View-Based Query Determinacy for Walk-Based Semantics

Nadime Francis
LIGM, Université Gustave Eiffel, CNRS, Marne-la-Vallée, France

Abstract

The view-based query determinacy problem asks, given a view $\mathcal{V}$ and a query Q whether the information contained in $\mathcal{V}$ always suffices to answer the query Q. It is a notoriously hard problem that is known to be undecidable even in very restricted settings. Here, we study it in the context of graph databases and regular path queries and views evaluated under several "walk-based semantics", that is, semantics in which queries and views return the matched walks in full as opposed to the classical "endpoint semantics". Our main finding is that view-based query determinacy is decidable for regular path queries under both trail and shortest walk semantics.

2012 ACM Subject Classification Theory of computation → Database theory

Keywords and phrases graph databases, regular path queries, trail semantics, shortest walk semantics, view-based query determinacy

Digital Object Identifier 10.4230/LIPIcs.ICDT.2026.22

Acknowledgements I gratefully thank Yann Strozecki for reading early versions of this work and providing constructive feedback.

1 Introduction

The *view-based query determinacy* problem is a classical static analysis problem in database theory. It arises in many reasoning scenarios, including data exchange and integration, query optimization and data privacy. Given a set of queries $\mathcal{V}$ called the *view* and a target query Q, one has to decide whether, for all databases D, the answers to the queries in $\mathcal{V}$, denoted as $\mathcal{V}(D)$ always suffice to compute $Q(D)$ regardless of the underlying data that was used to compute $\mathcal{V}(D)$. When this is the case, we say that $\mathcal{V}$ *determines* Q. Formally, this means that there exists a functional dependency from $\mathcal{V}$ to Q: for all databases D and D', if $\mathcal{V}(D) = \mathcal{V}(D')$, then $Q(D) = Q(D')$.

This problem is known to be computationally very hard in many settings, even for the most basic query and view languages. It is already undecidable for conjunctive queries and conjunctive views in relational databases [11, 12]. Some fragments and restrictions of the problem are known to be decidable, but the strength of the restrictions can be seen as a testimony to the original problem's hardness. These include chain queries (that is, conjunctive queries whose underlying graph is a path) [2] with a narrow extension in [6] and the *monotone* determinacy setting, in which the functional dependencies from the view to the target query is furthermore assumed to be monotonic [19].

In the graph database setting, the problem turns out, perhaps unsurprisingly, to also be computationally hard. The building block for most query languages over graphs are *regular path queries* [18]: queries which look for walks in the graph whose label conforms to a given regular expression. Unfortunately, query determinacy is undecidable for regular path queries and views, even for a very restricted form of regular expression with no Kleene star [10]. As in the relational case, the monotone setting is decidable, as shown in [9] using techniques from [4]. However, these results do not tell the whole story: in existing work, regular path queries are evaluated under the so-called *endpoint* semantics, where queries only return the endpoints of walks in the database whose label conforms to the query. While theoretically

very sound, the endpoint semantics do not capture the behavior of real-life systems, in which the matching walks are returned in full. To the best of our knowledge, there are no known results for query determinacy under walk-based semantics.

Moving from endpoint semantics to walk-based semantics leads to many complications. Indeed, since databases can contain cycles, it may happen that the set of walks to be returned by a given query is infinite. To address this issue, real-life systems evaluate queries under specific *semantics*, that is, criteria that ensure the finiteness of the output. The most common semantics are *trail semantics*, which filters out walks that contain repeated edges, and *shortest walk semantics*, which only returns the shortest witness for each fixed source and target vertex. These semantics come at a severe cost in complexity (testing emptiness for a given regular path query is NP-complete under trail semantics [17], which makes query evaluation intractable) and expressivity (by filtering out potentially interesting results). These semantics are nonetheless commonly seen in practice, which strongly implies that there is a real demand for walk-based semantics. Notable examples include Cypher, which is evaluated under trail semantics [8] and GSQL [21] and G-Core [3], which both use shortest walk semantics. Finally, the language GQL [5, 7], which has been designed as a standard for graph query languages, implements both trail and shortest walk semantics.

This article addresses the query determinacy problem in the graph database setting for regular path queries and views that are evaluated under walk-based semantics. In addition to the trail and shortest walk semantics, we also consider the *all-walks* semantics which simply returns all walks that match the query, even though the set of answers could be infinite. This is not only a theoretical consideration. We will see that in the context of query determinacy, the decision procedure for this semantics actually applies to all semantics that are "edge-covering" [16]: if an edge of the graph is useful to the query (it occurs in at least one matching walk), then this edge should appear in at least one walk of the answer.

Note that it is not immediately obvious whether query determinacy under walk-based semantics is harder than under endpoints semantics: we are assigned a harder task (computing the walks that match the query *in full*) but we are given a view that provides much more information. This is similar to the case of *full* conjunctive queries and views in the relational setting, where it is assumed that all variables in the body of the query are also output variables in the head. This setting is known to be decidable [2] with a simple criterion: full views determine the query if they cover all atoms of a specific canonical database. The graph database setting turns out to also be decidable under walk-based semantics but poses new challenges. First, regular path queries are essentially infinite (albeit regular) unions of conjunctive queries, which means that there would naively be infinitely many databases against which determinacy must be checked. Second, the semantics under which queries and views are evaluated might filter out necessary information, even though the walks containing said information do match the given regular languages.

Contributions. We start by showing how the classical query determinacy problem translates to our setting. As a side result, we remark that the query rewriting problem using views [19] turns out to be trivial in our setting: the rewriting, when it exists, will simply be the original query itself. Then our main contribution is to prove upper and lower complexity bounds for the problem of deciding query determinacy under all-walks, trail and shortest walk semantics. First, we establish a lower bound that covers all three semantics:

▶ **Theorem 1.** *Determinacy for regular path queries and views is* PSPACE*-hard under all-walks, trail and shortest walk semantics, even if the automata that define the queries and views are deterministic.*

In this work, we assume that the regular path queries are given as finite automata. Thus, it is a natural concern whether the complexity depends on the properties of the specific automaton that implements the query. Our result, however, shows that the lower bound holds even in the favorable case where the automaton is deterministic. Next, we prove a matching upper bound for all-walks and trail semantics, which makes the problem PSPACE-complete:

▶ **Theorem 2.** *Determinacy for regular path queries and views can be decided in* PSPACE *under all-walks and trail semantics, even for nondeterministic query and view automata.*

Finally, we prove a matching upper bound for shortest walk semantics, which turns out to be our most difficult result:

▶ **Theorem 3.** *Determinacy for regular path queries and views can be decided in* PSPACE *under shortest walk semantics, provided that the query is given as a deterministic automaton.*

This upper bound only holds when the *query* is given as a deterministic automaton (the automata defining the view may be nondeterministic). One could always choose to determinize the input to use this decision algorithm, but then the upper bound becomes EXPSPACE, and no longer matches the lower bound. Unfortunately, this gap remains as an open question: we neither have a better upper bound for the nondeterministic case, nor a tighter lower bound when the automaton can be nondeterministic.

Outline. Section 2 defines the data and query models of our setting. Section 3 defines the *view-based query determinacy* problem and shows preliminary results that apply to all three semantics, including the PSPACE lower bound. Section 4 proves the upper bound for the all-walks and trail semantics. While interesting in its own right, this section also serves as a warm-up for Section 5 where we prove the upper bound for the shortest walk semantics. Indeed, the techniques in Section 5 reuse and expand upon the core ideas of Section 4.

2 Preliminaries

In this section, we define the data and query models that are used throughout the article. Particular attention should be given to Section 2.3 that defines the walk-based semantics under which regular path queries are to be evaluated.

2.1 Graph databases

We represent graph databases as multi-edge multi-label graphs. All results naturally translate to the traditional theoretical model where databases are more simply represented as edge-labeled graphs [18]. Here, we chose to use a more general model that more closely resembles the *property graph* data model used in practice [5], where both vertices and edges are assumed to have a unique identifier. In particular, it is not enough to know that a vertex x is connected to a vertex y with an a-labeled edge: we will also need the specific identifier e of said edge.

▶ **Definition 4** (Database). *A* database D *is a tuple* $(\Sigma, \textsc{Node}, \textsc{Edge}, \textsc{Src}, \textsc{Tgt}, \textsc{Lbl})$ *where:*

- Σ *is a finite set of symbols, or* labels*;*
- $\textsc{Node}$ *is a finite set of* vertices*;*
- $\textsc{Edge}$ *is a finite set of* edges*;*
- $\textsc{Src} : \textsc{Edge} \to \textsc{Node}$ *is the* source *function;*
- $\textsc{Tgt} : \textsc{Edge} \to \textsc{Node}$ *is the* target *function;*
- $\textsc{Lbl} : \textsc{Edge} \to 2^{\Sigma}$ *is the* labeling *function.*

▶ **Definition 5** (Walk). *A* walk π *in a database* D *is a nonempty finite sequence of alternating vertices and edges of the form* $\pi = (x_0, e_0, x_1, \ldots, e_{k-1}, x_k)$ *where* $k \geq 0$, $x_0, \ldots, x_k \in \textsc{Node}$, $e_0, \ldots, e_{k-1} \in \textsc{Edge}$, *such that,* $\forall i < k,\ \textsc{Src}(e_i) = x_i$ *and* $\textsc{Tgt}(e_i) = x_{i+1}$.

We call k *the* length *of* π *and denote it by* $\textsc{Len}(\pi)$. *We extend the functions* Src, Tgt *and* Lbl *to the walks in* D *as follows. For each walk* $\pi = (x_0, e_0, x_1, \ldots, e_{k-1}, x_k)$ *in* D, $\textsc{Src}(\pi) = x_0$, $\textsc{Tgt}(\pi) = x_k$, *and* $\textsc{Lbl}(\pi) = \{a_0 a_1 \cdots a_{k-1} \mid \forall i < k,\ a_i \in \textsc{Lbl}(e_i)\}$. *Finally, we write* $s \xrightarrow{\pi} t$ *to say that* $\textsc{Src}(\pi) = s$ *and* $\textsc{Tgt}(\pi) = t$.

We say that two walks $\pi = (x_0, e_0, x_1, \ldots, e_{k-1}, x_k)$ *and* $\pi' = (y_0, f_0, y_1, \ldots, f_{\ell-1}, y_\ell)$ concatenate *if* $\textsc{Tgt}(\pi) = \textsc{Src}(\pi')$, *in which case we define their* concatenation $\pi \cdot \pi'$ *as*

$$\pi \cdot \pi' = (x_0, e_0, x_1, \ldots, e_{k-1}, x_k, f_0, y_1, \ldots, f_{\ell-1}, y_\ell)$$

that is, we concatenate the two lists and remove one occurrence of the repeated vertex $x_k = y_0$.

A walk is said to be simply labeled *if each edge of the walk has at most one label.*

Finally, a walk is called a trail *if it contains no repeated edge. It is furthermore called a* simple walk *if it contains no repeated vertex.*

When we are given a walk, typically as returned by a query, we assume that it is given together with all the data that occurs in the vertices and edges along the walk, namely their identifiers and labels. This is in line with real-life database systems, where a returned object can be subsequently inspected for any data it might contain. It will be useful to consider this data as its own database, as defined below.

▶ **Definition 6** (Data in a walk). *Let* π *be a walk in a database* D. *We define* $\textsc{Data}(\pi)$ *as the restriction of* D *to the vertices and edges that occur in* π. *Formally,* $\textsc{Data}(\pi)$ *is the database* $D_\pi = (\Sigma, \textsc{Node}_\pi, \textsc{Edge}_\pi, \textsc{Src}_\pi, \textsc{Tgt}_\pi, \textsc{Lbl}_\pi)$ *defined as:*

- $\textsc{Node}_\pi = \{x \mid \pi = \pi_1 \cdot (x) \cdot \pi_2\}$
- $\textsc{Edge}_\pi = \{e \mid \pi = \pi_1 \cdot (x_1, e, x_2) \cdot \pi_2\}$
- $\textsc{Src}_\pi = \textsc{Src}_{|\,\textsc{Edge}_\pi}$, $\textsc{Tgt}_\pi = \textsc{Tgt}_{|\,\textsc{Edge}_\pi}$, $\textsc{Lbl}_\pi = \textsc{Lbl}_{|\,\textsc{Edge}_\pi}$

2.2 Queries and Automata

A regular path query (RPQ) is defined by a regular language. In real-life systems, RPQs are typically input with a syntax that resembles regular expressions. Here, we define queries using finite state automata, which makes formal proofs easier and allows a finer complexity analysis depending on whether the given automaton is *deterministic*. This does not affect the applicability of our upper bounds, as regular expressions can be translated to nondeterministic automata in linear time (see for instance [20]). Note that the same does not hold for *deterministic* automata, which is why the deterministic case enjoys better bounds.

Formally, lower bounds do not immediately translate from one setting to the other. Fortunately the proof for regular expressions happens to be extremely similar (see Remark 22). In examples, we will informally use regular expressions, which are more human-readable.

▶ **Definition 7** (Automaton). *A nondeterministic* automaton A *is a 5-tuple* $(\Sigma, S, \Delta, I, F)$ *where* Σ *is a finite set of symbols,* S *is a finite set of* states, $I \subseteq S$ *is the set of* initial *states,* $\Delta \subseteq S \times \Sigma \times S$ *is the set of* transitions *and* $F \subseteq S$ *is the set of* final *or* accepting *states. We denote as* $\Delta(q, a)$ *the set* $\{p \mid (q, a, p) \in \Delta\}$, *and for a set* $K \subseteq S$, $\Delta(K, a) = \cup_{q \in K} \Delta(q, a)$.

An automaton A *is* deterministic *if* $|I| \leq 1$, *and for all* $q, a \in S \times \Sigma$, $|\Delta(q, a)| \leq 1$. *In that case, we will interpret* Δ *as a partial function* $\delta : S \times \Sigma \to S$, *written in lowercase to avoid confusion.*

We denote by $L(A)$ *the* language *of* A, *that is, the set of words* w *such that* $\Delta(I, w) \cap F \neq \emptyset$.

A *regular path query* Q is defined by an automaton A. We use $L(Q)$ and $L(A)$ interchangeably. An RPQ Q *matches* walks which carry a label that belong to $L(Q)$:

▶ **Definition 8** (Match). *Let D be a database and π be a walk in D. We say that π* matches *an RPQ Q if $\textsc{Lbl}(\pi) \cap L(Q) \neq \emptyset$. We denote as $\textsc{Match}(Q, D)$ the (possibly infinite) set of all walks of D that match Q.*

Warning. For simplicity, we assume that languages never contain the empty word ε. This is without loss of generality: the corner case where $\varepsilon \in L(Q)$ can be solved separately (see Appendix A) and excluding it avoids special cases and technical overhead in subsequent statements and proofs. From now on, matched walks always contain at least one edge.

2.3 Walk-based Semantics

When a database contains cycles, it may happen that an RPQ matches infinitely many walks. While this is not an issue in a theoretical context, real-life systems impose criteria that guarantee finiteness, known as *semantics*. We say that a semantics is *walk-based* if it returns a subset of $\textsc{Match}(Q, D)$. Let Q be an RPQ and D be a database. Let X be a semantics. We denote as $Q_X(D)$ the set of answers to Q under X. We focus on three specific walk-based semantics defined below:

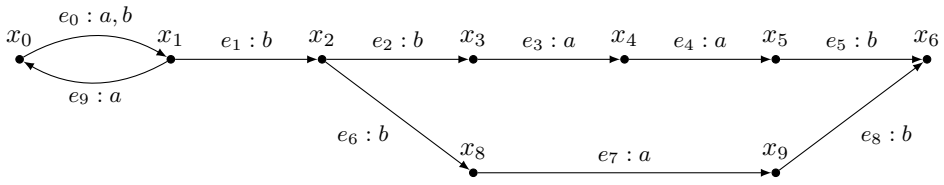
- *All-walks semantics*: $Q_{walk}(D) = \textsc{Match}(Q, D)$.
- *Trail semantics*: $Q_{trail}(D) = \{\pi \in \textsc{Match}(Q, D) \mid \pi \text{ is a trail}\}$.
- *Shortest walk semantics*: for each pair of vertices s, t, we define the *minimal match length* $\lambda_{s,t}$ as $\lambda_{s,t} = \min(\{\textsc{Len}(\pi) \mid \pi \in \textsc{Match}(Q, D) \text{ and } s \xrightarrow{\pi} t\})$. Under shortest walk semantics, Q only returns the walks whose length is minimal between their endpoints: $Q_{short}(D) = \{\pi \in \textsc{Match}(Q, D) \mid \textsc{Len}(\pi) = \lambda_{\textsc{Src}(\pi),\textsc{Tgt}(\pi)}\}$.

We will make use of some well-known properties of all-walks and trail semantics:

- RPQ under $X \in \{walk, trail\}$ are *monotonic*: if $D \subseteq D'$, then $Q_X(D) \subseteq Q_X(D')$.
- RPQ under *walk* (resp. *trail*) are *closed under homomorphism* (resp. edge-injective homomorphism): if φ is a homomorphism (resp. edge-injective homomorphism) from D to another database D', and $\pi \in Q_X(D)$, then $\varphi(\pi) \in Q_X(\varphi(D))$.
  Indeed, $\textsc{Lbl}(\pi) \subseteq \textsc{Lbl}(\varphi(\pi))$, thus if $\textsc{Lbl}(\pi) \cap L(Q) \neq \emptyset$, then $\textsc{Lbl}(\varphi(\pi)) \cap L(Q) \neq \emptyset$.
  Additionally, if π is a trail and φ is edge-injective, then $\varphi(\pi)$ is a trail.

Remark that neither property holds for shortest walk semantics: adding a new walk in D can make previous query answers disappear, as they might no longer be the shortest walks.

▶ **Example 9.** Let D be the database depicted below:



Let Q be the query defined by $b(a^+b)^+$, that is, $L(Q)$ contains all words which start with a b, end with another b, and without two consecutive b. Then we can evaluate Q over D under the three semantics. For the sake of readability, only the edges are listed in the walks below. Vertices are implicitly defined by the adjacent edges.

- $Q_{trail}(D) = \{(e_2e_3e_4e_5), (e_6e_7e_8)\}$
- $Q_{short}(D) = \{(e_0e_9e_0), (e_0e_9e_0e_1), (e_6e_7e_8)\}$
- $Q_{walk}(D)$ is an infinite set. In addition to walks in Q_{short} and Q_{trail}, it also contains walks described by the following regular expression: $e_0(e_9e_0)^+(\varepsilon + e_1)$.

Remark that $Q_{short}(D)$ does not contain the "top" walk $(e_2e_3e_4e_5)$, because $\lambda_{x_2,x_6} = 3$, as witnessed by the "bottom" walk $(e_6e_7e_8)$. Conversely, $Q_{trail}(D)$ does not contain any of the "looping" walks, such as $(e_0e_9e_0e_1)$, because they repeat e_0.

3 View-based Query Determinacy

This section defines the view-based query determinacy problem in our setting, and proves several useful preliminary results that are the foundations of the following sections. It concludes with the proof of Theorem 1 that provides a PSPACE lower bound for all three semantics.

▶ **Definition 10** (View). *A* view $\mathcal{V}$ *is a finite set of queries. If all queries in* $\mathcal{V}$ *are regular path queries, we say that* $\mathcal{V}$ *is a* regular path view.

▶ **Definition 11** (View image under X). *Let* $\mathcal{V}$ *be a regular path view. Let* X *be an RPQ semantics and let* D *be a database. The* view image *of* D *through* $\mathcal{V}$ *under* X *is the tuple* $\mathcal{V}_X(D) = (V_X(D))_{V\in\mathcal{V}}$.

▶ Remark 12. The view image of a database D is a tuple of sets, one set for each query V in the view $\mathcal{V}$. These sets could be infinite if X does not guarantee a finite output.

We now have all the tools to translate the classical definition of view-based query determinacy (see [19] for a reference) to our setting:

▶ **Definition 13** (Determinacy under X). *Let* $\mathcal{V}$ *be a view and* Q *be a query. We say that* $\mathcal{V}$ determines Q *under* X*, and we write* $\mathcal{V} \twoheadrightarrow_X Q$ *if the following holds:*

$$\forall D_1, D_2,\ \mathcal{V}_X(D_1) = \mathcal{V}_X(D_2) \ \Rightarrow\ Q_X(D_1) = Q_X(D_2)$$

Informally, this definition states that $\mathcal{V}$ determines Q if and only if there is a functional dependency from view images to query answers: as soon as two databases have the same view image, they must also return the same answers to the query.

Our goal is to solve the *view-based query determinacy problem* defined as:

PROBLEM	:	QUERY DETERMINACY UNDER SEMANTICS X
INPUT	:	A view $\mathcal{V} = (V_1, \ldots, V_n)$, a query Q
QUESTION	:	Does $\mathcal{V} \twoheadrightarrow_X Q$?

As in Definition 6, it will be useful to aggregate all data contained in a view image and consider it as a database:

▶ **Definition 14** (Data in a view under X). *Let* D *be a database,* $\mathcal{V}$ *be a regular path view and* X *be an RPQ semantics. We extend Definition 6 to view images as follows:*

$$\textsc{Data}(\mathcal{V}_X(D)) = \bigcup_{V\in\mathcal{V}} \bigcup_{\pi\in V_X(D)} \textsc{Data}(\pi)$$

▶ Remark 15 (Edge-covering Semantics). $\textsc{Data}(\mathcal{V}_X(D))$ is a substructure of D. As such, it is always finite, even if X is not a semantics that guarantees a finite output.

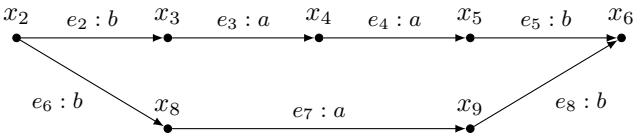
This finite structure is what gives meaning to all-walks semantics beyond being just a theoretical curiosity. Indeed, in our setting, all-walks semantics captures the family of RPQ semantics that are "edge-covering" [16]. Intuitively, edge-covering semantics return all edges that are useful to the query at least once. Formally, a semantics X is said to be "edge-covering" if and only if, for all queries Q and databases D, for all edges e in D, if there exists a walk $\pi \cdot (x, e, y) \cdot \mu \in \textsc{Match}(Q, D)$, then there exists a walk $\pi' \cdot (x, e, y) \cdot \mu' \in Q_X(D)$.

Let X be an edge-covering semantics. Then, it is not hard to see that for all views $\mathcal{V}$, $\textsc{Data}(\mathcal{V}_X(D)) = \textsc{Data}(\mathcal{V}_{walk}(D))$. We will see that the content of $\textsc{Data}(\mathcal{V}_X(D))$ alone suffices to decide determinacy, without requiring the more precise information contained in $\mathcal{V}_X(D)$. Thus, results for all-walks semantics naturally extend to edge-covering semantics.

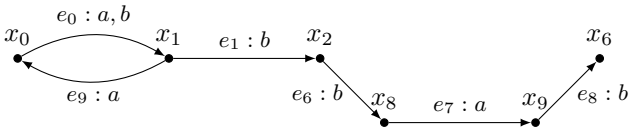
An example of edge-covering semantics that has been studied in theoretical work is the "shortest witness" semantics [14]: for each vertices s, t and edge e, this semantics returns the shortest walks from s to t that contain e. Though not commonly implemented in practice, a similar behavior can usually be achieved in real-life systems that allow performing multiple matches and joining the resulting walks, as in GQL or Neo4j.

▶ **Example 16.** Recall the database D from Example 9 and query Q defined by $b(a^+b)^+$. Let $\mathcal{V}$ be the view that contains only Q. Then $\textsc{Data}(\mathcal{V}_{walk}(D)) = D$, as all edges of D occur in some walk of $Q_{walk}(D)$. The case of trail and shortest walk semantics are given below:

- $\textsc{Data}(\mathcal{V}_{trail}(D)) =$



- $\textsc{Data}(\mathcal{V}_{short}(D)) =$



As is depicted above, under both trail and shortest walk semantics, some part of the database is "hidden" to the view.

The following two lemmas hint at the fact that $\textsc{Data}(\mathcal{V}_X(D))$ actually contains all relevant information for answering query determinacy:

▶ **Lemma 17** (No hidden walks). *Let $X \in \{walk, short, trail\}$, D be a database, $\mathcal{V}$ be a regular path view and Q be an RPQ. If there exists $\pi \in Q_X(D)$ such that π is not a walk in $\textsc{Data}(\mathcal{V}_X(D))$, then $\mathcal{V} \not\twoheadrightarrow_X Q$.*

Proof. Let everything be defined as in the statement of the lemma. Let $\pi \in Q_X(D)$ such that π is not a walk in $\textsc{Data}(\mathcal{V}_X(D))$. Then there exists an edge e that occurs in π but not in $\textsc{Data}(\mathcal{V}_X(D))$. Let D' be a copy of D without this edge. Then one simply has to remark that $\mathcal{V}_X(D) = \mathcal{V}_X(D')$, but $\pi \notin Q_X(D')$. ◀

▶ **Lemma 18** (View image from view data). *Let $X \in \{walk, short, trail\}$. Let D be a database. Then $\mathcal{V}_X(D) = \mathcal{V}_X(\textsc{Data}(\mathcal{V}_X(D)))$.*

Proof. Let $V \in \mathcal{V}$. We show that $V_X(D) = V_X(\textsc{Data}(\mathcal{V}_X(D)))$.

($\subseteq$) Let $\pi \in V_X(D)$. Then, by definition, π is a walk in $\textsc{Data}(\mathcal{V}_X(D))$. Thus, π matches V. Since $\textsc{Data}(\mathcal{V}_X(D))$ is a substructure of D, if π is a trail in D, it is also a trail in $\textsc{Data}(\mathcal{V}_X(D))$. Similarly, if π is a shortest walk between its endpoints in D, it is also a shortest walk in $\textsc{Data}(\mathcal{V}_X(D))$.

($\supseteq$) Let $\pi \in V_X(\textsc{Data}(\mathcal{V}_X(D)))$. As in the previous case, π is a walk of D that matches V, and if π is a trail in $\textsc{Data}(\mathcal{V}_X(D))$, then it is a trail in D. The only interesting question is whether π is a shortest walk in D when $X = \mathit{short}$. Assume for the sake of contradiction that there is a walk $\pi' \in V_{\mathit{short}}(D)$ such that π' has the same endpoints as π and is shorter than π. Then, by definition, π' is a walk in $\textsc{Data}(\mathcal{V}_{\mathit{short}}(D))$. Thus $\pi \notin V_X(\textsc{Data}(\mathcal{V}_{\mathit{short}}(D)))$, which is a contradiction. ◀

▶ Remark 19. Lemma 18 means that under $X \in \{\mathit{walk}, \mathit{trail}, \mathit{short}\}$, only $\textsc{Data}(\mathcal{V}_X(D))$ matters. We do not have to keep track of the specific view $V \in \mathcal{V}$ which returned each element of $\mathcal{V}_X(D)$. Be careful that this property does not generally hold: one can easily design "exotic" semantics that are specifically tailored to falsify this property, such as:

- "Second shortest walk": return π if and only if its length is the second shortest length.
- "Shortest walk hybrid": return $Q_{\mathit{walk}}(D)$ if it is finite, otherwise return $Q_{\mathit{short}}(D)$.

Lemma 17 and Lemma 18 imply that, in our setting, determinacy admits an alternate and much simpler definition:

▶ **Proposition 20** (Determinacy: alternate definition)**.** *Let $X \in \{\mathit{walk}, \mathit{trail}, \mathit{short}\}$. Let $\mathcal{V}$ be a regular path view and Q be an RPQ. Then $\mathcal{V} \twoheadrightarrow_X Q$ if and only if for all databases D, for all walks $\pi \in Q_X(D)$, π is a walk in $\textsc{Data}(\mathcal{V}_X(D))$.*

Proof.

($\Rightarrow$) This is immediately given as the contraposition of Lemma 17.

($\Leftarrow$) Assume that for all databases D, for all walks $\pi \in Q_X(D)$, π is a walk in $\textsc{Data}(\mathcal{V}_X(D))$. Let D_1 and D_2 be two databases such that $\mathcal{V}_X(D_1) = \mathcal{V}_X(D_2)$. Let $\pi \in Q_X(D_1)$. Then $\pi \in \textsc{Data}(\mathcal{V}_X(D_1))$, which yields $\pi \in \textsc{Data}(\mathcal{V}_X(D_2))$. If $X = \mathit{walk}$ (resp. $X = \mathit{trail}$), the result immediately follows: π is then a walk (resp. trail) in D_2 that matches Q, thus $\pi \in Q_X(D_2)$, which means $\mathcal{V} \twoheadrightarrow_X Q$.

Assume now that $X = \mathit{short}$. Since π is a walk of D_2 that matches Q, there are two cases: either $\pi \in Q_{\mathit{short}}(D_2)$ and there is nothing more to prove, or there exists a shorter walk μ in $Q_{\mathit{short}}(D_2)$ that has the same endpoints as π. In that case, the assumption yields that μ is a walk in $\textsc{Data}(\mathcal{V}_X(D_2))$. Thus it is a walk in $\textsc{Data}(\mathcal{V}_X(D_1))$. This means that π is not a shorter walk in D_1 either, which is a contradiction. ◀

When the view determines the query, one may be interested in finding *rewritings*, that is, queries whose answer on the view image coincides with the answer to the original query on the original data [19]. This task may require additional techniques and ideas. In our setting, it turns out that the two tasks coincide. Indeed, Proposition 20 together with Lemma 18 lead to the corollary below, which states that when the view determines the query, a rewriting of the query is the query itself.

▶ **Corollary 21** (Rewriting)**.** *Let $X \in \{\mathit{walk}, \mathit{trail}, \mathit{short}\}$. Let $\mathcal{V}$ be a regular path view and Q be an RPQ such that $\mathcal{V} \twoheadrightarrow_X Q$. Let D be a database. Then $Q_X(D) = Q_X(\textsc{Data}(\mathcal{V}_X(D)))$.*

We are now ready to prove the first complexity result: deciding determinacy is PSpace-hard under all-walks, trail and shortest walk semantics.

▶ **Theorem 1.** *Determinacy for regular path queries and views is* PSPACE*-hard under all-walks, trail and shortest walk semantics, even if the automata that define the queries and views are deterministic.*

Proof. Let $X \in \{walk, trail, short\}$. The proof is done by reduction from the INTERSECTION EMPTINESS problem [15], which asks, given n deterministic automata $A_1, \ldots, A_n$ whether $L(A_1) \cap \ldots \cap L(A_n) = \emptyset$. Here, we will use the equivalent UNION UNIVERSALITY problem: given n deterministic automata $A_1, \ldots, A_n$, does $L(A_1) \cup \ldots \cup L(A_n) = \Sigma^*$?

Let $start, end$ be two fresh symbols that do not occur in Σ. We define A as the usual 3-state deterministic automaton that accepts $start \cdot \Sigma^* \cdot end$ and we extend each automaton A_i into a deterministic automaton A'_i with a new initial and final state such that $L(A'_i) = start \cdot L(A_i) \cdot end$.

Then it remains to remark that Proposition 20 implies that $\{A'_i\} \twoheadrightarrow_X A$ if and only if $L(A_1) \cup \ldots \cup L(A_n)$ is universal. ◀

▶ Remark 22. This lower bounds also holds when the input are given as regular expressions, with a very similar reduction from the REGULAR EXPRESSION CONTAINMENT problem. [13]

4 All-walks and Trail Semantics

The goal of this section is to prove Theorem 2. To do so, we provide a decision procedure that works in PSPACE. The main idea is to show that determinacy has the *walk counterexample* property: if there is a database D that falsifies the alternate definition of determinacy provided by Proposition 20, then D can be assumed to be a simply labeled simple walk.

▶ **Proposition 23** (Determinacy on simple walks)**.** *Let $X \in \{walk, trail\}$. Let V be a regular path view and Q be an RPQ. Then $\mathcal{V} \twoheadrightarrow_X Q$ if and only if, for all databases that consist of only a simply labeled simple walk π matching Q,* $\textsc{Data}(\mathcal{V}_X(\pi)) = \textsc{Data}(\pi)$.

Proof.

($\Rightarrow$) If $\mathcal{V} \twoheadrightarrow_X Q$, then Lemma 17 implies that π is a walk in $\textsc{Data}(\mathcal{V}_X(\pi))$. Since $\textsc{Data}(\mathcal{V}_X(\pi))$ is a substructure of $\textsc{Data}(\pi)$, this implies that $\textsc{Data}(\mathcal{V}_X(\pi)) = \textsc{Data}(\pi)$.

($\Leftarrow$) Conversely, assume that for all databases that consist of only a simply labeled simple walk π matching Q, $\textsc{Data}(\mathcal{V}_X(\pi)) = \textsc{Data}(\pi)$. Let D be an arbitrary database and π be a (not necessarily simple) walk in $Q_X(D)$.

Since π matches Q, there exists $w \in \textsc{Lbl}(\pi)$ such that $w \in L(Q)$. Let μ be an unfolding of π that is simply labeled by w. Formally, μ is a copy of π in which each extra occurrence of a repeating edge or vertex of π is replaced by a fresh copy. Additionally, only one label of each edge is kept, so that $\textsc{Lbl}(\mu) = w$. Then, the assumption applied to the database that contains only μ yields $\textsc{Data}(\mathcal{V}_X(\mu)) = \textsc{Data}(\mu)$.

Assume $X = walk$ (resp. $X =$ trail). Let φ be the homomorphism (resp. edge-injective homomorphism) that folds μ into π, that is, φ maps each vertex and edge of μ to the original object in π. Since X is closed under homomorphism (resp. edge-injective homomorphism), $\textsc{Data}(\varphi(\mu)) \subseteq \textsc{Data}(\mathcal{V}_X(\varphi(\mu)))$, which means $\textsc{Data}(\pi) \subseteq \textsc{Data}(\mathcal{V}_X(\pi))$.

Moreover, X is monotone, thus $\textsc{Data}(\mathcal{V}_X(\pi)) \subseteq \textsc{Data}(\mathcal{V}_X(D))$. This implies that π is a walk (resp. trail) in $\textsc{Data}(\mathcal{V}_X(D))$. Therefore, Proposition 20 implies that $\mathcal{V} \twoheadrightarrow_X Q$. ◀

Proposition 23 leads to the definition of a *counterexample*, that is, a specific word in $L(Q)$ whose existence proves that $\mathcal{V} \not\twoheadrightarrow_X Q$.

▶ **Definition 24** (Counterexample under all-walks and trail semantics). *Let $X \in \{walk, trail\}$. Let $\mathcal{V}$ be a regular path view and Q be an RPQ. A counterexample for $\mathcal{V}$ and Q under X is a word $w \in L(Q)$ such that:*

- *$w = u \cdot a \cdot v$ for some $u, v \in \Sigma^*$ and $a \in \Sigma$;*
- *for all $u_1, u_2, v_1, v_2 \in \Sigma^*$ such that $u = u_1 \cdot u_2$ and $v = v_2 \cdot v_1$, for all $V \in \mathcal{V}$, $u_2 \cdot a \cdot v_2 \notin L(V)$.*

▶ **Proposition 25** (Counterexample criterion). *Let $X \in \{walk, trail\}$. Let $\mathcal{V}$ be a regular path view and Q be an RPQ. $\mathcal{V} \not\twoheadrightarrow_X Q$ if and only if there exists a counterexample for $\mathcal{V}$ and Q under X.*

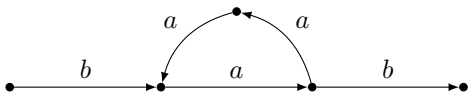
Proof. Proposition 25 is a simple consequence of Proposition 23. Indeed, $\mathcal{V} \not\twoheadrightarrow_X Q$ if and only if there is a simply labeled simple walk π that matches Q such that $\textsc{Data}(\mathcal{V}_X(\pi)) \neq \textsc{Data}(\pi)$, that is, there is an edge e of π such that e is not returned by any $V \in \mathcal{V}$. It then remains to remark that a suitable counterexample is $w = \textsc{Lbl}(\pi)$, with $a = \textsc{Lbl}(e)$. ◀

▶ **Example 26.** We present a few examples of determinacy under $X \in \{walk, trail\}$. Let $L(Q) = b(a^+b)^+$ as in Example 9.

1. $L(V_1) = ba^+ + a^+b$. Then $\{V_1\} \twoheadrightarrow_X Q$. Indeed, words $w \in L(Q)$ consist of concatenated blocks of the form ba^+b, which are entirely "seen" by the view:

$$\cdots \overbrace{\underbrace{baaaaaab}_{\in L(V_1)}\underbrace{aaaaaaab}_{\in L(V_1)}}^{\in L(V_1)\quad \in L(V_1)} \cdots$$

   While this example seems quite simple, it is interesting to remark that this is a case where $\mathcal{V}$ *does not* determine Q under shortest walk semantics, as we will see in Example 29.
2. $L(V_2) = ba^+b$. With the same argument, $\{V_2\} \twoheadrightarrow_X Q$. Remark that, under trail semantics, parts of the database that would match the query might be hidden to the view due to repeated edges. However, these parts will also *not* be usable by the query.



   In the example above, under $X = trail$, the a-cycle is hidden to the view (no trail that matches V_2 contains the top vertex), but it is also not in any answer to Q. Under $X = walk$, the a-cycle appears both in the view image and in the query answers. The fact that cycles are never an issue is a consequence of Proposition 23.
3. $L(V_3) = ba(aa)^* + (aa)^*ab$. In this case, the view only "sees" sequences of a of odd length, but those can combine to cover any length, as shown below. Hence, $\{V_3\} \twoheadrightarrow_X Q$.

$$\cdots \overbrace{\underbrace{baaaab}_{\in L(V_3)}}^{\in L(V_3)} \cdots$$

   However, for $L(V_3') = ba(aaa)^* + (aaa)^*ab$, $\{V_3'\} \not\twoheadrightarrow_X Q$. In the picture below, the word $w = baaab \in L(Q)$ but the middle a is "hidden".

$$\overbrace{b\ a}^{\in L(V_3')}\ a\ \overbrace{a\ b}^{\in L(V_3')}$$

Remark that in all examples, $\mathcal{V}$ contains a single query. This can actually be assumed without loss of generality, as a consequence of the proof of Theorem 2. Under $X \in \{trail, walk\}$, $\mathcal{V} \twoheadrightarrow_X Q$ if and only if $V_{all} \twoheadrightarrow_X Q$, where $L(V_{all}) = \bigcup_{V \in \mathcal{V}} L(V)$. We will see in Example 29 that this is not true under $X = short$.

Proposition 25 is the main ingredient for proving Theorem 2: we show that the set of counterexamples is actually a regular language, whose emptiness can be checked in PSPACE.

▶ **Theorem 2.** *Determinacy for regular path queries and views can be decided in* PSPACE *under all-walks and trail semantics, even for nondeterministic query and view automata.*

Proof. Let $X \in \{walk, short\}$. Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a regular path view and Q be a regular path query. Let $L_{\mathcal{V},Q}$ be the set of counterexamples for $\mathcal{V} \twoheadrightarrow_X Q$ as defined in Definition 24. Let $A_{\mathcal{V}} = (\Sigma, S_{\mathcal{V}}, \Delta_{\mathcal{V}}, I_{\mathcal{V}}, F_{\mathcal{V}})$ be any automaton for $L(V_1) \cup \ldots \cup L(V_n)$ and $A_Q = (\Sigma, S_Q, \Delta_Q, I_Q, F_Q)$ be an automaton for Q. Our goal is to construct an automaton $A_{\mathcal{V},Q}$ of at most exponential size such that $L(A_{\mathcal{V},Q}) = L_{\mathcal{V},Q}$, whose emptiness can then be checked in PSPACE.

The existence of $A_{\mathcal{V},Q}$ can be shown by using well-known closure properties of regular languages. We do this in Appendix B. Here, we will instead show a direct construction of $A_{\mathcal{V},Q}$. In addition to proving Theorem 2, this also serves as an introduction to the main result of Section 5, which expands upon the construction shown here.

Intuitively, $A_{\mathcal{V},Q}$ runs on a word $w \in L(Q)$ for which it nondeterministically guesses the factorization $u \cdot a \cdot v$ that yields the position a that should be hidden to $\mathcal{V}$. Then $A_{\mathcal{V},Q}$ runs $\mathcal{V}$ in parallel on each possible starting position in u, and checks that $\mathcal{V}$ does not accept a factor of w that contains a.

Formally, $A_{\mathcal{V},Q} = (\Sigma, S_{\mathcal{V},Q}, \Delta_{\mathcal{V},Q}, I_{\mathcal{V},Q}, F_{\mathcal{V},Q})$ is defined as follows:

- $S_{\mathcal{V},Q} = S_Q \times 2^{S_{\mathcal{V}}} \times \{0, 1\}$
- $I_{\mathcal{V},Q} = I_Q \times 2^{I_{\mathcal{V}}} \times \{0\}$
- $F_{\mathcal{V},Q} = F_Q \times 2^{S_{\mathcal{V}} - F_{\mathcal{V}}} \times \{1\}$
- For $q \in S_Q, P \subseteq S_{\mathcal{V}}$ and $b \in \Sigma$, $\Delta_{\mathcal{V},Q}$ contains transitions of the following forms:
  1. $(q, P, 0) \xrightarrow{b} \Delta_Q(q, b) \times \{\Delta_{\mathcal{V}}(P, b) \cup I_{\mathcal{V}}\} \times \{0\}$
  2. $(q, P, 0) \xrightarrow{b} \Delta_Q(q, b) \times \{\Delta_{\mathcal{V}}(P, b)\} \times \{1\}$ if $\Delta_{\mathcal{V}}(P, b) \cap F_{\mathcal{V}} = \emptyset$
  3. $(q, P, 1) \xrightarrow{b} \Delta_Q(q, b) \times \{\Delta_{\mathcal{V}}(P, b)\} \times \{1\}$ if $\Delta_{\mathcal{V}}(P, b) \cap F_{\mathcal{V}} = \emptyset$

▷ Claim 27. $L(A_{\mathcal{V},Q}) = L_{\mathcal{V},Q}$.

Assume that $w \in L(A_{\mathcal{V},Q})$. Then $A_{\mathcal{V},Q}$ has an accepting run over w that starts in some state $(q_0, P_0, 0)$ and ends in $(q_n, P_n, 1)$. Remark that transitions of the form (2) have to be used exactly once, as they are the only transitions that change the last component from 0 to 1, and there is no transition that changes 1 back to 0. This defines a unique factorization of w as $u \cdot a \cdot v$, where the transition is used when reading a after having read u.

Then, we prove by a straightforward induction that the state (q_i, P_i, f_i) that is reached after reading the prefix w_i of w of length i satisfies the following:

- $q_i \in \Delta_Q(I_Q, w_i)$ simulates a run of A_Q on w_i.
- If $i \leq \textsc{Len}(u)$, then P_i contains all states of $A_{\mathcal{V}}$ that can be reached at position i by starting on *any* position before i. That is, $p \in P_i$ if and only if there exists a suffix u_2 of w_i such that $p \in \Delta_{\mathcal{V}}(I_V, u_2)$.
- If $i \geq \textsc{Len}(u \cdot a)$, then P_i contains all states of $A_{\mathcal{V}}$ that can be reached at the current position by starting on *any* position of u. That is, $p \in P_i$ if and only if w_i if of the form $u_1 \cdot u_2 \cdot a \cdot v_2$, with $u = u_1 \cdot u_2$ and $p \in \Delta_{\mathcal{V}}(I_V, u_2 \cdot a \cdot v_2)$.

We conclude the proof by remarking that transitions of the form (2) and (3) enforce that $P \cap F_{\mathcal{V}} = \emptyset$. This means that for all factorizations of w as $u_1 \cdot u_2 \cdot a \cdot v_2 \cdot v_1$ with $u = u_1 \cdot u_2$ and $v = v_2 \cdot v_1$, we have $u_2 \cdot a \cdot v_2 \notin L(A_{\mathcal{V}})$, which is precisely what Definition 24 requires. Thus $w \in L_{\mathcal{V},Q}$.

The converse direction is proved in a very similar way. Assume that $w \in L_{\mathcal{V},Q}$. Then $w \in L(Q)$ and is of the form $w = u \cdot a \cdot v$. We then exhibit an accepting run of $A_{\mathcal{V},Q}$ on w that uses the nondeterministic transition (2) precisely when reading a.

▷ Claim 28. Emptiness of $A_{\mathcal{V},Q}$ can be checked in PSpace.

This is done by performing a reachability test from any initial state of $A_{\mathcal{V},Q}$ to any accepting state of $A_{\mathcal{V},Q}$. As $A_{\mathcal{V},Q}$ is a graph of exponential size, it is well known that this can be done in PSpace.

Therefore, emptiness of $L_{\mathcal{V},Q}$ can be checked in PSpace. Proposition 25 states that $L_{\mathcal{V},Q} = \emptyset$ if and only if $\mathcal{V} \twoheadrightarrow_X Q$ for $X \in \{walk, trail\}$. This concludes the proof. ◀

5 Shortest Walk Semantics

In this section, we prove Theorem 3. The proof reuses the ideas and the general approach of Section 4, which should be read first. However, the case of shortest walk semantics requires a much more careful and technical analysis. Section 5.1 presents the general idea and highlights what changes as compared to Section 4. Section 5.2 proves the main technical ingredient. Finally, Section 5.3 gives the PSpace decision procedure.

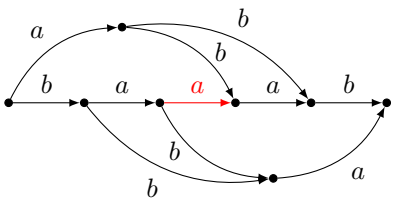
5.1 General Idea

As in Section 4, the starting point of our reasoning is Proposition 20: let Q be a regular path query and $\mathcal{V}$ be a regular path view. Then $V \twoheadrightarrow_{short} Q$ if and only if, for each database D and shortest walk π of D that matches Q, each edge e of π belongs to some walk in $V_{short}(D)$. What differs compared to Section 4 is that there are *two* cases that might cause an edge e to be missing from the view image:

1. Either e does not belong to any walk that matches some $V \in \mathcal{V}$, in which case the semantics under which V is evaluated does not matter.
2. Or e *does* belong to some walk π that matches $V \in \mathcal{V}$, *but* all such walks are not shortest walks: one can always find a shorter walk π' with the same endpoints that also matches V. Thus V does not return π under shortest walk semantics.

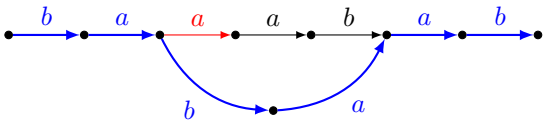
Case 2 hints at the fact that Proposition 23 is no longer true: under shortest walk semantics, determinacy does not have the walk counterexample property. There exists a regular path view $\mathcal{V}$ and an RPQ Q such that $\mathcal{V} \not\twoheadrightarrow_{short} Q$, but all databases that witness this fact are *not* simple walks. This is shown in Example 29 below.

▶ **Example 29.** Recall Q and V_1 defined by $L(Q) = b(a^+b)^+$ and $L(V_1) = ba^+ + a^+b$ from Example 26. Let $\mathcal{V} = \{V_1\}$. Then $\mathcal{V} \not\twoheadrightarrow_{short} Q$, as witnessed by the following database D:



Indeed, $Q_{short}(D)$ contains the "horizontal" walk labeled *baaab*. However, the middle edge (highlighted in red) does not belong to $\mathcal{V}_{short}(D)$. Indeed, despite belonging to the "horizontal" walks labeled *baa*, *baaa*, *aab* and *aaab*, none of these walks are shortest walks. For instance, the walk labeled *baa* is of length 3, but the "top" walk labeled *ab* is shorter, matches V_1 and has the same endpoints. However, we already know from Example 26 that $\mathcal{V}$ does determine Q on databases that are simple walks.

Interestingly, this example only works when the matched walk contains a single sequence of a. Indeed, when π contains two sequences of a, either the bottom or top "shortcuts" make it so that the "horizontal" walk is no longer a shortest walk, and thus need not be returned.

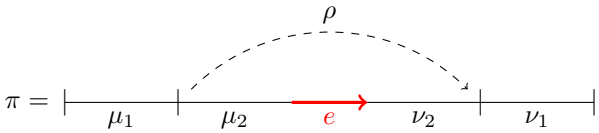


As shown on the picture above, if one tries to "hide" the red edge with the bottom "shortcut", then the blue walk becomes the shortest walk that matches Q from the leftmost vertex to the rightmost vertex. Therefore, the fact that the red edge is missing from the view is not an issue anymore. Thus, as counterintuitive as it may seem, $\mathcal{V} \twoheadrightarrow_{short} b(a^+b)(a^+b)^+$.

Another important remark is that, contrary to what is explained in Example 26, under shortest walk semantics, we cannot assume that all queries of $\mathcal{V}$ are grouped in a single query $L(V_{all}) = \bigcup_{V \in \mathcal{V}} L(V)$. Indeed, one can see that $\{(ba^+), (a^+b)\} \twoheadrightarrow_{short} b(a^+b)^+$: in that case, a walk labeled *baa* can only be shortcut by a walk labeled *ba*, and not by a walk labeled *ab*.

As illustrated in Example 29, counterexamples in the case of shortest walk semantics are more involved than in the case of walk and trail semantics. Here, a counterexample is a pair (π, e), where π is a walk that matches Q and e is an edge of π, such that, for each $V \in \mathcal{V}$, for each factorization of π as $\mu_1 \cdot \mu_2 \cdot e \cdot \nu_2 \cdot \nu_1$, one of the following holds. See the figure below to help visualize notations.

1. $\textsc{Lbl}(\mu_2 \cdot (e) \cdot \nu_2) \notin L(V)$.
2. There is a ρ such that $\textsc{Len}(\rho) \leq \textsc{Len}(\mu_2 \cdot e \cdot \nu_2)$, $\textsc{Lbl}(\rho) \in L(V)$ and $\textsc{Lbl}(\mu_1 \cdot \rho \cdot \nu_1) \notin L(Q)$.



As in the case of all-walks and trail semantics, it turns out that determinacy can be decided by only looking at the labels of counterexamples: we show in Section 5.2 how to prove a counterpart to Proposition 25 for shortest walk semantics. Then, we show in Section 5.3 that the set of these labels is a regular language. Thus, deciding determinacy amounts to checking the emptiness of this language. When Q is given as a deterministic automaton, we show that we can define an automaton of exponential size that precisely accepts this language and conclude that determinacy can be decided in PSpace.

5.2 Counterexample under Shortest Semantics

Definition 30 and Proposition 31 defined below are the key technical ingredients of Theorem 3.

▶ **Definition 30** (Counterexample under shortest semantics). *Let $\mathcal{V}$ be a regular path view and Q be an RPQ. A counterexample for $\mathcal{V}$ and Q under short is a word $w = u \cdot a \cdot v$, with $u, v \in \Sigma^*$ and $a \in \Sigma$ such that $w \in L(Q)$, and for all $u_1, u_2, v_1, v_2 \in \Sigma^*$ such that $u = u_1 \cdot u_2$ and $v = v_2 \cdot v_1$, for all $V \in \mathcal{V}$, one of the following holds:*

- $u_2 \cdot a \cdot v_2 \notin L(V)$;
- *there exists $r \in L(V)$ such that $\textsc{Len}(r) < \textsc{Len}(u_2 \cdot a \cdot v_2)$ and $u_1 \cdot r \cdot v_1 \notin L(Q)$.*

▶ **Proposition 31.** *Let $\mathcal{V}$ be a regular path view and Q be an RPQ. $\mathcal{V} \not\twoheadrightarrow_{short} Q$ if and only if there exists a counterexample for $\mathcal{V}$ and Q under short.*

Proof.

($\Leftarrow$) Assume that there exists a counterexample $w = u \cdot a \cdot v$. Then we build a database D as follows:

1. Add a simple walk π with $\textsc{Lbl}(\pi) = w$. Let μ and ν be the walks such that $\pi = \mu \cdot e \cdot \nu$ with $\textsc{Lbl}(\mu) = u$, $\textsc{Lbl}(\nu) = v$ and $\textsc{Lbl}(e) = a$.
2. For each $V \in \mathcal{V}$, for each factorization of μ as $\mu_1 \cdot (x) \cdot \mu_2$ and ν as $\nu_2 \cdot (y) \cdot \nu_1$, with x, y two vertices of D:
   - If $\textsc{Lbl}(\mu_2 \cdot e \cdot \nu_2) \notin L(V)$, there is nothing to do, and we continue with the next factorization.
   - If $\textsc{Lbl}(\mu_2 \cdot e \cdot \nu_2) \in L(V)$, we choose a word $r \in L(V)$ such that $\textsc{Len}(r) < \textsc{Len}(\mu_2 \cdot a \cdot \nu_2)$ and $\textsc{Lbl}(\mu_1 \cdot r \cdot v_1) \notin L(Q)$. We know that such a word exists by definition of w. Then we add to D a simple walk $\pi_{V,x,y}$ using only fresh vertices from x to y with $\textsc{Lbl}(\pi_{V,x,y}) = r$.

Then it is easy to check that π is the shortest walk from its source to its target that matches Q. Thus $\pi \in Q_{short}(D)$. However, e is not in $\mathcal{V}_{short}(D)$. Indeed, for every $V \in \mathcal{V}$, for every walk that contains e, either the walk is not matched by V, or D contains a strictly shorter walk with the same endpoints that matches V and does not contain e.

($\Rightarrow$) Assume that $\mathcal{V} \not\twoheadrightarrow_{short} Q$. Proposition 20 implies that there exists a database D and a walk π in D such that $\pi \in Q_{short}(D)$ and π is not a walk in $\textsc{Data}(\mathcal{V}_{short}(D))$. Since π is not a walk in $\textsc{Data}(\mathcal{V}_{short}(D))$, at least one edge e of π does not belong to $\mathcal{V}_{short}(D)$. Thus, π is of the form $\mu \cdot e \cdot \nu$.

Let $w = \textsc{Lbl}(\pi)$, $u = \textsc{Lbl}(\mu)$, $v = \textsc{Lbl}(\nu)$ and $a = \textsc{Lbl}(e)$. Then $w = u \cdot a \cdot v$. It remains to show that w is a counterexample.

Let $u = u_1 \cdot u_2$ and $v = v_2 \cdot v_1$ be factorizations of u and v. Let $\mu = \mu_1 \cdot (x) \cdot \mu_2$ be the corresponding factorization of μ, that is, $\textsc{Lbl}(\mu_1) = u_1$ and $\textsc{Lbl}(\mu_2) = u_2$. Similarly, let $\nu = \nu_2 \cdot (y) \cdot \nu_1$ be the corresponding factorization of ν.

Remark that $\mu_2 \cdot e \cdot \nu_2$ is a walk from (x) to (y). Let $V \in \mathcal{V}$. Since e does not belong to $\mathcal{V}_{short}(D)$, then $\mu_2 \cdot e \cdot \nu_2 \notin V(D)$. There are two possibilities:

1. $\mu_2 \cdot e \cdot \nu_2$ does not match V. This precisely means that $u_2 \cdot a \cdot v_2 \notin L(V)$, which satisfies the first requirement for counterexamples.
2. $\mu_2 \cdot e \cdot \nu_2$ matches V but is not a shortest walk among walks from (x) to (y) that match V. Thus D contains a shorter walk ρ from (x) to (y) such that $\rho \in V_{short}(D)$. Let $r = \textsc{Lbl}(\rho)$. Then $\textsc{Len}(r) < \textsc{Len}(u_2 \cdot a \cdot v_2)$ and $r \in L(V)$. Finally, remark that $\mu_1 \cdot \rho \cdot \nu_1$ is shorter than π and has the same endpoints. Thus $\mu_1 \cdot \rho \cdot \nu_1$ does not match Q, otherwise $\pi \notin Q_{short}(D)$. This implies that $u_1 \cdot r \cdot v_1 \notin L(Q)$, which satisfies the second requirement for counterexamples. ◀

5.3 Decision Procedure

We now have all ingredients to prove Theorem 3 below. This is done by showing that the set of counterexamples as defined in Definition 30 is actually a regular language, whose emptiness is equivalent to determinacy, as proved in Proposition 31. It now remains to give an automaton of at most exponential size that precisely accepts this language.

▶ **Theorem 3.** *Determinacy for regular path queries and views can be decided in* PSPACE *under shortest walk semantics, provided that the query is given as a deterministic automaton.*

Proof. Let $\mathcal{V} = \{V_1, \dots, V_n\}$ be a regular path view and Q be a regular path query. Let $L_{\mathcal{V},Q}$ be the set of counterexamples for $\mathcal{V} \twoheadrightarrow_{short} Q$, as defined in Definition 30. For each $V \in \mathcal{V}$, let $A_V = (\Sigma, S_V, \Delta_V, I_V, F_V)$ be any automaton for $L(V)$ and $A_Q = (\Sigma, S_Q, \delta_Q, q_0, F_Q)$ be a *deterministic* automaton for $L(Q)$. The automaton $A_{\mathcal{V},Q}$ for $L_{\mathcal{V},Q}$ is a much more involved version of the automaton used in the proof of Theorem 2, which should be read first.

Intuitively, $A_{\mathcal{V},Q}$ runs on a word $w \in L(Q)$ for which it nondeterministically guesses the factorization $u \cdot a \cdot v$ and the position a that should be hidden to $\mathcal{V}$. For each factor w' (denoted as $u_2 \cdot a \cdot v_2$ in Definition 30) of w that contains a, for each $V \in \mathcal{V}$, either w' does not belong to $L(V)$ (as in Theorem 2) or $A_{\mathcal{V},Q}$ nondeterministically guesses a "shortcut" for w': a word $r \in L(V)$ such that $\textsc{Len}(r) < \textsc{Len}(w')$, and maintains information to ensure that using this "shortcut" (ie, replacing w' with r in w) does not create an accepting run of A_Q.

Instead of giving an explicit transition table for $A_{\mathcal{V},Q}$, which would be very technical and not very informative, we give a high-level description of the information that are maintained and propagated throughout a computation of $A_{\mathcal{V},Q}$ on $w = u \cdot a \cdot v$. At step (i) of the computation, a state of $A_{\mathcal{V},Q}$ encodes the following information:

- The state $q^{(i)} \in S_Q$ of the computation of A_Q at step (i) on w.
- A bit $\textsc{Mid}^{(i)}$ that nondeterministically flips to 1 to say when $u \cdot a$ has been read.
- For each $V \in \mathcal{V}$, an array $\textsc{Run}_V^{(i)}[p, n] \subseteq S_V$ that maps each $p \in S_Q$ and integer $n \leq |S_Q| \times |S_V| + 1$ to the states of S_V that are reachable after a run of A_V on a factor of w that started before $\textsc{Mid}$ flipped, at step $(i - n)$, with the added information that $q^{(i-n)} = p$. Note that, thanks to standard pumping lemma techniques, we only count n up to $N_V = |S_Q| \times |S_V| + 1$, as will be explained below.
  This is a major difference with the proof of Theorem 2. Knowing when A_V accepts a factor that contains a is not enough: we also need to know the length n of said factor (in order to find a shorter "shortcut") and the state p of A_Q at the beginning of said factor (to ensure that using the "shortcut" does not yield an accepting run of A_Q).
- A set $\textsc{Err}^{(i)} \subseteq S_Q$ that contains "error" states of A_Q, that is, states that can be reached at step (i) by taking one of the "shortcuts". $\textsc{Err}$ must contain no accepting state at the end of the computation. $\textsc{Err}^{(i)}$ is populated as follows:
  - **(a)** If $\textsc{Mid}^{(i)} = 0$, $\textsc{Err}^{(i)} = \emptyset$. There are no "shortcuts" before $u \cdot a$.
  - **(b)** Otherwise, $\textsc{Err}^{(i)} \leftarrow \delta_Q(\textsc{Err}^{(i-1)}, w_i)$, where w_i is the ith symbol in w. We propagate the states in $\textsc{Err}$ according to the transition function of A_Q.
  - **(c)** Then, if $\textsc{Run}_V^{(i)}[p, n] \cap F_V \neq \emptyset$, this means that the suffix of length n at step (i) would reveal a through V. Nondeterministically guess a word $r \in L(V)$ such that $\textsc{Len}(r) < n$ and add $q_{err} = \delta_Q(p, r)$ to $\textsc{Err}^{(i)}$: using this "shortcut" brings A_Q to q_{err} at step (i).

Item (c) is the most critical part. Its correctness hinges on the following key remarks:

- If no suitable $r \in L(V)$ exists, the run aborts and rejects w. This corresponds to the case where there is no way to prevent V from revealing a.

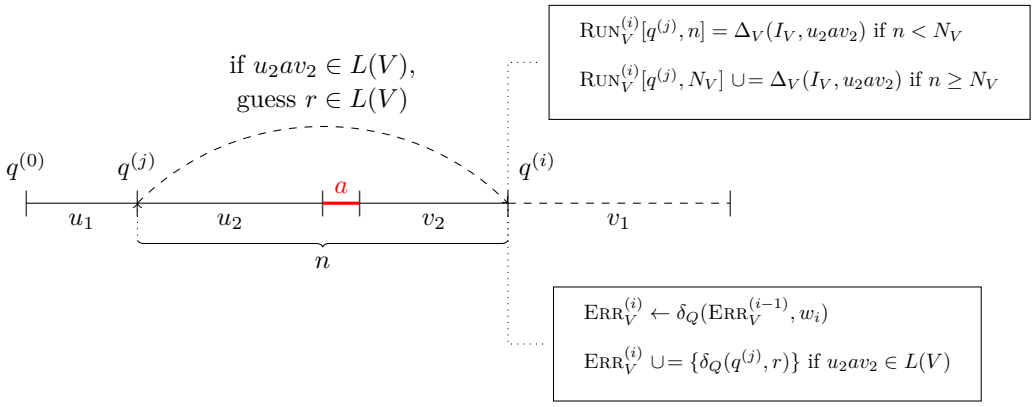


Figure 1 Notations and invariants for the proof of Theorem 3 at step (i) *after* the hidden edge.

- When $n \geq N_V$, we no longer need to know the precise value of n. Indeed, due to standard pumping lemma techniques, if there exists a word $r \in L(V)$ of length n such that $\delta_Q(p, r) = q_{err}$, then there exists a similar r' with $\text{LEN}(r') < N_V$.
- A_Q being deterministic is crucial here. It ensures that at step $(i - n)$, A_Q is in a unique state p. Keeping track of all possible runs of a nondeterministic automaton would require $\text{RUN}_V^{(i)}$ to be of exponential size.

Figure 1 provides a visual help for notations and update rules on a step (i) that occurs *after* the hidden edge, shown in red. Remark that each state only stores a polynomial amount of information, which means that $A_{\mathcal{V},Q}$ is of exponential size in the size of Q and $\mathcal{V}$. Thus, checking its emptiness can be done in PSPACE.

It remains to show that $L(A_{\mathcal{V},Q}) = L_{\mathcal{V},Q}$. As in the proof of Theorem 2, this is done by a technical but not very informative induction that shows that the invariants on Figure 1 are maintained. The precise update rules for $A_{\mathcal{V},Q}$ are given in Appendix C, along with the induction hypotheses. This concludes the proof. ◀

6 Conclusion

We have shown that the view-based query determinacy problem is decidable for regular path queries and views under three walk-based semantics: all-walks, trail and shortest walk semantics. We focused on trail and shortest walk semantics due to their prevalence in real-life systems, but our techniques naturally extend to other semantics. On the one hand, our analysis of trail semantics naturally extends to semantics that filter out results based on topological constraints (called *restrictors* in GQL [5], or *filter-based* in [16]), such as *acyclic* or *simple walk* semantics. On the other hand, the general idea for shortest walk semantics most likely[1] applies to other ranking semantics (called *selectors* in GQL, or *order-based* in [16]), which return the "best" results according to some criterion, such as *k-shortest* or *cheapest*.

Complexity-wise, we proved that the problem is PSPACE-hard under all three semantics, and provided a matching upper bound under all-walks and trail semantics. The case of shortest walk semantics is not entirely solved: our results only show that the problem is

[1] This claim is more prudent than for *restrictors* due to the wide variety of imaginable ranking criteria!

PSPACE-complete when the query is given as a deterministic automaton. Determinizing the input in order to apply our decision procedure would yield an EXPSPACE upper bound, which does not match the known lower bound. We leave the search for a tighter lower bound or a more efficient upper bound for future work.

Note that we have only considered the *static analysis* version of the problem. Another interesting direction for future work is the study of view-based query processing tasks where the view image is given as input, such as computing *certain answers* and *non-answers* [1].

References

1. Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 254–263, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/275487.275516.
2. Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011. doi:10.1016/j.tcs.2010.12.031.
3. Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *SIGMOD*, pages 1421–1432. ACM, 2018. doi:10.1145/3183713.3190654.
4. Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless regular views. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 247–258, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/543613.543646.
5. Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. In *SIGMOD'22*. ACM, 2022.
6. Nadime Francis. Asymptotic Determinacy of Path Queries using Union-of-Paths Views. *Theory of Computing Systems*, 61(1):156–190, July 2017. doi:10.1007/s00224-016-9697-x.
7. Nadime Francis, Amélie Gheerbrant, Paolo Guagliardo, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Liat Peterfreund, Alexandra Rogova, and Domagoj Vrgoč. A Researcher's Digest of GQL. In Floris Geerts and Brecht Vandevoort, editors, *ICDT'23*, volume 255 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPIcs.ICDT.2023.1.
8. Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, pages 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3183713.3190657.
9. Nadime Francis, Luc Segoufin, and Cristina Sirangelo. Datalog Rewritings of Regular Path Queries using Views. *Logical Methods in Computer Science*, 11(4), December 2015. doi:10.2168/LMCS-11(4:14)2015.
10. Grzegorz Głuch, Jerzy Marcinkowski, and Piotr Ostropolski-Nalewaja. The First Order Truth Behind Undecidability of Regular Path Queries Determinacy. In *22nd International Conference on Database Theory (ICDT 2019)*, volume 127 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:18, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICDT.2019.15.
11. Tomasz Gogacz and Jerzy Marcinkowski. The hunt for a red spider: Conjunctive query determinacy is undecidable. In *Logic in Computer Science (LICS), 2015 30th Annual ACM/IEEE Symposium on*, pages 281–292, United States, 2015. Institute of Electrical and Electronics Engineers. doi:10.1109/LICS.2015.35.

12 Tomasz Gogacz and Jerzy Marcinkowski. Red spider meets a rainworm: Conjunctive query finite determinacy is undecidable. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 121–134, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2902251.2902288.

13 Harry B. Hunt, Daniel J. Rosenkrantz, and Thomas G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12(2):222–268, 1976. doi:10.1016/S0022-0000(76)80038-4.

14 Sara Khichane. Study of new semantics for regular path queries. Master's thesis, Université Paris-Cité, 2024.

15 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 254–266, 1977. doi:10.1109/SFCS.1977.16.

16 Victor Marsault and Antoine Meyer. Designing and comparing rpq semantics. In *ICDT'26*, 2026. To appear.

17 Wim Martens, Matthias Niewerth, and Tina Popp. A trichotomy for regular trail queries. *Logical Methods in Computer Science*, Volume 19, Issue 4, December 2023. doi:10.46298/lmcs-19(4:20)2023.

18 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995. doi:10.1137/S009753979122370X.

19 Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Trans. Database Syst.*, 35(3), 2010. doi:10.1145/1806907.1806913.

20 Jean-Éric Pin, editor. *Handbook of Automata Theory*, volume 1. EMS, 2021.

21 TigerGraph Team. TigerGraph Documentation – version 3.1, 2021. URL: https://docs.tigergraph.com/.

A Handling Empty Words in Query and View Languages

In this section, we show how the case where the views or the query accept the empty word ε reduces to the case where ε is forbidden, by using no more than Definition 13.

▶ **Proposition 32.** *Let $X \in \{walk, trail, short\}$. Let Q be a regular path query and $\mathcal{V}$ be a regular path view. Then $\mathcal{V} \twoheadrightarrow_X Q$ if and only if:*

1. *If $\varepsilon \in L(Q)$, then there exists $V \in \mathcal{V}$ such that $\varepsilon \in L(V)$;*
2. *$\{\tilde{V} \mid V \in \mathcal{V}\} \twoheadrightarrow_X \tilde{Q}$, where $\tilde{A}$ is defined as any automaton such that $L(\tilde{A}) = L(A) \setminus \{\varepsilon\}$.*

Proof. Let $\tilde{\mathcal{V}} = \{\tilde{V} \mid V \in \mathcal{V}\}$. Assume that $\mathcal{V} \twoheadrightarrow_X Q$.

- Assume that $\varepsilon \in L(Q)$. Let D be a database that contains only a single isolated vertex x, and let D' be the empty database. Then $Q_X(D) = \{(x)\}$ and $Q_X(D') = \emptyset$. Assume for contradiction that for all $V \in \mathcal{V}$, $\varepsilon \notin L(V)$. Then $V_X(D) = V_X(D') = \emptyset$. However, $Q_X(D) \neq Q_X(D')$, thus $\mathcal{V} \not\twoheadrightarrow_X Q$.
- Assume for contradiction that $\tilde{\mathcal{V}} \not\twoheadrightarrow_X \tilde{Q}$. Then there exist two databases D and D' such that $\tilde{\mathcal{V}}_X(D) = \tilde{\mathcal{V}}_X(D')$ but $\tilde{Q}_X(D) \neq \tilde{Q}_X(D')$. Since $\varepsilon \notin L(\tilde{Q})$, this means that there exists an edge e that occurs only in one of $\tilde{Q}_X(D)$ or $\tilde{Q}_X(D')$. Since Q and $\tilde{Q}$ only differ on walks of length 0, e also only occur in one of $Q_X(D)$ or $Q_X(D')$. Remark now that since $\tilde{\mathcal{V}}_X(D) = \tilde{\mathcal{V}}_X(D')$, $\mathcal{V}_X(D)$ and $\mathcal{V}_X(D')$ only differ on walks of length 0, that is, isolated vertices in the view images. We build two new databases E and E' from D and D' by adding these isolated vertices to both E and E'. It remains to remark that $\mathcal{V}_X(E) = \mathcal{V}_X(E')$ while e still only occur in one of $Q_X(E)$ or $Q_X(E')$. Thus $\mathcal{V} \not\twoheadrightarrow_X Q$.

Conversely, assume that Properties 1 and 2 hold. Let D and D' be two databases such that $\mathcal{V}_X(D) = \mathcal{V}_X(D')$. In particular, $\mathcal{V}_X(D)$ and $\mathcal{V}_X(D')$ contain the same walks of length greater than 0. Thus $\tilde{V}_X(D) = \tilde{V}_X(D')$, from which (2) implies that $\tilde{Q}_X(D) = \tilde{Q}_X(D')$. If

$\varepsilon \notin L(Q)$, this immediately yields $Q_X(D) = Q_X(D')$, hence $\mathcal{V} \twoheadrightarrow_X Q$. Otherwise, $Q_X(D)$ and $Q_X(D')$ might differ on walks of length 0. However, (1) gives $V \in \mathcal{V}$ such that $\varepsilon \in L(V)$. Since $V_X(D) = V_X(D')$, D and D' must have the same walks of length 0 (that is, D and D' have the same set of vertices). Thus $Q_X(D) = Q_X(D')$ which concludes the proof. ◀

B Alternate Construction of $A_{\mathcal{V},Q}$ for the Proof of Theorem 2

Let $\mathcal{V} = \{V_1, \ldots, V_n\}$ be a regular path view and Q be a regular path query. By using standard automaton construction and closure properties, we show that there exists an automaton $A_{\mathcal{V},Q}$ of exponential size (in the size of the automata that define $\mathcal{V}$ and Q) that accepts $L_{\mathcal{V},Q}$, the set of counterexamples for $\mathcal{V}$ and Q under *walk* and *trail* (see Definition 24).

Let $\dot{\Sigma}$ be a disjoint copy of Σ, whose symbols are called *marked* symbols. For $a \in \Sigma$, we use $\dot{a}$ to denote its copy in $\dot{\Sigma}$.

Let M_Q be the automaton that accepts words in $L(Q)$ with a single marked symbol, that is $L(M_Q) = \{u \cdot \dot{a} \cdot v \mid a \in \Sigma, u \cdot a \cdot v \in L(Q)\}$. M_Q is simply constructed as a copy of any automaton A_Q for $L(Q)$ that nondeterministically guesses when to add the marked symbol, and is thus of polynomial size in the size of A_Q.

Let $A_{\mathcal{V}}$ be any automaton for $L(V_1) \cup \ldots \cup L(V_n)$. Similarly, let $B_{\mathcal{V}}$ be a copy of $A_{\mathcal{V}}$ that accepts words in $L(A_{\mathcal{V}})$ with a single marked symbol. Finally, let $M_{\mathcal{V}}$ be an automaton such that $L(M_{\mathcal{V}}) = \Sigma^* \cdot L(B_{\mathcal{V}}) \cdot \Sigma^*$. Again, $M_{\mathcal{V}}$ can be guaranteed to be of polynomial size in the size of $A_{\mathcal{V}}$. Intuitively, $M_{\mathcal{V}}$ contains the words in which $\mathcal{V}$ "sees" the marked symbol.

Finally, let $H_{\mathcal{V}}$ be an automaton for $\left(\overline{L(M_{\mathcal{V}})}\right) \cap \left(\Sigma^* \cdot \dot{\Sigma} \cdot \Sigma^*\right)$. Intuitively, $H_{\mathcal{V}}$ accepts the words in which the marked symbol is "hidden" to $\mathcal{V}$. Indeed, if $w = u \cdot \dot{a} \cdot v$ is such that $\mathcal{V}$ sees $\dot{a}$, then $w = u_1 \cdot u_2 \cdot \dot{a} \cdot v_2 \cdot v_1$ with $u_2 \cdot a \cdot v_2 \in L(V)$ for some $V \in \mathcal{V}$. This means that $u_2 \cdot \dot{a} \cdot v_2 \in L(B_{\mathcal{V}})$ and thus $u_1 \cdot u_2 \cdot \dot{a} \cdot v_2 \cdot v_1 \in L(M_{\mathcal{V}})$. Remark that $H_{\mathcal{V}}$ can be guaranteed to be at most of exponential size in the size of $M_{\mathcal{V}}$, with a standard construction for the complement of a nondeterministic automaton.

It then remains to remark that $L_{\mathcal{V},Q}$, is simply $L(H_{\mathcal{V}}) \cap L(M_Q)$ with the mark removed. Thus $A_{\mathcal{V},Q}$ can be constructed as an automaton of at most exponential size.

C Additional Material for the Proof of Theorem 3

This section completes the proof of Theorem 3 given in Section 5.3.

C.1 Update Rules

A formal description of the transitions of $A_{\mathcal{V},Q}$ are given by the update rules below. Notations follow the notations of the proof in Section 5.3.

Initial state.

- $q^{(0)} = q_0$, the initial state of A_Q.
- $\textsc{Run}_V^{(0)}[q_0, 0] = I_V$. All other values are set to $\emptyset$.
- $\textsc{Err}^{(0)} = \emptyset$.

Computation step strictly before $u \cdot a$ when reading a symbol $b \in \Sigma$.

- $q^{(i+1)} \leftarrow \delta_Q(q^{(i)}, b)$
- $\textsc{Err}^{(i+1)} \leftarrow \emptyset$.
- For each $V \in \mathcal{V}, p \in S_Q, 0 < n \leq N_V$:

- If $n < N_V$, $\textsc{Run}_V^{(i+1)}[p, n+1] \leftarrow \Delta_V(\textsc{Run}_V^{(i)}[p, n], b)$.
- If $n = N_V$, $\textsc{Run}_V^{(i+1)}[p, n] \cup= \Delta_V(\textsc{Run}_V^{(i)}[p, n], b)$.

- For each $V \in \mathcal{V}, p \in S_Q$, $\textsc{Run}_V^{(i+1)}[p, 0] \leftarrow \emptyset$ if $p \neq q$ and $\textsc{Run}_V^{(i+1)}[q, 0] \leftarrow I_V$.

Computation step on or after $u \cdot a$, when reading a symbol $b \in \Sigma$.

- $q^{(i+1)} \leftarrow \delta_Q(q^{(i)}, b)$
- $\textsc{Err}^{(i+1)} \leftarrow \delta_Q(\textsc{Err}^{(i)}, b)$.
- For each $V \in \mathcal{V}, p \in S_Q, 0 < n \leq N_V$:
  - If $n < N_V$, $\textsc{Run}_V^{(i+1)}[p, n+1] \leftarrow \Delta_V(\textsc{Run}_V^{(i)}[p, n], b)$.
  - If $n = N_V$, $\textsc{Run}_V^{(i+1)}[p, n] \cup= \Delta_V(\textsc{Run}_V^{(i)}[p, n], b)$.
- For each $V \in \mathcal{V}, p \in S_Q$, $\textsc{Run}_V^{(i+1)}[p, 0] \leftarrow \emptyset$.
- Then, for each $V \in \mathcal{V}, p \in S_Q, 0 < n \leq N_V$:
  - If $\textsc{Run}_V^{(i+1)}[p, n] \cap F_V = \emptyset$, skip this step.
  - Otherwise, nondeterministically choose a state q_{err} such that there exists a word $r \in L(V)$ with $\textsc{Len}(r) < n$ and $q_{err} = \delta_Q(p, r)$. If no such r exists, abort the run.
  - $\textsc{Err}^{(i+1)} \leftarrow \textsc{Err}^{(i+1)} \cup \{q_{err}\}$.

End state. Accepting states are states where $\textsc{Mid} = 1$, $q \in F_Q$ and $\textsc{Err} \cap F_Q = \emptyset$.

C.2 Induction Hypotheses

The correctness of $A_{\mathcal{V},Q}$ comes from the following claim, that imitates Definition 30.

▷ Claim 33. Let $w = u \cdot a \cdot v \in \Sigma^*$ and assume that $A_{\mathcal{V},Q}$ has a run on w such that $\textsc{Mid}$ flips to 1 after reading $u \cdot a$ if and only if there exists a family of words $(r_{V,k,\ell})_{V \in \mathcal{V}, k < \textsc{Len}(ua), \textsc{Len}(ua) \leq \ell \leq \textsc{Len}(w)}$ such that, for each factorization of w as $u_1 \cdot u_2 \cdot a \cdot v_2 \cdot v_1$, for all $V \in \mathcal{V}$, one of the following holds:

1. $u_2 \cdot a \cdot v_2 \notin L(V)$;
2. $r_{V,k,\ell} \in L(V)$, where $k = \textsc{Len}(u_1)$, $\ell = \textsc{Len}(u \cdot a \cdot v_2)$, $\textsc{Len}(r_{V,k,\ell}) < \textsc{Len}(u_2 \cdot a \cdot v_2)$.

Moreover, this run is accepting if and only if $w \in L(Q)$ and in each case 2, $u_1 \cdot r_{V,k,\ell} \cdot v_1 \notin L(Q)$.

The proof of this claim is done by inductively constructing the family $(r_{V,\ell,k})$ along a run of $A_{\mathcal{V},Q}$. The construction relies on the following induction hypotheses that hold at step (i) of the computation on $w = w_1 \dots w_d$.

- $q^{(0)} = q_0$ and for $i > 0, q^{(i)} = \delta_Q(q_0, w_i)$, where w_i is the prefix of length i of w.
- For $n < N_V$ and $i - n < \textsc{Len}(u \cdot a)$:

$$\textsc{Run}_V^{(i)}[q^{(i-n)}, n] = \Delta(I_V, w_{i-n} \dots w_i)$$

- For $n = N_V$ and $i - n < \textsc{Len}(u \cdot a)$ and $p \in S_Q$:

$$\textsc{Run}_V^{(i)}[p, n] = \bigcup_{\substack{m \geq n \\ q^{(i-m)} = p}} \Delta(I_V, w_{i-m} \dots w_i)$$

- For $i < \textsc{Len}(u \cdot a)$, $\textsc{Err}^{(i)} = \emptyset$.

- For $i \geq \textsc{Len}(u \cdot a)$:

$$\textsc{Err}^{(i)} = \bigcup_{\substack{k<\ell\leq i \\ \textsc{Run}_V^{(\ell)}[p,\min(\ell-k,N_V)]\cap F_V\neq\emptyset}} \delta_Q(q_0, w_1 \dots w_k \cdot r_{V,k,\ell} \cdot w_{\ell+1} \dots w_i)$$

  where, in the equation above, p is any state in S_Q, $w_1 \dots w_k$ is replaced by ε if $k = 0$ and $w_{\ell+1} \dots w_i$ is replaced by ε if $\ell = i$.

Finally, the run is accepting if and only if $q^{(d)} \in F_Q$ and $\textsc{Err}^{(d)} \cap F_Q = \emptyset$, which is exactly what the claim requires.