arXiv:2309.15270v1 [cs.DB] 26 Sep 2023

Consistent Query Answering for Primary Keys on Path Queries*

PARASCHOS KOUTRIS AND XIATING OUYANG
University of Wisconsin-Madison, WI, USA

JEF WIJSEN
University of Mons, Belgium

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 $\mathsf{CERTAINTY}(q)$ takes a database as input, and asks whether or not each repair satisfies q. It is known that for any self-join-free Boolean conjunctive query q, $\mathsf{CERTAINTY}(q)$ is in **FO**, **L**-complete, or **coNP**-complete. In particular, $\mathsf{CERTAINTY}(q)$ is in **FO** for any self-join-free Boolean path query q. In this paper, we show that if self-joins are allowed, the complexity of $\mathsf{CERTAINTY}(q)$ 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.

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) [3], 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: $\mathsf{CERTAINTY}(q)$

Input: A database instance **db**.

Question: Does q evaluate to true on every repair of **db**?

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

Conjecture 1. *For each Boolean conjunctive query q,* $\mathsf{CERTAINTY}(q)$ *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 ([32])**.** *For each self-join-free Boolean conjunctive query q,* $\mathsf{CERTAINTY}(q)$ *is in* **FO**, **L***-complete, or* **coNP***-complete, and it is decidable which of the three cases applies.*

---

*This paper is an evolved version of a paper with the same title and authors published at ACM PODS'21 [25]. In particular, the proof of Lemma 9 in the current paper is new and replaces a flawed proof in the earlier version, and the technical treatment in the current Section 6.3 strengthens some earlier results.

R	$\underline{1}$	2
	a	a
	a	b
	b	a
	b	b

S	$\underline{1}$	2
	a	a
	a	b
	b	a
	b	b

Figure 1: An inconsistent database instance **db**.

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. Self-joins happen to significantly change the complexity landscape laid down in Theorem 1; this is illustrated by Example 2. Self-join-freeness is a simplifying assumption that is also used outside CQA (e.g., [15, 4, 16]).

Example 1. 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 $q_2 = \exists x \exists y (R(\underline{x}, y) \wedge S(\underline{y}, x))$, where the primary key positions are underlined. Consider the inconsistent database instance **db** in Figure 1. We have that **db** is a "no"-instance of $\mathsf{CERTAINTY}(q_2)$, 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 $\mathsf{CERTAINTY}(q_1)$. 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 2. 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 $q_2 = \exists x \exists y \exists z (R(\underline{x}, z) \wedge S(\underline{y}, z))$. $\mathsf{CERTAINTY}(q_2)$ is known to be **coNP**-complete, whereas it is easily verified that $\mathsf{CERTAINTY}(q_1)$ is in **FO**, by observing that a database instance is a "yes"-instance of $\mathsf{CERTAINTY}(q_1)$ if and only if it satisfies $\exists x \exists y (R(\underline{x}, y))$. ☐

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

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

which we call *path queries*. The primary key positions are underlined. 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, which is currently beyond reach. If all R_i's are distinct (i.e., if there are no self-joins), then $\mathsf{CERTAINTY}(q)$ is known to be in **FO** for path queries q. However, when self-joins are allowed, the complexity landscape of $\mathsf{CERTAINTY}(q)$ for path queries exhibits a tetrachotomy, as stated by the following main result of our paper.

Theorem 2. *For each Boolean path query q, $\mathsf{CERTAINTY}(q)$ 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 and Theorem 2, it is striking that there are path queries q for which $\mathsf{CERTAINTY}(q)$ 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 2 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 R, S, X, Y etc., 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 $\mathsf{CERTAINTY}(q_1)$ 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 $\mathsf{CERTAINTY}(q_2)$ 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**).

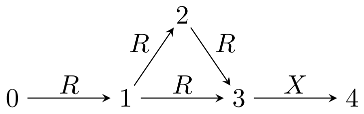


Figure 2: An example database instance **db** for $q_2 = RRX$.

The situation is still different for $q_3 = ARRX$, for which it will be shown that $\mathsf{CERTAINTY}(q_3)$ 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.

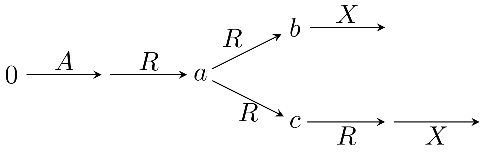


Figure 3: An example database instance **db** for $q_3 = ARRX$.

Organization. Section 2 introduces the preliminaries. In Section 3, the statement of Theorem 3 gives the syntactic conditions for deciding the complexity of $\mathsf{CERTAINTY}(q)$ 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 $\mathbf{db}$ of facts using only the relation names of the schema. We write $\mathsf{adom}(\mathbf{db})$ for the active domain of $\mathbf{db}$ (i.e., the set of constants that occur in $\mathbf{db}$). A *block* of $\mathbf{db}$ is a maximal set of key-equal facts of $\mathbf{db}$. Whenever a database instance $\mathbf{db}$ 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 $\mathbf{db}$ is *consistent* if it contains no two distinct facts that are key-equal (i.e., if no block of $\mathbf{db}$ contains more than one fact). A *repair* of $\mathbf{db}$ is an inclusion-maximal consistent subset of $\mathbf{db}$.

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 $\mathsf{vars}(q)$ the set of variables that occur in q. The set q represents the first-order sentence

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

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

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 $\mathsf{CERTAINTY}(q)$ takes as input a database instance $\mathbf{db}$, and asks whether q is satisfied by every repair of $\mathbf{db}$. It is straightforward that for every Boolean conjunctive query q, $\mathsf{CERTAINTY}(q)$ 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,..., x_{k+1} are distinct variables, and R_1, R_2,..., 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 $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ for path queries q. Let R be any relation name in q, and let u, v, w be (possibly empty) words over the alphabet of relation names of q.

$\mathcal{C}_1$**:** Whenever $q = uRvRw$, q is a prefix of $uRvRvRw$.

$\mathcal{C}_2$**:** 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.

$\mathcal{C}_3$**:** 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: $\mathcal{C}_1$ is tantamount to saying that q is a prefix of every word to which q rewinds; and $\mathcal{C}_3$ 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 1. *Let q be a path query. If q satisfies $\mathcal{C}_1$, then q satisfies $\mathcal{C}_2$; and if q satisfies $\mathcal{C}_2$, then q satisfies $\mathcal{C}_3$.*

The main part of this paper comprises a proof of the following theorem, which refines the statement of Theorem 2 by adding syntactic conditions. The theorem is illustrated by Example 3.

Theorem 3. *For every path query q, the following complexity upper bounds obtain:*

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

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

- *if q violates $\mathcal{C}_1$, then* $\mathsf{CERTAINTY}(q)$ *is* **NL***-hard;*

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

Example 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 $\mathsf{CERTAINTY}(q_1)$ 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 $\mathcal{C}_3$, but violates $\mathcal{C}_1$. Since q_2 vacuously satisfies $\mathcal{C}_2$ (because no relation name occurs three times in q_2), it is correct to conclude that $\mathsf{CERTAINTY}(q_2)$ 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 $\mathcal{C}_3$ but violates $\mathcal{C}_1$. It can be verified that q_3 violates $\mathcal{C}_2$ by writing it as follows:

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

We have $X = v_1 \neq v_2 = Y$, but $Rw = RY$ is not a prefix of $Rv_1 = RX$. Thus, $\mathsf{CERTAINTY}(q_3)$ 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 $\mathsf{CERTAINTY}(q_4)$ is **coNP**-complete. ☐

4 Regexes for $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$

In this section, we show that the conditions $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ can be captured by regular expressions (or regexes) on path queries, which will be used in the proof of Theorem 3. Since these results are within the field of *combinatorics of words*, we will use the term *word* rather than *path query*.

Definition 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 $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$, $\mathcal{B}_1$, $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, $\mathcal{B}_3$ with the set of all words satisfying this condition. Note then that $\mathcal{B}_1 \subsetneq \mathcal{B}_{2a} \cap \mathcal{B}_3$. The results in the remainder of this section can be summarized as follows:

- $\mathcal{C}_1 = \mathcal{B}_1$ (Lemma 1)
- $\mathcal{C}_2 = \mathcal{B}_{2a} \cup \mathcal{B}_{2b}$ (Lemma 3)
- $\mathcal{C}_3 = \mathcal{B}_{2a} \cup \mathcal{B}_{2b} \cup \mathcal{B}_3$ (Lemma 2)

Moreover, Lemma 3 characterizes $\mathcal{C}_3 \setminus \mathcal{C}_2$.

Lemma 1. *For every word q, the following are equivalent:*

1. *q satisfies $\mathcal{C}_1$; and*
2. *q satisfies $\mathcal{B}_1$.*

Lemma 2. *For every word q, the following are equivalent:*

1. *q satisfies $\mathcal{C}_3$; and*
2. *q satisfies $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, or $\mathcal{B}_3$.*

Definition 2 (First and last symbol)**.** For a nonempty word u, we write $\mathsf{first}(u)$ and $\mathsf{last}(u)$ for, respectively, the first and the last symbol of u. ☐

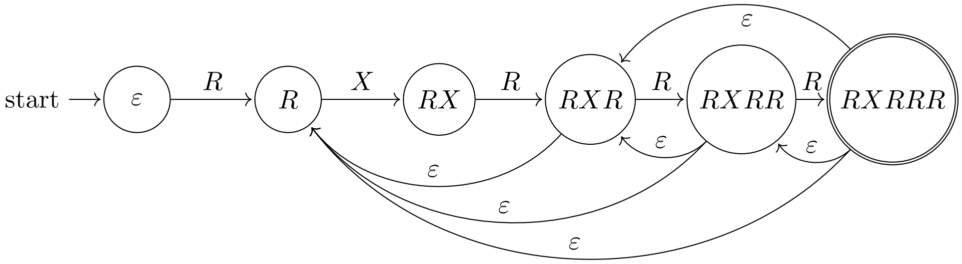


Figure 4: The $\mathsf{NFA}(q)$ automaton for the path query $q = RXRRR$.

Lemma 3. *Let q be a word that satisfies $\mathcal{C}_3$. Then, the following three statements are equivalent:*

1. *q violates $\mathcal{C}_2$;*
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 $\mathsf{last}(u) \cdot wuvu \cdot \mathsf{first}(v)$ is a factor of q; or*
   - *(b) $v = \varepsilon$, $w \neq \varepsilon$, and $\mathsf{last}(u) \cdot w\,(u)^2 \cdot \mathsf{first}(u)$ 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 $\mathcal{C}_2$, $\mathcal{B}_{2a}$, and $\mathcal{B}_{2b}$ implies $\mathcal{C}_3$, it is correct to conclude that the equivalence between the first two items in Lemma 3 does not need the hypothesis that q must satisfy $\mathcal{C}_3$.

5 Automaton-Based Perspective

In this section, we prove an important lemma, Lemma 7, which will be used for proving the complexity upper bounds in Theorem 3.

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 $\mathsf{NFA}(q)$.

Definition 3 ($\mathsf{NFA}(q)$). Every word q gives rise to a nondeterministic finite automaton (NFA) with ε-moves, denoted $\mathsf{NFA}(q)$, as follows.

States: The set of states is the set of prefixes of q. The empty word ε is a prefix 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 $\mathsf{NFA}(RXRRR)$. 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 $\mathsf{NFA}(q)$, which straightforwardly results in Lemma 4. Then, Lemma 5 gives alternative ways for expressing $\mathcal{C}_1$ and $\mathcal{C}_3$.

Definition 4. Let q be a path query, represented as a word over the alphabet of relation names. We define the language $\mathcal{L}^{\looparrowright}(q)$ as the smallest set of words such that

(a) q belongs to $\mathcal{L}^{\looparrowright}(q)$; and

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

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

Lemma 4. *For every path query q, the automaton $\mathsf{NFA}(q)$ accepts the language $\mathcal{L}^{\looparrowright}(q)$.*

Lemma 5. *Let q be a path query. Then,*

1. *q satisfies $\mathcal{C}_1$ if and only if q is a prefix of each $p \in \mathcal{L}^{\looparrowright}(q)$;*
2. *q satisfies $\mathcal{C}_3$ if and only if q is a factor of each $p \in \mathcal{L}^{\looparrowright}(q)$.*

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

We now show the $\Longrightarrow$ direction in both items. To this end, we call an application of the rule (b) in Definition 4 a *rewind*. By construction, each word in $\mathcal{L}^{\looparrowright}(q)$ can be obtained from q by using k rewinds, for some nonnegative integer k. Let q_k be a word in $\mathcal{L}^{\looparrowright}(q)$ 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 $\mathcal{L}^{\looparrowright}(q)$ 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;
- otherwise $uRvR$ is a proper prefix of q. Let $q = uRvRt$ where t is nonempty. Since q satisfies $\mathcal{C}_1$, 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 $\mathcal{L}^{\looparrowright}(q)$ 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 can decompose $q_k = u^- q^- RvRq^+ w^+$ where $q = q^- RvRq^+$, $u = u^- q^-$ and $w = q^+ w^+$. Since q satisfies $\mathcal{C}_3$, 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 $\mathsf{NFA}(q)$ by changing its start state, as defined next.

Definition 5. If u is a prefix of q (and thus u is a state in $\mathsf{NFA}(q)$), then $\mathsf{S\text{-}NFA}(q, u)$ is the automaton obtained from $\mathsf{NFA}(q)$ by letting the initial state be u instead of the empty word. Note that $\mathsf{S\text{-}NFA}(q, \varepsilon) = \mathsf{NFA}(q)$. It may be helpful to think of the first S in $\mathsf{S\text{-}NFA}(q, u)$ as “$\underline{\text{S}}$tart 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 7, 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 6 (Automata on database instances)**.** Let **db** be a database instance. A *path (in* **db***)* is defined as a sequence $R_1(\underline{c_1}, c_2)$, $R_2(\underline{c_2}, c_3)$, $\ldots$, $R_n(\underline{c_n}, c_{n+1})$ of facts in **db**. Such a path is said to *start in* c_1. We call $R_1 R_2 \cdots R_n$ 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 $\mathsf{start}(q, \mathbf{r})$ 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 $\mathsf{NFA}(q)$. ☐

Example 4. 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 $\mathcal{L}^{\looparrowright}(q)$ is $RR\,(R)^{*}\,X$. We have $\mathsf{start}(q, \mathbf{r}_1) = \{0, 1\}$ and $\mathsf{start}(q, \mathbf{r}_2) = \{0\}$. ☐

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

Lemma 6. *Let q be a path query, and $\mathbf{db}$ a database instance. There exists a repair $\mathbf{r}^*$ of $\mathbf{db}$ such that for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$.*

Informally, we think of the next Lemma 7 as a *reification lemma*. The notion of *reifiable variable* was coined in [40, 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 7 (Reification Lemma for $\mathcal{C}_3$)**.** *Let q be a path query that satisfies $\mathcal{C}_3$. Then, for every database instance $\mathbf{db}$, the following are equivalent:*

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

Proof. $\boxed{1 \Longrightarrow 2}$ Assume (1). By Lemma 6, there exists a repair $\mathbf{r}^*$ of $\mathbf{db}$ such that for every repair $\mathbf{r}$ of $\mathbf{db}$, $\mathsf{start}(q, \mathbf{r}^*) \subseteq \mathsf{start}(q, \mathbf{r})$. Since $\mathbf{r}^*$ satisfies q, there is a path $R_1(\underline{c_1}, c_2)$, $R_2(\underline{c_2}, c_3)$, $\dots$, $R_n(\underline{c_n}, c_{n+1})$ in $\mathbf{r}^*$ such that $q = R_1 R_2 \cdots R_n$. Since q is accepted by $\mathsf{NFA}(q)$, we have $c_1 \in \mathsf{start}(q, \mathbf{r}^*)$. It follows that $c_1 \in \mathsf{start}(q, \mathbf{r})$ for every repair $\mathbf{r}$ of $\mathbf{db}$.

$\boxed{2 \Longrightarrow 1}$ Let $\mathbf{r}$ be any repair of $\mathbf{db}$. By our hypothesis that (2) holds true, there is some $c \in \mathsf{start}(q, \mathbf{r})$. Therefore, there is a path in $\mathbf{r}$ that starts in c and is accepted by $\mathsf{NFA}(q)$. Let p be the trace of this path. By Lemma 4, $p \in \mathcal{L}^{\looparrowright}(q)$. Since q satisfies $\mathcal{C}_3$ by the hypothesis of the current lemma, it follows by Lemma 5 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 6

We will use the following definition.

Definition 7 (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 $\mathsf{S\text{-}NFA}(q, u)$ 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 $\mathsf{S\text{-}NFA}(q, \mathbf{r})$ have R as their last relation name. ☐

Example 5. 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 $\mathsf{NFA}(q)$ has states $\{\varepsilon, R, RR, RRX\}$ and accepts the regular language $RR\,(R)^*\,X$. Since $\mathsf{S\text{-}NFA}(q, \varepsilon)$ 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 $\mathsf{S\text{-}NFA}(q, R)$, 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 $\mathsf{NFA}(q)$. ☐

Lemma 8. *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 $\mathsf{S\text{-}NFA}(q, u)$. Let vR be a state with $|v| > |u|$. Thus, by construction, $\mathsf{NFA}(q)$ has a backward transition with label ε from state vR to state uR.

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

From Lemma 8, it follows that $\mathsf{ST}_q(f, \mathbf{r})$ is completely determined by the shortest word in it.

Definition 8. Let q be a path query and $\mathbf{db}$ a database instance. For every fact $f \in \mathbf{db}$, we define:

$$\mathsf{cqaST}_q(f, \mathbf{db}) := \bigcap \{\mathsf{ST}_q(f, \mathbf{r}) \mid \mathbf{r} \text{ is a repair of } \mathbf{db} \text{ that contains } f\},$$

where $\bigcap X = \bigcap_{S \in X} S$. ☐

It is to be noted here that whenever $\mathbf{r}_1$ and $\mathbf{r}_2$ are repairs containing f, then by Lemma 8, $\mathsf{ST}_q(f, \mathbf{r}_1)$ and $\mathsf{ST}_q(f, \mathbf{r}_2)$ are comparable by set inclusion. Therefore, informally, $\mathsf{cqaST}_q(f, \mathbf{db})$ is the $\subseteq$-minimal states set of f over all repairs that contain f.

Definition 9 (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 9. *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. Construct a repair $\mathbf{r}^*$ as follows. For every block $\mathbf{blk}$ of $\mathbf{db}$, insert into $\mathbf{r}^*$ a fact f of $\mathbf{blk}$ such that $\mathsf{cqaST}_q(f, \mathbf{db}) = \bigcap\{\mathsf{cqaST}_q(g, \mathbf{db}) \mid g \in \mathbf{blk}\}$. More informally, we insert into $\mathbf{r}^*$ a fact f from $\mathbf{blk}$ with a states set that is $\subseteq$-minimal over all repairs and all facts of $\mathbf{blk}$. We first show the following claim.

Claim 1. For every fact f in $\mathbf{r}^*$, we have $\mathsf{ST}_q(f, \mathbf{r}^*) = \mathsf{cqaST}_q(f, \mathbf{db})$.

Proof. Let f_1 be an arbitrary fact in $\mathbf{r}^*$. We show $\mathsf{ST}_q(f_1, \mathbf{r}^*) = \mathsf{cqaST}_q(f_1, \mathbf{db})$.

$\boxed{\supseteq}$ Obvious, because $\mathbf{r}^*$ is itself a repair of $\mathbf{db}$ that contains f_1.

$\boxed{\subseteq}$ Let $f_1 = R_1(\underline{c_0}, c_1)$. Assume by way of a contradiction that there is $p_1 \in \mathsf{ST}_q(f_1, \mathbf{r}^*)$ such that $p_1 \notin \mathsf{cqaST}_q(f_1, \mathbf{db})$. Then, for some (possibly empty) prefix p_0 of q, there is a sequence:

$$p_0 \xrightarrow{\varepsilon} p_0' \xrightarrow{f_1=R_1(\underline{c_0},c_1)} p_1 \xrightarrow{\varepsilon} p_1' \xrightarrow{f_2=R_2(\underline{c_1},c_2)} p_2 \quad \cdots \quad p_{n-1} \xrightarrow{\varepsilon} p_{n-1}' \xrightarrow{f_n=R_n(\underline{c_{n-1}},c_n)} p_n = q, \tag{1}$$

where $f_1, f_2, \ldots, f_n \in \mathbf{r}^*$, for each $i \in \{1, \ldots, n\}$, $p_i = p_{i-1}' R_i$, and for each $i \in \{0, \ldots, n-1\}$, either $p_i' = p_i$ or p_i' is a strict prefix of p_i such that p_i' and p_i agree on their rightmost relation name. Informally, the sequence (1) represents an accepting run of $\mathsf{S\text{-}NFA}(q, p_0)$ in $\mathbf{r}^*$. Since $p_1 \in \mathsf{ST}_q(f_1, \mathbf{r}^*) \setminus \mathsf{cqaST}_q(f_1, \mathbf{db})$, we can assume a largest index $\ell \in \{1, \ldots, n\}$ such that $p_\ell \in \mathsf{ST}_q(f_\ell, \mathbf{r}^*) \setminus \mathsf{cqaST}_q(f_\ell, \mathbf{db})$. By construction of $\mathbf{r}^*$, there is a repair $\mathbf{s}$ such that $f_\ell \in \mathbf{s}$ and $\mathsf{ST}_q(f_\ell, \mathbf{s}) = \mathsf{cqaST}_q(f_\ell, \mathbf{db})$. Consequently, $p_\ell \notin \mathsf{ST}_q(f_\ell, \mathbf{s})$. We distinguish two cases:

Case that $\ell = n$. Thus, the run (1) ends with

$$\cdots \quad p_{\ell-1} \xrightarrow{\varepsilon} p_{\ell-1}' \xrightarrow{f_\ell=R_\ell(\underline{c_{\ell-1}},c_\ell)} p_\ell = q.$$

Thus, the rightmost relation name in q is R_ℓ. Since $f_\ell \in \mathbf{s}$, it is clear that $p_\ell \in \mathsf{ST}_q(f_\ell, \mathbf{s})$, a contradiction.

Case that $\ell < n$. Thus, the run (1) includes

$$\cdots \quad p_{\ell-1} \xrightarrow{\varepsilon} p_{\ell-1}' \xrightarrow{f_\ell=R_\ell(\underline{c_{\ell-1}},c_\ell)} p_\ell \xrightarrow{\varepsilon} p_\ell' \xrightarrow{f_{\ell+1}=R_{\ell+1}(\underline{c_\ell},c_{\ell+1})} p_{\ell+1} \quad \cdots,$$

where $\ell + 1$ can be equal to n. Clearly, $p_{\ell+1} \in \mathsf{ST}_q(f_{\ell+1}, \mathbf{r}^*)$. Assume without loss of generality that $\mathbf{s}$ contains $f_{\ell+1}' := R_{\ell+1}(\underline{c_\ell}, c_{\ell+1}')$, which is key-equal to $f_{\ell+1}$ (possibly $c_{\ell+1}' = c_{\ell+1}$). From $p_\ell \notin \mathsf{ST}_q(f_\ell, \mathbf{s})$, it follows $p_{\ell+1} \notin \mathsf{ST}_q(f_{\ell+1}', \mathbf{s})$. Consequently, $p_{\ell+1} \notin \mathsf{cqaST}_q(f_{\ell+1}', \mathbf{db})$. By our construction of r^*, we have $p_{\ell+1} \notin \mathsf{cqaST}_q(f_{\ell+1}, \mathbf{db})$. Consequently, $p_{\ell+1} \in \mathsf{ST}_q(f_{\ell+1}, \mathbf{r}^*) \setminus \mathsf{cqaST}_q(f_{\ell+1}, \mathbf{db})$, which contradicts that ℓ was chosen to be the largest such an index possible.

The proof of Claim 1 is now concluded. ◁

To conclude the proof of the lemma, let $\mathbf{r}$ be any repair of $\mathbf{db}$, and let $f \in \mathbf{r}^*$ and $f' \in \mathbf{r}$ be two key-equal facts in $\mathbf{db}$. By Claim 1 and the construction of $\mathbf{r}^*$, we have that $\mathsf{ST}_q(f, \mathbf{r}^*) = \mathsf{cqaST}_q(f, \mathbf{db}) \subseteq \mathsf{cqaST}_q(f', \mathbf{db}) \subseteq \mathsf{ST}_q(f', \mathbf{r})$, as desired. ☐

We can now give the proof of Lemma 6.

Proof of Lemma 6. Let $\mathbf{db}$ be a database instance. Then by Lemma 9, 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. ☐

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

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

$$N \leftarrow N \cup \underbrace{\{\langle c, u\rangle\}}_{\text{forward}} \cup \underbrace{\{\langle c, w\rangle \mid \mathsf{NFA}(q) \text{ has a backward transition from } w \text{ to } u\}}_{\text{backward}}.$$

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 $\mathcal{C}_3$.

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>`

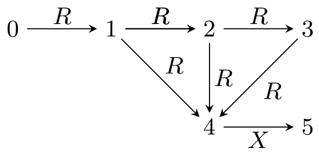


Figure 6: Example run of our algorithm for $q = RRX$, on the database instance **db** shown at the right.

6 Complexity Upper Bounds

We now show the complexity upper bounds of Theorem 3.

6.1 A PTIME Algorithm for $\mathcal{C}_3$

We now specify a polynomial-time algorithm for $\mathsf{CERTAINTY}(q)$, for path queries q that satisfy condition $\mathcal{C}_3$. The algorithm is based on the automata defined in Definition 5, and uses the concept defined next.

Definition 10 (Relation $\vdash_q$)**.** Let q be a path query and **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 **db** has a path that starts in c and is accepted by $\mathsf{S\text{-}NFA}(q, u)$. ☐

An algorithm that decides the relation $\vdash_q$ can be used to solve $\mathsf{CERTAINTY}(q)$ for path queries satisfying $\mathcal{C}_3$. Indeed, by Lemma 7, for path queries satisfying $\mathcal{C}_3$, **db** is a "yes"-instance for the problem $\mathsf{CERTAINTY}(q)$ 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 $\mathsf{S\text{-}NFA}(q, q)$. Then, the *Iterative Rule* is executed until N remains unchanged; it intrinsically reflects the constructive proof of Lemma 9: $\mathbf{db} \vdash_q \langle c, u\rangle$ if and only if for every fact $f = R(\underline{c}, d) \in \mathbf{db}$, we have $uR \in \mathsf{cqaST}_q(f, \mathbf{db})$. Figure 6 shows an example run of the algorithm in Figure 5. The next lemma states the correctness of the algorithm.

Lemma 10. *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 $c \in \mathsf{adom}(\mathbf{db})$ and every prefix u of q,*

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

Proof. $\boxed{\Longleftarrow}$ Proof by contraposition. Assume $\langle c, u\rangle \notin N$. The proof shows the construction of a repair $\mathbf{r}$ of **db** such that $\mathbf{r}$ has no path that starts in c and is accepted by $\mathsf{S\text{-}NFA}(q, u)$. Such a repair shows $\mathbf{db} \not\vdash_q \langle c, u\rangle$.

We explain which fact of an arbitrary block $R(\underline{a}, *)$ of **db** will be inserted in $\mathbf{r}$. 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 $\mathbf{r}$. 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 $\mathbf{r}$. We remark that this repair $\mathbf{r}$ is constructed in exactly the same way as the repair $\mathbf{r}^*$ built in the proof of Lemma 9.

Assume for the sake of contradiction that there is a path π in $\mathbf{r}$ that starts in c and is accepted by $\mathsf{S\text{-}NFA}(q, u)$. Let $\pi := R_1(\underline{c_0}, c_1)$, $R_2(\underline{c_1}, c_2)$, $\dots$, $R_n(\underline{c_{n-1}}, c_n)$ 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 $i \in \{1, \dots, n\}$ 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 $R_i(\underline{c_{i-1}}, c_i) \in \mathbf{r}$.

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

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"*.

$\boxed{\Longrightarrow}$ 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 $\mathsf{S\text{-}NFA}(q,q)$, 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 $\mathbf{db}$ 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 $\mathbf{r}$ be an arbitrary repair of $\mathbf{db}$. 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 $\mathbf{r}$ has a path that starts in d_i and is accepted by $\mathsf{S\text{-}NFA}(q,uR)$. Clearly, this path can be left extended with $R(\underline{c},d_i)$, and this left extended path is accepted by $\mathsf{S\text{-}NFA}(q,u)$. Note incidentally that the path in $\mathbf{r}$ may already use $R(\underline{c},d_i)$, in which case the path is cyclic. Since $\mathbf{r}$ 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 exists 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, $\mathsf{S\text{-}NFA}(q,u)$ has an ε-transition from state u to vS. Let $\mathbf{r}$ be an arbitrary repair of $\mathbf{db}$. By the reasoning in the previous case, $\mathbf{r}$ has a path that starts in c and is accepted by $\mathsf{S\text{-}NFA}(q,vS)$. We claim that $\mathbf{r}$ has a path that starts in c and is accepted by $\mathsf{S\text{-}NFA}(q,u)$. Indeed, $\mathsf{S\text{-}NFA}(q,u)$ can use the ε-transition to reach the state vS, and then behave like $\mathsf{S\text{-}NFA}(q,vS)$. This concludes the proof. ☐

The following corollary is now immediate.

Corollary 1. *Let q be a path query. Let $\mathbf{db}$ be a database instance, and $c \in \mathsf{adom}(\mathbf{db})$. Then, the following are equivalent:*

1. $c \in \mathsf{start}(q,\mathbf{r})$ *for every repair* $\mathbf{r}$ *of* $\mathbf{db}$*; 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 11. *For each path query q satisfying $\mathcal{C}_3$,* $\mathsf{CERTAINTY}(q)$ *is expressible in Least Fixpoint Logic, and hence is in* **PTIME**.

Proof. For a path query q, define the following formula in LFP [33]:

$$\psi_q(s,t) := \left[\mathbf{lfp}_{N,x,z}\varphi_q(N,x,z)\right](s,t), \tag{2}$$

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 $\mathbf{db}$ and constant c, $\mathbf{db} \models \alpha(c)$ if and only if $c \in \mathsf{adom}(\mathbf{db})$. 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 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 $\mathtt{q}$, $\mathtt{uR}$, $\mathtt{uv}$ from first-order variables x, z. It is easily verified that the LFP query (2) expresses the algorithm of Figure 5. ☐

Since the formula (2) in the proof of Lemma 11 uses universal quantification, it is not in Existential Least Fixpoint Logic, which is equal to $\mathsf{DATALOG}_{\neg}$ [33, Theorem 10.18].

6.2 FO-Rewritability for $\mathcal{C}_1$

We now show that if a path query q satisfies $\mathcal{C}_1$, then $\mathsf{CERTAINTY}(q)$ is in **FO**, and a first-order rewriting for q can be effectively constructed.

Definition 11 (First-order rewriting)**.** If q is a Boolean query such that $\mathsf{CERTAINTY}(q)$ 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 $\mathsf{CERTAINTY}(q)$; and
2. **db** satisfies ψ. ☐

Definition 12. If $q = \{R_1(\underline{x_1}, x_2),\ R_2(\underline{x_2}, x_3),\ \dots,\ R_k(\underline{x_k}, x_{k+1})\}$, $k \geq 1$, and c is a constant, then $q_{[c]}$ is the Boolean conjunctive query $q_{[c]} := \{R_1(\underline{c}, x_2), R_2(\underline{x_2}, x_3), \dots, R_k(\underline{x_k}, x_{k+1})\}$. ☐

Lemma 12. *For every nonempty path query q and constant c, the problem $\mathsf{CERTAINTY}(q_{[c]})$ 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, $n \to n+1$. 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,

$$\exists z\,(\varphi(z) \wedge z = d) \text{ is a first-order rewriting for } p_{[d]}. \tag{3}$$

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

$$\psi(x) = \exists y\,(R(\underline{x}, y)) \wedge \forall z\,(R(\underline{x}, z) \to \varphi(z))\,. \tag{4}$$

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 $\mathsf{CERTAINTY}(q_{[c]})$ 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 $\exists y\,(R(\underline{x}, y))$ in (4), 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 (3) 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 $\mathsf{CERTAINTY}(q_{[c]})$. Then **db** must obviously satisfy $\exists y\,(R(\underline{c}, y))$. Therefore, **db** includes a block $R(\underline{c}, *)$. Let **r** be an arbitrary repair of **db**. There exists d such that $R(\underline{c}, d) \in \mathbf{r}$. Since **r** satisfies $q_{[c]}$, it follows that **r** satisfies $p_{[d]}$. Since **r** is an arbitrary repair, the induction hypothesis (3) 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 13. *For every path query q that satisfies $\mathcal{C}_1$, the problem $\mathsf{CERTAINTY}(q)$ is in **FO**, and a first-order rewriting for q can be effectively constructed.*

Proof. By Lemmas 5 and 7, a database instance **db** is a "yes"-instance for $\mathsf{CERTAINTY}(q)$ if and only if there is a constant c (which depends on **db**) such that **db** is a "yes"-instance for $\mathsf{CERTAINTY}(q_{[c]})$. By Lemma 12, 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 $\mathcal{C}_2$

We show that $\mathsf{CERTAINTY}(q)$ is in **NL** if q satisfies $\mathcal{C}_2$ by expressing it in linear Datalog with stratified negation. The proof will use the syntactic characterization of $\mathcal{C}_2$ established in Lemma 3.

Lemma 14. *For every path query q that satisfies $\mathcal{C}_2$, the problem $\mathsf{CERTAINTY}(q)$ is in linear Datalog with stratified negation (and hence in **NL**).*

In the remainder of this section, we develop the proof of Lemma 14.

Definition 13. Let q be a path query. We define $\mathsf{NFA}^{\min}(q)$ as the automaton that accepts w if w is accepted by $\mathsf{NFA}(q)$ and no proper prefix of w is accepted by $\mathsf{NFA}(q)$. ☐

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

Example 6. Let $q = RXRYR$. Then, $RXRYRYR$ is accepted by $\mathsf{NFA}(q)$, but not by $\mathsf{NFA}^{\min}(q)$, because the proper prefix $RXRYR$ is also accepted by $\mathsf{NFA}(q)$. ☐

Definition 14. 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 15. *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 16. *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 ($s = s_1 \cdot v_2$):

$$\begin{array}{l}\left(s\,(uv)^{\ell_1}\,uv_1\right)\cdot v_2\,(uv)^{\ell_2}\,wv_1\cdot\underline{v_2\,(uv)^{\ell_2}\,wv_1}\cdot(v_2);\ \text{or}\\ (s_1)\cdot v_2\,(uv)^{\ell}\,wv_1\cdot\underline{v_2\,(uv)^{\ell}\,wv_1}\cdot(v_2).\end{array}$$

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

Definition 15. Let $\mathbf{db}$ be a database instance, and q a path query.

For $a, b \in \mathsf{adom}(\mathbf{db})$, we write $\mathbf{db} \models a \xrightarrow{q} b$ if there exists a path in $\mathbf{db}$ from a to b with trace q. Even more formally, $\mathbf{db} \models a \xrightarrow{q} b$ if $\mathbf{db}$ contains facts $R_1(\underline{a_1}, a_2), R_2(\underline{a_2}, a_3), \dots, R_{|q|}(\underline{a_{|q|}}, a_{|q|+1})$ such that $R_1R_2 \cdots R_{|q|} = q$. We write $\mathbf{db} \models a \xrightarrow{q_1} b \xrightarrow{q_2} c$ as a shorthand for $\mathbf{db} \models a \xrightarrow{q_1} b$ and $\mathbf{db} \models b \xrightarrow{q_2} c$.

We write $\mathbf{db} \models a \overset{q}{\twoheadrightarrow} b$ if there exists a *consistent path* in $\mathbf{db}$ from a to b with trace q, where a path is called consistent if it does not contain two distinct key-equal facts.

A constant $c \in \mathsf{adom}(\mathbf{db})$ is called *terminal for q in* $\mathbf{db}$ if for some (possibly empty) proper prefix p of q, there is a consistent path in $\mathbf{db}$ with trace p that cannot be right extended to a consistent path in $\mathbf{db}$ with trace q. ☐

Note that for every $c \in \mathsf{adom}(\mathbf{db})$, we have $c \overset{\varepsilon}{\twoheadrightarrow} c$. Clearly, if q is self-join-free, then $c \xrightarrow{q} d$ implies $c \overset{q}{\twoheadrightarrow} d$ (the converse implication holds vacuously true).

Example 7. Let $\mathbf{db} = \{R(\underline{c}, d), S(\underline{d}, c), R(\underline{c}, e), T(\underline{e}, f)\}$. Then, c is terminal for $RSRT$ in $\mathbf{db}$ because the path $R(\underline{c}, d), S(\underline{d}, c)$ cannot be right extended to a consistent path with trace $RSRT$, because d has no outgoing T-edge. Note incidentally that $\mathbf{db} \models c \overset{RS}{\twoheadrightarrow} c \overset{RT}{\twoheadrightarrow} f$, but $\mathbf{db} \not\models c \overset{RSRT}{\twoheadrightarrow} f$. ☐

Lemma 17. *Let $\mathbf{db}$ be a database instance, and $c \in \mathsf{adom}(\mathbf{db})$. Let q be a path query. Then, c is terminal for q in $\mathbf{db}$ if and only if $\mathbf{db}$ is a "no"-instance of $\mathsf{CERTAINTY}(q_{[c]})$, with $q_{[c]}$ as defined by Definition 12.*

Proof. $\boxed{\Longrightarrow}$ Straightforward. $\boxed{\Longleftarrow}$ Assume $\mathbf{db}$ is a "no"-instance of $\mathsf{CERTAINTY}(q_{[c]})$. Then, there is a repair $\mathbf{r}$ of $\mathbf{db}$ such that $\mathbf{r} \not\models q_{[c]}$. The empty path is a path in $\mathbf{r}$ that starts in c and has trace ε, which is a prefix of q. We can therefore assume a longest prefix p of q such there exists a path π in $\mathbf{r}$ that starts in c and has trace p. Since $\mathbf{r}$ is consistent, π is consistent. From $\mathbf{r} \not\models q_{[c]}$, it follows that p is a proper prefix of q. By Definition 15, c is terminal for q in $\mathbf{db}$. ☐

We can now give the proof of Lemma 14.

Proof of Lemma 14. Assume q satisfies $\mathcal{C}_2$. By Lemma 3, 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 $q = s\,(uv)^{k-1}\,wv$.

We first define a unary predicate P (which depends on q) such that $\mathbf{db} \models P(d)$ if for some $\ell \geq 0$, there are constants $d_0, d_1, \dots, d_\ell \in \mathsf{adom}(\mathbf{db})$ with $d_0 = d$ such that:

(i) $\mathbf{db} \models d_0 \xrightarrow{uv} d_1 \xrightarrow{uv} d_2 \xrightarrow{uv} \cdots \xrightarrow{uv} d_\ell$;

(ii) for every $i \in \{0, 1, \dots, \ell\}$, d_i is terminal for wv in $\mathbf{db}$; and

(iii) either d_ℓ is terminal for uv in $\mathbf{db}$, or $d_\ell \in \{d_0, \dots, d_{\ell-1}\}$.

Claim 2. The definition of the predicate P does not change if we replace item (i) by the stronger requirement that for every $i \in \{0, 1, \dots, \ell - 1\}$, there exists a path π_i from d_i to d_{i+1} with trace uv such that the composed path $\pi_0 \cdot \pi_1 \cdots \pi_{\ell-1}$ is consistent.

Proof. It suffices to show the following statement by induction on increasing l:

> whenever there exist $l \geq 1$ and constants $d_0, d_1, \dots, d_l$ with $d_0 = d$ such that conditions (i), (ii), and (iii) hold, there exist another constant $k \geq 1$ and constants $c_0, c_1, \dots, c_k$ with $c_0 = d$ such that conditions (i), (ii), and (iii) hold, and, moreover, for each $i \in \{0, 1, \dots, k-1\}$, there exists a path π_i from c_i to c_{i+1} such that the composed path $\pi_0 \cdot \pi_1 \cdots \pi_{k-1}$ is consistent.

Basis $l = 1$. Then we have $\mathbf{db} \models d_0 \xrightarrow{uv} d_1$, witnessed by a path π_0. Since uv is self-join-free, the path π_0 is consistent. The claim thus follows with $k = l = 1$, $c_0 = d_0$ and $c_1 = d_1$.

Inductive step $l \to l + 1$. Assume that the statement holds for any integer in $\{1, 2, \dots, l\}$. Suppose that there exist $l \geq 2$ and constants $d_0, d_1, \dots, d_{l+1}$ with $d_0 = d$ such that conditions (i), (ii), and (iii) hold.

For $i \in \{0, \dots, l\}$, let π_i be a path with trace uv from d_i to d_{i+1} in $\mathbf{db}$. The claim holds if the composed path $\pi_0 \cdot \pi_1 \cdots \pi_l$ is consistent, with $k = l + 1$ and $c_i = d_i$ for $i \in \{0, 1, \dots, l + 1\}$.

Now, assume that for some $i < j$, the paths that show $\mathbf{db} \models d_i \xrightarrow{uv} d_{i+1}$ and $\mathbf{db} \models d_j \xrightarrow{uv} d_{j+1}$ contain, respectively, $R(\underline{a}, b_1)$ and $R(\underline{a}, b_2)$ with $b_1 \neq b_2$. It is easily verified that

$$\mathbf{db} \models d_0 \xrightarrow{uv} d_1 \xrightarrow{uv} d_2 \xrightarrow{uv} \cdots \xrightarrow{uv} d_i \xrightarrow{uv} d_{j+1} \xrightarrow{uv} \cdots \xrightarrow{uv} d_{l+1},$$

where the number of uv-steps is strictly less than $l + 1$. Informally, we follow the original path until we reach $R(\underline{a}, b_1)$, but then follow $R(\underline{a}, b_2)$ instead of $R(\underline{a}, b_1)$, and continue on the path that proves $\mathbf{db} \models d_j \xrightarrow{uv} d_{j+1}$. Then the claim holds by applying the inductive hypothesis on constants $d_0, d_1, \dots, d_i, d_{j+1}, \dots, d_{l+1}$.

The proof is now complete. ☐

Since we care about the expressibility of the predicate P in Datalog, Claim 2 is not cooked into the definition of P. The idea is the same as in an **NL**-algorithm for reachability: if there exists a directed path from s to t, then there is such a path without repeated vertices; but we do not care for repeated vertices when computing reachability.

Claim 3. The definition of predicate P does not change if we require that for $i \in \{0, 1, \dots, \ell - 1\}$, d_i is not terminal for uv in $\mathbf{db}$.

Proof. Assume that for some $0 \leq i < \ell$, d_i is terminal for uv in $\mathbf{db}$. Then, all conditions in the definition are satisfied by choosing ℓ equal to j. ☐

Claim 3 is not cooked into the definition of P to simplify the the encoding of P in Datalog.

Next, we define a unary predicate O such that $\mathbf{db} \models O(c)$ for a constant c if $c \in \mathsf{adom}(\mathbf{db})$ and one of the following holds true:

1. c is terminal for $s\,(uv)^{k-1}$ in $\mathbf{db}$; or
2. there is a constant $d \in \mathsf{adom}(\mathbf{db})$ such that both $\mathbf{db} \models c \xrightarrow{s(uv)^{k-1}} d$ and $\mathbf{db} \models P(d)$.

Claim 4. Let $c \in \mathsf{adom}(\mathbf{db})$. The following are equivalent:

(I) there is a repair $\mathbf{r}$ of $\mathbf{db}$ that contains no path that starts in c and whose trace is in the language of the regular expression $s\,(uv)^{k-1}\,(uv)^{*}\,wv$; and

(II) $\mathbf{db} \models O(c)$.

Proof. Let $wv = S_0 S_1 \cdots S_{m-1}$ and $uv = R_0 R_1 \cdots R_{n-1}$.

$\boxed{\text{(I)} \implies \text{(II)}}$ Assume that item (I) holds true. Let the first relation name of s be R_i. Starting from c, let π be a maximal (possibly infinite) path in $\mathbf{r}$ that starts in c and has trace $R_i R_{i+1} R_{i+2} \cdots$, where addition is modulo n. Since $\mathbf{r}$ is consistent, π is deterministic. Since $\mathbf{r}$ is finite, π contains only finitely many distinct edges. Therefore, π ends either in a loop or in an edge $R_j(\underline{d}, e)$ such that $\mathbf{db} \models \neg\exists y R_{j+1}(\underline{e}, y)$ (recall that $\mathbf{r}$ contains a fact from every block of $\mathbf{db}$). Assume that π has a prefix π' with trace $s\,(uv)^{k-1}$; if e occurs at the non-primary key position of the last R_{n-1}-fact of π' or of any R_{n-1}-fact occurring afterwards in π, then it follows from item (I) that there exist a (possibly empty) prefix pS_j of wv and a constant $f \in \mathsf{adom}(\mathbf{r})$ such that $\mathbf{r} \models e \xrightarrow{p} f$ and $\mathbf{db} \models \neg\exists y S_j(\underline{f}, y)$. It is now easily verified that $\mathbf{db} \models O(c)$.

$\boxed{\text{(II)} \implies \text{(I)}}$ Assume $\mathbf{db} \models O(c)$. It is easily verified that the desired result holds true if c is terminal for $s\,(uv)^{k-1}$ in $\mathbf{db}$. Assume from here on that c is not terminal for $s\,(uv)^{k-1}$ in $\mathbf{db}$. That is, for every repair $\mathbf{r}$ of $\mathbf{db}$, there is a constant d such that $\mathbf{r} \models c \xrightarrow{s(uv)^{k-1}} d$. Then, there is a consistent path α with trace $s\,(uv)^{k-1}$ from c to some constant $d \in \mathsf{adom}(\mathbf{db})$ such that $\mathbf{db} \models P(d)$, using the stronger definition of P implied by Claims 2 and 3. Let $d_0, \ldots, d_\ell$ be as in our (stronger) definition of $P(d)$, that is, first, $d_1, \ldots, d_{\ell-1}$ are not terminal for uv in $\mathbf{db}$ (cf. Claim 3), and second, there is a $\subseteq$-minimal consistent subset π of $\mathbf{db}$ such that $\pi \models d_0 \xrightarrow{uv} d_1 \xrightarrow{uv} d_2 \xrightarrow{uv} \cdots \xrightarrow{uv} d_\ell$ (cf. Claim 2). We construct a repair $\mathbf{r}$ as follows:

1. insert into $\mathbf{r}$ all facts of π;
2. for every $i \in \{0, \ldots, \ell\}$, d_i is terminal for wv in $\mathbf{db}$. We ensure that $\mathbf{r} \models d_i \xrightarrow{S_0 S_1 \cdots S_{j_i}} e_i$ for some $j_i \in \{0, \ldots, m-2\}$ and some constant e_i such that $\mathbf{db} \models \neg\exists y S_{j_i+1}(\underline{e_i}, y)$;
3. if d_ℓ is terminal for uv in $\mathbf{db}$, then we ensure that $\mathbf{r} \models d_\ell \xrightarrow{R_0 R_1 \cdots R_j} e$ for some $j \in \{0, \ldots, n-2\}$ and some constant e such that $\mathbf{db} \models \neg\exists y S_{j+1}(\underline{e}, y)$;
4. insert into $\mathbf{r}$ the facts of α that are not key-equal to a fact already in $\mathbf{r}$; and
5. complete $\mathbf{r}$ into a $\subseteq$-maximal consistent subset of $\mathbf{db}$.

Since $\mathbf{r}$ is a repair of $\mathbf{db}$, there exists a path δ with trace $s\,(uv)^{k-1}$ in $\mathbf{r}$ that starts from c. If $\delta \neq \alpha$, then δ must contain a fact of π that was inserted in step 1. Consequently, no matter whether $\delta = \alpha$ or $\delta \neq \alpha$, the endpoint of δ belongs to $\{d_0, \ldots, d_\ell\}$. It follows that there is a (possibly empty) path from δ's endpoint to d_ℓ whose trace is of the form $(uv)^{*}$. Two cases can occur:

- d_ℓ is terminal for uv in $\mathbf{db}$.
- d_ℓ is not terminal for uv in $\mathbf{db}$. Then there is $j \in \{0, \ldots, \ell-1\}$ such that $d_j = d_\ell$. Then, there is a path of the form $(uv)^{*}$ that starts from δ's endpoint and eventually loops.

Since, by construction, each d_i is terminal for wv in $\mathbf{r}$, it will be the case that δ cannot be extended to a path in $\mathbf{r}$ whose trace is of the form $s\,(uv)^{k}\,(uv)^{*}\,wv$. ☐

Claim 5. The unary predicate O is expressible in linear Datalog with stratified negation.

Proof. The construction of the linear Datalog program is straightforward. Concerning the computation of predicates P and O, note that it can be checked in **FO** whether or not a constant c is terminal for some path query q, by Lemmas 12 and 17. The only need for recursion comes from condition (i) in the definition of the predicate P, which searches for a directed path of a particular form. We give a program for $q = UVUVWV$, where `c(X)` states that `X` is a constant, and `ukey(X)` states that `X` is the primary key of some U-fact. `consistent(X1, X2, X3, X4)` is true if either `X1` $\neq$ `X3` or `X2` $=$ `X4` (or both).

```
uvterminal(X) :- c(X), not ukey(X).
uvterminal(X) :- u(X,Y), not vkey(Y).
wvterminal(X) :- c(X), not wkey(X).
wvterminal(X) :- w(X,Y), not vkey(Y).

uv2terminal(X) :- uvterminal(X).
uv2terminal(X1) :- u(X1,X2), v(X2,X3), uvterminal(X3).

uvpath(X1,X3) :- u(X1,X2), v(X2,X3), wvterminal(X1), wvterminal(X2), wvterminal(X3).
uvpath(X1,X4) :- uvpath(X1,X2), u(X2,X3), v(X3,X4), wvterminal(X3), wvterminal(X4).

p(X) :- uvterminal(X), wvterminal(X). %%% the empty path.
p(X) :- uvpath(X,Y), uvterminal(Y).
p(X) :- uvpath(X,Y), uvpath(Y,Y). %%% p and uvpath are not mutually recursive.

o(X) :- uv2terminal(X).
o(X1) :- u(X1,X2), v(X2,X3), u(X3,X4), v(X4,X5), consistent(X1,X2,X3,X4), consistent(X2,X3,X4,X5), p(X5).
```

The above program is in linear Datalog with stratified negation. It is easily seen that any path query satisfying $\mathcal{B}_{2b}$ admits such a program for the predicate O. ☐

By Lemmas 7, 15, and 16, 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** that contains no path that starts in c_i and whose trace is in the language of the regular expression $s\,(uv)^{k-1}\,(uv)^{*}\,wv$.

By Claim 4, item (b) holds true if and only if for every $c \in \mathsf{adom}(\mathbf{db})$, $\mathbf{db} \models \neg O(c)$. It follows from Claim 5 that the latter test is in linear Datalog with stratified negation, which concludes the proof of Lemma 14. ☐

7 Complexity Lower Bounds

In this section, we show the complexity lower bounds of Theorem 3. For a path query $q = \{R_1(x_1, x_2), \ldots, R_k(x_k, x_{k+1})\}$ and constants a, b, we define the following database instances:

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

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 $\mathcal{C}_1$, then CERTAINTY(q) is **NL**-hard, and therefore not in **FO**.

Lemma 18. *If a path query q violates $\mathcal{C}_1$, then* CERTAINTY(q) *is* **NL**-*hard.*

Proof. Assume that q does not satisfy $\mathcal{C}_1$. 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 8 shows an example of the above construction. Observe that the only conflicts in $\mathbf{db}$ occur in R-facts outgoing from a same vertex.

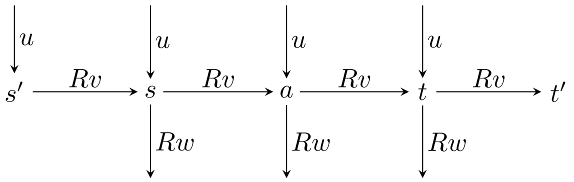


Figure 8: Database instance for the **NL**-hardness reduction from the graph G with $V = \{s, a, t\}$ and $E = \{(s, a), (a, t)\}$.

We now show that there exists a directed path from s to t in G if and only if there exists a repair of $\mathbf{db}$ 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.

$\boxed{\impliedby}$ Proof by contradiction. Suppose that there is no directed path from s to t in G. Let $\mathbf{r}$ be any repair of $\mathbf{db}$; we will show that $\mathbf{r}$ satisfies q. Indeed, there exists a maximal path $P = x_0, x_1, \dots, x_n$ 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 19. *If a path query q violates $\mathcal{C}_3$, then* $\mathsf{CERTAINTY}(q)$ *is* **coNP***-hard.*

Proof. If q does not satisfy $\mathcal{C}_3$, 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 $\mathsf{CERTAINTY}(q)$. 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 $\mathbf{db}$ for $\mathsf{CERTAINTY}(q)$ 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 9 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 $\mathbf{db}$. 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 $\mathbf{db}$ that does not satisfy q.

$\boxed{\implies}$ 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 $\phi_z^{\perp}[RvRw]$;

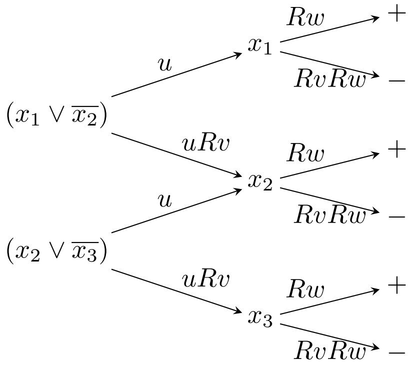


Figure 9: Database instance for the **coNP**-hardness reduction from the formula $\psi = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3})$.

- 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 $\mathcal{C}_3$. We conclude by contradiction that $\mathbf{r}$ falsifies q.

$\boxed{\Longleftarrow}$ Suppose that there exists a repair $\mathbf{r}$ of $\mathbf{db}$ 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 20. *If a path query q violates $\mathcal{C}_2$, then* CERTAINTY(p) *is* **PTIME***-hard.*

Proof. Suppose q violates $\mathcal{C}_2$. If q also violates $\mathcal{C}_3$, then the problem CERTAINTY(q) is **PTIME**-hard since it is **coNP**-hard by Lemma 19. 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 [18]:

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 $\mathbf{db}$ for CERTAINTY(q) as follows:

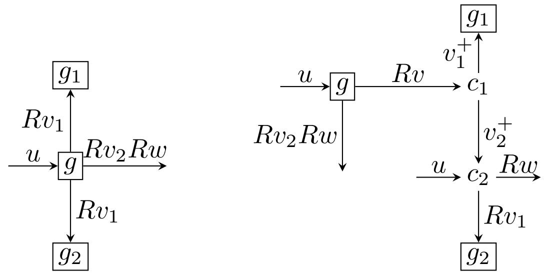
- for the output gate o, we add $\phi_\perp^o[uRv_1]$;
- for each input variable x with $\sigma(x) = 1$, we add $\phi_x^\perp[Rv_2Rw]$;
- for each gate g, we add $\phi_\perp^g[u]$ and $\phi_g^\perp[Rv_2Rw]$;
- for each AND gate $g = g_1 \wedge g_2$, we add
$$\phi_g^{g_1}[Rv_1] \cup \phi_g^{g_2}[Rv_1].$$
  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

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

  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 10. We next show that the output gate o is evaluated to 1 under σ if and only if each repair of **db** satisfies q.



(a) AND gate (b) OR gate

Figure 10: Gadgets for the **PTIME**-hardness reduction.

$\boxed{\implies}$ 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 $\phi_{c_2}^\perp[Rw] \not\subseteq \mathbf{r}$, 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.

$\boxed{\impliedby}$ 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 $k \geq 1$, $\mathcal{L}^{\looparrowright}(q)$ contains $uRv_1\left(Rv_2\right)^k Rw$. It is now easily verified that for large enough values of k, uRv_1Rv_2w is not a factor of $uRv_1\left(Rv_2\right)^k Rw$. By Lemmas 5 and 19, $\mathsf{CERTAINTY}(q)$ is **coNP**-hard. ☐

8 Path Queries with Constants

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

Definition 16 (Generalized path queries)**.** A *generalized path query* is a Boolean conjunctive query of the following form:

$$q = \{R_1(\underline{s_1}, s_2), R_2(\underline{s_2}, s_3), \dots, R_k(\underline{s_k}, s_{k+1})\}, \tag{5}$$

where s_1, s_2,..., s_{k+1} are constants or variables, all distinct, and R_1, R_2,..., 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

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

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

Example 8. 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 $\mathsf{CERTAINTY}(q)$ is in **FO**. This explains why the complexity classification in the remainder of this section will only depend on $\mathsf{char}(q)$.

Lemma 21. *For any generalized path query q, $\mathsf{CERTAINTY}(p)$ 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 16.

Definition 17. Let $q = \{R_1(\underline{x_1}, x_2)$, $R_2(\underline{x_2}, x_3)$, $\dots$, $R_k(\underline{x_k}, x_{k+1})\}$ 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 18 (Prefix homomorphism)**.** Let

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

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 $i \in \{1, \dots, k\}$, $R_i(\underline{\theta(s_i)}, \theta(s_{i+1})) \in p$. Such a homomorphism is a *prefix homomorphism* if $\theta(s_1) = t_1$. ☐

Example 9. 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 $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ 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, $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. Likewise, the following theorem degenerates to Theorem 3 for path queries without constants.

Theorem 4. *For every generalized path query q, the following complexity upper bounds obtain:*

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

The following complexity lower bounds obtain:

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

Finally, the proof of Theorem 4 reveals that for generalized path queries q containing at least one constant, the complexity of $\mathsf{CERTAINTY}(q)$ exhibits a trichotomy (instead of a tetrachotomy as in Theorem 4).

Theorem 5. *For any generalized path query q containing at least one constant, the problem* $\mathsf{CERTAINTY}(q)$ *is either in* **FO***,* **NL***-complete, or* **coNP***-complete.*

9 Related Work

Inconsistencies in databases have been studied in different contexts [8, 21, 22]. Consistent query answering (CQA) was initiated by the seminal work by Arenas, Bertossi, and Chomicki [3]. After twenty years, their contribution was acknowledged in a *Gems of PODS session* [5]. An overview of complexity classification results in CQA appeared recently in the *Database Principles* column of SIGMOD Record [41].

The term $\mathsf{CERTAINTY}(q)$ was coined in [39] 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 $\mathsf{CERTAINTY}(q)$ for the class of self-join-free Boolean conjunctive queries started with the work by Fuxman and Miller [17], and was further pursued in [23, 26, 27, 28, 30, 32], which eventually revealed that the complexity of $\mathsf{CERTAINTY}(q)$ 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. It remains decidable whether or not $\mathsf{CERTAINTY}(q)$ is in **FO** for self-join-free Boolean conjunctive queries with negated atoms [29], with respect to multiple keys [31], and with unary foreign keys [20], all assuming that q is self-join-free.

Little is known about $\mathsf{CERTAINTY}(q)$ beyond self-join-free conjunctive queries. Fontaine [14] showed that if we strengthen Conjecture 1 from conjunctive queries to unions of conjunctive queries, then it implies Bulatov's dichotomy theorem for conservative CSP [6]. This relationship between CQA and CSP was further explored in [34]. In [1], the authors show the **FO** boundary for $\mathsf{CERTAINTY}(q)$ for constant-free Boolean conjunctive queries q using a single binary relation name with a singleton primary key. Figueira et al. [13] have recently discovered a simple fixpoint algorithm that solves $\mathsf{CERTAINTY}(q)$ when q is a self-join free conjunctive query or a path query such that $\mathsf{CERTAINTY}(q)$ is in **PTIME**.

The counting variant of the problem $\mathsf{CERTAINTY}(q)$, denoted $\sharp\mathsf{CERTAINTY}(q)$, asks to count the number of repairs that satisfy some Boolean query q. For self-join-free Boolean conjunctive queries, $\sharp\mathsf{CERTAINTY}(q)$ exhibits a dichotomy between **FP** and ♯**PTIME**-complete [37]. This dichotomy has been shown to extend to self-joins if primary keys are singletons [38], and to functional dependencies [7].

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 [2, 9, 10, 11, 12, 19, 24, 35, 36].

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 $\mathsf{CERTAINTY}(q)$ 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, 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] F. N. Afrati, P. G. Kolaitis, and A. Vasilakopoulos. Consistent answers of conjunctive queries on graphs. *CoRR*, abs/1503.00650, 2015.

[2] A. Amezian El Khalfioui, J. Joertz, D. Labeeuw, G. Staquet, and J. Wijsen. Optimization of answer set programs for consistent query answering by means of first-order rewriting. In *CIKM*, pages 25–34. ACM, 2020.

[3] M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79. ACM Press, 1999.

[4] C. Berkholz, J. Keppeler, and N. Schweikardt. Answering conjunctive queries under updates. In *PODS*, pages 303–318. ACM, 2017.

[5] L. E. Bertossi. Database repairs and consistent query answering: Origins and further developments. In *PODS*, pages 48–58. ACM, 2019.

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

[7] M. Calautti, E. Livshits, A. Pieris, and M. Schneider. Counting database repairs entailing a query: The case of functional dependencies. In *PODS*, pages 403–412. ACM, 2022.

[8] J. Chomicki and J. Marcinkowski. Minimal-change integrity maintenance using tuple deletions. *Inf. Comput.*, 197(1-2):90–121, 2005.

[9] J. Chomicki, J. Marcinkowski, and S. Staworko. Hippo: A system for computing consistent answers to a class of SQL queries. In *EDBT*, volume 2992 of *Lecture Notes in Computer Science*, pages 841–844. Springer, 2004.

[10] A. A. Dixit and P. G. Kolaitis. A sat-based system for consistent query answering. In *SAT*, volume 11628 of *Lecture Notes in Computer Science*, pages 117–135. Springer, 2019.

[11] A. A. Dixit and P. G. Kolaitis. Consistent answers of aggregation queries via SAT. In *ICDE*, pages 924–937. IEEE, 2022.

[12] Z. Fan, P. Koutris, X. Ouyang, and J. Wijsen. LinCQA: Faster consistent query answering with linear time guarantees. *Proc. ACM Manag. Data*, 1(1):38:1–38:25, 2023.

[13] D. Figueira, A. Padmanabha, L. Segoufin, and C. Sirangelo. A simple algorithm for consistent query answering under primary keys. In *ICDT*, volume 255 of *LIPIcs*, pages 24:1–24:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[14] G. Fontaine. Why is it hard to obtain a dichotomy for consistent query answering? *ACM Trans. Comput. Log.*, 16(1):7:1–7:24, 2015.

[15] C. Freire, W. Gatterbauer, N. Immerman, and A. Meliou. The complexity of resilience and responsibility for self-join-free conjunctive queries. *Proc. VLDB Endow.*, 9(3):180–191, 2015.

[16] C. Freire, W. Gatterbauer, N. Immerman, and A. Meliou. New results for the complexity of resilience for binary conjunctive queries with self-joins. In *PODS*, pages 271–284. ACM, 2020.

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

[18] L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, July 1977.

[19] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Trans. Knowl. Data Eng.*, 15(6):1389–1408, 2003.

[20] M. Hannula and J. Wijsen. A dichotomy in consistent query answering for primary keys and unary foreign keys. In *PODS*, pages 437–449. ACM, 2022.

[21] L. A. Kahale, A. M. Khamis, B. Diab, Y. Chang, L. C. Lopes, A. Agarwal, L. Li, R. A. Mustafa, S. Koujanian, R. Waziry, et al. Meta-analyses proved inconsistent in how missing data were handled across their included primary trials: A methodological survey. *Clinical Epidemiology*, 12:527–535, 2020.

[22] Y. Katsis, A. Deutsch, Y. Papakonstantinou, and V. Vassalos. Inconsistency resolution in online databases. In *ICDE*, pages 1205–1208. IEEE Computer Society, 2010.

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

[24] P. G. Kolaitis, E. Pema, and W. Tan. Efficient querying of inconsistent databases with binary integer programming. *Proc. VLDB Endow.*, 6(6):397–408, 2013.

[25] P. Koutris, X. Ouyang, and J. Wijsen. Consistent query answering for primary keys on path queries. In *PODS*, pages 215–232. ACM, 2021.

[26] P. Koutris and D. Suciu. A dichotomy on the complexity of consistent query answering for atoms with simple keys. In *ICDT*, pages 165–176. OpenProceedings.org, 2014.

[27] P. Koutris and J. Wijsen. The data complexity of consistent query answering for self-join-free conjunctive queries under primary key constraints. In *PODS*, pages 17–29. ACM, 2015.

[28] P. Koutris and J. Wijsen. Consistent query answering for self-join-free conjunctive queries under primary key constraints. *ACM Trans. Database Syst.*, 42(2):9:1–9:45, 2017.

[29] P. Koutris and J. Wijsen. Consistent query answering for primary keys and conjunctive queries with negated atoms. In *PODS*, pages 209–224. ACM, 2018.

[30] P. Koutris and J. Wijsen. Consistent query answering for primary keys in logspace. In *ICDT*, volume 127 of *LIPIcs*, pages 23:1–23:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[31] P. Koutris and J. Wijsen. First-order rewritability in consistent query answering with respect to multiple keys. In *PODS*, pages 113–129. ACM, 2020.

[32] P. Koutris and J. Wijsen. Consistent query answering for primary keys in datalog. *Theory Comput. Syst.*, 65(1):122–178, 2021.

[33] L. Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004.

[34] C. Lutz and F. Wolter. On the relationship between consistent query answering and constraint satisfaction problems. In *ICDT*, volume 31 of *LIPIcs*, pages 363–379. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[35] M. Manna, F. Ricca, and G. Terracina. Taming primary key violations to query large inconsistent data via ASP. *Theory Pract. Log. Program.*, 15(4-5):696–710, 2015.

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

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

[38] D. Maslowski and J. Wijsen. Counting database repairs that satisfy conjunctive queries with self-joins. In *ICDT*, pages 155–164. OpenProceedings.org, 2014.

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

[40] J. Wijsen. Certain conjunctive query answering in first-order logic. *ACM Trans. Database Syst.*, 37(2):9:1–9:35, 2012.

[41] J. Wijsen. Foundations of query answering on inconsistent databases. *SIGMOD Rec.*, 48(3):6–16, 2019.

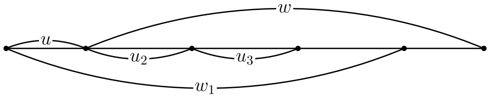
A Proofs for Section 4

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 22. *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 19 (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 20 (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 23 (Repeating lemma)**.** *Let q be a word that satisfies $\mathcal{C}_3$. 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 22, 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 22, r is a prefix of $(uR)^{|r|}$, which means that RuR is right-repeating.

This concludes the proof. ☐

Definition 21. If q is a word over an alphabet Σ, then $\mathsf{symbols}(q)$ is the set that contains all (and only) the symbols that occur in q.

Lemma 24 (Self-join-free episodes)**.** *Let q be a word that satisfies $\mathcal{C}_3$. 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 $L \notin \mathsf{symbols}(\ell)$, it must be that ℓ has a factor MmM such that Mm is self-join-free. By Lemma 23, MmM must be left-repeating or right-repeating, which requires $L \in \mathsf{symbols}(Mm)$, a contradiction. ☐

A.2 Proof of Lemma 1

Proof of Lemma 1. The implication 2 $\Longrightarrow$ 1 is obvious. To show 1 $\Longrightarrow$ 2, assume that q satisfies $\mathcal{C}_1$. 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 $\mathcal{C}_1$, q is a prefix of $\ell RmRmRr$. It follows that Rr is a prefix of $RmRr$. By Lemma 22, Rr is a prefix of $(Rm)^{|r|+1}$. It follows that there is a k such that q is a prefix of $\ell\,(Rm)^k$. $\square$

A.3 Proof of Lemma 2

Proof of Lemma 2. The proof of 2 $\Longrightarrow$ 1 is straightforward. We show next the direction 1 $\Longrightarrow$ 2. To this end, assume that q satisfies $\mathcal{C}_3$. 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 $\mathsf{symbols}(L\ell L)$, $\mathsf{symbols}(RrR)$, and $\mathsf{symbols}(m)$ are pairwise disjoint. Then, by Lemma 23, $L\ell L$ must be left-repeating, and RrR right-repeating. By Lemma 24, $L\ell$ 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 $q := \cdots\ \overbrace{L\ell\rlap{\underbrace{}_{e_2}}RoL}^{e_1}\,rR\ \cdots$, where it can be assumed that e_1 is the right-most episode that is left-repeating. Then, $\ell \neq \varepsilon \neq r$ implies $\mathsf{first}(\ell) \neq \mathsf{first}(r)$ (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 $\overbrace{L\ell\rlap{\underbrace{}_{e_2}}RoL}^{e_1}\,rR\ \cdot oL$. Then, q rewinds to a word p with factor:

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

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 $q := \cdots\ \overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,rL\ \cdots$, 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 $\mathsf{first}(\ell) \neq \mathsf{first}(r)$, or else e_1 would not be right-most, a contradiction. The word q rewinds to a word p with factor $\overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,r\,\overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,rL$, 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 $\overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,rL \cdot \mathsf{first}(r)$, and p has suffix $\overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,r\,\overbrace{L\ell\rlap{\underbrace{}_{e_2}}L}^{e_1}\,rL \cdot \mathsf{first}(r)$. If we left-align these suffixes, then there is a mismatch between $\mathsf{first}(r)$ 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. $\square$

A.4 Proof of Lemma 3

Proof of Lemma 3. Assume that q satisfies $\mathcal{C}_3$. By Lemma 2, q satisfies $\mathcal{B}_{2a}$, $\mathcal{B}_{2b}$, or $\mathcal{B}_3$.

$\boxed{1 \Longrightarrow 2}$ 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 $\mathcal{C}_2$ 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 $\mathsf{last}(u) \cdot wuvu \cdot \mathsf{first}(v)$ is a factor of q; or
- $v = \emptyset$, $w \neq \emptyset$ and $\mathsf{last}(u) \cdot w\,(u)^2 \cdot \mathsf{first}(u)$ is a factor of q.

$\boxed{3 \implies 1}$ 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 $\mathcal{C}_2$.

Consider next the case $v = \varepsilon$ (whence $w \neq \varepsilon$). Let $u = \hat{u}R$. By item (3b), q has a factor $R \cdot w\hat{u}R\hat{u}R \cdot \mathsf{first}(u)$, with three consecutive occurrences of R. Since $w\hat{u} \neq \hat{u}$ and $\mathsf{first}(u) \neq \mathsf{first}(w)$, it follows that q falsifies $\mathcal{C}_2$. ☐

B Proofs for Section 8

B.1 Proof of Lemma 21

Lemma 21 is an immediate corollary of Lemma 27, which states that whenever a generalized path query starts with a constant, then $\mathsf{CERTAINTY}(q)$ is in **FO**. Its proof needs two helping lemmas.

Lemma 25. *Let $q = q_1 \cup q_2 \cup \cdots \cup q_k$ be a Boolean conjunctive query such that for all $1 \leq i < j \leq k$, $\mathsf{vars}(q_i) \cap \mathsf{vars}(q_j) = \emptyset$. Then, the following are equivalent for every database instance* **db***:*

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

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

$\boxed{1 \implies 2}$ 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 $\mathsf{CERTAINTY}(q_1)$ and $\mathsf{CERTAINTY}(q_2)$.

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

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

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 $\mathsf{CERTAINTY}(q)$. ☐

Lemma 26. *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 $\mathsf{CERTAINTY}(q)$ *to* $\mathsf{CERTAINTY}(p)$*.*

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

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

$\boxed{\Longrightarrow}$ Assume $\mathbf{db}$ is a "yes"-instance for $\mathsf{CERTAINTY}(q)$. 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 $\mathbf{db}$. Then there exists a valuation μ with $\mu(q) \subseteq \mathbf{r} \setminus \{N(\underline{c}, d)\}$. Consider the valuation μ^+ from $\mathsf{vars}(q) \cup \{s_{k+1}, s_{k+2}\}$ to $\mathsf{adom}(\mathbf{db}) \cup \{c, d\}$ that agrees with μ on $\mathsf{vars}(q)$ 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 $\mathsf{CERTAINTY}(p)$.

$\boxed{\Longleftarrow}$ Assume that $\mathbf{db} \cup \{N(\underline{c}, d)\}$ is a "yes"-instance for the problem $\mathsf{CERTAINTY}(p)$. Let $\mathbf{r}$ be any repair of $\mathbf{db}$. 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 $\mathbf{db}$ contains only one N-fact, we have $\theta(s_{k+1}) = c$. It follows that $\theta(q) \subseteq \mathbf{r}$, as desired. ☐

Lemma 27. *Let q be a generalized path query with*

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

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

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 $i \in \{1, 2, \dots, \ell\}$, 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 $\mathsf{CERTAINTY}(q_i)$ is in $\mathbf{FO}$ for each $1 \leq i \leq \ell$. Indeed, if $s_{j_{i+1}}$ is a variable, then the claim follows by Lemma 12; if $s_{j_{i+1}}$ is a constant, then the claim follows by Lemma 26 and Lemma 12.

Since by construction, $q = q_1 \cup q_2 \cup \cdots \cup q_\ell$, we conclude that $\mathsf{CERTAINTY}(q)$ is in $\mathbf{FO}$ by Lemma 25. ☐

The proof of Lemma 21 is now simple.

Proof of Lemma 21. If q contains no constants, the lemma holds trivially. Otherwise, $\mathsf{CERTAINTY}(p)$ is in $\mathbf{FO}$ by Lemma 27. ☐

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

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

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

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

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

Example 10. 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 $\mathsf{ext}(q) = R(\underline{x}, y), S(\underline{y}, z), N(\underline{z}, u)$. ☐

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

Lemma 28. *For every generalized path query q, there is a first-order reduction from* $\mathsf{CERTAINTY}(q)$ *to* $\mathsf{CERTAINTY}(\mathsf{char}(q))$.

Proof. Let $p := q \setminus \mathsf{char}(q)$. Since $\mathsf{vars}(\mathsf{char}(q)) \cap \mathsf{vars}(p) = \emptyset$, Lemmas 25 and 27 imply that the following are equivalent for every database instance **db**:

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

To conclude the proof, it suffices to observe that $\mathsf{CERTAINTY}(p)$ is in **FO** by Lemma 27. ☐

Lemma 29. *For every generalized path query q, there is a first-order reduction from* $\mathsf{CERTAINTY}(\mathsf{char}(q))$ *to* $\mathsf{CERTAINTY}(\mathsf{ext}(q))$.

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 $\mathsf{CERTAINTY}(\mathsf{char}(q))$ to $\mathsf{CERTAINTY}(\mathsf{ext}(q))$ by Lemma 26. ☐

B.3 Complexity Upper Bounds in Theorem 4

Lemma 30. *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* $\mathsf{ext}(q)$ *satisfies $\mathcal{C}_2$.*

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 $\mathsf{ext}(q)$ is a factor of every word to which $\mathsf{ext}(q)$ 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 $\mathsf{ext}(q)$ satisfies $\mathcal{C}_3$.

The remaining test for $\mathcal{C}_2$ 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 $\mathsf{ext}(q)$ satisfies $\mathcal{C}_2$. ☐

Lemma 31. *For every generalized path query q,*

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

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, $\mathsf{ext}(q)$ is self-join-free, and thus $\mathsf{ext}(q)$ satisfies $\mathcal{C}_1$.

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 30, $\mathsf{ext}(q)$ satisfies $\mathcal{C}_2$. Since $\mathcal{C}_2$ logically implies $\mathcal{C}_3$, q satisfies $\mathcal{C}_3$. ☐

We can now prove the upper bounds in Theorem 4.

Proof of upper bounds in Theorem 4. Since first-order reductions compose, by Lemmas 28 and 29, there is a first-order reduction from the problem $\mathsf{CERTAINTY}(q)$ to $\mathsf{CERTAINTY}(\mathsf{ext}(q))$. The upper bound results then follow by Lemma 31. ☐

B.4 Complexity Lower Bounds in Theorem 4

The complexity lower bounds in Theorem 4 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 30, a violation of $\mathcal{D}_2$ implies a violation of $\mathcal{D}_3$.

In the proof of Lemma 18, 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 18 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 11. Since the constant c occurs at most twice in q by Definition 16, 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 18.

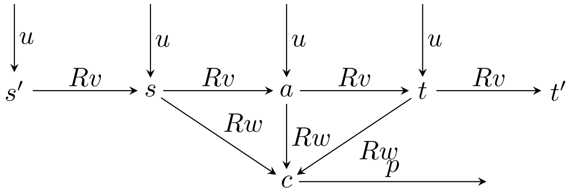


Figure 11: Database instance for the revised **NL**-hardness reduction from the graph G with $V = \{s, a, t\}$ and $E = \{(s, a), (a, t)\}$.

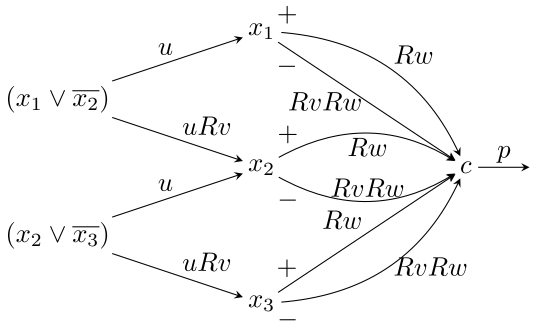


Figure 12: Database instance for the revised **coNP**-hardness reduction from the formula $\psi = (x_1 \lor \overline{x_2}) \land (x_2 \lor \overline{x_3})$.

In the proof of Lemma 19, 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 19 to be

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

An example is shown in Figure 12. 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 $\phi^z_C[uRv]$, $\phi^c_z[Rw]$, and the canonical copy of p; or
- $\mathbf{r}$ contains $\phi^z_C[u]$, $\phi^c_z[RvRw]$, and the canonical copy of p.

coNP-hardness can now be proved as in the proof of Lemma 19.

B.5 Proof of Theorem 5

Proof of Theorem 5. Immediate consequence of Theorem 4 and Lemma 30. ☐