrowright}(q)$ is the smallest language that contains q and is closed under rewinding.

Lemma 5.3. *For every path query q, the automaton $\text{NFA}(q)$ accepts the language $\mathcal{L}^{\curvearrowright}(q)$.*

Lemma 5.4. *Let q be a path query. Then,*

(1) q satisfies C_1 if and only if q is a prefix of each $p \in \mathcal{L}^{\curvearrowright}(q)$;
(2) q satisfies C_3 if and only if q is a factor of each $p \in \mathcal{L}^{\curvearrowright}(q)$.

Proof. $\boxed{\Longleftarrow \text{ in (1) and (2)}}$ This direction is trivial, because whenever $q = uRvRw$, we have that $uRvRvRw \in \mathcal{L}^{\curvearrowright}(q)$.

We now show the $\Longrightarrow$ direction in both items. To this end, we call an application of the rule (b) in Definition 5.2 a *rewind*. By construction, each word in $\mathcal{L}^{\curvearrowright}(q)$ can be obtained from q by using k rewinds, for some nonnegative integer k. Let q_k be a word in $\mathcal{L}^{\curvearrowright}(q)$ that can be obtained from q by using k rewinds.

$\boxed{\Longrightarrow \text{ in (1)}}$ We use induction on k to show that q is a prefix of q_k. For he induction basis, $k = 0$, we have that q is a prefix of $q_0 = q$. We next show the induction step $k \to k+1$. Let $q_{k+1} = uRvRvRw$ where $q_k = uRvRw$ is a word in $\mathcal{L}^{\curvearrowright}(q)$ obtained with k rewinds. By the induction hypothesis, we can assume a word s such that $q_k = q \cdot s$.

- If q is a prefix of $uRvR$, then $q_{k+1} = uRvRvRw$ trivially contains q as a prefix.

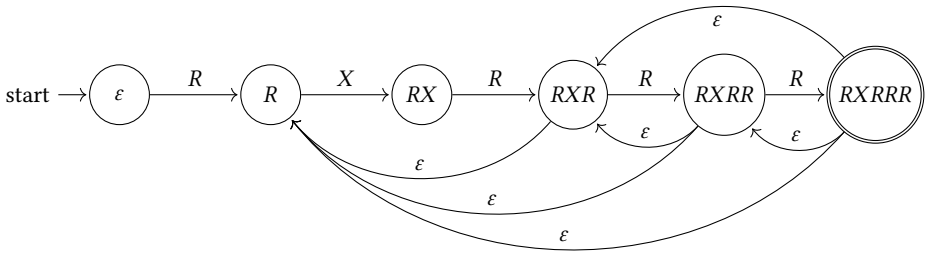


Figure 4: The NFA(q) automaton for the path query $q = RXRRR$.

- If $uRvR$ is a proper prefix of q, let $q = uRvRt$ where t is nonempty. Since q satisfies C_1, Rt is a prefix of Rv. Then $q_{k+1} = uRvRvRw$ contains $q = u \cdot Rv \cdot Rt$ as a prefix.

$\boxed{\Longrightarrow \text{ in (2)}}$ We use induction on k to show that q is a factor of q_k. For the induction basis, $k = 0$, we have that q is a prefix of $q_0 = q$. For the induction step, $k \to k+1$, let $q_{k+1} = uRvRvRw$ where $q_k = uRvRw$ is a word in $\mathcal{L}^{\rightsquigarrow}(q)$ obtained with k rewinds. By the induction hypothesis, $q_k = uRvRw$ contains q as a factor. If q is a factor of either $uRvR$ or $RvRw$, then $q_{k+1} = uRvRvRw$ contains q as a factor. Otherwise, we may decompose $q_k = u^-q^-RvRq^+w^+$ where $q = q^-RvRq^+$, $u = u^-q^-$ and $w = q^+w^+$. Since q satisfies C_3, the word $q^-RvRvRq^+$, which is a factor of q_{k+1}, contains q as a factor. □

In the technical treatment, it will be convenient to consider the automaton obtained from NFA(q) by changing its start state, as defined next.

Definition 5.5. If u is a prefix of q (and thus u is a state in NFA(q)), then S-NFA(q, u) is the automaton obtained from NFA(q) by letting the initial state be u instead of the empty word. Note that S-NFA(q, ε) = NFA(q). It may be helpful to think of the first S in S-NFA(q, u) as "Start at u." □

5.2 Reification Lemma

In this subsection, we first define how an automaton executes on a database instance. We then state an helping lemma which will be used in the proof of Lemma 5.9, which constitutes the main result of Section 5. To improve the readability and logical flow of our presentation, we postpone the proof of the helping lemma to Section 5.3.

Definition 5.6 (Automata on database instances). Let **db** be a database instance. A *path (in* **db***)* is defined as a sequence $R_1(\underline{c_1}, c_2)$, $R_2(\underline{c_2}, c_3)$, …, $R_n(\underline{c_n}, c_{n+1})$ of facts in **db**. Such a path is said to *start in* c_1. We call $R_1R_2 \cdots R_n$ the *trace* of this path. A path is said to be *accepted* by an automaton if its trace is accepted by the automaton.

Let q be a path query and $\mathbf{r}$ be a consistent database instance. We define start($q, \mathbf{r}$) as the set containing all (and only) constants $c \in$ adom($\mathbf{r}$) such that there is a path in $\mathbf{r}$ that starts in c and is accepted by NFA(q). □

Example 5.7. Consider the query $q_2 = RRX$ and the database instance of Figure 2. Let $\mathbf{r}_1$ and $\mathbf{r}_2$ be the repairs containing, respectively, $R(\underline{1}, 2)$ and $R(\underline{1}, 3)$. The only path with trace RRX in $\mathbf{r}_1$ starts in 1; and the only path with trace RRX in $\mathbf{r}_2$ starts in 0. The regular expression for $\mathcal{L}^{\rightsquigarrow}(q)$ is $RR(R)^*X$. We have start($q, \mathbf{r}_1$) = {0, 1} and start($q, \mathbf{r}_2$) = {0}. □

The following lemma tells us that, among all repairs, there is one that is inclusion-minimal with respect to start($q, \cdot$). In the preceding example, the repair $\mathbf{r}_2$ minimizes start($q, \cdot$).

Lemma 5.8. *Let q be a path query, and* **db** *a database instance. There exists a repair* $\mathbf{r}^*$ *of* **db** *such that for every repair* $\mathbf{r}$ *of* **db**, start($q, \mathbf{r}^*$) ⊆ start($q, \mathbf{r}$).

Informally, we think of the next Lemma 5.9 as a *reification lemma*. The notion of *reifiable variable* was coined in [34, Definition 8.5], to refer to a variable x in a query $\exists x\,(\varphi(x))$ such that whenever that query is true in every repair of a database instance, then there is a constant c such that $\varphi(c)$ is true in every repair. The following lemma captures a very similar concept.

Lemma 5.9 (Reification Lemma for C_3). *Let q be a path query that satisfies C_3. Then, for every database instance* **db**, *the following are equivalent:*

(1) **db** *is a "yes"-instance of* CERTAINTY(q)*; and*
(2) there exists a constant c (which depends on **db***) such that for every repair* $\mathbf{r}$ *of* **db**, $c \in$ start($q, \mathbf{r}$).

Proof. $\boxed{1 \Longrightarrow 2}$ Assume (1). By Lemma 5.8, there exists a repair $\mathbf{r}^*$ of **db** such that for every repair $\mathbf{r}$ of **db**, start($q, \mathbf{r}^*$) ⊆ start($q, \mathbf{r}$). Since $\mathbf{r}^*$ satisfies q, there is a path $R_1(\underline{c_1}, c_2)$, $R_2(\underline{c_2}, c_3)$, …, $R_n(\underline{c_n}, c_{n+1})$ in $\mathbf{r}^*$ such that $q = R_1R_2 \cdots R_n$. Since q is accepted by NFA(q), we have $c_1 \in$ start($q, \mathbf{r}^*$). It follows that $c_1 \in$ start($q, \mathbf{r}$) for every repair $\mathbf{r}$ of **db**.

$\boxed{2 \Longrightarrow 1}$ Let $\mathbf{r}$ be any repair of **db**. By our hypothesis that (2) holds true, there is some $c \in$ start($q, \mathbf{r}$). Therefore, there is a path in $\mathbf{r}$ that starts in c and is accepted by NFA(q). Let p be the trace of this path. By Lemma 5.3, $p \in \mathcal{L}^{\rightsquigarrow}(q)$. Since q satisfies C_3 by the hypothesis of the current lemma, it follows by Lemma 5.4 that q is a factor of p. Consequently, there is a path in $\mathbf{r}$ whose trace is q. It follows that $\mathbf{r}$ satisfies q. □

5.3 Proof of Lemma 5.8

We will use the following definition.

Definition 5.10 (States Set). This definition is relative to a path query q. Let $\mathbf{r}$ be a consistent database instance, and let f be an R-fact in $\mathbf{r}$, for some relation name R. The *states set* of f in $\mathbf{r}$, denoted $\mathsf{ST}_q(f, \mathbf{r})$, is defined as the smallest set of states satisfying the following property, for all prefixes u of q:

> if S-NFA(q, u) accepts a path in $\mathbf{r}$ that starts with f,
> then uR belongs to $\mathsf{ST}_q(f, \mathbf{r})$.

Note that if f is an R-fact, then all states in S-NFA$(q, \mathbf{r})$ have R as their last relation name. □

Example 5.11. Let $q = RRX$ and $\mathbf{r} = \{R(\underline{a}, b), R(\underline{b}, c), R(\underline{c}, d), X(\underline{d}, e), R(\underline{d}, e)\}$. Then NFA$(q)$ has states $\{\varepsilon, R, RR, RRX\}$ and accepts the regular language $RR(R)^*X$. Since S-NFA(q, ε) accepts the path $R(\underline{b}, c), R(\underline{c}, d), X(\underline{c}, d)$, the states set $\mathsf{ST}_q(R(\underline{b}, c), \mathbf{r})$ contains $\varepsilon \cdot R = R$. Since the latter path is also accepted by S-NFA(q, R), we also have $R \cdot R \in \mathsf{ST}_q(R(\underline{b}, c), \mathbf{r})$. Finally, note that $\mathsf{ST}_q(R(\underline{d}, e), \mathbf{r}) = \emptyset$, because there is no path that contains $R(\underline{d}, e)$ and is accepted by NFA(q). □

Lemma 5.12. *Let q be a path query, and $\mathbf{r}$ a consistent database instance. If $\mathsf{ST}_q(f, \mathbf{r})$ contains state uR, then it contains every state of the form vR with $|v| \geq |u|$.*

Proof. Assume $uR \in \mathsf{ST}_q(f, \mathbf{r})$. Then f is an R-fact and there is a path $f \cdot \pi$ in $\mathbf{r}$ that is accepted by S-NFA(q, u). Let vR be a state with $|v| > |u|$. Thus, by construction, NFA(q) has a backward transition with label ε from state vR to state uR.

It suffices to show that $f \cdot \pi$ is accepted by S-NFA(q, v). Starting in state v, S-NFA(q, v) traverses f (reaching state vR) and then uses the backward transition (with label ε) to reach the state uR. From there on, S-NFA(q, v) behaves like S-NFA(q, u). □

From Lemma 5.12, it follows that $\mathsf{ST}_q(f, \mathbf{r})$ is completely determined by the shortest word in it.

Definition 5.13 (Preorder $\preceq_q$ on repairs). Let $\mathbf{db}$ be a database instance. For all repairs $\mathbf{r}, \mathbf{s}$ of $\mathbf{db}$, we define $\mathbf{r} \preceq_q \mathbf{s}$ if for every $f \in \mathbf{r}$ and $g \in \mathbf{s}$ such that f and g are key-equal, we have $\mathsf{ST}_q(f, \mathbf{r}) \subseteq \mathsf{ST}_q(g, \mathbf{s})$.

Clearly, $\preceq_q$ is a reflexive and transitive binary relation on the set of repairs of $\mathbf{db}$. We write $\mathbf{r} \prec_q \mathbf{s}$ if both $\mathbf{r} \preceq_q \mathbf{s}$ and for some $f \in \mathbf{r}$ and $g \in \mathbf{s}$ such that f and g are key-equal, $\mathsf{ST}_q(f, \mathbf{r}) \subsetneq \mathsf{ST}_q(g, \mathbf{s})$. □

Lemma 5.14. *Let q be a path query. For every database instance $\mathbf{db}$, there is a repair $\mathbf{r}^*$ of $\mathbf{db}$ such that for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathbf{r}^* \preceq_q \mathbf{r}$.*

Proof. It suffices to show that whenever $\mathbf{r}$ and $\mathbf{s}$ are repairs of $\mathbf{db}$ such that $\mathbf{r} \not\preceq_q \mathbf{s}$, there exists a repair $\mathbf{r}^*$ of $\mathbf{db}$ such that $\mathbf{r}^* \prec_q \mathbf{r}$.

Let $\alpha : \mathbf{r} \to \mathbf{s}$ be the unique bijection such that for every $f \in \mathbf{r}$, the facts f and $\alpha(f)$ are key-equal. Note that α is the identity on facts in $\mathbf{r} \cap \mathbf{s}$. Define

$$\mathbf{r}^{\Delta} = \{f \in \mathbf{r} \mid \mathsf{ST}_q(\alpha(f), \mathbf{s}) \subsetneq \mathsf{ST}_q(f, \mathbf{r})\};$$
$$\mathbf{r}^{*} = (\mathbf{r} \setminus \mathbf{r}^{\Delta}) \cup \{\alpha(f) \mid f \in \mathbf{r}^{\Delta}\}.$$

From Lemma 5.12 and $\mathbf{r} \not\preceq_q \mathbf{s}$, it follows that $\mathbf{r}^{\Delta} \neq \emptyset$. We show $\mathbf{r}^* \preceq_q \mathbf{r}$. To this end, let $h \in \mathbf{r}^*$ and $uR \in \mathsf{ST}_q(h, \mathbf{r}^*)$. Thus, h is an R-fact, and we need to show $uR \in \mathsf{ST}_q(h', \mathbf{r})$, where h' is the fact in $\mathbf{r}$ that is key-equal to h (possibly $h' = h$). There is a path π in $\mathbf{r}^*$ that starts with h and is accepted by S-NFA(q, u). If π uses only facts in $\mathbf{r}$, then the desired result obtains vacuously. Otherwise let g be the leftmost fact in π such that $g \notin \mathbf{r}$ (possibly $g = h$). Thus, $g \in \mathbf{s}$. Let $f = \alpha^{-1}(g)$. So there are (possibly empty) paths π_1, π_2 such that $\pi = \pi_1 \cdot g \cdot \pi_2$. Since $g \in \mathbf{r}^* \setminus \mathbf{r}$, it follows $f \in \mathbf{r}^{\Delta}$, hence $\mathsf{ST}_q(g, \mathbf{s}) \subsetneq \mathsf{ST}_q(f, \mathbf{r})$. It follows that the path $\pi_1 \cdot f$ can be right extended in a path in $\mathbf{r}$ that is accepted by S-NFA(q, u). Consequently, $uR \in \mathsf{ST}_q(h', \mathbf{r})$.

Finally, we show that $\mathbf{r}^* \prec_q \mathbf{r}$. To this end, take any f in $\mathbf{r}^{\Delta}$. Since $\mathsf{ST}_q(\alpha(f), \mathbf{s}) \subsetneq \mathsf{ST}_q(f, \mathbf{r})$, we can assume $uR \in \mathsf{ST}_q(f, \mathbf{r})$ such that $uR \notin \mathsf{ST}_q(\alpha(f), \mathbf{s})$. Assume for the sake of contradiction that $uR \in \mathsf{ST}_q(\alpha(f), \mathbf{r}^*)$. Then, there exists a path π in $\mathbf{r}^*$ that starts with $\alpha(f)$ and is accepted by S-NFA(q, u). Since $uR \notin \mathsf{ST}_q(\alpha(f), \mathbf{s})$, the path π must use a fact in $\mathbf{r} \setminus \mathbf{s}$. We can assume $\pi = \pi_1 \cdot h \cdot \pi_2$ such that h is the leftmost fact in π belonging to $\mathbf{r} \setminus \mathbf{s}$. From $h \in \mathbf{r}^*$ and $h \notin \mathbf{s}$, it follows $h \notin \mathbf{r}^{\Delta}$. Since $h \notin \mathbf{r}^{\Delta}$, we have $\mathsf{ST}_q(h, \mathbf{r}) \subseteq \mathsf{ST}_q(\alpha(h), \mathbf{s})$. But this entails that $\pi_1 \cdot \alpha(h)$ can be right extended to a path in $\mathbf{s}$ that is accepted by S-NFA(q, u), hence $uR \in \mathsf{ST}_q(\alpha(f), \mathbf{s})$, a contradiction. □

We can now give the proof of Lemma 5.8.

Proof of Lemma 5.8. Let $\mathbf{db}$ be a database instance. Then by Lemma 5.14, there is a repair $\mathbf{r}^*$ of $\mathbf{db}$ such that for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathbf{r}^* \preceq_q \mathbf{r}$. It suffices to show that for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$. To this end, consider any repair $\mathbf{r}$ and $c \in \mathsf{start}(q, \mathbf{r}^*)$. Let R be the first relation name of q. Since $c \in \mathsf{start}(q, \mathbf{r}^*)$, there is $d \in \mathsf{adom}(\mathbf{r}^*)$ such that $R \in \mathsf{ST}_q(R(\underline{c}, d), \mathbf{r}^*)$. Then, there is a unique $d' \in \mathsf{adom}(\mathbf{r})$ such that $R(\underline{c}, d') \in \mathbf{r}$, where it is possible that $d' = d$. From $\mathbf{r}^* \preceq_q \mathbf{r}$, it follows $\mathsf{ST}_q(R(\underline{c}, d), \mathbf{r}^*) \subseteq \mathsf{ST}_q(R(\underline{c}, d'), \mathbf{r})$. Consequently, $R \in \mathsf{ST}_q(R(\underline{c}, d'), \mathbf{r})$, which implies $c \in \mathsf{start}(q, \mathbf{r})$. This conclude the proof. □

6 COMPLEXITY UPPER BOUNDS

We now show the complexity upper bounds of Theorem 3.2.

6.1 A PTIME Algorithm for C_3

We now specify a polynomial-time algorithm for CERTAINTY(q), for path queries q that satisfy condition C_3. The algorithm is based on the automata defined in Definition 5.5, and uses the concept defined next.

Definition 6.1 (Relation $\vdash_q$). Let q be a path query and $\mathbf{db}$ a database instance. For every $c \in \mathsf{adom}(q)$ and every prefix u of q, we write $\mathbf{db} \vdash_q \langle c, u \rangle$ if every repair of $\mathbf{db}$ has a path that starts in c and is accepted by S-NFA(q, u). □

An algorithm that decides the relation $\vdash_q$ can be used to solve CERTAINTY(q) for path queries satisfying C_3. Indeed, by Lemma 5.9, for path queries satisfying C_3, $\mathbf{db}$ is a "yes"-instance for the problem CERTAINTY(q) if and only if there is a constant $c \in \mathsf{adom}(\mathbf{db})$ such that $\mathbf{db} \vdash_q \langle c, u \rangle$ with $u = \varepsilon$.

Figure 5 shows an algorithm that computes $\{\langle c, u \rangle \mid \mathbf{db} \vdash_q \langle c, u \rangle\}$ as the fixed point of a binary relation N. The *Initialization Step* inserts into N all pairs $\langle c, q \rangle$, which is correct because $\mathbf{db} \vdash_q \langle c, q \rangle$ holds vacuously, as q is the accepting state of S-NFA(q, q). Then, the *Iterative Rule* is executed until N remains unchanged. This rule makes sure that, whenever a pair $\langle c, vS \rangle$ is added, all pairs $\langle c, wS \rangle$ with $|w| > |v|$ are also added. This is correct because $\mathbf{db} \vdash_q \langle c, vS \rangle$ and $|w| > |v|$ implies $\mathbf{db} \vdash_q \langle c, wS \rangle$, as S-NFA$(q, wS)$ has an ε-transition from state wS to uS. Figure 6 shows an example run of

Initialization Step: $N \leftarrow \{\langle c, q\rangle \mid c \in \text{adom}(\mathbf{db})\}$.

Iterative Rule: **if** uR is a prefix of q, and
$R(\underline{c}, *)$ is a nonempty block in **db** s.t. for every $R(\underline{c}, y) \in \mathbf{db}$, $\langle y, uR\rangle \in N$
then
$$N \leftarrow N \cup \underbrace{\{\langle c, u\rangle\}}_{\text{forward}} \cup \underbrace{\{\langle c, w\rangle \mid \text{NFA}(q) \text{ has a backward transition from } w \text{ to } u\}}_{\text{backward}}.$$

Figure 5: Polynomial-time algorithm for computing $\{\langle c, u\rangle \mid \mathbf{db} \vdash_q \langle c, u\rangle\}$, for a fixed path query q satisfying C_3.

Iteration	Tuples added to N
init.	<0, RRX>, <1, RRX>, <2, RRX>, <3, RRX>, <4, RRX>, <5, RRX>
1	<4, RR>
2	<3, R>, <3, RR>
3	<2, R>, <2, RR>
4	<1, R>, <1, RR>
5	<0, R>, <0, RR>, <0, ε>

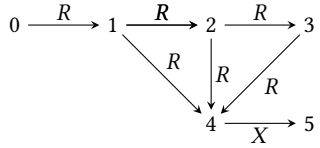


Figure 6: Example run of our algorithm for $q = RRX$, on the database instance db shown at the right.

the algorithm in Figure 5. The next lemma states the correctness of the algorithm.

Lemma 6.2. *Let q be a path query. Let* **db** *be a database instance. Let N be the output relation returned by the algorithm in Figure 5 on input* **db**. *Then, for every $c \in \text{adom}(\mathbf{db})$ and every prefix u of q,*

$$\langle c, u\rangle \in N \text{ if and only if } \mathbf{db} \vdash_q \langle c, u\rangle.$$

Proof. $\boxed{\Longleftarrow}$ Proof by contraposition. Assume $\langle c, u\rangle \notin N$. The proof shows the construction of a repair **r** of **db** such that **r** has no path that starts in c and is accepted by S-NFA(q, u). Such a repair shows $\mathbf{db} \nvdash_q \langle c, u\rangle$.

We explain which fact of an arbitrary block $R(\underline{a}, *)$ of **db** will be inserted in **r**. Among all prefixes of q that end with R, let u_0R be the longest prefix such that $\langle a, u_0\rangle \notin N$. If such u_0R does not exist, then an arbitrarily picked fact of the block $R(\underline{a}, *)$ is inserted in **r**. Otherwise, the *Iterative Rule* in Figure 5 entails the existence of a fact $R(\underline{a}, b)$ such that $\langle b, u_0R\rangle \notin N$. Then, $R(\underline{a}, b)$ is inserted in **r**.

Assume for the sake of contradiction that there is a path π in **r** that starts in c and is accepted by S-NFA(q, u). Let $\pi := R_1(\underline{c_0}, c_1)$, $R_2(\underline{c_1}, c_2)$, …, $R_n(\underline{c_{n-1}}, c_n)$ where $c_0 = c$. Since $\langle c_0, u\rangle \notin N$ and $\langle c_n, q\rangle \in N$, there is a longest prefix u_0 of q, where $|u_0| \geq |u|$, and $i \in \{1, \ldots, n\}$ such that $\langle c_{i-1}, u_0\rangle \notin N$ and $\langle c_i, u_0R_i\rangle \in N$. From $\langle c_{i-1}, u_0\rangle \notin N$, it follows that **db** contains a fact $R_i(\underline{c_{i-1}}, d)$ such that $\langle d, u_0R_i\rangle \notin N$. Then $R_i(\underline{c_{i-1}}, c_i)$ would not be chosen in a repair, contradicting $R_i(\underline{c_{i-1}}, c_i) \in \mathbf{r}$.

$\boxed{\Longrightarrow}$ Assume that $\langle c, u\rangle \in N$, Let ℓ be the number of executions of the *Iterative Rule* that were used to insert $\langle c, u\rangle$ in N. We show $\mathbf{db} \vdash_q \langle c, u\rangle$ by induction on ℓ.

The basis of the induction, $\ell = 0$, holds because the *Initialization Step* is obviously correct. Indeed, since q is an accepting state of S-NFA(q, q), we have $\mathbf{db} \vdash_q \langle c, q\rangle$. For the inductive step, $\ell \to \ell + 1$, we distinguish two cases.

Case that $\langle c, u\rangle$ is added to N by the forward *part of the* Iterative Rule. That is, $\langle c, u\rangle$ is added because **db** has a block $\{R(\underline{c}, d_1), \ldots, R(\underline{c}, d_k)\}$ with $k \geq 1$ and for every $i \in \{1, \ldots, k\}$, we have that $\langle d_i, uR\rangle$ was added to N by a previous execution of the *Iterative Rule*. Let **r** be an arbitrary repair of **db**. Since every repair contains exactly one fact from each block, we can assume $i \in \{1, \ldots, k\}$ such that $R(\underline{c}, d_i) \in \mathbf{r}$. By the induction hypothesis, $\mathbf{db} \vdash_q \langle d_i, uR\rangle$ and thus **r** has a path that starts in d_i and is accepted by S-NFA(q, uR). Clearly, this path can be left extended with $R(\underline{c}, d_i)$, and this left extended path is accepted by S-NFA(q, u). Note incidentally that the path in **r** may already use $R(\underline{c}, d_i)$, in which case the path is cyclic. Since **r** is an arbitrary repair, it is correct to conclude $\mathbf{db} \vdash_q \langle c, u\rangle$.

Case that $\langle c, u\rangle$ is added to N by the backward *part of the* Iterative Rule. Then, there is a relation name S and words v, w, such that $u = vSwS$, and $\langle c, u\rangle$ is added because $\langle c, vS\rangle$ was added in the same iteration. Then, S-NFA(q, u) has an ε-transition from state u to vS. Let **r** be an arbitrary repair of **db**. By the reasoning in the previous case, **r** has a path that starts in c and is accepted by S-NFA(q, vS). We claim that **r** has a path that starts in c and is accepted by S-NFA(q, u). Indeed, S-NFA(q, u) can use the ε-transition to reach the state vS, and then behave like S-NFA(q, vS). This concludes the proof. □

The following corollary is now immediate.

Corollary 6.3. *Let q be a path query. Let* **db** *be a database instance, and $c \in \text{adom}(\mathbf{db})$. Then, the following are equivalent:*

(1) $c \in \text{start}(q, \mathbf{r})$ for every repair **r** *of* **db***; and*
(2) $\langle c, \epsilon\rangle \in N$, where N is the output of the algorithm in Figure 5.

Finally, we obtain the following tractability result.

Lemma 6.4. *For each path query q satisfying C_3,* CERTAINTY(q) *is expressible in Least Fixpoint Logic, and hence is in* **PTIME**.

Proof. For a path query q, define the following formula in LFP [27]:

$$\psi_q(s, t) := \left[\mathbf{lfp}_{N,x,z}\varphi_q(N, x, z)\right](s, t), \tag{1}$$

where $\varphi_q(N, x, z)$ is given in Figure 7. Herein, $\alpha(x)$ denotes a first-order query that computes the active domain. That is, for every database instance **db** and constant c, $\mathbf{db} \models \alpha(c)$ if and only if $c \in \text{adom}(\mathbf{db})$. Further, $u \leq v$ means that u is a prefix of v; and $u < v$ means that u is a proper prefix of v. Thus, $u < v$ if and

$$\varphi_q(N,x,z) := \left(\begin{array}{ll} & (\alpha(x) \wedge z = \text{`q'}) \\ \vee & \left(\bigvee_{\mathsf{uR} \leq \mathsf{q}} \left((z = \text{`u'}) \wedge \exists y R(\underline{x}, y) \wedge \forall y \left(R(\underline{x}, y) \rightarrow N(y, \text{`uR'})\right)\right)\right) \\ \vee & \left(\bigvee_{\substack{\varepsilon < \mathsf{u} < \mathsf{uv} \leq \mathsf{q} \\ \mathsf{last(u)}=\mathsf{last(v)}}} (N(x, \text{`u'}) \wedge z = \text{`uv'})\right) \end{array} \right)$$

Figure 7: Definition of $\varphi_q(N,x,z)$. The predicate $\alpha(x)$ states that x is in the active domain, and $<$ is shorthand for *"is a strict prefix of".*

only if $u \leq v$ and $u \neq v$. The formula $\varphi_q(N,x,z)$ is positive in N, which is a 2-ary predicate symbol. It is understood that the middle disjunction ranges over all nonempty prefixes uR of q (possibly $u = \varepsilon$). The last disjunction ranges over all pairs (u, uv) of distinct nonempty prefixes of q that agree on their last symbol. We used a different typesetting to distinguish the constant words q, uR, uv from first-order variables x, z. It is easily verified that the LFP query (1) expresses the algorithm of Figure 5. □

Since the formula (1) in the proof of Lemma 6.4 uses universal quantification, it is not in Existential Least Fixpoint Logic, which is equal to DATALOG$_\neg$ [27, See Theorem 10.18]. It can be shown that there exists a path query q such that CERTAINTY(q) is in **PTIME** but not expressible in stratified Datalog.

6.2 FO-Rewritability for C_1

We now show that if a path query q satisfies C_1, then CERTAINTY(q) is in **FO**, and a first-order rewriting for q can be effectively constructed.

Definition 6.5 (First-order rewriting). If q is a Boolean query such that CERTAINTY(q) is in **FO**, then a *(consistent) first-order rewriting* for q is a first-order sentence ψ such that for every database instance **db**, the following are equivalent:

(1) **db** is a "yes"-instance of CERTAINTY(q); and
(2) **db** satisfies ψ. □

Definition 6.6. If $q = \{R_1(\underline{x_1}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_k(\underline{x_k}, x_{k+1})\}$, $k \geq 1$, and c is a constant, then $q_{[c]}$ is the Boolean conjunctive query $q_{[c]} := \{R_1(\underline{c}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_k(\underline{x_k}, x_{k+1})\}$. □

Lemma 6.7. *For every nonempty path query q and constant c, the problem* CERTAINTY($q_{[c]}$) *is in* **FO**. *Moreover, it is possible to construct a first-order formula $\psi(x)$, with free variable x, such that for every constant c, the sentence $\exists x\,(\psi(x) \wedge x = c)$ is a first-order rewriting for $q_{[c]}$.*

Proof. The proof inductively constructs a first-order rewriting for $q_{[c]}$, where the induction is on the number n of atoms in q. For the basis of the induction, $n = 1$, we have $q_{[c]} = R(\underline{c}, y)$. Then, the first-order formula $\psi(x) = \exists y R(\underline{x}, y)$ obviously satisfies the statement of the lemma.

We next show the induction step, $n \rightarrow n+1$. Let $R(\underline{x_1}, x_2)$ be the left-most atom of q, and assume that $p := q \setminus \{R(\underline{x_1}, x_2)\}$ is a path query with $n \geq 1$ atoms. By the induction hypothesis, it is possible to construct a first-order formula $\varphi(z)$, with free variable z, such that for every constant d,

$$\exists z\,(\varphi(z) \wedge z = d) \text{ is a first-order rewriting for } p_{[d]}. \tag{2}$$

We now define $\psi(x)$ as follows:

$$\psi(x) = \exists y \left(R(\underline{x}, y)\right) \wedge \forall z \left(R(\underline{x}, z) \rightarrow \varphi(z)\right). \tag{3}$$

We will show that for every constant c, $\exists x\,(\psi(x) \wedge x = c)$ is a first-order rewriting for $q_{[c]}$. To this end, let **db** be a database instance. It remains to be shown that **db** is a "yes"-instance of CERTAINTY($q_{[c]}$) if and only if **db** satisfies $\exists x\,(\psi(x) \wedge x = c)$.

$\boxed{\Longleftarrow}$ Assume **db** satisfies $\exists x\,(\psi(x) \wedge x = c)$. Because of the conjunct $\exists y \left(R(\underline{x}, y)\right)$ in (3), we have that **db** includes a block $R(\underline{c}, *)$. Let **r** be a repair of **db**. We need to show that **r** satisfies $q_{[c]}$. Clearly, **r** contains $R(\underline{c}, d)$ for some constant d. Since **db** satisfies $\exists z\,(\varphi(z) \wedge z = d)$, the induction hypothesis (2) tells us that **r** satisfies $p_{[d]}$. It is then obvious that **r** satisfies $q_{[c]}$.

$\boxed{\Longrightarrow}$ Assume **db** is a "yes"-instance for CERTAINTY($q_{[c]}$). Then **db** must obviously satisfy $\exists y \left(R(\underline{c}, y)\right)$. Therefore, **db** includes a block $R(\underline{c}, *)$. Let **r** be an arbitrary repair of **db**. There exists d such that $R(\underline{c}, d) \in$ **r**. Since **r** satisfies $q_{[c]}$, it follows that **r** satisfies $p_{[d]}$. Since **r** is an arbitrary repair, the induction hypothesis (2) tells us that **db** satisfies $\exists z\,(\varphi(z) \wedge z = d)$. It is then clear that **db** satisfies $\exists x\,(\psi(x) \wedge x = c)$. □

Lemma 6.8. *For every path query q that satisfies C_1, the problem* CERTAINTY(q) *is in* **FO**, *and a first-order rewriting for q can be effectively constructed.*

Proof. By Lemmas 5.4 and 5.9, a database instance **db** is a "yes"-instance for CERTAINTY(q) if and only if there is a constant c (which depends on **db**) such that **db** is a "yes"-instance for CERTAINTY($q_{[c]}$). By Lemma 6.7, it is possible to construct a first-order rewriting $\exists x\,(\psi(x) \wedge x = c)$ for $q_{[c]}$. It is then clear that $\exists x\,(\psi(x))$ is a first-order rewriting for q. □

6.3 An NL Algorithm for C_2

We show that CERTAINTY(q) is in **NL** if q satisfies C_2. The proof will use the syntactic characterization of C_2 established in Lemma 4.5.

Lemma 6.9. *For every path query q that satisfies C_2, the problem* CERTAINTY(q) *is in* **NL**.

In the remainder of this section, we develop the proof of Lemma 6.9.

Definition 6.10. Let q be a path query. We define $\mathsf{NFA}^{\min}(q)$ as the automaton that accepts w if w is accepted by NFA(q) and no proper prefix of w is accepted by NFA(q). □

It is well-known that such an automaton $\mathsf{NFA}^{\min}(q)$ exists.

Example 6.11. Let $q = RXRYR$. Then, $RXRYRYR$ is accepted by NFA(q), but not by $\mathsf{NFA}^{\min}(q)$, because the proper prefix $RXRYR$ is also accepted by NFA(q). □

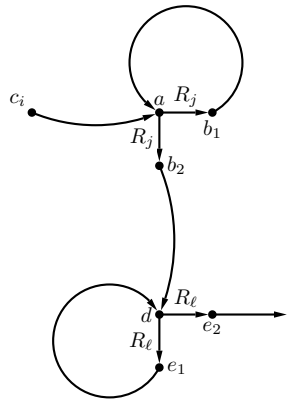


Figure 8: A repair r such that $c_i \notin \mathsf{start}(q, \mathbf{r})$ can be constructed by choosing $R_j(\underline{a}, b_2)$ and $R_\ell(\underline{d}, e_1)$. No short-cutting should occur during the $s\,(uv)^{k-1}$ part, because condition (6) in the proof of Lemma 6.9 may not hold there. If the certificate ended because its length is $|\mathbf{db}| + |q| + 1$, then the last cycle must be maintained.

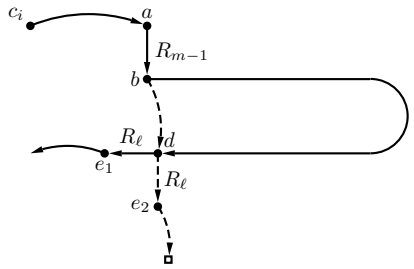


Figure 9: A repair r such that $c_i \notin \mathsf{start}(q, \mathbf{r})$ can be constructed by choosing $R_\ell(\underline{d}, e_2)$. The dashed path falsifies the query $\exists x\,(p(x) \wedge x = b)$, because it cannot be continued in the square node.

Definition 6.12. Let q be a path query and $\mathbf{r}$ be a consistent database instance. We define $\mathsf{start}^{\min}(q, \mathbf{r})$ as the set containing all (and only) constants $c \in \mathsf{adom}(\mathbf{r})$ such that there is a path in $\mathbf{r}$ that starts in c and is accepted by $\mathsf{NFA}^{\min}(q)$. □

Lemma 6.13. *Let q be a path query. For every consistent database instance $\mathbf{r}$, we have that $\mathsf{start}(q, \mathbf{r}) = \mathsf{start}^{\min}(q, \mathbf{r})$.*

Proof. By construction, $\mathsf{start}^{\min}(q, \mathbf{r}) \subseteq \mathsf{start}(q, \mathbf{r})$. Next assume that $c \in \mathsf{start}(q, \mathbf{r})$ and let π be the path that starts in c and is accepted by $\mathsf{NFA}(q)$. Let π^- be the shortest prefix of π that is accepted by $\mathsf{NFA}(q)$. Since π^- starts in c and is accepted by $\mathsf{NFA}^{\min}(q)$, it follows $c \in \mathsf{start}^{\min}(q, \mathbf{r})$. □

Lemma 6.14. *Let $u \cdot v \cdot w$ be a self-join-free word over the alphabet of relation names. Let s be a suffix of uv that is distinct from uv. For every integer $k \geq 0$, $\mathsf{NFA}^{\min}(s\,(uv)^k\,wv)$ accepts the language of the regular expression $s\,(uv)^k\,(uv)^*\,wv$.*

Proof. Let $q = s\,(uv)^k\,wv$. Since $u \cdot v \cdot w$ is self-join-free, applying the rewinding operation, zero, one, or more times, in the part of q that precedes w will repeat the factor uv. This gives words of the form $s\,(uv)^\ell\,wv$ with $\ell \geq k$. The difficult case is where we rewind a factor of q that itself contains w as a factor. In this case, the rewinding operation will repeat a factor of the form $v_2\,(uv)^\ell\,wv_1$ such that $v = v_1v_2$ and $v_2 \neq \varepsilon$, which results in words of one of the following forms ($s = s_1 \cdot v_2$):

$$\Big(s\,(uv)^{\ell_1}\,uv_1\Big)\cdot v_2\,(uv)^{\ell_2}\,wv_1\cdot\underline{v_2\,(uv)^{\ell_2}\,wv_1}\cdot(v_2);\ \text{or}$$
$$(s_1)\cdot v_2\,(uv)^{\ell}\,wv_1\cdot\underline{v_2\,(uv)^{\ell}\,wv_1}\cdot(v_2).$$

These words have a prefix belonging to the language of the regular expression $s\,(uv)^k\,(uv)^*\,wv$. □

We can now give the proof of Lemma 6.9.

Proof of Lemma 6.9. Assume q satisfies C_2. By Lemma 4.5, q satisfies $\mathcal{B}_{2a}$ or $\mathcal{B}_{2b}$. We treat the case that q satisfies $\mathcal{B}_{2b}$ (the case that q satisfies $\mathcal{B}_{2a}$ is even easier). We have that q is a factor of $(uv)^k\,wv$, where k is chosen as small as possible, and uvw is self-join-free. The proof is straightforward if $k = 0$; we assume $k \geq 1$ from here on. To simplify notation, we will show the case where q is a suffix of $(uv)^k\,wv$; our proof can be easily extended to the case where q is not a suffix, at the price of some extra notation. There is a suffix s of uv such that $q = s\,(uv)^{k-1}\,wv$. By Lemmas 5.9, 6.13, and 6.14, the following are equivalent:

(a) **db** is a "no"-instance of CERTAINTY(q); and
(b) for every constant $c_i \in \mathsf{adom}(q)$, there is a repair $\mathbf{r}$ of **db** such that there is no path in $\mathbf{r}$ that starts in c_i and whose trace is in the language of the regular expression $s\,(uv)^{k-1}\,(uv)^*\,wv$.

From here on, let $m := |uv|$ and let $R_0, \ldots, R_{m-1}$ be relation names such that

$$R_0R_1 \ldots R_{m-1} = uv. \tag{4}$$

We will say that $R_{i+1 \bmod m}$ is the symbol that immediately follows $R_{i \bmod m}$ (for $0 \leq i \leq m-1$). Thus, we view (4) as a cyclic list. In particular, R_0 immediately follows R_{m-1}.

We use the certificate definition of **NL**. Recall that a read-once logspace verifier has three tapes. A read-once certificate tape, a read-only input tape, and a logspace-bounded working tape. The database instance **db** is on the read-only input tape. The certificate will show that for all constants $c_1, c_2, \ldots, c_n$, listed in the order in which they occur in **db**, there is a repair satisfying the previous item (b). Note that a same constant can occur multiple times. This will prove the lemma, since **NL** is closed under complement. The sub-certificate for each c_i will be a finite sequence of facts in **db**, of the form:

$$S_1(\underline{b_1}, b_2), S_2(\underline{b_2}, b_3), S_3(\underline{b_3}, b_4), \ldots \tag{5}$$

such that

(1) $b_1 = c_i$;
(2) S_1 is the first symbol of s if $s \neq \varepsilon$; otherwise $S_1 = R_0$; and
(3) for every $j \geq 2$, we have that S_j immediately follows S_{j-1} in the cyclic list (4).

E.g., for $uv = ABC$ and $s = BC$, the sequence $S_1S_2S_3S_4S_5S_6S_7S_8$ reads $BCABCABC$. Let $p(x)$ be the query obtained from the path

query wv by treating the leftmost variable, called x, as a free variable. For example, if $wv = V_1V_2V_3$, then

$$p(x) = \exists y \exists z \exists t(V_1(\underline{x}, y) \wedge V_2(\underline{y}, z) \wedge V_3(\underline{z}, t)).$$

The expected sub-certificate must satisfy the following constraint:

> for every $i \geq |s\,(uv)^{k-1}|$, if the sequence (5) contains $S_i(\underline{b_i}, b_{i+1})$ with $S_i = R_{m-1}$, then **db** is a "no"-instance of the problem CERTAINTY($\exists x\,(p(x) \wedge x = b_{i+1})$). Note that the latter problem is in **FO**. (6)

Continuing our previous example, if $uv = ABC$, $S = BC$, and $k = 2$, then $|s\,(uv)^{k-1}| = 5$. Then, we distinguish the C-facts $S_5 = C(\underline{b_5}, b_6)$, $S_8 = C(\underline{b_8}, b_9)$, $S_{11} = C(\underline{b_{11}}, b_{12})$,..., and check that for $b_i \in \{b_6, b_9, b_{12}, \dots\}$, **db** is a "no"-instance of

$$\text{CERTAINTY}(\exists x\,(p(x) \wedge x = b_i)).$$

Finally, every sub-certificate of the form (5) ends when it reaches S_{N+1} with $N = |\mathbf{db}| + |q|$, which means that two database facts must occur twice in the sub-certificate. A sub-certificate can be shorter if it reaches $S_\ell(\underline{b_\ell}, b_{\ell+1})$, but there is no R_i-fact $R_i(\underline{b_{\ell+1}}, \_)$ $(0 \leq i \leq m-1)$ such that R_i immediately follows S_ℓ in the cyclic list (4).

We claim that the following are equivalent:

(1) a sub-certificate of the form (5) with $b_1 = c_i$ exists; and
(2) there is a repair $\mathbf{r}$ of **db** such that $c_i \notin \text{start}(q, \mathbf{r})$.

The implication $2 \implies 1$ is straightforward. We next show $1 \implies 2$. To this end, assume that a sub-certificate of the form (5) with $b_1 = c_i$ exists. We argue that such a sub-certificate can be turned into a repair $\mathbf{r}$ of **db** such that $c_i \notin \text{start}(q, \mathbf{r})$. Notice that the sub-certificate may be inconsistent, because it can first contain $R_\ell(\underline{d}, e_1)$ and then $R_\ell(\underline{d}, e_2)$ (possibly $e_1 = e_2$), implying a cycle. A logspace verifier cannot detect such distinct, key-equal facts. Nonetheless, $\mathbf{r}$ can be made consistent by short-cutting cycles, which boils down to establishing a shorter sub-certificate for $c_i \notin \text{start}(q, \mathbf{r})$. This is illustrated by Figure 8. In addition to the removal of distinct, key-equal facts, it is essential that any sequence of the form (5) in $\mathbf{r}$ satisfies (6), that is, for every fact $R_{m-1}(\underline{b_i}, b_{i+1})$ referred to in (6), it must be the case that $\mathbf{r} \not\models \exists x\,(p(x) \wedge x = b_{i+1})$. We argue that this is possible. To this end, let v, which is a suffix of (4), be $v = R_o R_{o+1} \dots R_{m-1}$ $(0 \leq o \leq m-1)$ and assume that **db** has a sequence $\mathbf{s} := R_o(\underline{d_o}, d_{o+1}), R_{o+1}(\underline{d_{o+1}}, d_{o+2}), \dots, R_j(\underline{d_j}, d_{j+1})$, with $j < m-1$, such that **db** contains no fact $R_{j+1}(\underline{d_{j+1}}, \_)$. Such an "incompletable" sequence $\mathbf{s}$ can be used in (5). Informally, R_i-facts $(o \leq i \leq j)$ that are needed to establish $\mathbf{r} \not\models \exists x\,(p(x) \wedge x = b_{i+1})$, can also be used in sub-certificates. This is illustrated by Figure 9.

To conclude the proof, it is easily verified that our certificates can be checked by a logspace verifier. The memory needs include an index ranging over the constants $c_1, c_2, \dots, c_n$ that form the database instance **db**, and a counter that can count up to $|\mathbf{db}| + |q| + 1$. Furthermore, the logspace verifier has to issue some queries in **FO**. □

7 COMPLEXITY LOWER BOUNDS

In this section, we show the complexity lower bounds of Theorem 3.2. For a path query $q = \{R_1(x_1, x_2), \dots, R_k(x_k, x_{k+1})\}$ and constants a, b, we define the following database instances:

$$\begin{aligned}
\phi_a^b[q] &:= \{R_1(a, \square_2), R_2(\square_2, \square_3), \dots, R_k(\square_k, b)\} \\
\phi_a^\perp[q] &:= \{R_1(a, \square_2), R_2(\square_2, \square_3), \dots, R_k(\square_k, \square_{k+1})\} \\
\phi_\perp^b[q] &:= \{R_1(\square_1, \square_2), R_2(\square_2, \square_3), \dots, R_k(\square_k, b)\}
\end{aligned}$$

where the symbols $\square_i$ denoted fresh constants not occurring elsewhere. Significantly, two occurrences of $\square_i$ will represent different constants.

7.1 NL-Hardness

We first show that if a path query violates C_1, then CERTAINTY(q) is **NL**-hard, and therefore not in **FO**.

Lemma 7.1. *If a path query q violates C_1, then* CERTAINTY(q) *is* **NL***-hard.*

Proof. Assume that q does not satisfy C_1. Then, there exists a relation name R such that $q = uRvRw$ and q is not a prefix of $uRvRvRw$. It follows that Rw is not a prefix of $RvRw$. Since $Rv \neq \varepsilon$, there exists no (conjunctive query) homomorphism from q to uRw.

The problem REACHABILITY takes as input a directed graph $G(V, E)$ and two vertices $s, t \in V$, and asks whether G has a directed path from s to t. This problem is **NL**-complete and remains **NL**-complete when the inputs are acyclic graphs. Recall that **NL** is closed under complement. We present a first-order reduction from REACHABILITY to the complement of CERTAINTY(q), for acyclic directed graphs.

Let $G = (V, E)$ be an acyclic directed graph and $s, t \in V$. Let $G' = (V \cup \{s', t'\}, E \cup \{(s', s), (t, t')\})$, where s', t' are fresh vertices. We construct an input instance **db** for CERTAINTY(q) as follows:

- for each vertex $x \in V \cup \{s'\}$, we add $\phi_\perp^x[u]$;
- for each edge $(x, y) \in E \cup \{(s', s), (t, t')\}$, we add $\phi_x^y[Rv]$; and
- for each vertex $x \in V$, we add $\phi_x^\perp[Rw]$.

This construction can be executed in **FO**. Figure 10 shows an example of the above construction. Observe that the only conflicts in **db** occur in R-facts outgoing from a same vertex.

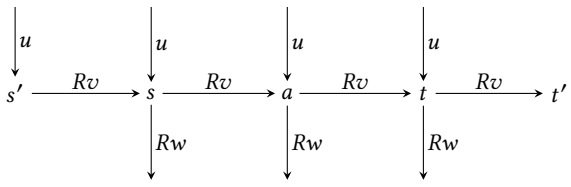


Figure 10: Database instance for the NL-hardness reduction from the graph G with $V = \{s, a, t\}$ and $E = \{(s, a), (a, t)\}$.

We now show that there exists a directed path from s to t in G if and only if there exists a repair of **db** that does not satisfy q.

$\boxed{\implies}$ Suppose that there is a directed path from s to t in G. Then, G' has a directed path $P = s, x_0, x_1, \dots, t, t'$. Then, consider the repair $\mathbf{r}$ that chooses the first R-fact from $\phi_x^y[Rv]$ for each edge (x, y) on the path P, and the first R-fact from $\phi_y^\perp[Rw]$ for each y not on the path P. We show that $\mathbf{r}$ falsifies q. Assume for the sake of contradiction that $\mathbf{r}$ satisfies q. Then, there exists a valuation θ for the variables in q such that $\theta(q) \subseteq \mathbf{r}$. Since, as argued in the

beginning of this proof, there exists no (conjunctive query) homomorphism from q to uRw, it must be that all facts in $\theta(q)$ belong to a path in $\mathbf{r}$ with trace $u\,(Rv)^k$, for some $k \geq 0$. Since, by construction, no constants are repeated on such paths, there exists a (conjunctive query) homomorphism from q to $u\,(Rv)^k$, which implies that Rw is a prefix of $RvRw$, a contradiction. We conclude by contradiction that $\mathbf{r}$ falsifies q.

$\boxed{\Longleftarrow}$ Proof by contradiction. Suppose that there is no directed path from s to t in G. Let $\mathbf{r}$ be any repair of **db**; we will show that $\mathbf{r}$ satisfies q. Indeed, there exists a maximal path $P = x_0, x_1, \ldots, x_n$ such that $x_0 = s'$, $x_1 = s$, and $\phi_{x_i}^{x_{i+1}}[Rv] \subseteq \mathbf{r}$. By construction, s' cannot reach t' in G', and thus $x_n \neq t'$. Since P is maximal, we must have $\phi_{x_n}^{\perp}[Rw] \subseteq \mathbf{r}$. Then $\phi_{\perp}^{x_{n-1}}[u] \cup \phi_{x_{n-1}}^{x_n}[Rv] \cup \phi_{x_n}^{\perp}[Rw]$ satisfies q. □

7.2 coNP-Hardness

Next, we show the **coNP**-hard lower bound.

Lemma 7.2. *If a path query q violates C_3, then* CERTAINTY(q) *is* **coNP***-hard.*

Proof. If q does not satisfy C_3, then there exists a relation R such that $q = uRvRw$ and q is not a factor of $uRvRvRw$. Note that this means that there is no homomorphism from q to $uRvRvRw$. Also, u must be nonempty (otherwise, $q = RvRw$ is trivially a suffix of $RvRvRw$). Let S be the first relation of u.

The proof is a first-order reduction from SAT to the complement of CERTAINTY(q). The problem SAT asks whether a given propositional formula in CNF has a satisfying truth assignment.

Given any formula ψ for SAT, we construct an input instance **db** for CERTAINTY(q) as follows:

- for each variable z, we add $\phi_z^{\perp}[Rw]$ and $\phi_z^{\perp}[RvRw]$;
- for each clause C and positive literal z of C, we add $\phi_C^z[u]$;
- for each clause C and variable z that occurs in a negative literal of C, we add $\phi_C^z[uRv]$.

This construction can be executed in **FO**. Figure 11 depicts an example of the above construction. Intuitively, $\phi_z^{\perp}[Rw]$ corresponds to setting the variable z to true, and $\phi_z^{\perp}[RvRw]$ to false. There are two types of conflicts that occur in **db**. First, we have conflicting facts of the form $S(\underline{C}, *)$; resolving this conflict corresponds to the clause C choosing one of its literals. Moreover, for each variable z, we have conflicting facts of the form $R(\underline{z}, *)$; resolving this conflict corresponds to the variable z choosing a truth assignment.

We show now that ψ has a satisfying truth assignment if and only if there exists a repair of **db** that does not satisfy q.

$\boxed{\Longrightarrow}$ Assume that there exists a satisfying truth assignment σ for ψ. Then for any clause C, there exists a variable $z_C \in C$ whose corresponding literal is true in C under σ. Consider the repair $\mathbf{r}$ that:

- for each variable z, it chooses the first R-fact of $\phi_z^{\perp}[Rw]$ if $\sigma(z)$ is true, otherwise the first R-fact of $\phi_z^{\perp}[RvRw]$;
- for each clause C, it chooses the first S-fact of $\phi_C^z[u]$ if z_C is positive in C, or the first S-fact of $\phi_C^z[uRv]$ if z_C is negative in C.

Assume for the sake of contradiction that $\mathbf{r}$ satisfies q. Then we must have a homomorphism from q to either uRw or $uRvRvRw$. But the former is not possible, while the latter contradicts C_3. We conclude by contradiction that $\mathbf{r}$ falsifies q.

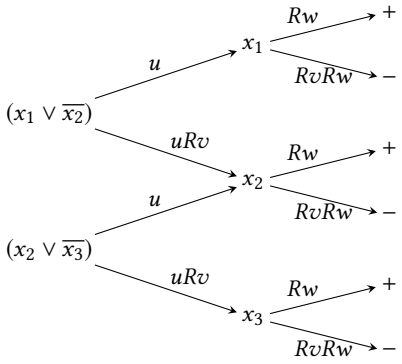


Figure 11: Database instance for the coNP-hardness reduction from the formula $\psi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3})$.

$\boxed{\Longleftarrow}$ Suppose that there exists a repair $\mathbf{r}$ of **db** that falsifies q. Consider the assignment σ:

$$\sigma(z) = \begin{cases} \text{true} & \text{if } \phi_z^{\perp}[Rw] \subseteq \mathbf{r} \\ \text{false} & \text{if } \phi_z^{\perp}[RvRw] \subseteq \mathbf{r} \end{cases}$$

We claim that σ is a satisfying truth assignment for ψ. Indeed, for each clause C, the repair must have chosen a variable z in C. If z appears as a positive literal in C, then $\phi_C^z[u] \subseteq \mathbf{r}$. Since $\mathbf{r}$ falsifies q, we must have $\phi_z^{\perp}[Rw] \subseteq \mathbf{r}$. Thus, $\sigma(z)$ is true and C is satisfied. If z appears in a negative literal, then $\phi_C^z[uRv] \subseteq \mathbf{r}$. Since $\mathbf{r}$ falsifies q, we must have $\phi_z^{\perp}[RvRw] \subseteq \mathbf{r}$. Thus, $\sigma(z)$ is false and C is again satisfied. □

7.3 PTIME-Hardness

Finally, we show the **PTIME**-hard lower bound.

Lemma 7.3. *If a path query q violates C_2, then* CERTAINTY(p) *is* **PTIME***-hard.*

Proof. Suppose q violates C_2. If q also violates C_3 , then the problem CERTAINTY(q) is **PTIME**-hard since it is **coNP**-hard by Lemma 7.2. Otherwise, it is possible to write $q = uRv_1Rv_2Rw$, with three consecutive occurrences of R such that $v_1 \neq v_2$ and Rw is not a prefix of Rv_1. Let v be the maximal path query such that $v_1 = vv_1^+$ and $v_2 = vv_2^+$. Thus $v_1^+ \neq v_2^+$ and the first relation names of v_1^+ and v_2^+ are different.

Our proof is a reduction from the Monotone Circuit Value Problem (MCVP) known to be **PTIME**-complete [13]:

Problem: MCVP

Input: A monotone Boolean circuit C on inputs $x_1, x_2, \ldots, x_n$ and output gate o; an assignment $\sigma : \{x_i \mid 1 \leq i \leq n\} \to \{0, 1\}$.

Question: What is the value of the output o under σ?

We construct an instance **db** for CERTAINTY(q) as follows:

- for the output gate o, we add $\phi_{\perp}^{o}[uRv_1]$;
- for each input variable x with $\sigma(x) = 1$, we add $\phi_x^{\perp}[Rv_2Rw]$;
- for each gate g, we add $\phi_{\perp}^{g}[u]$ and $\phi_g^{\perp}[Rv_2Rw]$;
- for each AND gate $g = g_1 \wedge g_2$, we add

$$\phi_g^{g_1}[Rv_1] \cup \phi_g^{g_2}[Rv_1].$$

Here, g_1 and g_2 can be gates or input variables; and

- for each OR gate $g = g_1 \vee g_2$, we add

$$\phi_g^{c_1}[Rv] \cup \phi_{c_1}^{g_1}[v_1^+] \cup \phi_{c_1}^{c_2}[v_2^+]$$
$$\cup\ \phi_\perp^{c_2}[u] \cup \phi_{c_2}^{g_2}[Rv_1] \cup \phi_{c_2}^{\perp}[Rw]$$

  where c_1, c_2 are fresh constants.

This construction can be executed in **FO**. An example of the gadget constructions is shown in Figure 12. We next show that the output gate o is evaluated to 1 under σ if and only if each repair of **db** satisfies q.

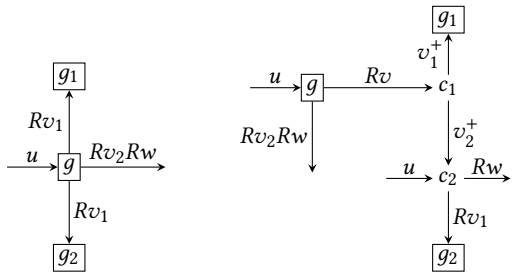


Figure 12: Gadgets for the PTIME-hardness reduction.

$\boxed{\Longrightarrow}$ Suppose the output gate o is evaluated to 1 under σ. Consider any repair $\mathbf{r}$. We construct a sequence of gates starting from o, with the invariant that every gate g evaluates to 1, and there is a path of the form uRv_1 in $\mathbf{r}$ that ends in g. The output gate o evaluates to 1, and also we have that $\phi_\perp^o[uRv_1] \subseteq \mathbf{r}$ by construction. Suppose that we are at gate g. If there is a Rv_2Rw path in $\mathbf{r}$ that starts in g, the sequence ends and the query q is satisfied. Otherwise, we distinguish two cases:

(1) $g = g_1 \wedge g_2$. Then, we choose the gate with $\phi_g^{g_i}[Rv_1] \subseteq \mathbf{r}$. Since both gates evaluate to 1 and $\phi_\perp^g[u] \subseteq \mathbf{r}$, the invariant holds for the chosen gate.
(2) $g = g_1 \vee g_2$. If g_1 evaluates to 1, we choose g_1. Observe that $\phi_\perp^g[u] \cup \phi_g^{c_1}[Rv] \cup \phi_{c_1}^{g_1}[v_1^+]$ creates the desired uRv_1 path. Otherwise g_2 evaluates to 1. If $\phi_{c_2}^{\perp}[Rw] \subseteq \mathbf{r}$, then there is a path with trace uRv_1 ending in g, and a path with trace Rv_2Rw starting in g, and therefore $\mathbf{r}$ satisfies q. If $\phi_{c_2}^{\perp}[Rw] \nsubseteq \mathbf{r}$, we choose g_2 and the invariant holds.

If the query is not satisfied at any point in the sequence, we will reach an input variable x evaluated at 1. But then there is an outgoing Rv_2Rw path from x, which means that q must be satisfied.

$\boxed{\Longleftarrow}$ Proof by contraposition. Assume that o is evaluated to 0 under σ. We construct a repair $\mathbf{r}$ as follows, for each gate g:

- if g is evaluated to 1, we choose the first R-fact in $\phi_g^{\perp}[Rv_2Rw]$;
- if $g = g_1 \wedge g_2$ and g is evaluated to 0, let g_i be the gate or input variable evaluated to 0. We then choose $\phi_g^{g_i}[Rv_1]$;
- if $g = g_1 \vee g_2$ and g is evaluated to 0, we choose $\phi_g^{c_1}[Rv]$; and
- if $g = g_1 \vee g_2$, we choose $\phi_{c_2}^{g_2}[Rv_1]$.

For a path query p, we write $\mathsf{head}(p)$ for the variable at the key-position of the first atom, and $\mathsf{rear}(p)$ for the variable at the non-key position of the last atom.

Assume for the sake of contradiction that $\mathbf{r}$ satisfies q. Then, there exists some valuation θ such that $\theta(uRv_1Rv_2Rw) \subseteq \mathbf{r}$. Then the gate $g^* := \theta(\mathsf{head}(Rv_1))$ is evaluated to 0 by construction. Let $g_1 := \theta(\mathsf{rear}(Rv_1))$. By construction, for $g^* = g_1 \wedge g_2$ or $g^* = g_1 \vee g_2$, we must have $\phi_g^{g_1}[Rv_1] \subseteq \mathbf{r}$ and g_1 is a gate or an input variable also evaluated to 0. By our construction of $\mathbf{r}$, there is no path with trace Rv_2Rw outgoing from g_1. However, $\theta(Rv_2Rw) \subseteq \mathbf{r}$, this can only happen when g_1 is an OR gate, and one of the following occurs:

- Case that $|Rw| \leq |Rv_1|$, and the trace of $\theta(Rv_2Rw)$ is a prefix of $Rvv_2^+Rv_1$. Then Rw is a prefix of Rv_1, a contradiction.
- Case that $|Rw| > |Rv_1|$, and $Rvv_2^+Rv_1$ is a prefix of the trace of $\theta(Rv_2Rw)$. Consequently, Rv_1 is a prefix of Rw. Then, for every $k \geq 1$, $\mathcal{L}^{\rightsquigarrow}(q)$ contains $uRv_1 (Rv_2)^k Rw$. It is now easily verified that for large enough values of k, uRv_1Rv_2w is not a factor of $uRv_1 (Rv_2)^k Rw$. By Lemmas 5.4 and 7.2, CERTAINTY(q) is **coNP**-hard. □

8 PATH QUERIES WITH CONSTANTS

We now extend our complexity classification of CERTAINTY(q) to path queries in which constants can occur.

Definition 8.1 (Generalized path queries). A *generalized path query* is a Boolean conjunctive query of the following form:

$$q = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_k(\underline{s_k}, s_{k+1})\}, \tag{7}$$

where $s_1, s_2, \ldots, s_{k+1}$ are constants or variables, all distinct, and R_1, $R_2, \ldots, R_k$ are (not necessarily distinct) relation names. Significantly, every constant can occur at most twice: at a non-primary-key position and the next primary-key-position.

The *characteristic prefix* of q, denoted by $\mathsf{char}(q)$, is the longest prefix

$$\{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_\ell(\underline{s_\ell}, s_{\ell+1})\}, 0 \leq \ell \leq k$$

such that no constant occurs among $s_1, s_2, \ldots, s_\ell$ (but $s_{\ell+1}$ can be a constant). Clearly, if q is constant-free, then $\mathsf{char}(q) = q$. □

Example 8.2. If $q = \{R(\underline{x}, y), S(\underline{y}, 0), T(\underline{0}, 1), R(\underline{1}, w)\}$, where 0 and 1 are constants, then $\mathsf{char}(q) = \{R(\underline{x}, y), S(\underline{y}, 0)\}$. □

The following lemma implies that if a generalized path query q starts with a constant, then CERTAINTY(q) is in **FO**. This explains why the complexity classification in the remainder of this section will only depend on $\mathsf{char}(q)$.

Lemma 8.3. *For any generalized path query q,* CERTAINTY(p) *is in* **FO**, *where $p := q \setminus \mathsf{char}(q)$.*

We now introduce some definitions and notations used in our complexity classification. The following definition introduces a convenient syntactic shorthand for characteristic prefixes previously defined in Definition 8.1.

Definition 8.4. Let $q = \{R_1(\underline{x_1}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_k(\underline{x_k}, x_{k+1})\}$ be a path query. We write $[\![q, c]\!]$ for the generalized path query obtained from q by replacing x_{k+1} with the constant c. The constant-free path query q will be denoted by $[\![q, \top]\!]$, where $\top$ is a distinguished special symbol. □

Definition 8.5 (Prefix homomorphism). Let

$$q = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_k(\underline{s_k}, s_{k+1})\}$$
$$p = \{S_1(\underline{t_1}, t_2), S_2(\underline{t_2}, t_3), \ldots, R_\ell(\underline{s_\ell}, s_{\ell+1})\}$$

be generalized path queries. A *homomorphism from q to p* is a substitution θ for the variables in q, extended to be the identity on constants, such that for every $i \in \{1, \ldots, k\}$, $R_i(\underline{\theta(s_i)}, \theta(s_{i+1})) \in p$. Such a homomorphism is a *prefix homomorphism* if $\theta(s_1) = t_1$. □

Example 8.6. Let $q = \{R(\underline{x}, y), R(\underline{y}, 1), S(\underline{1}, z)\}$, and $p = \{R(\underline{x}, y), R(\underline{y}, z), R(\underline{y}, 1)\}$. Then $\mathsf{char}(q) = \{R(\underline{x}, y), R(\underline{y}, 1)\} = [\![RR, 1]\!]$ and $p = [\![RRR, 1]\!]$. There is a homomorphism from $\mathsf{char}(q)$ to p, but there is no prefix homomorphism from $\mathsf{char}(q)$ to p. □

The following conditions generalize C_1, C_2, and C_3 from constant-free path queries to generalized path queries. Let γ be either a constant or the distinguished symbol $\top$.

$\mathcal{D}_1$: Whenever $\mathsf{char}(q) = [\![uRvRw, \gamma]\!]$, there is a prefix homomorphism from $\mathsf{char}(q)$ to $[\![uRvRvRw, \gamma]\!]$.

$\mathcal{D}_2$: Whenever $\mathsf{char}(q) = [\![uRvRw, \gamma]\!]$, there is a homomorphism from $\mathsf{char}(q)$ to $[\![uRvRvRw, \gamma]\!]$; and whenever $\mathsf{char}(q) = [\![uRv_1Rv_2Rw, \gamma]\!]$ for consecutive occurrences of R, $v_1 = v_2$ or there is a prefix homomorphism from $[\![Rw, \gamma]\!]$ to $[\![Rv_1, \gamma]\!]$.

$\mathcal{D}_3$: Whenever $\mathsf{char}(q) = [\![uRvRw, \gamma]\!]$, there is a homomorphism from $\mathsf{char}(q)$ to $[\![uRvRvRw, \gamma]\!]$.

It is easily verified that if $\gamma = \top$, then $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ are equivalent to, respectively, C_1, C_2, and C_3. Likewise, the following theorem degenerates to Theorem 3.2 for path queries without constants.

Theorem 8.7. *For every generalized path query q, the following complexity upper bounds obtain:*

- *if q satisfies $\mathcal{D}_1$, then* CERTAINTY(q) *is in* **FO**;
- *if q satisfies $\mathcal{D}_2$, then* CERTAINTY(q) *is in* **NL**; *and*
- *if q satisfies $\mathcal{D}_3$, then* CERTAINTY(q) *is in* **PTIME**.

The following complexity lower bounds obtain:

- *if q violates $\mathcal{D}_1$, then* CERTAINTY(q) *is* **NL**-*hard;*
- *if q violates $\mathcal{D}_2$, then* CERTAINTY(q) *is* **PTIME**-*hard; and*
- *if q violates $\mathcal{D}_3$, then* CERTAINTY(q) *is* **coNP**-*complete.*

Finally, the proof of Theorem 8.7 reveals that for generalized path queries q containing at least one constant, the complexity of CERTAINTY(q) exhibits a trichotomy (instead of a tetrachotomy as in Theorem 8.7).

Theorem 8.8. *For any generalized path query q containing at least one constant, the problem* CERTAINTY(q) *is either in* **FO**, **NL**-*complete, or* **coNP**-*complete.*

9 RELATED WORK

Inconsistencies in databases have been studied in different contexts [6, 15, 16]. Consistent query answering (CQA) was initiated by the seminal work by Arenas, Bertossi, and Chomicki [2]. After twenty years, their contribution was acknowledged in a *Gems of PODS session* [4]. An overview of complexity classification results in CQA appeared recently in the *Database Principles* column of SIGMOD Record [35].

The term CERTAINTY(q) was coined in [33] to refer to CQA for Boolean queries q on databases that violate primary keys, one per relation, which are fixed by q's schema. The complexity classification of CERTAINTY(q) for the class of self-join-free Boolean conjunctive queries started with the work by Fuxman and Miller [12], and was further pursued in [18, 20–22, 24, 26], which eventually revealed that the complexity of CERTAINTY(q) for self-join-free conjunctive queries displays a trichotomy between **FO**, **L**-complete, and **coNP**-complete. A few extensions beyond this trichotomy result are known. The complexity of CERTAINTY(q) for self-join-free Boolean conjunctive queries with negated atoms was studied in [23]. For self-join-free Boolean conjunctive queries with respect to multiple keys, it remains decidable whether or not CERTAINTY(q) is in **FO** [25].

Little is known about CERTAINTY(q) beyond self-join-free conjunctive queries. Fontaine [9] showed that if we strengthen Conjecture 1.1 from conjunctive queries to unions of conjunctive queries, then it implies Bulatov's dichotomy theorem for conservative CSP [5]. This relationship between CQA and CSP was further explored in [28]. In [1], the authors show the **FO** boundary for CERTAINTY(q) for constant-free Boolean conjunctive queries q using a single binary relation name with a singleton primary key.

The counting variant of the problem CERTAINTY(q), denoted ♯CERTAINTY(q), asks to count the number of repairs that satisfy some Boolean query q. For self-join-free Boolean conjunctive queries, ♯CERTAINTY(q) exhibits a dichotomy between **FP** and ♯**PTIME**-complete [31]. It is known that this dichotomy also holds if self-joins are allowed under the restriction that primary keys are singletons [32].

In practice, systems supporting CQA have often used efficient solvers for Disjunctive Logic Programming, Answer Set Programming (ASP) or Binary Integer Programming (BIP), regardless of whether the CQA problem admits a first-order rewriting [7, 8, 14, 17, 19, 29, 30].

10 CONCLUSION

We established a complexity classification in consistent query answering relative to primary keys, for path queries that can have self-joins: for every path query q, the problem CERTAINTY(q) is in **FO**, **NL**-complete, **PTIME**-complete, or **coNP**-complete, and it is decidable in polynomial time in the size of q which of the four cases applies. If CERTAINTY(q) is in **FO** or in **PTIME**, rewritings of q can be effectively constructed in, respectively, first-order logic and Least Fixpoint Logic .

For binary relation names and singleton primary keys, an intriguing open problem is to generalize the form of the queries, from paths to directed rooted trees, DAGs, or general digraphs. The ultimate open problem is Conjecture 1.1, which conjectures that for every Boolean conjunctive query q, CERTAINTY(q) is either in **PTIME** or **coNP**-complete.

Acknowledgements. This work is supported by the National Science Foundation under grant IIS-1910014.

REFERENCES

[1] Foto N. Afrati, Phokion G. Kolaitis, and Angelos Vasilakopoulos. 2015. Consistent Answers of Conjunctive Queries on Graphs. *CoRR* abs/1503.00650 (2015).
[2] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. 1999. Consistent Query Answers in Inconsistent Databases. In *PODS*. ACM Press, 68–79.
[3] Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. 2017. Answering Conjunctive Queries under Updates. In *PODS*. ACM, 303–318.

[4] Leopoldo E. Bertossi. 2019. Database Repairs and Consistent Query Answering: Origins and Further Developments. In *PODS*. ACM, 48–58.
[5] Andrei A. Bulatov. 2011. Complexity of conservative constraint satisfaction problems. *ACM Trans. Comput. Log.* 12, 4 (2011), 24:1–24:66.
[6] 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
[7] 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.
[8] 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.
[9] 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.
[10] 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.
[11] 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.
[12] Ariel Fuxman and Renée J. Miller. 2007. First-order query rewriting for inconsistent databases. *J. Comput. Syst. Sci.* 73, 4 (2007), 610–635.
[13] Leslie M. Goldschlager. 1977. The Monotone and Planar Circuit Value Problems Are Log Space Complete for P. *SIGACT News* 9, 2 (July 1977), 25–29. https://doi.org/10.1145/1008354.1008356
[14] 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.
[15] 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.
[16] Yannis Katsis, Alin Deutsch, Yannis Papakonstantinou, and Vasilis Vassalos. 2010. Inconsistency resolution in online databases. In *ICDE*. IEEE Computer Society, 1205–1208.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] 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.
[23] Paraschos Koutris and Jef Wijsen. 2018. Consistent Query Answering for Primary Keys and Conjunctive Queries with Negated Atoms. In *PODS*. ACM, 209–224.
[24] 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.
[25] Paraschos Koutris and Jef Wijsen. 2020. First-Order Rewritability in Consistent Query Answering with Respect to Multiple Keys. In *PODS*. ACM, 113–129.
[26] Paraschos Koutris and Jef Wijsen. 2021. Consistent Query Answering for Primary Keys in Datalog. *Theory Comput. Syst.* 65, 1 (2021), 122–178.
[27] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.
[28] 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.
[29] 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.
[30] 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.
[31] Dany Maslowski and Jef Wijsen. 2013. A dichotomy in the complexity of counting database repairs. *J. Comput. Syst. Sci.* 79, 6 (2013), 958–983.
[32] Dany Maslowski and Jef Wijsen. 2014. Counting Database Repairs that Satisfy Conjunctive Queries with Self-Joins. In *ICDT*. OpenProceedings.org, 155–164.
[33] Jef Wijsen. 2010. On the first-order expressibility of computing certain answers to conjunctive queries over uncertain databases. In *PODS*. ACM, 179–190.
[34] Jef Wijsen. 2012. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.* 37, 2 (2012), 9:1–9:35.
[35] Jef Wijsen. 2019. Foundations of Query Answering on Inconsistent Databases. *SIGMOD Rec.* 48, 3 (2019), 6–16.

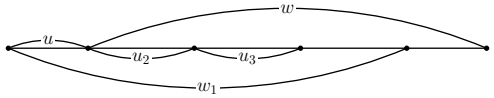
A PROOFS FOR SECTION 4

A.1 Preliminary Results

We define $(q)^k = \varepsilon$ if $k = 0$. The following lemma concerns words having a proper suffix that is also a prefix.

Lemma A.1. *If w is a prefix of the word uw with $u \neq \varepsilon$, then w is a prefix of $(u)^{|w|}$. Symmetrically, if u is a suffix uw with $w \neq \varepsilon$, then u is a suffix of $(w)^{|u|}$.*

Proof. Assume w is a prefix of uw with $u \neq \varepsilon$. The desired result is obvious if $|w| \leq |u|$, in which case w is a prefix of u. In the remainder of the proof, assume $|w| > |u|$. The desired result becomes clear from the following construction:



The word w_1 is the occurrence of w that is a prefix of uw. The word u_2 is the length-$|u|$ prefix of w. Obviously, $u_2 = u$. The word u_2u_3 is the length-$2|u|$ prefix of w. Obviously, $u_3 = u_2$. And so on. It is now clear that w is a prefix of $(u)^{|w|}$. Note that this construction requires $u \neq \varepsilon$. This concludes the proof. □

Definition A.2 (Episode). An *episode* of q is a factor of q of the form RuR such that R does not occur in u. Let $q = \ell RuRr$ where RuR is an episode. We say that this episode is *right-repeating (within q)* if r is a prefix of $(uR)^{|r|}$. Symmetrically, we say that this episode is *left-repeating* if ℓ is a suffix of $(Ru)^{|\ell|}$. □

For example, let $q = AMAA\overbrace{MAAM}^{e_1}A\overbrace{MAAM}^{e_2}AAMAB$. Then the episode called e_1 is left-repeating, while the episode e_2 is neither left-repeating nor right-repeating.

Definition A.3 (Offset). Let u and w be words. We say that u has *offset* n in w if there exists words p, s such that $|p| = n$ and $w = pus$. □

Lemma A.4 (Repeating lemma). *Let q be a word that satisfies C_3. Then, every episode of q is either left-repeating or right-repeating (or both).*

Proof. Let RuR be an episode in $q = \ell RuRr$. By the hypothesis of the lemma, q is a factor of $p := \ell \cdot Ru \cdot \underline{Ru} \cdot Rr$. Since $|q| - |p| = |u| + 1$, the offset of q in p is $\leq |u| + 1$. Since R does not occur in u, it must be that q is either a prefix or a suffix of p. We distinguish two cases:

Case that q is a suffix of p. Then, it is easily verified that ℓ is a suffix of ℓRu. By Lemma A.1, ℓ is a suffix of $(Ru)^{|\ell|}$, which means that RuR is left-repeating within q.

Case that q is a prefix of p. We have that r is a prefix of uRr. By Lemma A.1, r is a prefix of $(uR)^{|r|}$, which means that RuR is right-repeating.

This concludes the proof. □

Definition A.5. If q is a word over an alphabet Σ, then symbols(q) is the set that contains all (and only) the symbols that occur in q.

Lemma A.6 (Self-join-free episodes). *Let q be a word that satisfies C_3. Let $L\ell L$ be the right-most occurrence of an episode that is left-repeating in q. Then, $L\ell$ is self-join-free.*

Proof. Consider for the sake of contradiction that $L\ell$ is not self-join-free. Since $L \notin \text{symbols}(\ell)$, it must be that ℓ has a factor MmM such that Mm is self-join-free. By Lemma A.4, MmM must be left-repeating or right-repeating, which requires $L \in \text{symbols}(Mm)$, a contradiction. □

A.2 Proof of Lemma 4.2

Proof of Lemma 4.2. The implication $2 \implies 1$ is obvious. To show $1 \implies 2$, assume that q satisfies C_1. The desired result is obvious if q is self-join-free. Assume from here on that q is not self-join-free. Then, we can write $q = \ell RmRr$, such that ℓRm is self-join-free. That is, the second occurrence of R is the left-most symbol that occurs a second time in q. By C_1, q is a prefix of $\ell RmRmRr$. It follows that Rr is a prefix of $RmRr$. By Lemma A.1, Rr is a prefix of $(Rm)^{|r|+1}$. It follows that there is a k such that q is a prefix of $\ell\,(Rm)^k$. □

A.3 Proof of Lemma 4.3

Proof of Lemma 4.3. The proof of $2 \implies 1$ is straightforward. We show next the direction $1 \implies 2$. To this end, assume that q satisfies C_3. The desired result is obvious if q is self-join-free (let $j = k = 0$ in $\mathcal{B}_{2a}$). Assume that q has a factor $L\ell L \cdot m \cdot RrR$ where $L\ell L$ and RrR are episodes such that symbols($L\ell L$), symbols(RrR), and symbols(m) are pairwise disjoint. Then, by Lemma A.4, $L\ell L$ must be left-repeating, and RrR right-repeating. By Lemma A.6, $L\ell$ and rR are self-join-free. Then q is of the form $\mathcal{B}_{2a}$. By letting $j = 0$ or $k = 0$, we obtain the situation where the number of episodes that are factors of q is zero or one.

The only difficult case is where two episodes overlap. Assume that q has an episode that is left-repeating (the case of a right-repeating episode is symmetrical). Assume that this left-repeating episode is $e_1 := L\ell RoL$ in $q := \cdots \overbrace{L\ell \underbrace{RoL}_{e_2} rR}^{e_1} \cdots$, where it can be assumed that e_1 is the right-most episode that is left-repeating. Then, $\ell \neq \varepsilon \neq r$ implies $\text{first}(\ell) \neq \text{first}(r)$ (or else e_1 would not be right-most, a contradiction). By a similar reasoning, $\ell = \varepsilon$ implies $r \neq \varepsilon$. Therefore, it is correct to conclude $\ell \neq r$. It can also be assumed without loss of generality that r shares no symbols with e_1, by choosing R as the first symbol after e_1 that also occurs in e_1. Now assume e_2 is right-repeating, over a length $> |oL|$. Then q contains a factor $\overbrace{L\ell \underbrace{RoL}_{e_2} rR}^{e_1} \cdot oL$. Then, q rewinds to a word p with factor:

$$\overbrace{L\ell \underbrace{RoL}_{e_2} rR}^{e_1} \cdot o \overbrace{L|\ell \underbrace{RoL}_{e_2} rR}^{e_1} \cdot oL,$$

where the vertical bar | is added to indicate a distinguished position. It can now be verified that q is not a factor of p, because of the alternation of e_1 and e_2 which does not occur in q. This contradicts the hypothesis of the lemma. In particular, the words that start at position | are r and ℓ in, respectively, q and p. We conclude by contradiction that e_2 cannot be right-repeating over a length $> |oL|$. Thus, following the right-most occurrence of e_1, the word q can contain fresh word r, followed by RoL, which is a suffix of e_1. This is exactly the form $\mathcal{B}_{2b}$.

A remaining, and simpler, case is where two episodes overlap by a single symbol $R = L$, giving $q := \cdots \overbrace{L\ell \underbrace{L}_{e_2} rL}^{e_1} \cdots$, where e_1 is the right-most episode that is left-repeating, and L is the first symbol after e_1 that also occurs in e_1. Therefore, L does not occur in $\ell \cdot r$, and $\ell \neq r$. Indeed, if $\ell = \varepsilon = r$, then e_1 is not right-most; and if $\ell \neq \varepsilon \neq r$, then $\text{first}(\ell) \neq \text{first}(r)$, or else e_1 would not be right-most, a contradiction. The word q rewinds to a word p with factor $\overbrace{L\ell \underbrace{L}_{e_2}}^{e_1} r \overbrace{L\ell \underbrace{L}_{e_2}}^{e_1} rL$, and $|p| - |q| = |\ell| + |r| + 2$. It is easily verified that e_2 cannot be right-repeating for > 0 symbols. For instance, consider the case where $r \neq \varepsilon$ and e_2 is right-repeating for 1 symbol, meaning that q has suffix $\overbrace{L\ell \underbrace{L}_{e_2} rL}^{e_1} \cdot \text{first}(r)$, and p has suffix $\overbrace{L\ell \underbrace{L}_{e_2}}^{e_1} r \overbrace{L\ell \underbrace{L}_{e_2}}^{e_1} rL \cdot \text{first}(r)$. If we left-align these suffixes, then there is a mismatch between $\text{first}(r)$ and the leftmost symbol of $L\ell$. The other possibility is to right-align these suffixes, but then e_1 cannot be genuinely left-repeating within q. □

A.4 Proof of Lemma 4.5

Proof of Lemma 4.5. Assume that q satisfies C_3. By Lemma 4.3, q satisfies $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, or $\mathcal{B}_3$.

$\boxed{1 \implies 2}$ By contraposition. Assume that (2) does not hold. Then, either q satisfies $\mathcal{B}_{2a}$ or q satisfies $\mathcal{B}_{2b}$. Assume that $q = aRb_1Rb_2Rc$ for three consecutive occurrences of R such that $b_1 \neq b_2$. It suffices to show that Rc is a prefix of Rb_1. It is easily verified that $b_1 \neq b_2$ cannot happen if q satisfies $\mathcal{B}_{2a}$. Therefore, q satisfies $\mathcal{B}_{2b}$. The word in $(uv)^k\,wv$ in $\mathcal{B}_{2b}$ indeed allows for suffix $vu \cdot vw \cdot v$ where the first and second occurrence of v are followed, respectively, by u and w. Then, in q, we have that w is followed by a prefix of v, and therefore C_2 is satisfied.

$\boxed{2 \implies 3}$ The hypothesis is that q satisfies $\mathcal{B}_3$, but falsifies both $\mathcal{B}_{2a}$ and $\mathcal{B}_{2b}$. We can assume $k \geq 0$ and self-join-free word uvw such that q is a factor of $uw\,(uv)^k$, but q falsifies $\mathcal{B}_{2a}$ and $\mathcal{B}_{2b}$. It must be that $u \neq \varepsilon$ and the offset of q in $uw\,(uv)^k$ is $< |u|$, for otherwise q is a factor of $w\,(uv)^k$ and therefore satisfies $\mathcal{B}_{2a}$, a contradiction. Also, one of v or w must not be the empty word, or else q is a factor of $u\,(u)^k$, and therefore satisfies $\mathcal{B}_{2a}$ (and also satisfies $\mathcal{B}_{2b}$). We now consider the length of q. The word $uwuvu$ is a factor of $(wu)^2\,vu$, and thus satisfies $\mathcal{B}_{2b}$. If $v = \emptyset$, then the

word $uwuu$ is a factor of $(wu)^2\,u$, and thus satisfies $\mathcal{B}_{2b}$. It is now correct to conclude that one of the following must occur:

- $v \neq \emptyset$ and $\text{last}(u) \cdot wuvu \cdot \text{first}(v)$ is a factor of q; or
- $v = \emptyset$, $w \neq \emptyset$ and $\text{last}(u) \cdot w\,(u)^2 \cdot \text{first}(u)$ is a factor of q.

$\boxed{3 \Longrightarrow 1}$ Assume (3). Consider first the case $v \neq \varepsilon$. Let $u = \hat{u}R$ and $v = S\hat{v}$. We have $R \neq S$, since uv is self-join-free. By item (3a), q has a factor $R \cdot w\hat{u}RS\hat{v}\hat{u}R \cdot S$, with three consecutive occurrences of R. It is easily verified that $w\hat{u} \neq S\hat{v}\hat{u}$, and that RS is not a prefix of $Rw\hat{u}$. Therefore q falsifies C_2.

Consider next the case $v = \varepsilon$ (whence $w \neq \varepsilon$). Let $u = \hat{u}R$. By item (3b), q has a factor $R \cdot w\hat{u}R\hat{u}R \cdot \text{first}(u)$, with three consecutive occurrences of R. Since $w\hat{u} \neq \hat{u}$ and $\text{first}(u) \neq \text{first}(w)$, it follows that q falsifies C_2. □

B PROOFS FOR SECTION 8

B.1 Proof of Lemma 8.3

Lemma 8.3 is an immediate corollary of Lemma B.3, which states that whenever a generalized path query starts with a constant, then CERTAINTY(q) is in **FO**. Its proof needs two helping lemmas.

Lemma B.1. *Let $q = q_1 \cup q_2 \cup \cdots \cup q_k$ be a Boolean conjunctive query such that for all $1 \leq i < j \leq k$, $\text{vars}(q_i) \cap \text{vars}(q_j) = \emptyset$. Then, the following are equivalent for every database instance* **db***:*

(1) **db** *is a "yes"-instance for* CERTAINTY(q)*; and*
(2) for each $1 \leq i \leq k$, **db** *is a "yes"-instance for* CERTAINTY(q_i).

Proof. We give the proof for $k = 2$. The generalization to larger k is straightforward.

$\boxed{1 \Longrightarrow 2}$ Assume that (1) holds true. Then each repair $\mathbf{r}$ of **db** satisfies q, and therefore satisfies both q_1 and q_2. Therefore, **db** is a "yes"-instance for both CERTAINTY(q_1) and CERTAINTY(q_2).

$\boxed{2 \Longrightarrow 1}$ Assume that (2) holds true. Let $\mathbf{r}$ be any repair of **db**. Then there are valuations μ from $\text{vars}(q_1)$ to $\text{adom}(\mathbf{db})$, and θ from $\text{vars}(q_2)$ to $\text{adom}(\mathbf{db})$ such that $\mu(q_1) \subseteq \mathbf{r}$ and $\theta(q_2) \subseteq \mathbf{r}$. Since $\text{vars}(q_1) \cap \text{vars}(q_2) = \emptyset$ by construction, we can define a valuation σ as follows, for every variable $z \in \text{vars}(q_1) \cup \text{vars}(q_2)$:

$$\sigma(z) = \begin{cases} \mu(z) & \text{if } z \in \text{vars}(q_1) \\ \theta(z) & \text{if } z \in \text{vars}(q_2) \end{cases}$$

From $\sigma(q) = \sigma(q_1) \cup \sigma(q_2) = \mu(q_1) \cup \theta(q_2) \subseteq \mathbf{r}$, it follows that $\mathbf{r}$ satisfies q. Therefore, **db** is a "yes"-instance for CERTAINTY(q). □

Lemma B.2. *Let q be a generalized path query with*

$$q = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_k(\underline{s_k}, c)\},$$

where c is a constant, and each s_i is either a constant or a variable for all $i \in \{1, \ldots, k\}$. Let

$$p = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_k(\underline{s_k}, s_{k+1}), N(\underline{s_{k+1}}, s_{k+2})\},$$

where s_{k+1}, s_{k+2} are fresh variables to q and N is a fresh relation to q. Then there exists a first-order reduction from CERTAINTY(q) *to* CERTAINTY(p).

Proof. Let **db** be an instance for CERTAINTY(q) and consider the instance $\mathbf{db} \cup \{N(\underline{c}, d)\}$ for CERTAINTY(p) where d is a fresh constant to $\text{adom}(\mathbf{db})$.

We show that **db** is a "yes"-instance for CERTAINTY(q) if and only if $\mathbf{db} \cup \{N(\underline{c}, d)\}$ is a "yes"- instance for CERTAINTY(p).

$\boxed{\Longrightarrow}$ Assume **db** is a "yes"-instance for CERTAINTY(q). Let $\mathbf{r}$ be any repair of $\mathbf{db} \cup \{N(\underline{c}, d)\}$, and thus $\mathbf{r} \setminus \{N(\underline{c}, d)\}$ is a repair for **db**. Then there exists a valuation μ with $\mu(q) \subseteq \mathbf{r} \setminus \{N(\underline{c}, d)\}$. Consider the valuation μ^+ from $\text{vars}(q) \cup \{s_{k+1}, s_{k+2}\}$ to $\text{adom}(\mathbf{db}) \cup \{c, d\}$ that agrees with μ on $\text{vars}(q)$ and maps additionally $\mu^+(s_{k+1}) = c$ and $\mu^+(s_{k+2}) = d$. We thus have $\mu^+(p) \subseteq \mathbf{r}$. It is correct to conclude that $\mathbf{db} \cup \{N(\underline{c}, d)\}$ is a "yes"-instance for CERTAINTY(p).

$\boxed{\Longleftarrow}$ Assume that $\mathbf{db} \cup \{N(\underline{c}, d)\}$ is a "yes"-instance for the problem CERTAINTY(p). Let $\mathbf{r}$ be any repair of **db**. Then $\mathbf{r} \cup \{N(\underline{c}, d)\}$ is a repair of $\mathbf{db} \cup \{N(\underline{c}, d)\}$, and thus there exists some valuation θ with $\theta(p) \subseteq \mathbf{r} \cup \{N(\underline{c}, d)\}$. Since **db** contains only one N-fact, we have $\theta(s_{k+1}) = c$. It follows that $\theta(q) \subseteq \mathbf{r}$, as desired. □

Lemma B.3. *Let q be a generalized path query with*

$$q = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \ldots, R_k(\underline{s_k}, s_{k+1})\}$$

where s_1 is a constant, and each s_i is either a constant or a variable for all $i \in \{2, \ldots, k+1\}$. Then the problem CERTAINTY(q) *is in* **FO**.

Proof. Let the $1 = j_1 < j_2 < \cdots < j_\ell \leq k+1$ be all the indexes j such that s_j is a constant for some $\ell \geq 1$. Let $j_{\ell+1} = k+1$. Then for each $i \in \{1, 2, \ldots, \ell\}$, the query

$$q_i = \bigcup_{j_i \leq j < j_{i+1}} \{R_j(\underline{s_j}, s_{j+1})\}$$

is a generalized path query where each s_{j_i} is a constant.

We claim that CERTAINTY(q_i) is in **FO** for each $1 \leq i \leq \ell$. Indeed, if $s_{j_{i+1}}$ is a variable, then the claim follows by Lemma 6.7; if $s_{j_{i+1}}$ is a constant, then the claim follows by Lemma B.2 and Lemma 6.7.

Since by construction, $q = q_1 \cup q_2 \cup \cdots \cup q_\ell$, we conclude that CERTAINTY(q) is in **FO** by Lemma B.1. □

The proof of Lemma 8.3 is now simple.

Proof of Lemma 8.3. If q contains no constants, the lemma holds trivially. Otherwise, CERTAINTY(p) is in **FO** by Lemma B.3. □

B.2 Elimination of Constants

In this section, we show how constants can be eliminated from generalized path queries. The *extended query* of a generalized path query is defined next.

Definition B.4 (Extended query). Let q be a generalized path query. The *extended query* of q, denoted by $\text{ext}(q)$, is defined as follows:

- if q does not contain any constant, then $\text{ext}(q) := q$;
- otherwise, $\text{char}(q) = \{R_1(\underline{x_1}, x_2), R_2(\underline{x_2}, x_3), \ldots, R_\ell(\underline{x_\ell}, c)\}$ for some constant c. In this case, we define

  $$\text{ext}(q) := \{R_1(\underline{x_1}, x_2), \ldots, R_\ell(\underline{x_\ell}, x_{\ell+1}), N(\underline{x_{\ell+1}}, x_{\ell+2})\},$$

  where $x_{\ell+1}$ and $x_{\ell+2}$ are fresh variables and N is a fresh relation name not occurring in q. □

By definition, $\text{ext}(q)$ does not contain any constant.

Example B.5. Let $q = R(\underline{x}, y), S(\underline{y}, 0), T(\underline{0}, 1), R(\underline{1}, w)$ where 0 and 1 are constants. We have $\text{ext}(q) = R(\underline{x}, y), S(\underline{y}, z), N(\underline{z}, u)$. □

We show two lemmas which, taken together, show that the problem CERTAINTY(q) is first-order reducible to CERTAINTY($\text{ext}(q)$), for every generalized path query q.

Lemma B.6. *For every generalized path query q, there is a first-order reduction from* CERTAINTY(q) *to* CERTAINTY(char(q)).

Proof. Let $p := q \setminus \mathsf{char}(q)$. Since $\mathsf{vars}(\mathsf{char}(q)) \cap \mathsf{vars}(p) = \emptyset$, Lemmas B.1 and B.3 imply that the following are equivalent for every database instance **db**:

(1) **db** is a "yes"-instance for CERTAINTY(q); and
(2) **db** is a "yes"-instance for CERTAINTY(char(q)) and a "yes"-instance for CERTAINTY(p).

To conclude the proof, it suffices to observe that CERTAINTY(p) is in **FO** by Lemma B.3. □

Lemma B.7. *For every generalized path query q, there is a first-order reduction from* CERTAINTY(char(q)) *to* CERTAINTY(ext(q)).

Proof. Let q be a generalized path query. If q contains no constants, the lemma trivially obtains because $\mathsf{char}(q) = \mathsf{ext}(q) = q$. If q contains at least one constant, then there exists a first-order reduction from CERTAINTY(char(q)) to CERTAINTY(ext(q)) by Lemma B.2. □

B.3 Complexity Upper Bounds in Theorem 8.7

Lemma B.8. *Let q be a generalized path query that contains at least one constant. If q satisfies $\mathcal{D}_3$, then q satisfies $\mathcal{D}_2$ and* ext(q) *satisfies C_2.*

Proof. Assume that q satisfies $\mathcal{D}_3$. Let $\mathsf{char}(q) = [\![p, c]\!]$ for some constant c. We have $\mathsf{ext}(q) = p \cdot N$ where N is a fresh relation name not occurring in p.

We first argue that ext(q) is a factor of every word to which ext(q) rewinds. To this end, let $\mathsf{ext}(q) = uRvRwN$ where $p = uRvRw$. Since q satisfies $\mathcal{D}_3$, there exists a homomorphism from $\mathsf{char}(q) = [\![uRvRw, c]\!]$ to $[\![uRvRvRw, c]\!]$, implying that $uRvRw$ is a suffix of $uRvRvRw$. It follows that $uRvRwN$ is a suffix of $uRvRvRwN$. Hence ext(q) satisfies C_3.

The remaining test for C_2 is where $\mathsf{ext}(q) = uRv_1Rv_2RwN$ for consecutive occurrences of R. We need to show that either $v_1 = v_2$ or RwN is a prefix of Rv_1 (or both). We have $p = uRv_1Rv_2Rw$. Since q satisfies $\mathcal{D}_3$, there exists a homomorphism from $\mathsf{char}(q) = [\![uRv_1Rv_2Rw, c]\!]$ to $[\![uRv_1Rv_2Rv_2Rw, c]\!]$. Since c is a constant, the homomorphism must map Rv_1 to Rv_2, implying that $v_1 = v_2$. It is correct to conclude that q satisfies $\mathcal{D}_2$ and ext(q) satisfies C_2. □

Lemma B.9. *For every generalized path query q,*

- *if q satisfies $\mathcal{D}_1$, then* ext(q) *satisfies C_1;*
- *if q satisfies $\mathcal{D}_2$, then* ext(q) *satisfies C_2; and*
- *if q satisfies $\mathcal{D}_3$, then* ext(q) *satisfies C_3.*

Proof. The lemma holds trivially if q contains no constant. Assume from here on that q contains at least one constant.

Assume that q satisfies $\mathcal{D}_1$. Then $\mathsf{char}(q)$ must be self-join-free. In this case, ext(q) is self-join-free, and thus ext(q) satisfies C_1.

For the two remaining items, assume that q satisfies $\mathcal{D}_2$ or $\mathcal{D}_3$. Since $\mathcal{D}_2$ logically implies $\mathcal{D}_3$, q satisfies $\mathcal{D}_3$. By Lemma B.8, ext(q) satisfies C_2. Since C_2 logically implies C_3, q satisfies C_3. □

We can now prove the upper bounds in Theorem 8.7.

Proof of upper bounds in Theorem 8.7. Since first-order reductions compose, by Lemmas B.6 and B.7, there is a first-order reduction from the problem CERTAINTY(q) to CERTAINTY(ext(q)). The upper bound results then follow by Lemma B.9. □

B.4 Complexity Lower Bounds in Theorem 8.7

The complexity lower bounds in Theorem 8.7 can be proved by slight modifications of the proofs in Sections 7.1 and 7.2. We explain these modifications below for a generalized path query q containing at least one constant. Note incidentally that the proof in Section 7.3 needs no revisiting, because, by Lemma B.8, a violation of $\mathcal{D}_2$ implies a violation of $\mathcal{D}_3$.

In the proof of Lemma 7.1, let $\mathsf{char}(q) = [\![uRvRw, c]\!]$ where c is a constant and there is no prefix homomorphism from $\mathsf{char}(q)$ to $[\![uRvRvRw, c]\!]$. Let $p = q \setminus \mathsf{char}(q)$. Note that the path query uRv does not contain any constant. We revise the reduction description in Lemma 7.1 to be

- for each vertex $x \in V \cup \{s'\}$, we add $\phi^x_\perp[u]$;
- for each edge $(x, y) \in E \cup \{(s', s), (t, t')\}$, we add $\phi^y_x[Rv]$;
- for each vertex $x \in V$, we add $\phi^c_x[Rw]$; and
- add a canonical copy of p (which starts in the constant c).

An example is shown in Figure 13. Since the constant c occurs at most twice in q by Definition 8.1, the query q can only be satisfied by a repair including each of $\phi^x_\perp[u]$, $\phi^y_x[Rv]$, $\phi^c_y[Rw]$, and the canonical copy of p. **NL**-hardness can now be proved as in the proof of Lemma 7.1.

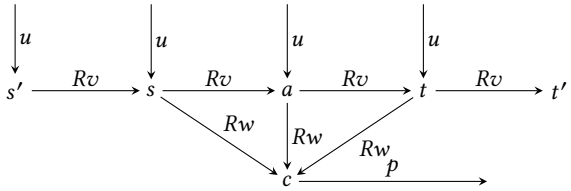


Figure 13: Database instance for the revised NL-hardness reduction from the graph G with $V = \{s, a, t\}$ and $E = \{(s, a), (a, t)\}$.

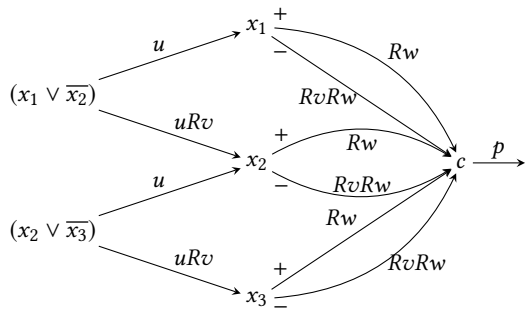


Figure 14: Database instance for the revised coNP-hardness reduction from the formula $\psi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3})$.

In the proof of Lemma 7.2, let $\mathsf{char}(q) = [\![uRvRw, c]\!]$ where c is a constant and there is no homomorphism from $\mathsf{char}(q)$ to $[\![uRvRvRw, c]\!]$. Let $p = q \setminus \mathsf{char}(q)$. Note that both path queries

uRv and u do not contain any constant. We revise the reduction description in Lemma 7.2 to be

- for each variable z, we add $\phi_z^c[Rw]$ and $\phi_z^c[RvRw]$;
- for each clause C and positive literal z of C, we add $\phi_C^z[u]$;
- for each clause C and variable z that occurs in a negative literal of C, we add $\phi_C^z[uRv]$; and
- add a canonical copy of p (which starts in the constant c).

An example is shown in Figure 14. Since the constant c occurs at most twice in q, the query q can only be satisfied by a repair $\mathbf{r}$ such that either

- $\mathbf{r}$ contains $\phi_C^z[uRv]$, $\phi_z^c[Rw]$, and the canonical copy of p; or
- $\mathbf{r}$ contains $\phi_C^z[u]$, $\phi_z^c[RvRw]$, and the canonical copy of p.

coNP-hardness can now be proved as in the proof of Lemma 7.2.

B.5 Proof of Theorem 8.8

Proof of Theorem 8.8. Immediate consequence of Theorem 8.7 and Lemma B.8. □