Datalog Rewritings of Regular Path Queries using Views

Nadime Francis, Luc Segoufin, Cristina Sirangelo

Datalog Rewritings of Regular Path Queries using Views

Nadime Francis
ENS-Cachan, INRIA

Luc Segoufin
INRIA, ENS-Cachan

Cristina Sirangelo
LSV at ENS-Cachan, INRIA, CNRS

Abstract

We consider query answering using views on graph databases, i.e. databases structured as edge-labeled graphs. We mainly consider views and queries specified by Regular Path Queries (RPQ). These are queries selecting pairs of nodes in a graph database that are connected via a path whose sequence of edge labels belongs to some regular language. We say that a view $\mathbf{V}$ determines a query Q if for all graph databases D, the view image $\mathbf{V}$(D) always contains enough information to answer Q on D. In other words, there is a well defined function from $\mathbf{V}$(D) to Q(D).

Our main result shows that when this function is monotone, there exists a rewriting of Q as a Datalog query over the view instance $\mathbf{V}$(D). In particular the rewriting query can be evaluated in time polynomial in the size of $\mathbf{V}$(D). Moreover this implies that it is decidable whether an RPQ query can be rewritten in Datalog using RPQ views.

1 Introduction

We consider the problem of answering queries using views on graph databases. Graph databases are relational databases where all relation symbols are binary. In other words a graph database can be viewed as an edge-labeled directed graph.

Graph-structured data can be found in many important scenarios. Typical examples are the semantic Web via the format RDF and social networks. Graph-structured data differs conceptually from relational databases in that the topology of the underlying graph is as important as the data it contains. Usual queries will thus test whether two nodes are connected and *how* they are connected [4].

In many contexts it is useful to know whether a given set of queries can be used to answer another query. A typical example is the data integration setting where data sources are described by views of a virtual global database. Queries over the global database are then rewritten as queries over the views. Another example is caching: answers to some set of queries against a data source are cached, and one wishes to know if a newly arrived query can be answered using the cached information, without accessing the source. This problem also finds application in the context of security and privacy. Suppose access to some of the information in a database is provided by a set of public views, but answers to other queries are to be kept secret. This requires verifying that the disclosed views do not provide enough information to answer the secret queries.

All these problems can be phrased in terms of views and query rewriting using views, which is a typical database problem, not specific to graph databases, that has received considerable attention (see [12, 13, 3] among others). When graph databases are concerned, the difference lies only in the kind of queries under consideration [6, 8, 7, 9].

Over graph databases, typical queries have at least the expressive power of *Regular Path Queries* (RPQ), defined in [10] (see also the survey [4]). An RPQ selects pairs of nodes connected by a path whose sequence of edge labels satisfies a given regular expression. A view, denoted by $\mathbf{V}$, is then specified using a finite set of RPQs. When evaluated over a graph database D, the view $\mathbf{V}$ yields a new graph database $\mathbf{V}(\mathrm{D})$ where each $V_i \in \mathbf{V}$ is a new edge relation symbol.

We are interested in knowing whether the view $\mathbf{V}$ always provides enough information to answer another RPQ query Q, i.e. whether Q(D) can be computed from $\mathbf{V}(\mathrm{D})$ for all databases D. When this is the case we say that $\mathbf{V}$ *determines* Q, and we look for a *rewriting* of Q using $\mathbf{V}$, i.e. a new query, in some query language, that expresses Q in terms of $\mathbf{V}$. We are then interested in finding an algorithm for evaluating the rewriting, i.e. an algorithm computing Q(D) from $\mathbf{V}(\mathrm{D})$.

These two related questions, determinacy and query rewriting, have been studied for relational databases and graph databases. Over relational databases, determinacy is undecidable already if the queries and views are defined by union of conjunctive queries, and its decidability status is open for views and queries specified by conjunctive queries (CQ) [13]. Over graph databases and RPQ queries and views, the decidability status of determinacy is also open [7]. Determinacy has been shown to be decidable in a scenario where views and queries can only test whether there is a path of distance k between the two nodes, for some given k [3]. This scenario lies at the intersection of CQ and RPQ and contains already non trivial examples. For instance the view Path_3 and Path_4, giving respectively the pairs of nodes connected by a path of length 3 and 4, determines the query Path_5 asking for the pairs of nodes connected by a path of length 5 [3] (see also Example 2 in Section 2).

Clearly when Q can be rewritten in terms of $\mathbf{V}$, the rewriting witnesses that $\mathbf{V}$ determines Q. On the other hand determinacy does not say that one can find a rewriting definable in a particular language, nor with particular computational properties.

It is then natural to ask which rewriting language $\mathcal{L}_\mathcal{R}$ is sufficiently powerful so that determinacy is equivalent to the existence of a rewriting definable in $\mathcal{L}_\mathcal{R}$. This clearly depends on the language used for defining the query and the view.

Consider again the case of Path_5 that is determined by Path_3 and Path_4. A rewriting $R(x, y)$ of Path_5 in terms of Path_3 and Path_4 is defined by:

$$\exists u\ (\mathrm{Path}_4(x, u) \wedge \forall v\ (\mathrm{Path}_3(v, u) \rightarrow \mathrm{Path}_4(v, y)))$$

and it can be shown that there is no rewriting definable in CQ, nor in RPQ (cf. Example 2). In the case of views and queries defined by CQs it is still an open problem to know whether first-order logic is a sufficiently powerful rewriting language. Even worse, it is not even known whether there always exists a rewriting that can be evaluated in time polynomial in the size of the view instance [13], ie. polynomial *data complexity*. A similar situation arises over graph-databases and RPQ views and queries [7].

It can be checked that in the example above there exists no monotone rewriting of Path_5 (see again Example 2). In particular, as RPQs define only monotone queries, no rewriting is definable in RPQ. Monotone query languages such as CQ, Datalog, RPQ and their extensions are of crucial importance in many database applications. The possibility of expressing rewritings in these languages is subject to a monotonicity restriction.

This is why in this paper we are considering a stronger notion of determinacy, referred to as *monotone determinacy*, by further requiring that the mapping from view instances to query results is *monotone*.

In the case when views and queries are defined by CQs, monotone determinacy can be shown to be equivalent to the existence of a rewriting in CQ [13]. As this latter problem is decidable [12], monotone determinacy for CQs is decidable.

We consider here monotone determinacy for graph databases and views and queries defined by RPQs. We first observe that monotone determinacy corresponds to the notion called *losslessness under the sound view assumption* in [7], where it was shown to be decidable. We then concentrate on the rewriting problem.

We know that there exist cases of monotone rewritings that are not expressible in RPQ [7] (see also Example 13 in Section 5). We thus need a more powerful language in order to express all monotone rewritings.

It is not too hard to show that if $\mathbf{V}$ determines Q then there exists a rewriting with NP data complexity, as well as a rewriting with coNP data complexity. Our main result shows that if moreover $\mathbf{V}$ determines Q in a monotone way, there exists a rewriting definable in Datalog, which therefore can be evaluated in polynomial time.

Our proofs are constructive, hence the Datalog rewriting can be computed from $\mathbf{V}$ and Q.

As a corollary this implies that it is decidable whether a query Q has a rewriting definable in Datalog using a view $\mathbf{V}$, where both $\mathbf{V}$ and Q are defined using RPQs. This comes from the fact that our main result implies that the existence of a rewriting in Datalog is equivalent to monotone determinacy, a decidable property as mentioned above.

Related work The work which is most closely related to ours is that of the "Four Italians". In particular, the notion of losslessness under the exact view assumption introduced in [7] corresponds to what we call determinacy; similarly the notion of losslessness under the sound view assumption corresponds to what we call monotone determinacy. Monotone determinacy is also mentioned in the thesis [14] under the name of "strong determinacy". It is shown there that it corresponds to the existence of a monotone rewriting.

A lot of attention has been devoted to the problem of computing the set of certain answers to a query w.r.t a set of views, under the sound view assumption (see the precise definition of certain answers in Section 6.1). For RPQ views and queries, the problem is shown to be equivalent to testing whether the given instance homomorphically embeds into a structure $T_{\mathrm{Q},\mathbf{V}}$ computed from the view $\mathbf{V}$ and the query Q [6]. In general this shows that the data complexity of computing the certain answers is coNP-complete. Building on results on Constraint Satisfaction Problems [11], it was also shown in [6] that for an RPQ view $\mathbf{V}$, an RPQ query Q and for each l, k, with $l \leq k$, there is a Datalog program $Q_{l,k}$ which is contained in the certain answers to Q given $\mathbf{V}$ and is, in a sense, maximally contained: i.e. $Q_{l,k}$ contains all Datalog programs which are contained in the certain answers and use at most l head variables and at most k variables in each rule.

If we assume that $\mathbf{V}$ determines Q in a monotone way, it is easy to see that the query computing the certain answers under the sound view assumption is a rewriting of Q using $\mathbf{V}$ (i.e the certain answers of a view instance $\mathbf{V}(\mathrm{D})$ are precisely the query result $\mathrm{Q}(\mathrm{D})$). However there are possibly other rewritings (they only need to agree on instances of the form $\mathbf{V}(\mathrm{D})$, but may possibly differ on instances not in the image of $\mathbf{V}$.) While computing the certain answers is coNP-hard, our main result shows that there exists another rewriting which is expressible in Datalog, and has therefore polynomial time data complexity.

Nevertheless our proof makes use of the structure $T_{\mathrm{Q},\mathbf{V}}$ mentioned above, and our Datalog rewriting turns out to be the query $Q_{l,k}$ associated with Q and $\mathbf{V}$ for some suitable values of l and k.

2 Preliminaries

Graph databases and paths A binary schema is a finite set of relation symbols of arity 2. All the schemas used in this paper are binary. A *graph database* D is a finite relational structure over a (binary) schema σ. We will also say a σ-structure. Alternatively D can be viewed as a directed edge-labeled graph with labels from the alphabet σ. The elements of the domain of D are referred to as *nodes*. The number of elements in D is denoted by $|\mathrm{D}|$. If A is a set of elements of D, we denote by $\mathrm{D}[A]$ the substructure of D induced by A.

Given a graph database D, a *path* π in D from x_0 to x_m is a finite sequence $\pi = x_0 a_0 x_1 \dots x_{m-1} a_{m-1} x_m$, where each x_i is a node of D, each a_i is in σ, and $a_i(x_i, x_{i+1})$ holds in D for each i. A *simple path* is a path such that no node occurs twice in the sequence. The *label* of π, denoted by $\lambda(\pi)$, is the word $a_0 a_1 \dots a_{m-1} \in \sigma^*$. By abuse of notation, we sometimes view a path π as a graph database, which contains only the nodes and edges that occur in the sequence.

Queries and query languages A *query* Q over a schema σ is a mapping associating to each graph database D over σ a finite relation Q(D) over the domain of D. We will only consider binary queries, that is queries that return binary relations, and work with the following query languages.

A *Regular Path Query* (often abbreviated as RPQ) over σ is given by a regular expression over the alphabet σ. If Q is an RPQ, we denote by $L(\mathrm{Q})$ the language corresponding to its regular expression. On a graph database, such a query selects all the pairs (x, y) of nodes such that there exists a path π from x to y with $\lambda(\pi) \in L(\mathrm{Q})$. For instance the query Path_3 of the introduction is an RPQ corresponding to the regular expression $\sigma\sigma\sigma$ (also denoted σ^3). Another example is the RPQ $(\sigma\sigma)^*$ that select pairs of nodes connected via a path of even length.

A *Context-Free Path Query* over σ is defined similarly but using a context-free grammar instead of a regular expression.

A *Conjunctive Regular Path Query* (sometimes abbreviated CRPQ) over σ is a conjunctive query whose atoms are specified using RPQs over σ. For instance the query

$$\exists z\ Q_1(x, z) \wedge Q_2(z, y) \wedge Q_3(z, y)$$

where $Q_1 = a^+$, $Q_2 = b$ and $Q_3 = c$ selects pairs of nodes (x, y) which are connected via a path labeled a^+b and another path labeled a^+c sharing their a^+ part. This cannot be expressed by an RPQ.

A *Datalog query* over schema σ is defined by a finite set of rules of the form

$$I(\bar{x}) :- I_1(\bar{x}_1) \wedge \cdots \wedge I_m(\bar{x}_m)$$

where each I_i is a relational symbol, either a symbol from σ, or an *internal symbol*. $I(\bar{x})$ is called the *head* of the rule and I must be an internal symbol. The variables $\bar{x}$ are among $\bar{x}_1 \ldots \bar{x}_m$ and the variables of $\bar{x}_i$ not occurring in $\bar{x}$ should be understood as existentially quantified. One of the internal symbols, referred to as the *goal*, is binary and is designated as being the output of the query. The evaluation of a Datalog query computes the internal relations incrementally starting from the empty ones by applying greedily the rules (see [2]).

It is easy to see that any Regular or Context-Free Path Query, and therefore any Conjunctive Regular Path Query, can be expressed in Datalog. Hence Datalog is the most expressive of the query languages presented above. It is also well known that each Datalog query can be evaluated in polynomial time, data complexity, using the procedure briefly sketched above.

We will consider restrictions of Datalog limiting the maximal arity of the internal symbols and the number of variables in each rule. This is classical in the context of Constraint Satisfaction Problems (CSP) [11] that we will use in Section 6. In the context of CSP, Datalog programs are boolean (i.e. the goal has arity 0) and $\mathrm{Datalog}_{l,k}$ denotes the fragment allowing at most k variables in each rule and internal symbols of arity at most l. Here we are dealing with *binary* Datalog programs. In order to stay close to the notations and results coming from CSP, we generalize this definition and let $\mathrm{Datalog}_{l,k}$ denote the Datalog programs having at most $k + r$ variables in each rule and internal symbols of arity at most $l + r$, where r is the arity of the goal, in our case $r = 2$.

Views If σ and τ are (binary) schemas, a *view* $\mathbf{V}$ from σ to τ is a set consisting of one binary query over σ for each symbol in τ. If $\mathbf{V}$ consists of the queries $\{V_1, \ldots, V_n\}$, with a little abuse of notation, we let each V_i also denote the corresponding symbol in τ. For a graph database D over σ, we denote by $\mathbf{V}(\mathrm{D})$ the graph database over τ where each binary symbol V_i is instantiated as $V_i(\mathrm{D})$. We say that a view consisting of the queries $\{V_1, \ldots, V_n\}$ is an RPQ view if each V_i is an RPQ. We define similarly Context-Free Path Query views and Conjunctive Regular Path Query views.

In what follows whenever we refer to a view $\mathbf{V}$ and a query Q, unless otherwise specified, we always assume that Q is over the schema σ and $\mathbf{V}$ is a view from σ to τ. A *view instance* E is a τ-structure such that $\mathrm{E} = \mathbf{V}(\mathrm{D})$ for some database D.

Determinacy and rewriting The notion of determinacy specifies when a query can be answered completely from the available view. The following definitions are taken from [13].

Definition 1 (Determinacy)**.** *We say that a view $\boldsymbol{V}$ determines a query Q if :*

$$\forall D, D', \quad \boldsymbol{V}(D) = \boldsymbol{V}(D') \;\Rightarrow\; Q(D) = Q(D')$$

In other words, Q(D) only depends on the view instance $\mathbf{V}$(D) and not on the particular database D yielding the view. Observe that determinacy says that there exists a function f defined on view images such that Q(D) = $f(\mathbf{V}(\mathrm{D}))$ for each database D. We call f the *function induced by Q using $\boldsymbol{V}$.*

A *rewriting* of Q using $\mathbf{V}$ is a query R over the schema τ such that R($\mathbf{V}$(D)) = Q(D) for all D.

Notice that there can be possibly many rewritings, while the function induced by Q using $\mathbf{V}$ is unique. In fact the domain of f is defined to be the set of view images, that is, all the τ-structures E such that there exists a database D with $\mathbf{V}$(D) = E. Thus, f is fully defined by the identity Q(D) = $f(\mathbf{V}(\mathrm{D}))$, and is therefore unique. On the contrary, rewritings are defined as queries over τ, which means that they are mappings defined over all τ-structures E, even those which are not of the form E = $\mathbf{V}$(D). In particular, this means that the condition Q(D) = R($\mathbf{V}$(D)) is not sufficient to fully define R, as it can take arbitrary values on τ-structures that are not of the form $\mathbf{V}$(D). Of course any rewriting coincides with the function f *when restricted to view images.*

Example 2. *Consider again the view $\boldsymbol{V}$ defined by the two RPQs $V_1 = \sigma^3$ and $V_2 = \sigma^4$ testing for the existence of a path of length 3 and 4, respectively. Let $Q = \sigma^5$ be the RPQ testing for the existence of a path of length 5.*

It turns out that $\boldsymbol{V}$ determines Q [3]. This is not immediate to see but, as mentioned in Section 1, one can verify that a rewriting of Q using $\boldsymbol{V}$ can be expressed in first-order by the following query:

$$\exists u\ (V_2(x,u) \wedge \forall v\ (V_1(v,u) \Rightarrow V_2(v,y)))$$

As shown in Figure 1, the function induced by Q using $\boldsymbol{V}$ is not monotone. This implies that no monotone query can be a rewriting, in particular there exists no CQ nor RPQ rewriting.

Consider now the RPQ $Q' = \sigma^2$. One can verify that $\boldsymbol{V}$ does not determine Q'. Indeed the database consisting of a single node with no edge, and the database consisting of a single path of length 2, have the same empty view but disagree on Q'.

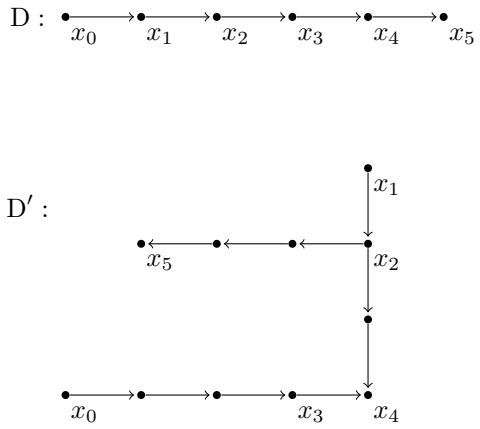


Fig. 1: Illustration for Example 2: D and D′ are such that $\mathbf{V}$(D) $\subseteq$ $\mathbf{V}$(D′), but $(x_0, x_5) \in$ Q(D), whereas $(x_0, x_5) \notin$ Q(D′). Hence the function induced by Q using $\mathbf{V}$ is not monotone.

It is important at this point to understand the difference between determinacy and rewriting. If $\mathbf{V}$ determines Q then there exists a rewriting of Q using $\mathbf{V}$. However there are possibly many

rewritings of Q using $\mathbf{V}$. Each of them agrees on the function induced by Q using $\mathbf{V}$ when restricted to view images, but can take arbitrary values on structures that are not in the image of the view. Consider for instance the view $\mathbf{V}$ and the query Q of Example 2. The query:

$$\exists u, u' \; V_2(x,u) \wedge V_1(x,u') \wedge \forall v \; (V_1(v,u) \Rightarrow V_2(v,y))$$

is also a rewriting of Q using $\mathbf{V}$. It is equivalent to the rewriting of Example 2 on τ-instances E such that $\mathrm{E} = \mathbf{V}(\mathrm{D})$ for some D. Indeed whenever $V_2(x,u)$ holds in $\mathbf{V}(\mathrm{D})$, the database D contains a path of length 4 from x to u, hence if u' is the node at distance 3 from x in this path, $V_1(x,u')$ also holds in $\mathbf{V}(\mathrm{D})$. However the two rewritings may differ on instances which are not in the view image, such as an instance consisting of a single V_2-labeled edge.

The *determinacy problem* for a query language $\mathcal{L}$ is the problem of deciding, given an input view $\mathbf{V}$ defined in $\mathcal{L}$ and a query Q of $\mathcal{L}$, whether $\mathbf{V}$ determines Q.

Determinacy does not say whether there exists a rewriting definable in a particular query language, or computable with a particular data complexity. This clearly depends on the language used for specifying the views and queries.

The *rewriting problem* for a query language $\mathcal{L}$ is the problem of finding a rewriting for a query Q of $\mathcal{L}$ using a view $\mathbf{V}$ defined in $\mathcal{L}$ whenever $\mathbf{V}$ determines Q.

These two problems have been thoroughly investigated in the case that $\mathcal{L}$ is RPQ [7, 6, 8, 9]. However the determinacy problem for RPQ remains wide open and it is not clear what would be a good (low data complexity) rewriting language for RPQ. Note that a similar situation arises in the case that $\mathcal{L}$ is CQ [13, 3].

3 Determinacy problem

We have already mentioned above that the determinacy problem for RPQs is open. For Context-Free Path Queries and for Conjunctive Regular Path Queries, determinacy is undecidable. Actually the problem is already undecidable when the query Q is an RPQ. These undecidability results are formalized in the two following propositions.

Proposition 3. *Given a Context-Free Path Query view $\mathbf{V}$ and a Regular Path Query Q, it is undecidable whether $\mathbf{V}$ determines Q.*

Proof. We prove this by reduction from the universality problem for context-free languages. Let L be a context-free language over some alphabet σ. Let \$ be a fresh symbol that does not appear in σ. Let $\mathbf{V} = \{V\}$ where V is defined by $L(V) = \$ \cdot L \cdot \$$. Let Q be defined by $L(\mathrm{Q}) = \$ \cdot \sigma^* \cdot \$$. Then $\mathbf{V}$ determines Q if and only if L is universal over σ.

- Assume that L is universal. Then $\mathrm{Q} = V$ and it is easy to check that $R = V$ is a rewriting of Q using $\mathbf{V}$.
- Conversely, assume that L is not universal. Then there exists $w \in \sigma^*$ such that $w \notin L$. Consider the database D consisting of a simple path labeled by $\$ \cdot w \cdot \$$, and the empty database D′. Then $\mathbf{V}(\mathrm{D}) = \emptyset = \mathbf{V}(\mathrm{D}')$, but $\mathrm{Q}(\mathrm{D})$ contains the first and last node of the path, whereas $\mathrm{Q}(\mathrm{D}')$ is empty. Hence, $\mathbf{V}$ does not determine Q.

□

Proposition 4. *Given a Conjunctive Regular Path Query view $\mathbf{V}$ and a Regular Path Query Q, it is undecidable whether $\mathbf{V}$ determines Q.*

Proof. We prove this by reduction from the word problem for graph databases.

Problem :	Word problem for graph databases
Input :	A list of pairs $(u_i, v_i)_{0<i\leq n}$, a pair (u, v), where u, v, u_i, v_i, for every i are words over σ, viewed as RPQs
Question :	Is the following statement true? For every graph database D over σ, if $\forall i, u_i(\mathrm{D}) = v_i(\mathrm{D})$, then $u(\mathrm{D}) = v(\mathrm{D})$

A straightforward reduction from the word problem for finite semigroups shows:

Lemma 5. *The word problem for graph databases is undecidable.*

Proof. We prove this by reduction from the word problem for finite semigroups. This problem has the same input as the word problem for graph databases but asks whether for all semigroup S and all homomorphism h from σ^* to S such that $h(u_i) = h(v_i)$ for all i, it is the case that $h(u) = h(v)$.

We now prove that any input is accepting for the word problem for finite semigroups if and only if it is accepting for the word problem for graph databases.

1. Assume that the input is accepting for the word problem for finite semigroups. Let D be a graph database such that for all i, $u_i(\mathrm{D}) = v_i(\mathrm{D})$. From D, we compute the semigroup S_D and the homomorphism $h : \sigma^* \to S_\mathrm{D}$ as follows:

   - The elements of S_D are the set of pairs $w(\mathrm{D})$ for all $w \in \sigma^*$. As D is finite S_D is finite.
   - Let x and y be two elements of S_D. Let $u, v \in \sigma^*$ such that $x = u(\mathrm{D})$ and $y = v(\mathrm{D})$. Then $x \cdot y$ is defined as $u \cdot v(\mathrm{D})$. It is easy to check that this operation is associative and well defined (i.e. does not depend on the specific choice of u and v).
   - For all $\alpha \in \sigma$ we set $h(\alpha) = \alpha(\mathrm{D})$. Hence for all $u \in \sigma^*$ we have $h(u) = u(D)$.

   By construction we therefore have for all i, $h(u_i) = h(v_i)$. Hence, $h(u) = h(v)$, which implies that $u(\mathrm{D}) = v(\mathrm{D})$.

2. Assume that the input is accepting for the word problem for graph databases. Let S be a finite semigroup, and h an homomorphism from σ^* to S, such that, for all i, $h(u_i) = h(v_i)$. From S and h, we define the graph database D_h as follows:

   - The sets of nodes of D_h is $h(\sigma^+) \cup \{\varepsilon\}$. This set is finite since $h(\sigma^+)$ is a subset of S.
   - Let x and y be two nodes of D_h. Then there is an edge α from x to y if either $x = \varepsilon$ and $y = h(\alpha)$ or $x \neq \varepsilon$ and $x \cdot h(\alpha) = y$.

   Assume that $(x, y) \in u_i(\mathrm{D}_h)$. Then either $x = \varepsilon$, hence $y = h(u_i) = h(v_i)$ and $(x, y) \in v_i(\mathrm{D}_h)$, or $x \cdot h(u_i) = y$, which implies that $x \cdot h(v_i) = y$ and $(x, y) \in v_i(\mathrm{D}_h)$. Hence, $u_i(\mathrm{D}_h) = v_i(\mathrm{D}_h)$ for all i and therefore $u(\mathrm{D}_h) = v(\mathrm{D}_h)$. Hence, $(\varepsilon, h(u)) \in v(\mathrm{D}_h)$, which implies that there is a path v from ε to $h(u)$ and thus that $h(u) = h(v)$.

□

Let $(u_i, v_i)_{0<i\leq n}$ and (u, v) be an input for the word problem. Let σ' be a copy of σ using only fresh symbols. For each $\alpha \in \sigma$, we use α' to denote the corresponding symbol in σ'. We define the following query and view:

- Q is the RPQ defined by $L(\mathrm{Q}) = \{u, v'\}$ where v' is a copy of v using symbols of σ'.
- For all $\alpha \in \sigma$, V_α is a query of the view defined by the RPQ $L_\alpha = \{\alpha, \alpha'\}$.
- For all i, V_i is also a query of the view defined by the RPQ $L_i = \{u_i, v_i'\}$, where v_i' is a copy of v_i using symbols of σ'.
- For all $\alpha, \beta \in \sigma$, $T_{\alpha,\beta}$ is a query of the view defined by the CRPQ: $\alpha(x, y) \wedge \exists z, t\ \beta'(z, t)$.
- For all $\alpha, \beta \in \sigma$, $T'_{\alpha,\beta}$ is a query of the view defined by the CRPQ: $\alpha'(x, y) \wedge \exists z, t\ \beta(z, t)$.

We now prove that $\mathbf{V} = \{V_\alpha, V_i, T_{\alpha,\beta}, T'_{\alpha,\beta} \mid \alpha, \beta \in \sigma, 0 < i \leq n\}$ determines Q if and only if the input is accepting for the word problem for graph databases.

1. Assume that the input is accepting for the word problem for graph databases. Let D and D′ be two graph databases such that $\mathbf{V}(\mathrm{D}) = \mathbf{V}(\mathrm{D}')$. Consider first the case where D uses symbols from both σ and σ', then $T_{\alpha,\beta}$ and $T'_{\alpha,\beta}$ reveal D entirely, which implies that $\mathrm{D} = \mathrm{D}'$, and thus $\mathrm{Q}(\mathrm{D}) = \mathrm{Q}(\mathrm{D}')$. Similarly, if both D and D′ use only symbols from σ, then V_α reveals D entirely ensuring that $\mathrm{D} = \mathrm{D}'$. It remains to consider the case where D only uses symbols from σ and D′ only uses symbols from σ'. Notice that, since $V_\alpha(\mathrm{D}) = V_\alpha(\mathrm{D}')$, then D and D′ are isomorphic (by renaming each α to α').

   Let $(x, y) \in u_i(\mathrm{D})$. Hence, $(x, y) \in V_i(\mathrm{D})$, which implies that $(x, y) \in V_i(\mathrm{D}')$, and finally that $(x, y) \in v'_i(\mathrm{D}')$. By isomorphism $(x, y) \in v_i(\mathrm{D})$. Similarly, we can show that $(x, y) \in v_i(\mathrm{D})$ implies $(x, y) \in u_i(\mathrm{D})$. Hence, $u(\mathrm{D}) = v(\mathrm{D})$. Let $(x, y) \in \mathrm{Q}(\mathrm{D})$. Then, $(x, y) \in u(\mathrm{D})$, which implies that $(x, y) \in v'(\mathrm{D}')$, and thus that $(x, y) \in \mathrm{Q}(\mathrm{D}')$. A similar reasoning also gives the converse, and we can conclude that $\mathbf{V}$ determines Q.

2. Assume that $\mathbf{V}$ determines Q. Let D be a graph database over σ that satisfies the condition for the word problem. Let D′ be the copy of D given by renaming the symbols in σ by the corresponding symbols in σ'. Remark now that $\mathbf{V}(\mathrm{D}) = \mathbf{V}(\mathrm{D}')$. Indeed, $V_\alpha(\mathrm{D}) = V_\alpha(\mathrm{D}')$ is given by the fact that D′ is a copy of D over σ'. $V_i(\mathrm{D}) = V_i(\mathrm{D}')$ is given by the fact that D satisfies the condition for the word problem. Finally, $T_{\alpha,\beta}(\mathrm{D}) = T_{\alpha,\beta}(\mathrm{D}') = T'_{\alpha,\beta}(\mathrm{D}) = T'_{\alpha,\beta}(\mathrm{D}') = \emptyset$ comes from the fact that D (resp. D′) uses only symbols from σ (resp. σ').

   Since $\mathbf{V}$ determines Q, this implies that $\mathrm{Q}(\mathrm{D}) = \mathrm{Q}(\mathrm{D}')$. Let $(x, y) \in u(\mathrm{D})$. Then $(x, y) \in \mathrm{Q}(\mathrm{D})$, which implies that $(x, y) \in \mathrm{Q}(\mathrm{D}')$. Hence, $(x, y) \in v'(\mathrm{D}')$, and since D′ is a copy of D, this yields $(x, y) \in v(\mathrm{D})$. A similar reasoning also gives the converse, and we can conclude that the input is accepting for the word problem for graph databases.

□

4 Views and Rewriting

We have seen in the previous section that knowing whether a given view $\mathbf{V}$ determines a given query Q is often computationally a difficult task. In this section we assume that $\mathbf{V}$ determines Q and we investigate how Q can be computed from the given view instance.

A possibility is to use the following generic algorithm :

Given a τ-structure E, compute a σ-structure D such that $\boldsymbol{V}(D) = E$ (reject if no such D exists) and return $Q(D)$.

As we know that $\mathbf{V}$ determines Q this procedure always returns the correct answers on view images. Therefore the query over τ defined by this algorithm is a rewriting of Q using $\mathbf{V}$.

For all the query languages considered in this paper, computing $\mathbf{V}(\mathrm{D})$ and $\mathrm{Q}(\mathrm{D})$ can be done in time polynomial in $|\mathrm{D}|$. Hence it remains to be able to test whether there exists a D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$ and, if yes, compute such a D.

The first issue, testing whether a τ-instance is in the image of the view, is already a challenging task and will be investigated in the next section. We start with the second problem, i.e. computing a D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$, if it exists.

4.1 Looking for a view preimage

We assume in this section that $\mathbf{V}$ is a view from σ to τ and that we are given a τ-structure E that is in the image of $\mathbf{V}$. We are now looking for a D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$, knowing that one such D exists. Our first result below shows that for RPQ views, if such a D exists then there is one whose size is polynomial in $|\mathrm{E}|$. It is essentially a pumping argument.

Lemma 6. *Let $\boldsymbol{V}$ be an RPQ view from σ to τ. Let E be a τ-structure. If $E = \boldsymbol{V}(D)$ for some D then $E = \boldsymbol{V}(D')$, for some D' of size quadratic in $|E|$.*

Proof. Let $\mathbf{V}$ and E be as in the statement of the lemma. We show that if there exists D such that $\mathrm{E} = \mathbf{V}(\mathrm{D})$ then there exists a new database D' of size $O(|\mathrm{E}|^2)$ such that $\mathbf{V}(\mathrm{D}') = \mathbf{V}(\mathrm{D})$. D' is obtained from D in several steps. First D is "normalized", without altering its view, so that nodes not occurring in E appear in only one path linking two nodes of E. The normalized D turns out to consist of a constant number of disjoint paths between each pair of nodes of E (where the constant only depends on the size of the view automaton). Then a Ramsey argument is used to show that these paths can be "cut" without changing the view. The resulting database D' thus consists of a constant number of paths of constant length between each pair of nodes of E. The size of D' is therefore $O(|\mathrm{E}|^2)$. We now formalize this argument.

Assume that there exists a database D such that $\mathrm{E} = \mathbf{V}(\mathrm{D})$. We prove the lemma by constructing a new database D' such that $\mathbf{V}(\mathrm{D}') = \mathbf{V}(\mathrm{D})$, with $|\mathrm{D}'| = O(|\mathrm{E}|^2)$.

Let $A = \langle S_{\mathbf{V}}, \delta_{\mathbf{V}}, q^0_{\mathbf{V}}, F_{\mathbf{V}} \rangle$ be the product automaton of all the deterministic minimal automata of all the regular expressions of the RPQs in $\mathbf{V}$. Let $N(\mathbf{V})$ be the number of states of A, i.e $|S_{\mathbf{V}}|$.

In what follows, for $w \in \sigma^*$, $\delta_{\mathbf{V}}(\cdot, w)$ denotes the function from $S_{\mathbf{V}}$ to $S_{\mathbf{V}}$ sending q to p such that there is a run of A on w starting in state q and arriving in state p.

We say that a path π from u to v in a database D' is $\mathbf{V}$-minimal if u, v are elements of $\mathbf{V}(\mathrm{D}')$ and no other nodes of π are in the domain of $\mathbf{V}(\mathrm{D}')$.

We first build a database D_1 such that :

- $\mathbf{V}(\mathrm{D}_1) = \mathbf{V}(\mathrm{D})$;
- each node of D_1 is in a $\mathbf{V}$-minimal path and no two $\mathbf{V}$-minimal paths in D_1 intersect;
- the number of $\mathbf{V}$-minimal paths in D_1 is bounded by $|\mathbf{V}(\mathrm{D})|^2 \cdot N(\mathbf{V})^{N(\mathbf{V})}$.

D_1 is constructed as follows: All elements of $\mathbf{V}(\mathrm{D})$ are elements of D_1. Moreover, for each function $f : S_{\mathbf{V}} \to S_{\mathbf{V}}$ and each pair (x, y) of elements of $\mathbf{V}(\mathrm{D})$, if there exists a $\mathbf{V}$-minimal path π from x to y in D and such that $f = \delta_{\mathbf{V}}(\cdot, \lambda(\pi))$, then we add to D_1 a copy of π that uses only fresh, non-repeating nodes, except for x and y. Figure 2 illustrates the main idea of this construction.

It is now easy to check that D_1 has the desired properties. The second bullet holds by construction. Clearly the number of $f : S_{\mathbf{V}} \to S_{\mathbf{V}}$ is bounded by $N(\mathbf{V})^{N(\mathbf{V})}$ hence the third bullet holds. It remains to check that $\mathbf{V}(\mathrm{D}_1) = \mathbf{V}(\mathrm{D})$. There is an obvious canonical homomorphism sending D_1 to D. Hence $\mathbf{V}(\mathrm{D}_1) \subseteq \mathbf{V}(\mathrm{D})$. For the converse direction, consider a path π witnessing the fact that $(u, v) \in \mathbf{V}(\mathrm{D})$. Decompose π into $\mathbf{V}$-minimal paths. By construction, each of these $\mathbf{V}$-minimal paths can be simulated in D_1. Hence $(u, v) \in \mathbf{V}(\mathrm{D}_1)$.

From D_1 we construct the desired D' by replacing each $\mathbf{V}$-minimal path of D_1 by another one whose length is bounded by a constant r and without affecting the view image. Altogether D' will have a size bounded by $r \cdot |\mathbf{V}(\mathrm{D})|^2 \cdot N(\mathbf{V})^{N(\mathbf{V})}$, hence polynomial in $|\mathbf{V}(\mathrm{D})|$ as desired.

Let r be the Ramsey's number that guarantees the existence of a monochromatic 3-clique in an r-clique using $N(\mathbf{V})^{N(\mathbf{V})} \cdot 2^{N(V)^{N(V)}}$ colors.

Consider a $\mathbf{V}$-minimal path $\pi = x a_0 x_1 a_1 \dots x_m a_m y$ in D_1 such that $m > r$. For $1 \leq s < t \leq m$ we denote by $\pi_{s \to t}$ the subpath of π that starts at position s and ends at position t, that is $\pi_{s \to t} = x_s a_s x_{s+1} a_{s+1} \dots a_{t-1} x_t$.

To each pair of nodes (x_i, x_j) in π with $i < j$, we attribute the color (f_{ij}, Δ_{ij}) where:

$$
\begin{aligned}
f_{ij} &= \delta_{\mathbf{V}}(\cdot, \lambda(\pi_{i \to j})) \\
\Delta_{ij} &= \{ f : S_{\mathbf{V}} \to S_{\mathbf{V}} \mid \exists \alpha, \quad && i < \alpha < j \text{ and} \\
& && f = \delta_{\mathbf{V}}(\cdot, \lambda(\pi_{i \to \alpha})) \}.
\end{aligned}
$$

Then, by our choice of r, we know that there exist $i < j < k$ such that $f_{ij} = f_{jk} = f_{ik}$ and $\Delta_{ij} = \Delta_{jk} = \Delta_{ik}$. Let π' be the path constructed from π by replacing the subpath $\pi_{i \to k}$ by $\pi_{j \to k}$.

Let D_2 be the database constructed from D_1 by replacing π by π'. We now prove that $\mathbf{V}(\mathrm{D}_2) = \mathbf{V}(\mathrm{D}_1)$. As D_2 still has all the properties of D_1 listed above, by repeating this operation until all $\mathbf{V}$-minimal paths have length less than r we eventually get the desired database D'.

Let $(u, v) \in \mathbf{V}(\mathrm{D}_1)$ as witnessed by a path μ in D_1. Then μ neither starts nor ends in an internal node of π as internal nodes do not appear in $\mathbf{V}(\mathrm{D}_1)$. Hence either μ does not use π or it uses all of it. In the former case, μ witnesses the fact that $(u, v) \in \mathbf{V}(\mathrm{D}_2)$. In the latter, notice that $f_{ik} = f_{jk}$ implies that $\lambda_{\mathbf{V}}(\cdot, \lambda(\pi)) = \lambda_{\mathbf{V}}(\cdot, \lambda(\pi'))$, hence replacing π by π' in μ witnesses the fact that $(u, v) \in \mathbf{V}(\mathrm{D}_2)$. Altogether we have shown that $\mathbf{V}(\mathrm{D}_1) \subseteq \mathbf{V}(\mathrm{D}_2)$.

Suppose now that $(u, v) \in \mathbf{V}(\mathrm{D}_2)$ as witnessed by a path μ in D_2. If μ does not go through x_j (i.e. x_j is not an internal node of μ), it is also a path in D_1 and $(u, v) \in \mathbf{V}(\mathrm{D}_1)$. If μ goes through x_j but does not end between x_j and x_k we can also conclude that $(u, v) \in \mathbf{V}(\mathrm{D}_1)$ using the fact that $f_{ik} = f_{jk}$. It remains to consider the case when μ ends with $x_j a_j \dots a_{\beta-1} x_\beta$ for some β with $j < \beta < k$ (in particular $v = x_\beta$). As $\Delta_{ij} = \Delta_{jk}$ there exists α with $i < \alpha < j$ such that $\delta_{\mathbf{V}}(\cdot, \lambda(\pi_{i\to\alpha})) = \delta_{\mathbf{V}}(\cdot, \lambda(\pi_{j\to\beta}))$. From this we can construct a path μ' in D_1 replacing in μ the segment $x_j a_j \dots a_{\beta-1} x_\beta$ by $x_i a_i \dots a_{\alpha-1} x_\alpha$, witnessing the fact that $(u, x_\alpha) \in \mathbf{V}(\mathrm{D}_1)$, a contradiction as x_α is not an element of $\mathbf{V}(\mathrm{D}_1)$. Altogether we have proved that $\mathbf{V}(\mathrm{D}_2) \subseteq \mathbf{V}(\mathrm{D}_1)$. Hence, $\mathbf{V}(\mathrm{D}_2) = \mathbf{V}(\mathrm{D}_1) = \mathbf{V}(\mathrm{D})$.

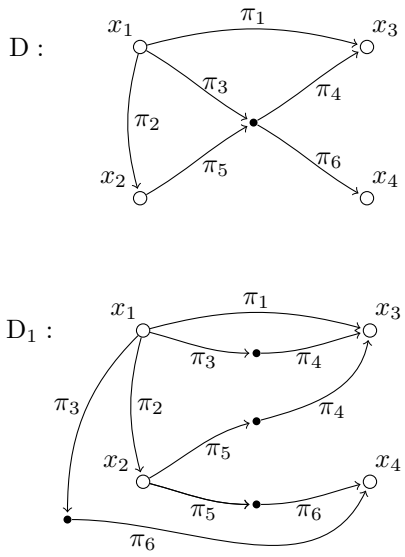


Fig. 2: Illustration of the transformation from D to D_1 in Lemma 6. Nodes are colored white or black depending on whether they appear in $\mathbf{V}(\mathrm{D})$ or not.

□

In view of Lemma 6, we know that if $\mathbf{V}$ determines Q then there exists a rewriting R with NP data complexity. Indeed R is the query computed by the following non-deterministic polynomial time algorithm: on an input τ-structure E, guess from E a database D of polynomial size, check that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$ and then evaluate Q on D. There also exists a rewriting with coNP data complexity, by considering all databases D of polynomial size such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$. Altogether we get:

Corollary 7. *Let $\mathbf{V}$ and Q be RPQs such that $\mathbf{V}$ determines Q. Then there exists a rewriting of Q using $\mathbf{V}$ with* NP *data complexity, and another one with* coNP *data complexity.*

It is not known whether, for RPQ views and queries, determinacy implies the existence of a rewriting with polynomial time data complexity. The complexity bounds of Corollary 7 are the current best known bounds. We will see in the next sections that if we further assume that the function induced by Q using $\mathbf{V}$ is *monotone* then there exists a rewriting of Q using $\mathbf{V}$ definable in Datalog and therefore computable in polynomial time.

Using a more intricate pumping argument it is possible to show that for any Conjunctive Regular Path Query view $\mathbf{V}$, the fact that a view instance is in the image of $\mathbf{V}$ can also be witnessed by a database of polynomial size. Hence Corollary 7 extends to Conjunctive Regular Path Queries.

However we will see that for Context-Free Path Query views there is no recursive bound on the size of a database yielding a given view instance. This will follow from Lemma 10 showing that, for Context-Free Path Query views, checking whether a view instance is in the image of the view is undecidable.

4.2 Testing for view images

We now consider the following problem. We are given a view $\mathbf{V}$ from σ to τ and a τ-structure E and we are asking whether there exists a σ-structure D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$.

Note that this problem is related to the previous one. In view of Lemma 6 we immediately get an NP algorithm for testing membership in the image of an RPQ view $\mathbf{V}$: on input E guess a database D of size polynomial in E and check $\mathbf{V}(\mathrm{D}) = \mathrm{E}$. We will see that testing for view images is NP-hard for Regular Path Query views and undecidable for Context-Free Path Query views.

Moreover one can show that if testing for view images can be done in PTime then, for Q and $\mathbf{V}$ such that $\mathbf{V}$ determines Q, then there exists a rewriting of Q using $\mathbf{V}$ with polynomial time data complexity. The polynomial time algorithm works as follows. On a view instance E, it first tests whether there exists a database D such that $\mathrm{E} = \mathbf{V}(\mathrm{D})$. If not it rejects. If yes, consider the schema adding two new letters a and b and consider the query $Q_{a,b}$ asking for a path in the language $a \cdot L(Q) \cdot b$. Define $\mathbf{V}'$ as $\mathbf{V} \cup \{\mathrm{Q}_{a,b}, V_a, V_b\}$ where V_a and V_b return all pairs of nodes linked by a and b respectively. For each pair (x, y) of nodes of E, let E' be E expanded with the empty relation for $\mathrm{Q}_{a,b}$, a single pair (u, x) for V_a and a single pair (y, v) for V_b where u and v are two new nodes. We then test whether E' is a view image. A simple argument shows that the test says yes iff $(x, y) \notin \mathrm{Q}(\mathrm{D})$ and the algorithm works in time polynomial in the size of E.

Unfortunately, as already mentioned, the test for view images is NP-hard already for RPQ views.

Lemma 8. *There is an RPQ view $\mathbf{V}$ from σ to τ such that given a τ-structure E it is* NP*-hard to test whether there exists a σ-structure D such that $\mathbf{V}(D) = E$.*

Proof. We reduce 3-Colorability to our problem. The proof is a simple variation of the reduction found in [5] to prove that computing certain answers under the sound view assumption is coNP-hard in data complexity.

Let $\sigma = \{rg, gr, bg, gb, rb, br\}$ and $\tau = \{V_1, V_2\}$. By abuse of notation, we will refer to an element of σ as $\alpha\beta$, with α and β two symbols in $\{r, g, b\}$, and $\alpha \neq \beta$. Let $\mathbf{V}$ be the following view from σ to τ:

- $\mathbf{V} = \{V_1, V_2\}$
- $L(V_1) = \{rg, gr, bg, gb, rb, br\}$
- $L(V_2) = \{\alpha_1\beta_1 \cdot \alpha_2\beta_2 \mid \beta_1 \neq \alpha_2\}$.

Let $G = (U, W)$ be a connected graph. From G we define a τ-structure E_G, in which the interpretation of V_1 is:

$$\{(x, y) \mid (x, y) \in W \text{ or } (y, x) \in W\}$$

and the interpretation of V_2 is the empty relation.

We show that G is 3-colorable iff there exists D such that $\mathbf{V}(\mathrm{D}) = E_G$.

Intuitively, the idea is that σ describes the colors of the edges of G, that is the color of the two end points of each edge. For instance, if x and y are linked by rg, then it should be understood that x is red and y is green. V_1 checks that each pair of nodes that are connected in G are colored with (at least) two different colors, and V_2 checks if there is any error, that is, if a node is required

to have more than one color. Since V_2 is empty, any graph database D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$ cannot have any such error, and would thus be 3-colorable.

More precisely, assume that G is 3-colorable. Then there exists a coloring function $c : U \to \{r, g, b\}$ such that $c(x) \neq c(y)$ for all $(x, y) \in W$. We define D as the σ-structure such that, for each $\alpha\beta \in \sigma$, the interpretation of $\alpha\beta$ in D is:

$$\{(x, y) \mid (x, y) \in W \text{ or } (y, x) \in W, \\ \text{and } c(x) = \alpha, c(y) = \beta\}.$$

It is then easy to check that $\mathbf{V}(\mathrm{D}) = \mathrm{E}_G$. Indeed, for all $x, y, z \in \mathrm{D}$, if $\alpha_1\beta_1(x, y)$ and $\alpha_2\beta_2(y, z)$ hold in D, then $\beta_1 = c(y) = \alpha_2$, hence $(x, z) \notin V_2(\mathrm{D})$, so $V_2(\mathrm{D})$ is empty.

Conversely, assume that there exists a graph database D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}_G$. Consider the coloring function $c : U \to \{r, g, b\}$ defined as: $c(x) = \alpha$ if there exists y such that $\alpha\beta(x, y)$ holds in D. Since $V_2(\mathrm{D})$ is empty, it is immediate to check that $c(x)$ is uniquely defined and that c is a proper 3-coloring of G. □

If we go from regular languages to context-free ones, then the problem becomes undecidable.

Lemma 9. *Let $\boldsymbol{V}$ be a Context-Free Path Query view from σ to τ. Let E be a τ-instance. Then it is undecidable whether there exists a σ-structure D such that $\boldsymbol{V}(D) = E$.*

Proof. We prove this by reduction from the universality problem for context-free languages. Let L be a context-free language over some alphabet σ. Let \$ be a fresh symbol that does not appear in σ. Let $\mathbf{V} = \{V_1, V_2\}$, where V_1 is defined by $L(V_1) = \$ \cdot L \cdot \$$ and V_2 is defined by $L(V_2) = \$ \cdot \sigma^* \cdot \$$. Finally, let E be the view instance that contains a single pair (x, y) in V_2 and no pair in V_1. Then there exists D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$ if and only if L is not universal over σ.

- Assume that there exists a database D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$. Then there exists a path π from x to y such that $\lambda(\pi) \in L(V_2)$. Hence there exists $w \in \sigma^*$ such that $\lambda(\pi) = \$ \cdot w \cdot \$$. However, $\lambda(\pi) \notin L(V_1)$. Hence $w \notin L$ and L is not universal.
- Conversely, assume that L is not universal. Then there exists $w \in \sigma^*$ such that $w \notin L$. Then it is easy to check that the database D consisting of a simple path labeled by $\$ \cdot w \cdot \$$ satisfies $\mathbf{V}(\mathrm{D}) = \mathrm{E}$.

□

A more intricate argument shows that undecidability already holds for a *fixed* view definition $\mathbf{V}$.

Lemma 10. *There exists a fixed Context-Free Path Query view $\boldsymbol{V}$ from σ to τ such that, given a τ-structure E, it is undecidable whether there exists a σ-structure D such that $\boldsymbol{V}(D) = E$.*

Proof. Let $\sigma = \{(, ;,), a, b, \$, 1\}$. Let $\boldsymbol{\sigma}$ be a copy of σ with fresh symbols. For $\alpha \in \sigma$, we denote by $\boldsymbol{\alpha}$ the corresponding symbol in $\boldsymbol{\sigma}$. For w a word, $\tilde{w}$ denote the word corresponding to w read from right to left. $\mathbf{V}$ consists of views that reveal each symbol in σ, that is, for all $\alpha \in \sigma$, $\mathbf{V}$ contains a view V_α defined by $L(V_\alpha) = \{\alpha\}$. Additionally, $\mathbf{V}$ contains the queries V_u, V_v, V'_u, V'_v, V_g and V_c defined by the following equations:

$$L(V_u) = \left\{ \begin{array}{c} \$ \cdot w \cdot \$ \cdot (i_1;v_1;u_1) \dots (i_n;v_n;u_n) \cdot \$ \mid \\ w, u_k, v_k \in \{\boldsymbol{a}, \boldsymbol{b}\}^*, i_k \in \mathbf{1}^*, u_1 \cdot \ldots \cdot u_n = \tilde{w} \end{array} \right\}$$

$$L(V_v) = \left\{ \begin{array}{c} \$ \cdot w \cdot \$ \cdot (i_1;v_1;u_1) \dots (i_n;v_n;u_n) \cdot \$ \mid \\ w, u_k, v_k \in \{\boldsymbol{a}, \boldsymbol{b}\}^*, i_k \in \mathbf{1}^*, v_1 \cdot \ldots \cdot v_n = \tilde{w} \end{array} \right\}$$

$$L(V'_u) = \left\{ \begin{array}{c} \$ \cdot w \cdot \$ \cdot (i_1;v_1;u_1) \dots (i_n;v_n;u_n) \cdot \$ \mid \\ w, u_k, v_k \in \{\boldsymbol{a}, \boldsymbol{b}\}^*, i_k \in \mathbf{1}^*, u_1 \cdot \ldots \cdot u_n \neq \tilde{w} \end{array} \right\}$$

$$L(V'_u) = \left\{ \begin{array}{c} \$ \cdot w \cdot \$ \cdot (i_1;v_1;u_1) \dots (i_n;v_n;u_n) \cdot \$ \mid \\ w, u_k, v_k \in \{\boldsymbol{a}, \boldsymbol{b}\}^*, i_k \in \mathbf{1}^*, v_1 \cdot \ldots \cdot v_n \neq \tilde{w} \end{array} \right\}$$

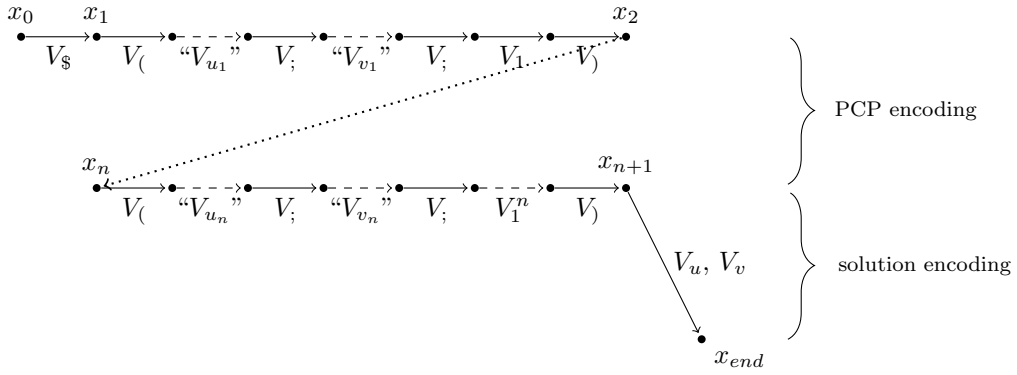
$$L(V_g) = \left\{ \begin{array}{c} \$ \cdot (u_1;v_1;i_1) \cdot \ldots \cdot (u_n;v_n;i_n) \cdot \$ \cdot \boldsymbol{\sigma}^* \cdot \$ \cdot \boldsymbol{\sigma}^* \cdot (\boldsymbol{i'};\boldsymbol{v'};\boldsymbol{u'}) \mid \\ u_k, v_k \in \{a,b\}^*, i_k \in 1^*, u', v' \in \{\boldsymbol{a},\boldsymbol{b}\}^*, i' \in \boldsymbol{1}^*, i' > i_n \end{array} \right\}$$

$$L(V_c) = \left\{ \begin{array}{c} \$ \cdot (u_1;v_1;i_1) \cdot \ldots \cdot (u_n;v_n;i_n) \cdot \$ \cdot \boldsymbol{\sigma}^* \cdot \$ \cdot \boldsymbol{\sigma}^* \cdot (\boldsymbol{i'};\boldsymbol{v'};\boldsymbol{u'}) \mid \\ u_k, v_k \in \{a,b\}^*, i_k \in 1^*, u', v' \in \{\boldsymbol{a},\boldsymbol{b}\}^*, i' \in \boldsymbol{1}^*, \\ \exists k, i_k = \varphi(i'), u_k \neq \varphi(\tilde{u'}) \text{ or } v_k \neq \varphi(\tilde{v'}) \end{array} \right\}$$

where φ is the function that maps each symbol in $\boldsymbol{\sigma}$ to the corresponding symbol in σ.

One can check that all these languages are actually context-free languages.

We now prove that, given a view instance E for this specific view $\mathbf{V}$, it is undecidable whether there exists a database D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$. We prove this by reduction from the Post Correspondence Problem (PCP). Let $(u_i, v_i, i)_{0<i\leq n}$ be an instance of PCP over $\{a, b\}$, where the third argument explicitly gives the index of each pair. We build the following view instance E:



We now show that there exists D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$ if and only if the PCP instance is satisfiable. Intuitively, E consists of two parts. The first part, from x_0 to x_{n+1} is the encoding of the PCP instance. It uses letters from σ that are all revealed by the view. All tuples are simply enumerated in the natural order, where the ith tuple is encoded between x_i and x_{i+1}. The dashed arrows V_{u_i} and V_{v_i} represent the correct succession of V_a and V_b that naturally encode u_i and v_i, whereas the V_1^i part is the unary encoding of i, the index of the tuple. The second part of the instance states the existence of a solution for this instance, and uses "hidden" letters from $\boldsymbol{\sigma}$. V_u and V_v states that there exists a solution, and the fact that all other views are empty checks that this solution is correct.

- Assume that there exists a database D such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$. Then there exists a path π from x_{n+1} to x_{end} such that $\lambda(\pi) \in L(V_u)$. Hence, this path is of the form $\$ \cdot w \cdot \$ \cdot (i_1;v'_1;u'_1) \ldots (i_m;v'_m;u'_m) \cdot \$$, where w is a word in $\boldsymbol{\sigma}^*$ and $u'_1 \ldots u'_m = \tilde{w}$. Remark that is also holds that $v'_1 \ldots v'_m = \tilde{w}$, otherwise $\lambda(\pi) \in V'_v$, which would imply that $(x_{n+1}, x_{end}) \in V'_v(\mathrm{D})$, and lead to a contradiction.

  Hence, $u'_1 \ldots u'_m = v'_1 \ldots v'_m$. It remains to show that each $(i_i;v'_i;u'_i)$ is an encoding of the mirror of some tuple in the PCP instance, which would imply a solution as $\tilde{u}'_m \ldots \tilde{u}'_1 = \tilde{v}'_m \ldots \tilde{v}'_1$. In other words, $u_{|i_m|} \ldots u_{|i_1|} = v_{|i_m|} \ldots v_{|i_1|}$.

  Assume that one of the $(i_i;v'_i;u'_i)$ is not the mirror of some tuple encoded in the first half of the instance. Remark that $|i_i| \leq n$. Otherwise, there exists a path whose label is in $L(V_g)$, which leads to a contradiction. Hence, either $u'_i \neq \tilde{u}_{|i_i|}$ or $v'_i \neq \tilde{v}_{|i_i|}$. Both cases lead to the existence of a path whose label is in $L(V_c)$, and thus to a contradiction.

- Assume that there exists a solution $i_1 \ldots i_m$ to the PCP instance. Then the database D that consists of the following simple path is such that $\mathbf{V}(\mathrm{D}) = \mathrm{E}$:

  $$\$(u_1;v_1;1) \ldots (u_n;v_n;1^n)\mathbf{\$u_{i_1}} \ldots \mathbf{u_{i_m}\$(1}^{i_m}\mathbf{;\tilde{v}_{i_m};\tilde{u}_{i_m})} \ldots \mathbf{(1}^{i_1}\mathbf{;\tilde{v}_{i_1};\tilde{u}_{i_1})\$}$$

  where $\mathbf{u_i}$ and $\mathbf{v_i}$ simply represent the corresponding u_i and v_i written using $\boldsymbol{a}$ and $\boldsymbol{b}$ instead of a and b.

□

Note that in the proof of Lemma 10 the view instance is a coding of a PCP instance and the corresponding database a coding of a solution. As there is no recursive bound on the size of a solution of a PCP instance, for Context-Free Path Query views, there are no recursive bound on the size of a database that yields a given view instance. This is to be compared with the polynomial bound for RPQ views shown in Lemma 6.

5 Monotone determinacy and rewriting

As Example 2 shows, there is an RPQ view $\mathbf{V}$ and an RPQ query Q such that $\mathbf{V}$ determines Q but the function induced by Q using $\mathbf{V}$ is not monotone, therefore having no RPQ rewriting. It is natural to wonder whether the monotonicity of the function induced by the query is the only limit for the existence of an RPQ rewriting. Recall from the introduction that if $\mathbf{V}$ and Q are defined using CQs and $\mathbf{V}$ determines Q, then the function induced by Q using $\mathbf{V}$ is monotone iff there exists a CQ rewriting. In the case of RPQ views and queries the analog does not hold. We will see that, even if we assume monotonicity, an RPQ rewriting need not exist; however in the next section we will show that a rewriting definable in Datalog always exists. We start by formalizing the notion of monotone determinacy.

Definition 11 (Monotone determinacy)**.** *We say that a view $\boldsymbol{V}$ determines a query Q in a monotone way if $\boldsymbol{V}$ determines Q and the function induced by Q using $\boldsymbol{V}$ is monotone.*

It is rather immediate to see that monotone determinacy is equivalent to the following property for $\mathbf{V}$ and Q:

$$\forall \mathrm{D}, \mathrm{D}', \quad \mathbf{V}(\mathrm{D}) \subseteq \mathbf{V}(\mathrm{D}') \;\Rightarrow\; \mathrm{Q}(\mathrm{D}) \subseteq \mathrm{Q}(\mathrm{D}')$$

This turns out to coincide with the notion of *losslessness under the sound view assumption* defined in [7], that was shown to be decidable, actually ExpSpace-complete, for RPQs.

Corollary 12. *The monotone determinacy problem for RPQs is* ExpSpace*-complete.*

Note that in the proof of Proposition 3, the rewriting is always monotone when the view determines the query. Therefore, for Context-Free Path Query views and RPQ queries, monotone determinacy is undecidable.

Recall from Example 2 that there exist a view and a query such that the view determines the query but not in a monotone way. We now assume given an RPQ view $\mathbf{V}$ and an RPQ query Q such that $\mathbf{V}$ determines Q in a monotone way. It was observed in [7] that even in this case there might be no rewriting definable in RPQ.

In fact, given $\mathbf{V}$ and Q defined using RPQs, it is decidable whether an RPQ rewriting exists and the problem is 2ExpSpace-complete [8]. As testing monotone determinacy is ExpSpace-complete, a simple complexity argument shows that an RPQ rewriting is not guaranteed to exist under monotone determinacy.

Here is a concrete example witnessing this fact.[1]

Example 13. *Let $\sigma = \{a, b, c\}$. Let Q and $\boldsymbol{V}$ be defined as follows:*

- $Q = ab^*a \mid ac^*a$
- $\boldsymbol{V} = \{V_1, V_2, V_3\}$ *with*
  - $V_1 = ab^*$
  - $V_2 = ac^*$
  - $V_3 = b^*a \mid c^*a$

[1] A similar example was claimed in [7, Example 4] but it seems that in this example $\mathbf{V}$ and Q are such that $\mathbf{V}$ does not determine Q.

One can verify that $\boldsymbol{V}$ *determines* Q *as witnessed by the following rewriting* $R(x, y)$*:*

$$\exists z\ V_1(x, z) \wedge V_2(x, z) \wedge V_3(z, y)$$

That R *is a rewriting is illustrated in Figure 3. Consider the database* D *of Figure 3 which is a typical database such that* $(x, y) \in Q(D)$*. The choice of* z *witnessing* $(x, y) \in R(\boldsymbol{V}(D))$ *is then immediate. Conversely, consider the database* D' *of Figure 3. It is a typical database such that* $(x, y) \in R(\boldsymbol{V}(D))$*. The top path shows that* $(x, y) \in Q(D)$*.*

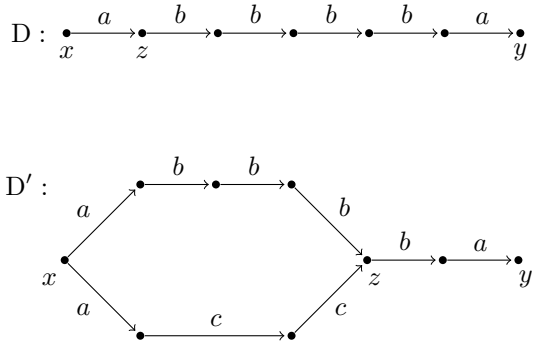


Fig. 3: Databases D and D′ for Example 13.

Since R *is monotone,* $\boldsymbol{V}$ *determines* Q *in a monotone way. It can also be shown (for instance using the decision procedure provided in [8]) that no RPQ rewriting exists.*

In the previous example we have exhibited a Conjunctive Regular Path Query rewriting. However the following example suggests that Conjunctive Regular Path Query is not expressive enough as a rewriting language.

Example 14. *Let* $\sigma = \{a\}$*. Let* $\boldsymbol{V}$ *and* Q *be defined as follows:*

- $Q = a(a^6)^* \mid aa(a^6)^*$ *(words of length 1 or 2 modulo 6)*
- $\boldsymbol{V} = \{V_1, V_2\}$ *with*
  - $V_1 = a \mid aa$ *(words of length 1 or 2)*
  - $V_2 = aa \mid aaa$ *(words of length 2 or 3)*

It can be verified that $\boldsymbol{V}$ *determines* Q *in a monotone way as witnessed by the following rewriting* $R(x, y)$*:*

$$\exists z\ V_1(x, z) \wedge T^*(z, y)$$

where $T(x, y)$ *is defined as:*

$$\exists z_1, z_2\ V_1(x, z_1) \wedge V_2(x, z_1) \wedge V_1(z_1, z_2) \wedge V_2(z_1, z_2) \wedge V_1(z_2, y) \wedge V_2(z_2, y)$$

The query T *is such that if* $T(x, y)$ *holds in* $\boldsymbol{V}(D)$*, then in* D *the nodes* x *and* y *are either linked by a path of length* 6 *or by both a path of length* 5 *and a path of length* 7*. This fact can be checked by a simple case analysis. One such case is illustrated in Figure 4. In this case there is no path of length* 6 *in* D*, but the top path has length* 5*, and the path starting with the bottom segment and then the last two top segments has length* 7*.*

From this, a simple induction shows that if $T^*(x, y)$ *holds in* $\boldsymbol{V}(D)$*, then in* D *the nodes* x *and* y *are either linked by a path of length* 0 *modulo* 6*, or by both a path of length* 1 *modulo* 6 *and a path of length* 5 *modulo* 6*.*

Assume now that $R(x, y)$ holds in $\boldsymbol{V}(D)$. Then in D there exists a z such that x is at distance 1 or 2 from z, and such that $T^*(z, y)$ holds in $\boldsymbol{V}(D)$. Assume first that z and y are at distance 0 modulo 6 in D. In this case, regardless of the distance between x and z, $Q(x, y)$ holds in D. Otherwise, in D there exist both a path of length 1 modulo 6 and a path of length 5 modulo 6 from z to y. Therefore, if x and z are at distance 1, the first path from z to y yields a path of length 2 modulo 6 and, if x and z are at distance 2, the second path from z to y yields a path of length 1 modulo 6, see Figure 5.

Conversely, it is easy to check that $R(x, y)$ holds in $\boldsymbol{V}(D)$ whenever $Q(x, y)$ holds in D. This follows from the fact that $T(x, y)$ holds in $\boldsymbol{V}(D)$ for all x and y that are at distance 6 in D.

Notice that R is monotone. A tedious combinatorial argument can show that R cannot be expressed as a Conjunctive Regular Path Query.

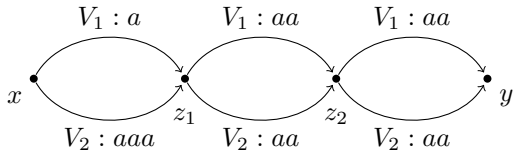


Fig. 4: Example 14: An arbitrary database D whose view satisfies $T(x, y)$. Each arrow of the form $V_i : w$ from a node u to a node v should be understood as a path from u to v whose label is w which witnesses $(u, v) \in V_i(D)$.

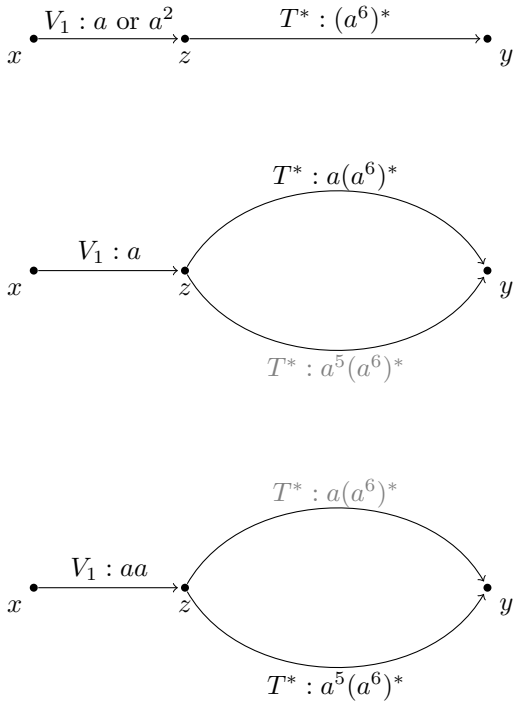


Fig. 5: The three cases of Example 14. The parts that are not used for Q are shaded out.

Remark 15. *The careful reader has probably noticed that in both examples above a rewriting can be expressed in* MSO*. As we will see later, it easily follows from the results of [6] that this is always true in general: if $\boldsymbol{V}$ and Q are defined by RPQs and $\boldsymbol{V}$ determines Q in a monotone way, then there exists a rewriting of Q using $\boldsymbol{V}$ definable in* MSO *(actually universal* MSO*).*

6 Datalog rewriting

In this section we prove our main result, namely:

Theorem 16. *If $\mathbf{V}$ and Q are RPQs and $\mathbf{V}$ determines Q in a monotone way then there exists a Datalog rewriting of Q using $\mathbf{V}$.*

Theorem 16 also implies that the monotone determinacy problem for RPQs coincides with the problem of the existence of a Datalog rewriting. The latter is therefore decidable by Corollary 12:

Corollary 17. *Let $\mathbf{V}$ and Q be RPQs. It is decidable,* ExpSpace*-complete, whether there exists a Datalog rewriting of Q using $\mathbf{V}$.*

Our proof being constructive, the Datalog rewriting can be computed from $\mathbf{V}$ and Q.

Main idea and sketch of the proof The starting point is the relationship between rewriting and certain answers under monotone determinacy. One can easily show that if the view determines the query in a monotone way then the certain answers query is a rewriting. However certain answers for RPQ views and queries are coNP-hard to compute [5]. Here we show that there exists another rewriting (which of course coincides with certain answers on view images) that is expressible in Datalog. This other rewriting is suggested by the relationship between certain answers and Constraint Satisfaction Problems (CSP). Following [6] we adopt here the homomorphism point of view for CSPs: Each CSP is defined by a structure, called *the template*, and its solutions are all the structures mapping homomorphically into the template.

Indeed [6] showed that, for RPQs $\mathbf{V}$ and Q, certain answers can be expressed as a CSP whose template depends only on $\mathbf{V}$ and Q. It is known from [11] that for every l and k with $l \leq k$, and every template, there exists a Datalog$_{l,k}$ query approximating the CSP defined by this template. Even if its Datalog$_{l,k}$ "approximation" does not compute precisely the CSP associated to $\mathbf{V}$ and Q, if it is exact *on view images*, then it is a rewriting. We show that if the view determines the query in a monotone way then there is an l and a k, depending only on $\mathbf{V}$ and Q, such that the Datalog$_{l,k}$ approximation is exact *on view images*. This proves the existence of a Datalog rewriting.

This is done in two steps. We first show that there exists a Datalog approximation which is exact on view images of simple path databases. Then we show how to lift this result on all view images. The first step is proved by a careful analysis of the properties of view images of simple path databases. The second steps exploits monotonicity.

We now provide more details.

6.1 Monotone rewritings, certain answers and CSP

Let $\mathbf{V}$ be a view from σ to τ and Q be a query on σ-structures. The certain answers of Q on a τ-structure E w.r.t. $\mathbf{V}$ are defined as

$$cert_{\mathrm{Q},\mathbf{V}}(\mathrm{E}) = \bigcap_{\mathrm{D} \mid \mathrm{E} \subseteq \mathbf{V}(\mathrm{D})} \mathrm{Q}(\mathrm{D})$$

This notion is usually referred to as certain answers under the *sound view assumption* or *open world assumption* in the literature [1, 9]. It is straightforward to check that if $\mathbf{V}$ determines Q in a monotone way, the query $cert_{\mathrm{Q},\mathbf{V}}$ is a rewriting of Q using $\mathbf{V}$, i.e. $cert_{\mathrm{Q},\mathbf{V}}(\mathbf{V}(\mathrm{D})) = \mathrm{Q}(\mathrm{D})$ for each σ-structure D.

Therefore any language known to express certain answers is a suitable rewriting language under monotone determinacy.

The following proposition, proved in [6], shows that, for RPQ views and queries, certain answers (and therefore rewritings) can be expressed as (the negation of) a CSP.

Proposition 18 ([6]). *Let $\boldsymbol{V}$ be an RPQ view from σ to τ and Q be an RPQ query over σ. There exists a τ-structure $T_{Q,\boldsymbol{V}}$ having a set of distinguished* source nodes *and a set of distinguished* target nodes *such that, if $\boldsymbol{V}$ determines Q in a monotone way, the following are equivalent, for each σ-structure D and each pair of nodes u, v of D:*

1. $(u, v) \in Q(D)$
2. $(u, v) \in cert_{Q,\boldsymbol{V}}(\boldsymbol{V}(D))$
3. *$\boldsymbol{V}(D)$ has no homomorphism to $T_{Q,\boldsymbol{V}}$ sending u to a source node and v to a target node.* [2]

In the sequel, by $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ (resp. $\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$) we refer to the set of all triplets (E, u, v) such that E is a τ-structure, u, v are nodes of E and, there is no homomorphism (resp. there is a homomorphism) from E to $T_{\mathrm{Q},\mathbf{V}}$ sending u to a source node and v to a target node[3]. In view of Proposition 18, if $\mathbf{V}$ determines Q in a monotone way, $(\mathbf{V}(\mathrm{D}), u, v) \in \neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ iff $(u, v) \in \mathrm{Q}(\mathrm{D})$.

Observe that $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ naturally defines a binary query associating with each τ-structure E the set of all pairs (u, v) of nodes of E such that $(\mathrm{E}, u, v) \in \neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$. By abuse of notation, when clear from the context, we will let $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ also denote this binary query.

Remark 19. *The structure $T_{Q,\boldsymbol{V}}$ of Proposition 18 can be effectively computed from Q and $\boldsymbol{V}$. Moreover observe that CSP($T_{Q,\boldsymbol{V}}$) can be expressed in existential* MSO*. This shows, as mentioned in Remark 15, that if $\boldsymbol{V}$ and Q are RPQs and $\boldsymbol{V}$ determines Q in monotone way, then there always exists a rewriting of Q using $\boldsymbol{V}$ definable in (universal)* MSO*; moreover this rewriting can be effectively computed from Q and $\boldsymbol{V}$.*

It is well known that the certain answers query is a rewriting that can be computed in coNP (this follows for instance from Proposition 18). Assuming coNP is not PTime, $cert_{\mathrm{Q},\mathbf{V}}$ cannot always be computed in polynomial time, not even under the assumption that $\mathbf{V}$ determines Q in a monotone way. Indeed it has been shown [5] that there exists Q and $\mathbf{V}$ defined by RPQs such that $cert_{\mathrm{Q},\mathbf{V}}$ has coNP-hard data complexity. An easy reduction from this problem shows that the lower bound remains valid if we further assume that $\mathbf{V}$ determines Q in a monotone way:

Proposition 20. *There exist an RPQ view $\boldsymbol{V}$ and an RPQ query Q such that $\boldsymbol{V}$ determines Q in a monotone way and it is* coNP*-hard to decide – given a τ-structure E and nodes (u, v) of E– whether $(u, v) \in cert_{Q,\boldsymbol{V}}(E)$.*

We show in the next section that when $\mathbf{V}$ determines Q in a monotone way there is another rewriting expressible in Datalog, hence computable in polynomial time. Before we do this we remark that the coNP complexity of $cert_{\mathrm{Q},\mathbf{V}}$ can be extended to Context-Free Path Query views and RPQ queries.

Proposition 21. *Let $\boldsymbol{V}$ be a Context-Free Path Query view and Q be a RPQ. Then $cert_{Q,\boldsymbol{V}}$ can be evaluated with* coNP *data complexity.*

Proof. Let $\mathbf{V}$ be a Context-Free Path Query view, and Q be a RPQ over some schema σ. We prove that $cert_{\mathrm{Q},\mathbf{V}}$ can be evaluated with coNP data complexity by reducing it to the case of regular path views. Let $A = \langle S, \delta, q_0, F\rangle$ be a deterministic minimal automaton for $L(\mathrm{Q})$. In what follows, $\delta(., w)$ denotes the function from S to S associating to a state p the state reached by A when reading w starting from p. For all $V \in \mathbf{V}$, we define the language L_V as:

$$L_V = \{w \in \sigma^* \mid \exists w' \in L(V) \;\; \delta(\cdot, w) = \delta(\cdot, w')\}.$$

[2] More precisely [6] further proved that 2. and 3. are equivalent not only for $\mathbf{V}(\mathrm{D})$ but for all τ-structures, and even without the assumption that $\mathbf{V}$ determines Q in a monotone way.

[3] CSP are usually defined as boolean problems, i.e. without the nodes u, v. As RPQ queries are binary, these parameters are necessary for our presentation. The problem $\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$, as defined here, can be viewed as a classical CSP problem by extending the signature with two unary predicates, interpreted as the source and the target nodes, as done in [6].

We claim that L_V is a regular language. To see this recall that for each function f from S to S the language L_f defined as $L_f = \{w \in \sigma^* \mid \delta(\cdot, w) = f\}$ is regular and notice that L_V is a union of such languages. We remark here for later that L_V is constructible as soon that it is decidable whether $L(V) \cap L_f$ is non empty. This is in particular the case when $L(V)$ is context-free.

We now define a new view $\tilde{\mathbf{V}}$ defined as the RPQ view:

$$\tilde{\mathbf{V}} = \{\tilde{V} \mid V \in \mathbf{V} \text{ and } L(\tilde{V}) = L_V\}$$

Let E be a view instance for $\mathbf{V}$. We define $\tilde{\mathrm{E}}$ as a copy of E where each V relation is replaced by $\tilde{V}$. Hence, $\tilde{\mathrm{E}}$ is a view instance for $\tilde{\mathbf{V}}$. We now show that:

$$cert_{\mathrm{Q},\mathbf{V}}(\mathrm{E}) = cert_{\mathrm{Q},\tilde{\mathbf{V}}}(\tilde{\mathrm{E}})$$

and thus $cert_{\mathrm{Q},\mathbf{V}}(\mathrm{E})$ can be evaluated in coNP in the size of $\tilde{\mathrm{E}}$, which is also the size of E.

- Assume that $(u, v) \in cert_{\mathrm{Q},\tilde{\mathbf{V}}}(\tilde{\mathrm{E}})$. Hence, for all D such that $\tilde{\mathbf{V}}(\mathrm{D}) \supseteq \tilde{\mathrm{E}}$, there exists a path π from u to v such that $\lambda(\pi) \in L(\mathrm{Q})$. Let D be a database such that $\mathbf{V}(\mathrm{D}) \supseteq \mathrm{E}$. Remark that, for all $V \in \mathbf{V}$, $L(V) \subseteq L(\tilde{V})$. Hence, $\tilde{\mathbf{V}}(\mathrm{D}) \supseteq \tilde{\mathrm{E}}$. Hence, there exists a path π in D from u to v such that $\lambda(\pi) \in L(\mathrm{Q})$, which means that $(u, v) \in cert_{\mathrm{Q},\mathbf{V}}(\mathrm{E})$.
- Conversely, assume that $(u, v) \notin cert_{\mathrm{Q},\tilde{\mathbf{V}}}(\tilde{\mathrm{E}})$. Hence, there exists a database D such that $\tilde{\mathbf{V}}(\mathrm{D}) \supseteq \tilde{\mathrm{E}}$, but no path from u to v in D satisfies Q. From D, we build a database D' as follows:
  - Start with D' as a copy of D.
  - For all $V \in \mathbf{V}$, for all $(x, y) \in \mathrm{E}$, if $(x, y) \in V$, then $(x, y) \in \tilde{V}$ in $\tilde{\mathrm{E}}$. We pick a path π in D' from x to y of label w' such that $w' \in L(\tilde{V})$. Hence, there exists $w \in L(V)$ such that $\delta(\cdot, w') = \delta(\cdot, w)$. Then, we add in D' a simple path from x to y using only fresh nodes of label w. Hence $(x, y) \in V(\mathrm{D}')$.

  Remark then that $\mathbf{V}(\mathrm{D}') \supseteq \mathrm{E}$. Let π' be a path from u to v in D'. Then π' is of the form $\pi' = \pi_1\mu_1\pi_2 \dots \pi_{n-1}\mu_{n-1}\pi_n$, where each π_i is a path that was originally in D and each μ_i is a new path using only fresh nodes. Then, for each μ_i, there exists a path ρ_i in D with the same starting and ending nodes and such that $\delta(\cdot, \lambda(\mu_i)) = \delta(\cdot, \lambda(\rho_i))$. Hence, we can define a path π of D as $\pi = \pi_1\rho_1\pi_2 \dots \pi_{n-1}\rho_{n-1}\pi_n$. Hence, $\delta(\cdot, \lambda(\pi')) = \delta(\cdot, \lambda(\pi))$.

  Since $(u, v) \notin cert_{\mathrm{Q},\tilde{\mathbf{V}}}(\tilde{\mathrm{E}})$, then $\delta(q_0, \lambda(\pi)) \notin F$. Hence, $\delta(q_0, \lambda(\pi')) \notin F$, which proves that $(u, v) \notin cert_{\mathrm{Q},\mathbf{V}}(\mathrm{E})$.

□

The proposition has the following consequence:

Corollary 22. *Let Q be a RPQ and $\boldsymbol{V}$ be a Context-Free Path Query view such that $\boldsymbol{V}$ determines Q in a monotone way. Then there exists a rewriting of Q using $\boldsymbol{V}$ that can be evaluated with* coNP *data complexity.*

Notice that the proof of Proposition 21 and therefore also Corollary 22 do not assume that the language defining the views are context-free and work with any language. However, in order to effectively construct the rewriting, it is necessary that the formalism used to define the views has a decidable emptiness test for the intersection with a regular language.

6.2 Existence of a Datalog rewriting

We now show that for each RPQ query Q and each RPQ view $\mathbf{V}$ such that $\mathbf{V}$ determines Q in a monotone way, there exists a Datalog rewriting.

The existence of such a rewriting stems from links between CSPs and Datalog. Recall from Proposition 18 that if $\mathbf{V}$ determines Q in a monotone way, $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$, viewed as a binary query, is a rewriting of Q using $\mathbf{V}$. It is known that to each CSP problem (i.e. arbitrary template), one can associate a canonical $\mathrm{Datalog}_{l,k}$ program, for each l, k, with $l \leq k$. This program can equivalently be described in terms of a two-player game, and can be thought of as a maximal "approximation" of the complement of a CSP problem, in a precise sense (the interested reader is referred to [11] for more details). Our main contribution consists in proving that, for some explicit values of l and k (depending on Q and $\mathbf{V}$), this $\mathrm{Datalog}_{l,k}$ approximation is "exact" when restricted to view images (i.e. computes precisely $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$), and is therefore a rewriting over such instances.

We now present the (l, k)-two-player game of [11], and its correspondence with Datalog.

Definition 23 ((l, k)-two-player game)**.** *Let l, k be two integers, with $l \leq k$, let E be a τ-structure and u, v be two nodes of E. The (l, k)-game on $(E, T_{Q,\mathbf{V}}, u, v)$ is played by two players as follows:*

- *The game begins with $A_0 = \emptyset$ and h_0 being the empty function over A_0.*

  For $i \geq 0$, round $i + 1$ is defined as follows:

- *Player 1 selects a set A_{i+1} of nodes of E, with $|A_{i+1}| \leq k$ and $|A_i \cap A_{i+1}| \leq l$.*

- *Player 2 responds by giving a homomorphism $h_{i+1} : E[A_{i+1}] \to T_{Q,\mathbf{V}}$ that coincides with h_i on $A_i \cap A_{i+1}$ and such that $h_{i+1}(u)$ is a source node and $h_{i+1}(v)$ is a target node whenever u or v are in A_{i+1}.*

Player 1 wins if at any point Player 2 has no possible move. Player 2 wins if she can play forever.

The existence of a winning strategy for Player 1 is expressible in Datalog:

Lemma 24 ([11, 6])**.** *Let l, k be two integers, with $l \leq k$, and Q and $\mathbf{V}$ be an RPQ query and an RPQ view. Then there exists a program $Q_{l,k}(x, y)$ in $\mathrm{Datalog}_{l,k}$ such that for every graph database E, $Q_{l,k}(E)$ is the set of pairs (u, v) such that Player 1 has a winning strategy for the (l, k)-two-player game on $(E, T_{Q,\mathbf{V}}, u, v)$.*

Moreover the program in the above lemma can be effectively constructed from $T_{\mathrm{Q},\mathbf{V}}$, and therefore from Q and $\mathbf{V}$. It will be simply denoted by $Q_{l,k}$ when Q and $\mathbf{V}$ are clear from the context.

We are now ready to state the main technical result of our paper.

Proposition 25. *Let $\mathbf{V}$ and Q be an RPQ view and an RPQ query such that $\mathbf{V}$ determines Q in a monotone way. There exists l such that $Q_{l,l+1}$ is a rewriting of Q using $\mathbf{V}$.*

Theorem 16 is an immediate consequence of this proposition. The rest of this section is devoted to proving Proposition 25. This is done in two steps. We first prove that there exists l such that $Q_{l,l+1}$ is a rewriting of Q using $\mathbf{V}$, when restricted to view images of simple path graph databases. We then show that this suffices for $Q_{l,l+1}$ to be a rewriting of Q using $\mathbf{V}$.

Observe that if there is a homomorphism from a τ-structure E to $T_{\mathrm{Q},\mathbf{V}}$ sending u to a source node and v to a target node, then Player 2 has a winning strategy for the (l, k)-two-player game on $(\mathrm{E}, T_{\mathrm{Q},\mathbf{V}}, u, v)$. This strategy consists in always playing the restriction of the homomorphism on the set selected by Player 1. In this sense the program $Q_{l,k}$ is a $\mathrm{Datalog}_{l,k}$ under-approximation of the $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ problem: if $(u, v) \in Q_{l,k}(\mathrm{E})$ then $(\mathrm{E}, u, v) \in \neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$. If moreover $\mathrm{E} = \mathbf{V}(\mathrm{D})$ for some σ-structure D then, by Proposition 18, $(u, v) \in Q_{l,k}(\mathbf{V}(\mathrm{D}))$ implies $(u, v) \in \mathrm{Q}(\mathrm{D})$. We will refer to this property by saying that $Q_{l,k}$ is always *sound*.

The converse inclusion does not necessarily hold. If $(u, v) \notin Q_{l,k}(\mathrm{E})$ then Player 2 has a winning strategy, but this only means that she can always exhibit partial homomorphisms from E

to $T_{\mathrm{Q},\mathbf{V}}$ (sometimes called *local consistency checking*); this is in general not sufficient to guarantee the existence of a suitable global homomorphism.

However here we are not interested in arbitrary τ-structures, but only structures of the form $\mathbf{V}(\mathrm{D})$ for some simple path graph database D. We now show that, thanks to the particular properties of these structures, local consistency checking is sufficient to obtain a global homomorphism, for some suitable l and $k = l + 1$. In other words, the program $Q_{l,l+1}$ computes precisely $\neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$ on views of simple path graph databases.

The case of simple path graph databases

Proposition 26. *Let $\boldsymbol{V}$ and Q be an RPQ view and an RPQ query. There exists l such that for every simple path database D from u to v,*

$$(u, v) \in Q_{l,l+1}(\boldsymbol{V}(D)) \text{ iff } (\boldsymbol{V}(D), u, v) \in \neg CSP(T_{Q,\boldsymbol{V}}).$$

In particular if $\boldsymbol{V}$ determines Q in a monotone way,

$$(u, v) \in Q_{l,l+1}(\boldsymbol{V}(D)) \text{ iff } (u, v) \in Q(D).$$

Proof. Let $\mathbf{V}$ and Q be an RPQ view and an RPQ query, and let D be a graph database consisting of a simple path from node u to node v. Assume $u, v \in \mathbf{V}(\mathrm{D})$.

We will show, in Lemma 29 below, that for large enough l, if Player 2 has a winning strategy on the game on $(\mathbf{V}(\mathrm{D}), T_{\mathrm{Q},\mathbf{V}}, u, v)$ then we can exhibit a homomorphism witnessing the fact that $(\mathbf{V}(\mathrm{D}), u, v) \in \mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$. Before that we prove crucial properties of $\mathbf{V}(\mathrm{D})$ which will be exploited in the sequel. For that we need the following simple definitions and claims.

Let D consist of the simple path $\pi = x_0 a_1 x_1 \dots x_{m-1} a_m x_m$, with $x_0 = u$ and $x_m = v$. Moreover let $\mathrm{E} = \mathbf{V}(\mathrm{D})$ and let $A = \langle S_{\mathbf{V}}, \delta_{\mathbf{V}}, q^0_{\mathbf{V}}, F_{\mathbf{V}} \rangle$ be the product automaton of all the deterministic minimal automata of all the regular expressions of the RPQs in $\mathbf{V}$. Let $N(\mathbf{V})$ be the number of states of A, i.e. $|S_{\mathbf{V}}|$.

In what follows, for $q \in S_{\mathbf{V}}$ and $w \in \sigma^*$, $\delta_{\mathbf{V}}(q, w)$ denotes the state $p \in S_{\mathbf{V}}$ such that there is a run of A on w starting in state q and arriving in state p.

For every $k \leq m + 1$, and every $i, j \leq k$, we say that $x_i \sim_k x_j$ in $\mathbf{V}(\mathrm{D})$ if, for all $V \in \mathbf{V}$, for all $r \geq k$,

$$(x_i, x_r) \in V(\mathrm{D}) \quad \Leftrightarrow \quad (x_j, x_r) \in V(\mathrm{D})$$

For all k, the relation $\sim_k$ is an equivalence relation over $\{x_i \mid i \leq k\}$. We now prove the main property of $\mathbf{V}(\mathrm{D})$, namely that the index of all $\sim_k$ is bounded by the size of $\mathbf{V}$.

Claim 27. *For all $k \leq m + 1$:*

$$\left| \{x_i \mid i \leq k\} / \sim_k \right| \leq N(\boldsymbol{V})$$

Proof. To each node x_i in π with $i \leq k$, we associate a state $\varphi(x_i) \in S_{\mathbf{V}}$ defined as :

$$\varphi(x_i) = \delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{i \to k}))$$

where $\pi_{s \to t}$ is defined as the subpath of π that starts at position s and ends at position t, that is $\pi_{s \to t} = x_s a_s x_{s+1} a_{s+1} \dots a_{t-1} x_t$.

Assume that there exist two nodes x_i and x_j, with $i, j \leq k$, that have the same image in φ. It follows that:

$$\delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{i \to k})) = \delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{j \to k}))$$

Let us prove that $x_i \sim_k x_j$. Assume that there exist $r \geq k$ and $V \in \mathbf{V}$ such that $(x_i, x_r) \in V(\mathrm{D})$. Then $\delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{i \to r}))$ is final for V. Remark that $\lambda(\pi_{i \to r}) = \lambda(\pi_{i \to k})\lambda(\pi_{k \to r})$, from which we can deduce that :

$$\delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{i \to r})) = \delta_{\mathbf{V}}(\varphi(x_i), \lambda(\pi_{k \to r}))$$

Hence,

$$\delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{i\to r})) = \delta_{\mathbf{V}}(\varphi(x_j), \lambda(\pi_{k\to r}))$$

We can now conclude that $\delta_{\mathbf{V}}(q^0_{\mathbf{V}}, \lambda(\pi_{j\to r}))$ is final for V, which means that $(x_j, x_r) \in V(\mathrm{D})$. A symmetric argument easily proves the other direction of the equivalence. Hence, $x_i \sim_k x_j$, and we can finally conclude that there cannot be more that $N(\mathbf{V})$ distinct equivalence classes of $\sim_k$ over the nodes $\{x_i \mid i \leq k\}$ of π. □

The following easily verified property of the equivalence relations $\sim_k$ will also be useful:

Claim 28. *Let $k_1, k_2 \leq m+1$, with $k_1 \leq k_2$. Let x and y be two elements of π that occur before x_{k_1}. Then $x \sim_{k_1} y$ implies $x \sim_{k_2} y$.*

We are now ready to prove the statement of the Proposition.

Let $l = |T_{\mathrm{Q},\mathbf{V}}| \cdot N(\mathbf{V})$. We prove that $(u, v) \in Q_{l,l+1}(\mathrm{E})$ iff $(\mathrm{E}, u, v) \in \neg\mathrm{CSP}(T_{\mathrm{Q},\mathbf{V}})$. In view of the fact that $Q_{l,l+1}$ encodes the $(l, l+1)$-two-player game in the sense of Lemma 24, it is enough to prove the following:

Lemma 29. *Player 2 has a winning strategy for the $(l, l+1)$-two-player game on $(E, T_{Q,\mathbf{V}}, u, v)$ iff there is an homomorphism from E to $T_{Q,\mathbf{V}}$ sending u to a source node and v to a target node.*

Proof. The right-left direction is obvious. If there is a suitable homomorphism $h : \mathrm{E} \to T_{\mathrm{Q},\mathbf{V}}$, then Player 2 has a winning strategy which consists in playing according to h.

Conversely, assume that Player 2 has a winning strategy for the $(l, l+1)$-two-player game on $(\mathrm{E}, T_{\mathrm{Q},\mathbf{V}}, u, v)$. Let $\{s_1, s_2, \ldots, s_r\}$ be an ordering of the elements of E, according to the order on π, that is, in such a way that $\forall j \leq k$, s_j occurs before s_k in π. Clearly $s_1 = u$ and $s_r = v$. If $r \leq l+1$, Player 1 can select all elements of E in a single round, and then Player 2 has to provide a full homomorphism from E to $T_{\mathrm{Q},\mathbf{V}}$, which concludes the proof.

Assume $r > l+1$. For ease of notations, we will number rounds starting from $l+1$. This can be seen just as a technicality, or equivalently as Player 1 selecting the empty set for the first l rounds. Since Player 2 has a winning strategy, she has, in particular, a winning response against the following play of Player 1 :

- On round $l+1$, Player 1 plays $A_{l+1} = \{s_1, \ldots, s_{l+1}\}$. Player 2 has to respond with a partial homomorphism h_{l+1}, which she can do, since she has a winning strategy.
- Assume that, on round i, A_i is of size $l+1$ and its element of biggest index is s_i (as it is the case on round $l+1$). Given the choice of l, the set A_i is sufficiently "big", that is by Claim 27, there exist two elements $s_j, s_k \in A_i$ such that $s_j \sim_i s_k$, and $h_i(s_j) = h_i(s_k)$. On round $i+1$, Player 1 picks $A_{i+1} = (A_i - \{s_j\}) \cup \{s_{i+1}\}$. This choice maintains that A_{i+1} is of size $l+1$ and that its element of biggest index is s_{i+1}. Once again, Player 2 has to respond with a partial homomorphism h_{i+1}, which she can do.
- Following this play, on round r, A_r contains s_r, the element of biggest index in E. From now on, we no longer care about Player 1's move, that is, we arbitrarily set $A_i = \emptyset$ for all $i > r$.

We can now define h as follows :

$$h(s_i) = \begin{cases} h_{l+1}(s_i) & \text{if } i \leq l+1 \\ h_i(s_i) & \text{if } l+1 < i \leq r \end{cases}$$

Observe that, by definition, the mapping h sends u to a source node and v to a target node (since so do all the h_i's used in the game). It remains to prove that h is an homomorphism from E to $T_{\mathrm{Q},\mathbf{V}}$. We prove by induction on $i \geq l+1$ that :

(H_1) h is a homomorphism from $\mathrm{E}[\{s_1, \ldots, s_i\}]$ to $T_{\mathrm{Q},\mathbf{V}}$.

(H_2) h coincides with h_i on A_i.

(H_3) for all $j \leq i$, there exists $s \in A_i$ such that $s_j \sim_i s$ and $h(s_j) = h(s)$.

Base case : For $i = l + 1$, the mapping h coincides by definition with h_{l+1} on $\{s_1, \ldots, s_{l+1}\}$. Hence, (H_1) and (H_3) follow easily.

Inductive case : Assume that there exists i with $l + 1 \leq i < r$ such that (H_1),(H_2) and (H_3) holds for i; we prove them for $i + 1$.

(H_2) Let $s \in A_{i+1}$. If $s = s_{i+1}$, then, by definition, $h(s_{i+1}) = h_{i+1}(s_{i+1})$. Otherwise, $s \in A_i \cap A_{i+1}$. (H_2) for i implies that $h(s) = h_i(s)$, and the definition of h_{i+1} thus yields $h_{i+1}(s) = h_i(s) = h(s)$. Hence, (H_2) holds for $i + 1$.

(H_3) Let $j \leq i + 1$. If $j = i + 1$, then $s_j \in A_{i+1}$, and the result is obvious. Otherwise, (H_3) for i implies that there exists $s \in A_i$ such that $s_j \sim_i s$ and $h(s_j) = h(s)$. From Claim 28, we deduce that $s_j \sim_{i+1} s$. If $s \in A_{i+1}$, there is nothing more to prove. Otherwise, it means that s is exactly the element that was removed from A_i on round $i + 1$, which means that there exists another element $s' \in A_i \cap A_{i+1}$ such that $s \sim_i s'$ and $h_i(s) = h_i(s')$. Then Claim 28 and (H_2) imply that $s_j \sim_{i+1} s'$ and $h(s_j) = h(s')$. Hence (H_3) holds for $i + 1$.

(H_1) By definition, h already preserves any self-loop. Moreover, (H_1) for i implies that h is a homomorphism from $\mathrm{E}[\{s_1, \ldots, s_i\}]$ to $T_{\mathrm{Q},\mathbf{V}}$. Hence, any edge between two elements of $\{s_1, \ldots, s_i\}$ in E is already preserved by h. Let $s_j \in \{s_1, \ldots, s_i\}$. Remark that, since π is a simple path, there are no edges from s_{i+1} to s_j in E. Thus, we just have to prove that all edges from s_j to s_{i+1} are preserved by h.

(H_3) for $i + 1$ implies that there exists an element $s \in A_{i+1}$ such that $s_j \sim_{i+1} s$ and $h(s_j) = h(s)$. Since h_{i+1} is a homomorphism on $\mathrm{E}[A_{i+1}]$, it preserves all edges from s to s_{i+1}. Moreover, (H_2) for $i + 1$ implies that h and h_{i+1} coincide on A_{i+1}, which means that h preserves all edges from s to s_{i+1}. Finally, the definition of $\sim_{i+1}$ implies that s_j and s have the same edges to s_{i+1}. Hence, h preserves all edges from s_j to s_{i+1}.

Finally, (H_1) applied for r proves that h is indeed a homomorphism from E to $T_{\mathrm{Q},\mathbf{V}}$.

This completes the proof of Lemma 29. □

Now assume $\mathbf{V}$ determines Q in a monotone way, then from Proposition 18 it immediately follows that $(u, v) \in Q_{l,l+1}(\mathbf{V}(\mathrm{D}))$ iff $(u, v) \in \mathrm{Q}(\mathrm{D})$. This completes the proof of Proposition 26. □

From simple paths to arbitrary graph databases Proposition 26 shows that if Q determines $\mathbf{V}$ in a monotone way then $Q_{l,l+1}$ is a rewriting of Q using $\mathbf{V}$, when restricted to simple path databases. It remains to lift this result to arbitrary graph databases. In a sense, the following result shows that the general case can always be reduced to the simple path case.

Proposition 30. *Let $\boldsymbol{V}$ and Q be an RPQ view and an RPQ query such that $\boldsymbol{V}$ determines Q in a monotone way. Assume $\mathcal{P}$ is a query of schema τ such that:*

1. *$\mathcal{P}$ is closed under homomorphisms: for all databases E, E', and all pair of elements (u, v) of E, if $(u, v) \in \mathcal{P}(E)$ and there exists a homomorphism $h : E \to E'$ then $(h(u), h(v)) \in \mathcal{P}(E')$.*
2. *$\mathcal{P}$ is sound and complete for all simple path databases: for all simple path databases D from u to v such that u and v are in the domain of $\boldsymbol{V}(D)$, we have $(u, v) \in \mathcal{P}(\boldsymbol{V}(D))$ iff $(u, v) \in Q(D)$.*
3. *$\mathcal{P}$ is always sound: for all graph databases D and elements u and v of $\boldsymbol{V}(D)$, if $(u, v) \in \mathcal{P}(\boldsymbol{V}(D))$ then $(u, v) \in Q(D)$.*

Then $\mathcal{P}$ is a rewriting of Q using $\boldsymbol{V}$.

Proof. Let D be a database, and (u, v) be a pair of elements of $\mathbf{V}(\mathrm{D})$, such that $(u, v) \in \mathrm{Q}(\mathrm{D})$. Then there exists in D a path π_0 from u to v, such that $\lambda(\pi_0) \in L(\mathrm{Q})$.

Consider the simple path $\pi = x_0 a_0 x_1 \dots x_m a_m x_{m+1}$ defined such that $\lambda(\pi) = \lambda(\pi_0)$. Since $\mathbf{V}$ determines Q in a monotone way and $\lambda(\pi) \in L(\mathrm{Q})$, then x_0 and x_{m+1} are in the domain of $\mathbf{V}(\pi)$, and $(x_0, x_{m+1}) \in \mathrm{Q}(\pi)$. Hence, (2) implies that $(x_0, x_{m+1}) \in \mathcal{P}(\mathbf{V}(\pi))$.

Additionally, it is clear that there exists a homomorphism h from π to D with $h(x_0) = u$ and $h(x_{m+1}) = v$. Observe that h extends to the views of π and D, that is h is an homomorphism from $\mathbf{V}(\pi)$ to $\mathbf{V}(\mathrm{D})$, and (1) thus implies that $(u, v) \in \mathcal{P}(\mathbf{V}(\mathrm{D}))$.

The other direction is immediately given by (3). □

We now have all the elements to prove Proposition 25. Let $\mathbf{V}$ and Q be an RPQ view and an RPQ query such that $\mathbf{V}$ determines Q in a monotone way. By Proposition 26 there exists l such that $Q_{l,l+1}$ is sound and complete over simple path databases. Moreover each Datalog query is preserved under homomorphisms, and we have already observed that all $Q_{l,k}$ are always sound. It then follows from Proposition 30 that there exists l such that $Q_{l,l+1}$ is a rewriting of Q using $\mathbf{V}$. This proves Proposition 25 and therefore Theorem 16.

7 Conclusions

We have seen that if an RPQ view $\mathbf{V}$ determines an RPQ query Q in a monotone way then a Datalog rewriting can be computed from $\mathbf{V}$ and Q. As a corollary it is decidable whether there exists a Datalog rewriting to an RPQ query using RPQ views.

These results extends to 2-way-RPQ. A 2-way-RPQ is defined using a regular expression over the alphabet $\sigma \cup \bar{\sigma}$. It asks for pairs of nodes linked by a 2-way-path using the symbol a for traversing an edge of label a in the direction of the arrow, and the symbol $\bar{a}$ for backward traversing an edge of label a. This query language has been studied in [9]. In particular [9] gives an extension of Corollary 12 and of Proposition 18 for 2-way-RPQ. Building from these two results it is possible to extend the results of Section 6 to 2-way-RPQs. The details are more complicated and omitted here, but the general idea is the same.

We may wonder whether a simpler query language than Datalog could suffice to express monotone rewritings of RPQ queries using RPQ views. For instance all examples we are aware of use only the transitive closure of binary Conjunctive Regular Path Queries. It is then natural to ask whether linear Datalog (where at most one internal predicate may occur in the body of each rule), using internal predicates of arity at most 2, can express all monotone rewritings. We leave this interesting question for future work.

Finally we conclude by mentioning that we don't know yet whether the monotone determinacy problem for Conjunctive Regular Path Query is decidable. Likewise, deciding whether an RPQ view determines an RPQ query, without the monotonicity assumption, is still an open problem.

References

[1] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 254–263, 1998.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] Foto N. Afrati. Determinacy and query rewriting for conjunctive queries and views. *Theoretical Computer Science*, 412(11):1005–1021, 2011.

[4] Pablo Barceló Baeza. Querying graph databases. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 175–188, 2013.

[5] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *Intl. Conf. on Data Engineering (ICDE)*, pages 389–398. IEEE, 2000.

[6] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing and constraint satisfaction. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 361–371. IEEE, 2000.

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Lossless regular views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 247–258. ACM, 2002.

[8] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Rewriting of regular expressions and regular path queries. *Journal of Computer and System Sciences*, 64(3):443–465, 2002.

[9] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query processing: On the relationship between rewriting, answering and losslessness. *Theoretical Computer Science*, 371(3):169–182, 2007.

[10] Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. A graphical query language supporting recursion. *SIGMOD Rec.*, 16(3):323–330, December 1987.

[11] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998.

[12] Alon Y. Levy, Alberto O. Mendelzon, Yehoshua Sagiv, and Divesh Srivastava. Answering queries using views. In *ACM Symp. on Principles of Database Systems (PODS)*, pages 95–104, 1995.

[13] Alan Nash, Luc Segoufin, and Victor Vianu. Views and queries: Determinacy and rewriting. *ACM Transactions on Database Systems*, 35(3), 2010.

[14] Jorge Pérez. *Schema Mapping Management in Data Exchange Systems*. PhD thesis, Escuela de Ingeniería, Pontificia Universidad Católica de Chile, 2011.