Graph Logics with Rational Relations: The Role of Word Combinatorics

Pablo Barceló Pablo Muñoz

Dept. of Computer Science
University of Chile
pbarcelo@dcc.uchile.cl pmunoz@dcc.uchile.cl

Abstract

Graph databases make use of logics that combine traditional first-order features with navigation on paths, in the same way logics for model checking do. However, modern applications of graph databases impose a new requirement on the expressiveness of the logics: they need comparing labels of paths based on word relations (such as prefix, subword, or subsequence). This has led to the study of logics that extend basic graph languages with features for comparing labels of paths based on regular relations, or the strictly more powerful rational relations. The evaluation problem for the former logic is decidable (and even tractable in data complexity), but already extending this logic with such a common rational relation as subword or suffix turns evaluation undecidable.

In practice, however, it is rare to have the need for such powerful logics. Therefore, it is more realistic to study the complexity of less expressive logics that still allow comparing paths based on practically motivated rational relations. Here we concentrate on the most basic such languages, which extend graph pattern logics with path comparisons based only on suffix, subword or subsequence. We pinpoint the complexity of evaluation for each one of these logics, which shows that all of them are decidable in elementary time (PSPACE or NEXPTIME). Furthermore, the extension with suffix is even tractable in data complexity (but the other two are not). In order to obtain our results we establish a link between the evaluation problem for graph logics and two important problems in word combinatorics: word equations with regular constraints and square shuffling.

Categories and Subject Descriptors H.2.3 [*Database Management*]: Languages—Query Languages

Keywords rational relations, logics for graphs, complexity of evaluation, regular path queries, word equations, shuffle

1. Introduction

Graph databases are important for applications in which the topology of the data is as important as data itself. Intuitively, a graph database represents objects (the nodes) and relationships between those objects (often modeled as labeled edges). The last years have witnessed an increasing interest in graph databases, due to the uprise of applications that need to manage and query massive and highly-connected data (e.g., the semantic web, social networks and biological networks). See [2] for a survey on graph database models, and [3, 23] for surveys on graph logical languages.

The logics used for specifying properties of graph databases combine standard first-order features with *navigational* ones. The latter allow to recursively traverse the edges of the graph while checking for the existence of paths satisfying given conditions. These navigational features are close in spirit to the ones used in logics for model checking [10]. Notably, basic evaluation tasks for both families of languages can be carried out using similar techniques based on automata.

The building block for navigational languages over graph databases is the class of *regular path queries* [11], or RPQs, that define pairs of nodes in graph databases linked by a path whose label satisfies a regular expression. Closing RPQs under conjunction and existential quantification gives rise to the *conjunctive* RPQs, or CRPQs [9]. Evaluation of CRPQs is NP-complete, but its *data complexity* – i.e., the complexity when the query is assumed to be fixed – is tractable (NLOGSPACE). The latter is considered to be acceptable in the database context [22].

It has been noticed that CRPQs fall short of expressive power for modern applications of graph databases due to their inability to compare paths [5]. For instance, semantic web languages compare paths based on semantic associations and biological sequences are compared in terms of their mutual edit distance, but these requirements cannot be expressed with CRPQs. To overcome this limitation, a family of *extended CRPQs* has been proposed [3]. The logics in this family extend CRPQs with the ability to compare labels of paths with elements from a set $\mathcal{S}$ of relations on words. Each such logic is denoted CRPQ($\mathcal{S}$) (or simply CRPQ(S) in the case when $\mathcal{S} = \{S\}$).

The first such logic to be studied was CRPQ(REG) [5], where REG is the class of *regular relations* on words [13], or equivalently, relations defined by *synchronous* n-ary automata. The class REG includes important relations on strings, such as prefix, equal length of words, and fixed edit distance. Using automata techniques it can be shown that CRPQ(REG) preserves the good data complexity properties of CRPQs (that is, evaluation of CRPQ(REG) queries is in NLOGSPACE in data complexity). Still, the expressiveness of this logic is limited for many applications; e.g., in biological networks or the semantic web one deals with subwords and subsequences, but these relations are *not* regular. They are *rational*; i.e., they can still be defined by automata, but those whose heads move asynchronously [6].

Adding rational relations to CRPQs has to be done carefully since the evaluation problem for CRPQ(RAT) is undecidable. How-

CSL-LICS 2014, July 14–18, 2014, Vienna, Austria.

http://dx.doi.org/10.1145/2603088.2603122

ever, we are not interested in all rational relations, but only in some particular ones often encountered in practice. The approach taken by Barceló, Figueira and Libkin in [4] was studying to what extent rational relations such as subword $\preceq_{\mathrm{sw}}$, suffix $\preceq_{\mathrm{suff}}$ or subsequence $\preceq_{\mathrm{ss}}$, can be added to CRPQ(REG) without losing decidability of evaluation. It was shown that this is not possible for the first two; i.e., evaluation for CRPQ(REG $\cup \{\preceq_{\mathrm{sw}}\}$) and CRPQ(REG $\cup \{\preceq_{\mathrm{suff}}\}$) is undecidable. On the other hand, evaluation for CRPQ(REG $\cup \{\preceq_{\mathrm{ss}}\}$) is decidable, but with very high data complexity (non-elementary, or not bounded by any stack of exponentials). Therefore, these languages are impractical.

In practice, however, it is uncommon to have the need to compare paths based on both the aforementioned rational relations and arbitrary regular relations in REG. Therefore, a more realistic approach would be to study the complexity of evaluation for less expressive logics, starting from those of the form CRPQ($\preceq$) – for $\preceq$ one of $\preceq_{\mathrm{sw}}$, $\preceq_{\mathrm{suff}}$ or $\preceq_{\mathrm{ss}}$ – that only allow to compare paths based on a *single* rational relation of interest. Our goal is to understand what is the cost of evaluation for such logics (and, in particular, if any of them can be evaluated efficiently in data complexity).

Some partial results in [4], obtained using automata techniques, show that this restriction dramatically reduces the complexity for two of the logics: evaluation for CRPQ($\preceq_{\mathrm{suff}}$) and CRPQ($\preceq_{\mathrm{ss}}$) is in NEXPTIME, and data complexity is in NP. On the other hand, such techniques were insufficient for determining the precise complexity of these problems and for establishing the decidability of evaluation for the logic CRPQ($\preceq_{\mathrm{sw}}$). In particular, they provide no answer to the question of which of these logics have good behavior in terms of data complexity.

In this paper we provide complete answers to the previous questions by establishing a "missing link" between the evaluation problem for these logics and important problems in word combinatorics. Before explaining those techniques and our results in depth, it is worth mentioning that our results do not intend to be specific to the aforementioned rational relations. For instance, our positive results are obtained in the most general possible way, so that they could be later used to obtain tractability of evaluation for CRPQs extended with different rational relations. Also, our negative lower bounds might serve as the ground over which the intractability of other graph logics can be established.

Proof techniques and main results We start by noticing that the evaluation problem for the logics CRPQ($\preceq_{\mathrm{suff}}$) and CRPQ($\preceq_{\mathrm{sw}}$) can be reduced in PSPACE to the solvability of word equations with regular constraints. The latter has been shown to be in PSPACE [12], based on ideas generated by the sophisticated Makanin's algorithm [19]. This immediately answers one of the questions left open in [4]: evaluation for CRPQ($\preceq_{\mathrm{sw}}$) is decidable in PSPACE. The evaluation of both CRPQ($\preceq_{\mathrm{sw}}$) and CRPQ($\preceq_{\mathrm{suff}}$) is known to be PSPACE-hard [4]; therefore, both problems are PSPACE-complete (which matches the complexity of evaluation for first-order logic).

We then move to study whether any of these languages can be evaluated in polynomial time in data complexity (that is, assuming the formula to be fixed). In this case the reduction to solvability of word equations can be constructed in logarithmic space and yields a *fixed* word equation, while the input consists of the regular constraints only. We identify a condition on this class of instances that ensures solvability in NLOGSPACE; such condition refers to the existence of only a finite number of *principal* solutions for the word equation (when regular constraints are not taken into account). We then show that equations generated by reduction from the evaluation problem for CRPQ($\preceq_{\mathrm{suff}}$) are of this form, and thus that CRPQ($\preceq_{\mathrm{suff}}$) can be evaluated in NLOGSPACE in data complexity.

This technique cannot be applied to CRPQ($\preceq_{\mathrm{sw}}$) since equations generated from this language do not have the finite number of principal solutions property (even in restricted settings). Moreover, we prove the quite surprising result that evaluation of CRPQ($\preceq_{\mathrm{sw}}$) is PSPACE-complete even in data complexity (i.e., there is a *fixed* CRPQ($\preceq_{\mathrm{sw}}$) formula for which evaluation is PSPACE-complete). This shows a striking difference between CRPQ($\preceq_{\mathrm{sw}}$) and CRPQ($\preceq_{\mathrm{suff}}$) in terms of the data complexity of evaluation. As a corollary to our techniques we obtain the following result of independent interest: There is a word equation e, such that checking solvability of e under regular constraints is PSPACE-complete. This result was not previoulsy known in the literature.

In the second part of the paper we study the complexity of evaluation for CRPQ($\preceq_{\mathrm{ss}}$). This case is different since we cannot reduce it to solvability of word equations with regular constraints. Instead, we have to use different techniques to prove that the previous bounds obtained in [4] for this problem are sharp. We start by showing that the evaluation problem for CRPQ($\preceq_{\mathrm{ss}}$) is NP-complete in data complexity. The lower bound is obtained by applying *word shuffling* techniques. In particular, we establish a simple reduction from the problem of *unshuffling a square*, i.e., the problem of checking whether a word w can be obtained by taking the shuffle of some word u with itself, which has been very recently proved to be NP-complete [8, 21]. We also prove that, in general, evaluation for CRPQ($\preceq_{\mathrm{ss}}$) is NEXPTIME-complete. In this case, we use a more cumbersome reduction from a suitable succinct version of the longest common subsequence problem. This proves that, in its full generality, the language CRPQ($\preceq_{\mathrm{ss}}$) is impractical.

Organization of the paper We present basic notation and results in Section 2 and a review of logics over graph databases in Section 3. Our results on the complexity of evaluation for the logics CRPQ($\preceq_{\mathrm{suff}}$) and CRPQ($\preceq_{\mathrm{sw}}$) are presented in Section 4, and those for CRPQ($\preceq_{\mathrm{ss}}$) in Section 5. We finish with our final remarks in Section 6.

2. Preliminaries

Practical relations on words Let Σ be a finite alphabet and assume that w, w', w'', u are words in Σ^* such that $w = w'uw''$. Then:

- u is a *subword* of w (written as $u \preceq_{\mathrm{sw}} w$),
- w' is a *prefix* of w (written as $w' \preceq_{\mathrm{pref}} w$), and
- w'' is a *suffix* of w (written as $w'' \preceq_{\mathrm{suff}} w$).

We say that w' is a *subsequence* of w (written as $w' \preceq_{\mathrm{ss}} w$) if w' is obtained by removing some letters (perhaps none) from w, i.e., $w = a_1 \dots a_n$, and $w' = a_{i_1} a_{i_2} \dots a_{i_k}$, where $1 \leq i_1 < i_2 < \dots < i_k \leq n$.

When we want to make explicit that relation $\preceq$ is over Σ, for $\preceq$ one of $\preceq_{\mathrm{pref}}$, $\preceq_{\mathrm{suff}}$, $\preceq_{\mathrm{sw}}$, or $\preceq_{\mathrm{ss}}$, we write $\preceq^{\Sigma}$.

Regular and rational relations We assume familiarity with nondeterministic finite automata (NFA) and regular expressions. We start by defining regular relations. Let Σ be a finite alphabet, $\perp \notin \Sigma$ a new alphabet letter, and $\Sigma_\perp := \Sigma \cup \{\perp\}$. Each tuple $\bar{w} = (w_1, \dots, w_n)$ of words from Σ^* can be viewed as a word over $\Sigma_\perp^n$ as follows: pad words w_i with $\perp$ so that they all are of the same length, and use as the k-th symbol of the new word the n-tuple of the k-th symbols of the padded words. Formally, let $|w_i|$ be the length of the word w_i and $\ell = \max_i |w_i|$. Then $w_1 \otimes \dots \otimes w_n$ is a word of length ℓ whose k-th symbol is $(a_1, \dots, a_n) \in \Sigma_\perp^n$ such that:

$$a_i = \begin{cases} \text{the } k\text{th letter of } w_i & \text{if } |w_i| \geq k \\ \perp & \text{otherwise.} \end{cases}$$

A relation $R \subseteq (\Sigma^*)^n$ is called a *regular n-ary relation* over Σ if there is an NFA (or equivalently, a regular expression) over $\Sigma^n_\perp$ that defines $\{w_1 \otimes \ldots \otimes w_n \mid (w_1, \ldots, w_n) \in R\}$. The class of regular relations is denoted by REG, and we write REG_n to denote the restriction of REG to relations of arity n.

Example 2.1. The binary relation $\preceq^\Sigma_{\text{pref}}$ is regular, as witnessed by the expression $\big(\bigcup_{a\in\Sigma}(a,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(\perp,a)\big)^*$. On the other hand, there is a finite alphabet Σ such that $\preceq^\Sigma_{\text{sw}}$ is not regular. Similarly for $\preceq_{\text{suff}}$ and $\preceq_{\text{ss}}$. □

There are two equivalent ways to define rational relations over Σ. One uses regular expressions, which are now built from tuples $\bar{a} \in (\Sigma \cup \{\varepsilon\})^n$ applying the usual operations of union, concatenation, and Kleene star. Alternatively, n-ary rational relations can be defined by means of n-tape automata, that have n heads for the tapes and one additional control; at every step, based on the state and the letters it is reading, the automaton can enter a new state and move some (but not necessarily all) tape heads. The classes of n-ary relations so defined are called *rational n-ary relations*; we use the notation RAT_n and RAT, as before. For technical reasons we assume that rational relations are syntactically given as n-tape automata, but we often switch to the regular expressions view in examples and proofs to facilitate readability.

Example 2.2. Binary relations $\preceq^\Sigma_{\text{suff}}$, $\preceq^\Sigma_{\text{sw}}$, and $\preceq^\Sigma_{\text{ss}}$ are all rational:

- The expression $\big(\bigcup_{a\in\Sigma}(\varepsilon,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(a,a)\big)^*$ defines $\preceq_{\text{suff}}$.
- The relation $\preceq_{\text{sw}}$ is defined by the expression $\big(\bigcup_{a\in\Sigma}(\varepsilon,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(a,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(a,a)\big)^* \cdot \big(\bigcup_{a\in\Sigma}(\varepsilon,a)\big)^*$.
- The expression $\big(\bigcup_{a\in\Sigma}(\varepsilon,a) \cup (a,a)\big)^*$ defines $\preceq_{\text{ss}}$. □

Clearly, $\mathsf{RAT}_1 = \mathsf{REG}_1$, as both correspond to the class of regular languages. On the other hand, we have strict inclusions $\mathsf{REG}_k \subsetneq \mathsf{RAT}_k$ for each $k > 1$; e.g., there is a finite alphabet Σ such that $\preceq^\Sigma_{\text{suff}} \in \mathsf{RAT}_2 - \mathsf{REG}_2$. Same for $\preceq_{\text{sw}}$ and $\preceq_{\text{ss}}$.

We do not distinguish between an NFA (resp., regular expression) S and the set of n-tuples of words it defines; e.g., we write $\bar{w} \in S$ to denote that the n-tuple $\bar{w}$ of words belongs to the language defined by S. Also, we abuse notation and write $\preceq_{\text{sw}}$ to denote the set that consists of each rational relation $\preceq^\Sigma_{\text{sw}}$, for Σ a finite alphabet. Same for $\preceq_{\text{suff}}$ and $\preceq_{\text{ss}}$.

The generalized intersection problem As shown in [4], the evaluation problem for logics of the form CRPQ($\mathcal{S}$) ($\mathcal{S} \subseteq \mathsf{RAT}$) can be stated in language-theoretical terms. Such reformulation is known as the *generalized intersection problem*. We introduce such problem below; its relationship with the complexity of evaluation is explained in Section 3.

For the sake of our results, it is sufficient to concentrate on the case when $\mathcal{S}$ is a set of binary relations on words. We write $[m]$ for $\{1, \ldots, m\}$. For an *index set* $I \subseteq [m]^2$, we assume that mappings $\lambda : I \to 2^{\mathcal{S}}$ are always of finite range, i.e., $|\lambda((i,j))|$ is finite, for each pair $(i,j) \in I$. The generalized intersection problem for $\mathcal{S}$ is the following decision problem:

PROBLEM:	GENINT($\mathcal{S}$)
INPUT:	A tuple $(L_1, \ldots, L_m, I, \lambda)$ such that the L_i's are NFAs over Σ, $I \subseteq [m]^2$, and $\lambda : I \to 2^{\mathcal{S}}$.
QUESTION:	Are there words $w_i \in L_i$, for $i \in [m]$, such that $(w_i, w_j) \in S$ for all $(i,j) \in I$ and $S \in \lambda((i,j))$?

Intuitively, GENINT($\mathcal{S}$) asks if there are words $w_i \in L_i$, for $1 \le i \le m$, that satisfy the constraints specified by I and λ. Each such constraint forces a particular pair (w_i, w_j) to belong to every relation in $\lambda((i,j))$.

For a fixed index set $I \subseteq [m]^2$, we shall write GENINT$_I(\mathcal{S})$; in that case, the input to the problem consists of the NFAs $L_1, \ldots, L_m$ and the (finite range) mapping λ only.

In the case when $\mathcal{S} = (\mathsf{REG}_2 \cup \preceq)$, for $\preceq$ one of $\preceq_{\text{sw}}$, $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$, there is a particular restriction of GENINT($\mathcal{S}$) we are interested in. This takes as input a regular relation $R \in \mathsf{REG}_2$ over Σ, and the problem is determining whether the intersection of R and $\preceq$ is nonempty. Notice that this corresponds to the restriction of GENINT($\mathcal{S}$) in which $I = \{(1,2)\}$ is fixed, and inputs are of the form $(L_1 = \Sigma^*, L_2 = \Sigma^*, \lambda)$ for some λ that satisfies $\lambda((1,2)) = \{R, \preceq^\Sigma\}$, for $R \in \mathsf{REG}_2$. We denote this restriction by $(\mathsf{REG}_2 \cap \preceq)$. In case that the alphabet Σ is also fixed, we write $(\mathsf{REG}_2 \cap_\Sigma \preceq)$.

Next we present some important results from [4] regarding the complexity of the generalized intersection problem. We later explain how they can be used to determine the complexity of evaluation for graph logics. We start with classes that extend REG_2 with rational relations $\preceq_{\text{sw}}$, $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$. In this case the problem becomes either undecidable or highly intractable:

Theorem 2.1. [4]

1. *If $\preceq$ is one of $\preceq_{\text{sw}}$ or $\preceq_{\text{suff}}$, then there is a finite alphabet Σ such that $(\mathsf{REG}_2 \cap_\Sigma \preceq)$ is undecidable.*
2. *The problem GENINT($\mathsf{REG}_2 \cup \preceq_{\text{ss}}$) is decidable, but there is a finite alphabet Σ such that $(\mathsf{REG}_2 \cap_\Sigma \preceq_{\text{ss}})$ is non-elementary.*

We consider now the cases when $\mathcal{S}$ is $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$. This restriction allows to reduce the complexity of GENINT($\mathcal{S}$).

Theorem 2.2. [4] *Let $\preceq$ be $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$. Then* GENINT($\preceq$) *is in* NEXPTIME. *For each fixed I,* GENINT$_I(\preceq)$ *is in* NP.

The decidability of GENINT($\preceq_{\text{sw}}$) was left open in [4].

3. Review of Logics over Graph Databases

Graph databases The standard abstraction of graph databases [2] is finite Σ-labeled graphs $G = (V, E)$, where V is a finite set of nodes, and $E \subseteq V \times \Sigma \times V$ is a set of labeled edges. A *path* ρ from v_0 to v_m in G is a sequence of edges (v_0, a_0, v_1), $(v_1, a_1, v_2), \cdots, (v_{m-1}, a_{m-1}, v_m)$ from E, for some $m \ge 0$. The *label* of ρ, denoted by $\kappa(\rho)$, is the word $a_0 \cdots a_{m-1} \in \Sigma^*$. Notice that $\kappa(\rho)$ is the empty word ε if $\rho = v$, for $v \in V$.

Graph logics The main building block for graph logics are *regular path queries*, or *RPQ*s [11]; they are expressions of the form

$$\varphi(x,y) \;=\; x \xrightarrow{L} y,$$

where L is a regular expression over Σ. Given a Σ-labeled graph $G = (V, E)$, an RPQ $\varphi(x,y)$ of the form above, and v, v' nodes of G, we have that $G \models \varphi(v, v')$ iff there is a path ρ from v to v' in G with $\kappa(\rho) \in L$.

Conjunctive RPQs, or CRPQ*s* [9], are the closure of RPQs under conjunction and existential quantification. Formally, they are expressions of the form

$$\varphi(\bar{x}) \;=\; \exists \bar{y} \bigwedge_{i=1}^{m} (u_i \xrightarrow{L_i} u'_i), \tag{1}$$

where variables u_i, u'_is come from $\bar{x}, \bar{y}$. The semantics naturally extends the semantics of RPQs: $\varphi(\bar{a})$ is true in G iff there is a tuple $\bar{b}$ of nodes such that $|\bar{b}| = |\bar{y}|$ and for every $i \le m$ and every v_i, v'_i interpreting u_i and u'_i in $(\bar{a}, \bar{b})$, respectively, there is a path ρ_i from v_i and v'_i whose label $\kappa(\rho_i)$ is in L_i.

CRPQs can further be extended to *compare* paths. For that, we need to name path variables, and choose a class $\mathcal{S}$ of allowed binary

relations on paths. The class CRPQ($\mathcal{S}$) consists of all formulas of the form:

$$\varphi(\bar{x}) \;=\; \exists \bar{y}\,\Big(\bigwedge_{i=1}^{m}(u_i \xrightarrow{\chi_i:L_i} u_i') \;\wedge\; \bigwedge_{(i,j)\in I}\;\bigwedge_{S\in\lambda((i,j))} S(\chi_i,\chi_j)\Big),$$

where $I \subseteq [m]^2$ and $\lambda : I \to 2^{\mathcal{S}}$. We use variables $\chi_1, \ldots, \chi_m$ to denote paths; these are quantified existentially. That is, the semantics of $G \models \varphi(\bar{a})$ is that there is a tuple $\bar{b}$ of nodes and paths ρ_k, for $k \leq m$, between v_k and v_k' (where, as before, v_k, v_k' are elements of $\bar{a}, \bar{b}$ interpreting u_k, u_k') such that $(\kappa(\rho_i), \kappa(\rho_j)) \in S$ whenever $(i,j) \in I$ and $S \in \lambda((i,j))$.

For instance, CRPQ(REG_2) extends CRPQs with the ability to compare pairs of labels of paths with regular relations, and CRPQ($\mathsf{REG}_2 \cup \preceq_{\text{ss}}$) extends the latter with the possibility to compare labels of paths with the subsequence relation.

Example 3.1. The CRPQ($\preceq_{\text{ss}}$) formula

$$\exists y, y'\ \big((x \xrightarrow{\chi:\Sigma^* a} y) \wedge (x \xrightarrow{\chi':\Sigma^* b} y') \wedge \chi \preceq_{\text{ss}} \chi'\big)$$

finds nodes v so that there are two paths starting from v, one ending with an a-edge, whose label is a subsequence of the other one, that ends with a b-edge. □

The evaluation problem For a logic CRPQ($\mathcal{S}$) this is the problem of, given a graph database G, a tuple $\bar{v}$ of nodes, and a formula $\varphi(\bar{x})$ in CRPQ($\mathcal{S}$), determine whether $G \models \varphi(\bar{v})$. This corresponds to the *combined complexity* of evaluation. In the context of databases, one is often interested in *data complexity*, when the typically small formula φ is fixed, and the input consists of the typically large structure $(G, \bar{v})$. Let $\mathcal{C}$ be a complexity class. As usual, we say that the evaluation of CRPQ($\mathcal{S}$) is in $\mathcal{C}$ in data complexity, if the evaluation of each formula in CRPQ($\mathcal{S}$) is in $\mathcal{C}$. The evaluation of CRPQ($\mathcal{S}$) is $\mathcal{C}$-hard in data complexity, if there is a formula in CRPQ($\mathcal{S}$) for which the evaluation is $\mathcal{C}$-hard. Finally, CRPQ($\mathcal{S}$) evaluation is $\mathcal{C}$-complete if it is both in $\mathcal{C}$ and $\mathcal{C}$-hard.

The complexity of evaluation for the logic CRPQ(REG_2) was studied in [5] using standard automata techniques. In particular, this logic is tractable in data complexity.

Theorem 3.1. [5] *The evaluation problem for* CRPQ(REG_2) *is* PSPACE*-complete, and* NP*-complete for* CRPQ*. The evaluation problem for* CRPQ(REG_2) *is* NLOGSPACE*-complete in data complexity.*

On the other hand, determining the complexity of logics of the form CRPQ($\mathcal{S}$), where $\mathcal{S} \subseteq \mathsf{RAT}_2$, is more difficult; in particular, it is equivalent to determining the complexity of GENINT($\mathcal{S}$) for suitable complexity classes. This is stated in the next lemma, which uses techniques from [4].

Lemma 3.2. *Let $\mathcal{C}$ be a complexity class closed under* PSPACE *reductions. If* GENINT($\mathcal{S}$) *is in $\mathcal{C}$, then evaluation of* CRPQ($\mathcal{S}$) *is in $\mathcal{C}$. Moreover, if* GENINT($\mathcal{S}$) *is $\mathcal{C}$-hard, then evaluation of* CRPQ($\mathcal{S}$) *is $\mathcal{C}$-hard.*

Furthermore, again applying techniques from [4] we prove that the data complexity of evaluation of CRPQ($\mathcal{S}$) can be studied in terms of suitable restrictions GENINT($\mathcal{S}$). From now on, we denote by $\text{GENINT}_{I,\Sigma,\lambda}(\mathcal{S})$ the restriction of GENINT($\mathcal{S}$) in which the index set $I \subseteq [m]^2$, the alphabet Σ, and the (finite range) mapping $\lambda : I \to 2^{\mathcal{S}}$ are fixed. The input to this problem consists only of regular expressions L_i, for $i \leq m$, over the fixed alphabet Σ.

Lemma 3.3. *Let $\mathcal{C}$ be a complexity class closed under* NLOGSPACE *reductions.*

1. *If for each $I \subseteq [m]^2$ it is the case that* $\text{GENINT}_I(\mathcal{S})$ *is in $\mathcal{C}$, then evaluation of* CRPQ($\mathcal{S}$) *is in $\mathcal{C}$ in data complexity.*
2. *If there is an index set $I \subseteq [m]^2$, a finite alphabet Σ, and a mapping $\lambda : I \to 2^{\mathcal{S}}$, such that* $\text{GENINT}_{I,\Sigma,\lambda}(\mathcal{S})$ *is $\mathcal{C}$-hard, then evaluation of* CRPQ($\mathcal{S}$) *is $\mathcal{C}$-hard in data complexity.*

Applying these two lemmas, together with Theorems 2.1 and 2.2, we can find complexity bounds for the evaluation for some important graph logics. This is summarized in the next two corollaries. The first one talks about logics of the form CRPQ($\mathsf{REG}_2 \cup \preceq$), for $\preceq$ one of $\preceq_{\text{sw}}$, $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$.

Corollary 3.4. [4]

1. *Let $\mathcal{S} = (\mathsf{REG}_2 \cup \preceq)$, for $\preceq$ one of $\preceq_{\text{sw}}$ or $\preceq_{\text{suff}}$. There is a* CRPQ($\mathcal{S}$) *formula φ such that the evaluation problem for φ is undecidable.*
2. *The evaluation problem for* CRPQ($\mathsf{REG}_2 \cup \preceq_{\text{ss}}$) *is decidable, but non-elementary even in data complexity.*

In other words, these logics are completely impractical, since the evaluation problem for them is either undecidable or very expensive in data complexity. Notice that the upper bound for CRPQ($\mathsf{REG}_2 \cup \preceq_{\text{ss}}$) follows directly from Lemma 3.2 and Theorem 2.1. On the other hand, the lower bounds follow from Lemma 3.3 and Theorem 2.1. This is because ($\mathsf{REG}_2 \cap_\Sigma \preceq$), for $\preceq$ one of $\preceq_{\text{sw}}$, $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$, is of the form $\text{GENINT}_{I,\Sigma',\lambda}(\mathsf{REG}_2 \cup \preceq)$, for some I, Σ' and λ [4].

The next corollary deals with logics of the form CRPQ($\preceq_{\text{ss}}$) and CRPQ($\preceq_{\text{suff}}$). It is shown that this restriction reduces dramatically the complexity of evaluation. The proof follows directly from Theorem 2.2 and Lemmas 3.2 and 3.3.

Corollary 3.5. [4] *The evaluation problem for* CRPQ($\preceq_{\text{ss}}$) *and* CRPQ($\preceq_{\text{suff}}$) *can be solved in* NEXPTIME*, and in* NP *in data complexity.*

The case of CRPQ($\preceq_{\text{sw}}$) was left open in [4]. Our goal is determining the precise complexity of evaluation for logics of the form CRPQ($\preceq$), for $\preceq$ one of $\preceq_{\text{sw}}$, $\preceq_{\text{suff}}$ or $\preceq_{\text{ss}}$. We do this in the following sections. Since the generalized intersection problem is of independent interest and allows for a clean presentation, we concentrate on studying the complexity of such problem and then transfer the results to the complexity of evaluation for the logics using Lemmas 3.2 and 3.3.

4. The logics CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$)

The relations $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ have an important property in common: they can be defined by word equations. This observation implies that GENINT($\preceq_{\text{suff}}$) and GENINT($\preceq_{\text{sw}}$) can be reduced in polynomial time to the problem of solving word equations with regular constraints, which is in PSPACE. It follows that evaluation of both CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$) is PSPACE-complete. This is explained in Section 4.1.

In order to understand the data complexity of these logics, we need to dig deeper in the classes of word equations they can be reduced to. In the case of CRPQ($\preceq_{\text{suff}}$) such class allows for efficient solvability, and as a consequence the data complexity of CRPQ($\preceq_{\text{suff}}$) is tractable (see Section 4.2). In the case of CRPQ($\preceq_{\text{sw}}$) such good properties are not preserved, and in fact we prove that the data complexity of CRPQ($\preceq_{\text{sw}}$) continues being PSPACE-complete (see Section 4.3).

4.1 Word equations and the generalized intersection problem

Let X be a countably infinite set of variables. A *word equation* over Σ [19] is an expression e of the form $\varphi = \psi$, where both φ and ψ are words over $\Sigma \cup X$. A *solution* for e is a mapping h from the variables that appear in e to Σ^* that unifies both sides of the equation, i.e., $h(\varphi) = h(\psi)$, assuming that $h(a) = a$ for each

symbol $a \in \Sigma$.[1] A *word equation with regular constraints* [12] is a tuple (e, ν), where e is a word equation and ν is a mapping that associates an NFA L_x over Σ with each variable x that appears in e. A solution for (e, ν) is a solution h for e over Σ that satisfies $h(x) \in L_x$, for each $x \in X$ that is mentioned in e.

A deep result due to Makanin states that the existence of solutions problem for word equations is decidable [19]. By applying somewhat different techniques, Plandowski proved that the problem is in PSPACE [20]. Then Gutiérrez et al. developed an extension of those techniqes to prove that the latter holds even for word equations with regular constraints:

Theorem 4.1. [12] *The existence of solutions problem for word equations with regular constraints is* PSPACE*-complete.*

Word equations can be used to define relations on words (see, e.g., [15, 18]). Formally, an n-ary relation R over Σ^* is *definable by word equations*, if there is a word equation e over Σ and variables $x_1, \ldots, x_n$ appearing in e such that:

$$R \;=\; \{(h(x_1), \ldots, h(x_n)) \mid h \text{ is a solution for } e\}.$$

We denote by EQ the set of binary relations that are definable by word equations. We assume that each such binary relation is specified as a word equation that defines it.

Example 4.1. Both $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ are in EQ. In fact, $\preceq_{\text{suff}}$ is the set of pairs (x, y) that satisfy the word equation $y = zx$, and $\preceq_{\text{sw}}$ is the set of pairs (x, y) that satisfy the word equation $y = zxw$. On the other hand, $\preceq_{\text{ss}}$ is not in EQ [14]. □

The fact that relations in EQ can be defined with word equations implies that the problem GENINT(EQ) boils down to the problem of solving word equations with regular constraints. We explain this with an example.

Example 4.2. Consider the index set

$$I_\diamond \;=\; \{(1,2), (1,3), (2,4), (3,4)\}$$

and an instance of $\text{GENINT}_{I_\diamond}(\preceq_{\text{suff}})$ of the form

$$(L_1, L_2, L_3, L_4, \lambda),$$

where the L_i's are NFAs over Σ, for $1 \leq i \leq 4$, and $\lambda((i,j)) = \{\preceq^{\Sigma}_{\text{suff}}\}$, for each $(i,j) \in I_\diamond$.

We are thus looking for the existence of words w_1, w_2, w_3, w_4 over Σ such that for each pair $(i,j) \in I_\diamond$ the following holds: $w_i \in L_i$, $w_j \in L_j$, and $w_i \preceq_{\text{suff}} w_j$. In other words, we are looking for pairs (w_i, w_j) that satisfy the equation with regular constraints

$$(w_j = u_{ij} w_i,\ \nu_{ij}),$$

where $\nu_{ij}(w_j) = L_j$, $\nu_{ij}(w_i) = L_i$, and $\nu_{ij}(u_{ij}) = \Sigma^*$.

Putting all this together we can prove that $(L_1, L_2, L_3, L_4, \lambda)$ is in $\text{GENINT}_{I_\diamond}(\preceq_{\text{suff}})$ iff the word equation with regular constraints (e, ν) has a solution, where e and ν are as follows: (1) e is the word equation

$$w_2 \# w_3 \# w_4 \# w_4 \;=\; u_{12} w_1 \# u_{13} w_1 \# u_{24} w_2 \# u_{34} w_3,$$

where $\#$ is a symbol not in Σ, and (2) $\nu(w_i) = L_i$, for each $1 \leq i \leq 4$, and $\nu(u_{i,j}) = \Sigma^*$, for each $(i,j) \in I_\diamond$. Clearly, (e, ν) can be constructed in logarithmic space (LOGSPACE) from $\mathcal{S}$. Notice that e only uses variables and the symbol $\#$, and its form depends exclusively on I and λ. □

Assume that $\Sigma_\#$ is the extension of finite alphabet Σ with a fresh symbol $\#$. Generalizing from the idea presented in the previous example, we can prove the following proposition:

[1] We assume, as usual, that if $\varphi = a_1 \ldots a_n$ then $h(\varphi) = h(a_1) \ldots h(a_n)$.

Proposition 4.2. *There is a* LOGSPACE *translation that, given NFAs $L_1, \ldots, L_m$ over Σ, an index set $I \subseteq [m]^2$, and a (finite range) mapping $\lambda : I \to 2^{\mathsf{EQ}}$, constructs a word equation with regular constraints (e, ν) over $\Sigma_\#$ such that $(L_1, \ldots, L_m, , I, \lambda) \in$ GENINT(EQ) iff (e, ν) has a solution.*

Also, the form of e depends exclusively on I and λ.

As a corollary to Proposition 4.2 and Theorem 4.1, we obtain that GENINT(EQ), and, thus, $\text{GENINT}(\preceq_{\text{suff}})$ and $\text{GENINT}(\preceq_{\text{sw}})$, are in PSPACE. It follows that the three problems are complete for this class. This is because the problem of checking for nonemptiness the language defined by the intersection of regular expressions $L_1, \ldots, L_m$, which is known to be PSPACE-hard [16], can be efficiently reduced to $\text{GENINT}(\preceq)$, for $\preceq$ one of $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$. This reduction can be carried out even in the restricted case in which the index set $I \subseteq [m]^2$ is *acyclic* [4]; that is, when the undirected graph defined by I over $[m]$ is acyclic. Summing up:

Corollary 4.3. *The problem* GENINT(EQ) *is in* PSPACE*.*

In particular, the problems $\text{GENINT}(\preceq_{\text{suff}})$ *and* $\text{GENINT}(\preceq_{\text{sw}})$ *are* PSPACE*-complete. The lower bound holds even in the case in which the index set I is acyclic.*

Complexity of CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$) We can now apply Lemma 3.2, and make use of the results in Corollary 4.3, to determine the precise complexity of evaluation for the logics CRPQ(EQ), CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$):

Theorem 4.4. *Evaluation for* CRPQ(EQ) *is in* PSPACE*.*

In particular, the evaluation problem for CRPQ($\preceq$)*, when $\preceq$ is either $\preceq_{\text{suff}}$ or $\preceq_{\text{sw}}$, is* PSPACE*-complete. This holds even for formulas of the form $\exists \bar{y}\, (\bigwedge_{i=1}^{m} (u_i \xrightarrow{\chi_i : L_i} u'_i) \wedge \bigwedge_{(i,j) \in I} \chi_i \preceq \chi_j)$ in which I is acyclic.*

Our next goal is to determine whether any of these logics is efficient in data complexity. We start with CRPQ($\preceq_{\text{suff}}$).

4.2 The data complexity of CRPQ($\preceq_{\text{suff}}$)

We prove here that evaluation of CRPQ($\preceq_{\text{suff}}$) is in NLOGSPACE in data complexity. From Lemma 3.3 it is sufficient to show that $\text{GENINT}_I(\preceq_{\text{suff}})$ is in NLOGSPACE, for each $I \subseteq [m]^2$. We explain below how to prove this.

Let $I \subseteq [m]^2$ be an index set and consider an instance of $\text{GENINT}_I(\preceq_{\text{suff}})$ of the form $(L_1, \ldots, L_m, \lambda)$, for $L_1, \ldots, L_m$ NFAs over Σ and $\lambda : I \to 2^{\{\preceq_{\text{suff}}\}}$. We can assume w.l.og. that $\lambda((i,j)) = \{\preceq^{\Sigma}_{\text{suff}}\}$, for each $(i,j) \in I$. Applying Proposition 4.2 we can construct in LOGSPACE a word equation with regular constraints (e, ν), such that (i) the form of e depends exclusively on I (since λ is uniquely determined by I in this case), and thus it is fixed, and (ii) $(L_1, \ldots, L_m, \lambda)$ is in $\text{GENINT}_I(S)$ iff (e, ν) has a solution over $\Sigma_\#$. From now on we denote such word equation e by $e(I)$, in order to make explicit its dependence on I only.

From the previous remarks we obtain the following: For each index set $I \subseteq [m]^2$, the problem $\text{GENINT}_I(S)$ reduces in LOGSPACE to the problem of solving the *fixed* word equation $e(I)$ under regular constraints. Notice that the alphabet Σ is not fixed in this case, but only the pattern described by $e(I)$.

As mentioned before, it was proved in [12] that checking the existence of solutions for word equations with regular constraints (e, ν) is complete for PSPACE. On the other hand, the algorithm provided in [12] does not yield better bounds when the word equation e is fixed (in fact, we prove later that this restriction remains hard for PSPACE). Here we study such problem but restricted to word equations of the form $e(I)$, i.e., word equations obtained by applying the translation of Proposition 4.2 to $\text{GENINT}_I(\preceq_{\text{suff}})$. We prove that under such restriction the problem becomes tractable. In order to do this we prove a stronger result. We first identify a

class $\mathcal{E}_{\text{fin}}$ of word equations such that solving each fixed equation $e \in \mathcal{E}_{\text{fin}}$ under regular constraints can be done in NLOGSPACE, and then prove that each equation of the form $e(I)$ is in $\mathcal{E}_{\text{fin}}$. The class $\mathcal{E}_{\text{fin}}$ consists of those word equations that only admit a finite number of *principal* solutions. We formally define this class below and establish its good behavior in our context.

Finite number of principal solutions For the sake of convenience, we assume from now on that word equations e are simply expressions of the form $\varphi = \psi$, where φ and ψ are words that consist of variables and constants. We do not assume as before that the alphabet Σ, where solutions for e are to be searched, is part of the definition of e. In fact, we freely interpret e over alphabets Σ that extend the set of constants that are mentioned in the equation.

Let e be a word equation and h_1, h_2 two solutions for e over Σ_1 and Σ_2, respectively. Then h_1 *divides* h_2 if there is a *continuous morphism* $\alpha : \Sigma_1^* \to \Sigma_2^*$ such that $h_2 = \alpha \circ h_1$. Recall that α is a morphism if (i) $\alpha(\varepsilon) = \varepsilon$, and (ii) for each $w \in \Sigma^*$ such that $w = a_1 \dots a_n$, it is the case that $\alpha(w) = \alpha(a_1) \dots \alpha(a_n)$. The morphism α is continuous if $\alpha(a) \neq \varepsilon$, for each $a \in \Sigma$. A solution h for e is *principal* [1] when it is divided by no other but itself (up to isomorphism). It is known that each word equation that has a solution has a principal solution [18]. We denote by $\mathcal{E}_{\text{fin}}$ the class of word equations e with only a finite number of principal solutions.

Example 4.3. The word equation $x = yz$ is in $\mathcal{E}_{\text{fin}}$. In fact, its only principal solution h is the one that satisfies $h(x) = ab$, $h(y) = a$ and $h(z) = b$. On the other hand, the equation $xy = yx$ does not belong to $\mathcal{E}_{\text{fin}}$; its principal solutions are all solutions of the form $h_{m,n}$, for $m, n > 0$ relatively primes, where $h_{m,n}(x) = a^m$ and $h_{m,n}(y) = a^n$. □

Next we establish the good behavior of $\mathcal{E}_{\text{fin}}$ in our context. We denote by $\mathsf{Weq}(e)$ the problem of evaluating the *fixed* word equation e under regular constraints. Formally, this takes as input an alphabet Σ that extends the set of constants that are mentioned in e, and a mapping ν that associates an NFA L_x over Σ with each variable x that is mentioned in e, and the question is whether (e, ν) has a solution.

Theorem 4.5. $\mathsf{Weq}(e)$ *is in* NLOGSPACE*, for each* $e \in \mathcal{E}_{\text{fin}}$.

Proof (idea): Consider an input to $\mathsf{Weq}(e)$ that consists of a finite alphabet Σ and a mapping ν that associates an NFA over Σ with each variable that is mentioned in e. We start by computing the (finite) set of principal solutions for e using Lentin's algorithm [17]. This can be done in constant time since e is fixed. It can be proved that (e, ν) has a solution over Σ iff there is a principal solution h for e for which the procedure we describe below does not fail.

Let Y and Σ_0 be the set of variables and constants mentioned in e, respectively. Assume that the principal solution h is a mapping from $Y \cup \Sigma_0$ to the set of words over a finite alphabet Δ. We can assume w.l.o.g that $\Sigma \cap \Delta = \Sigma_0$. From h we try to construct a solution for (e, ν) over Σ, by searching for a morphism $\varphi : \Delta^* \to \Sigma^*$ such that (i) $\varphi(a) = a$, for each $a \in \Sigma_0$, and (ii) $\varphi(h(y)) \in \nu(y)$, for each variable $y \in Y$. We explain how to do this with an example, since the full construction is a bit cumbersome.

Assume that $Y = \{x, y, z\}$, $\Delta = \{a, b, c\}$, and h satisifies the following: $h(x) = abc$, $h(y) = b$ and $h(z) = ac$. Suppose that we can guess states q_0, q_1, q_2, q_3 in $\nu(x)$, states r_0, r_1 in $\nu(y)$, and states s_0, s_1, s_2 in $\nu(z)$, such that q_0, r_0, s_0 are initial states, q_3, r_1, s_2 are final states, and the following holds:

- State (q_1, s_1) is reachable from (q_0, s_0) reading word w_1 over the NFA $\nu(x) \times \nu(z)$.
- State (q_2, r_1) is reachable from (q_1, r_0) reading word w_2 over the NFA $\nu(x) \times \nu(y)$.
- State (q_3, s_2) is reachable from (q_2, s_1) reading word w_3 over the NFA $\nu(x) \times \nu(z)$.

Then (e, ν) has a solution h' over Σ given as: $h'(x) = w_1 w_2 w_3$, $h'(y) = w_2$ and $h'(z) = w_1 w_3$. If, on the other hand, it is not possible to find such states, then we declare that h fails.

It is not hard to see how this idea can be extended to the general case. Notice that the number of states to be guessed is bounded by the maximum length of a word of the form $h(y)$, for $y \in Y$, and thus it is fixed. Each such state can be represented using logarithmic space. Furthermore, the number of variables in Y is fixed, and, therefore, each one of the reachability tasks can be carried out in NLOGSPACE using standard "on-the-fly" techniques. Thus, the procedure can be performed in NLOGSPACE for each principal solution of e. Since the number of such solutions is fixed, we can determine in NLOGSPACE whether the equation (e, ν) has a solution over Σ. □

Principal solutions and $\preceq_{\text{suff}}$ From our previous remarks, proving that $\text{GENINT}_I(\preceq_{\text{suff}})$ is in NLOGSPACE, for each $I \subseteq [m]^2$, amounts to proving that $\mathsf{Weq}(e(I))$ is in NLOGSPACE for each such I. Due to Theorem 4.5, it is sufficient to prove that $e(I) \in \mathcal{E}_{\text{fin}}$, for every $I \subseteq [m]^2$. This is proved below.

Lemma 4.6. *The word equation* $e(I)$ *is in* $\mathcal{E}_{\text{fin}}$*, for each* $I \subseteq [m]^2$.

Proof (idea): Consider an arbitrary word equation of the form $e(I)$, for $I \subseteq [m]^2$. We prove that $e(I)$ has a finite number of principal solutions. We start by defining *systems* of word equations. These are sets of the form $\{E_1, \dots, E_n\}$, where each E_i is an *extended* word equation. The latter are expressions of the form $\varphi_1 = \dots = \varphi_\ell$. In general, the φ_i's are words over $\Sigma \cup X$, but in our case we can restrict them to be simply words over X (i.e., the φ_i's are composed exclusively by variables). A solution for the extended equation $\varphi_1 = \dots = \varphi_\ell$ over Σ is a mapping h from X to Σ^* such that $h(\varphi_1) = \dots = h(\varphi_\ell)$. A solution h for the system $\{E_1, \dots, E_n\}$ is a mapping from X to Σ^* such that h is a solution for each E_i, $1 \leq i \leq n$. Principal solutions of systems of word equations are defined exactly in the same way than for word equations. While systems of word equations can be reduced to a single word equation, they are convenient for our proof.

We first note that $e(I)$ can be "reduced" to a system of word equations of a special form, which we call *suffix-like*. Formally, a system $\{E_1, \dots, E_n\}$ is suffix-like if the following holds:

1. For each $1 \leq i \leq n$, if E_i is of the form $\varphi_1 = \dots = \varphi_\ell$, then no φ_j contains repeated variables, for $1 \leq j \leq \ell$.
2. For each $1 \leq i, j \leq n$, if E_i is of the form $\varphi_1 = \dots = \varphi_\ell$ and E_j is of the form $\psi_1 = \dots = \psi_{\ell'}$, then the following holds for each $1 \leq k \leq \ell$, $1 \leq k' \leq \ell'$, and variable $x \in X$ that appears in both φ_k and $\psi_{k'}$: Assume $\varphi_k = pxs$ and $\psi_{k'} = p'xs'$. Then $s = s'$. In other words, a variable completely determines the suffixes of the components of the extended equations of the system where it appears.

Then we have the following:

Claim 4.7. *There is a suffix-like system* $\mathcal{E}(I)$ *of word equations with the same number of principal solutions than* $e(I)$.

In order to prove Claim 4.7, we first assume w.l.o.g. that I is a DAG, and then construct $\mathcal{E}(I)$ from $e(I)$ by applying variable substitutions following a topological order of I.

In view of Claim 4.7, to prove the theorem it is sufficient to show that suffix-like systems of word equations have a finite number of principal solutions. To do that, we apply Lentin's algorithm [17], a.k.a. *pig-pug*, that generates the set of principal solutions

of a system of word equations, and then show that such procedure always finishes when the system is suffix-like.

The pig-pug procedure iteratively converts a system of word equations into another system of word equations by nondeterministically guessing the lengths of the words associated with the variables. It can be shown that when the original system is suffix-like, a pig-pug application converts it into another system that is suffix-like. This is proved by induction on the number of extended word equations in the system.

Using the latter, it can be proved that successive applications of the pig-pug procedure over a suffix-like system of word equations always lead to systems of word equations in which no variable is repeated (i.e., in addition to condition (1) we have that no two different extended equations in the system share a variable, and for each extended equation in the system of the form $\varphi_1 = \cdots = \varphi_\ell$ it is the case that no distinct φ_i's share a variable). It is known that this kind of systems only have a finite number of principal solutions, which proves our result. □

From Lemma 4.6 and Theorem 4.5 we obtain:

Corollary 4.8. *The problem* $\text{GENINT}_I(\preceq_{\text{suff}})$ *is in* NLOGSPACE*, for each* $I \subseteq [m]^2$.

From Lemma 3.3 we now obtain our desired result: evaluation of $\text{CRPQ}(\preceq_{\text{suff}})$ is tractable in data complexity. An NLOGSPACE lower bound holds in this case since CRPQs are already hard for NLOGSPACE in data complexity.

Theorem 4.9. *The evaluation problem for* $\text{CRPQ}(\preceq_{\text{suff}})$ *is complete for* NLOGSPACE *in data complexity.*

We study next the data complexity of $\text{CRPQ}(\preceq_{\text{sw}})$.

4.3 The data complexity of CRPQ($\preceq_{\text{sw}}$)

It is not hard to prove that word equations obtained by applying Proposition 4.2 to $\text{GENINT}_I(\preceq_{\text{sw}})$ are not in $\mathcal{E}_{\text{fin}}$, and, thus, we cannot study the data complexity of $\text{CRPQ}(\preceq_{\text{sw}})$ along the lines developed in the previous section. This is consistent with the results we obtain below. In fact, we show that while the combined complexity of evaluation for $\text{CRPQ}(\preceq_{\text{sw}})$ and $\text{CRPQ}(\preceq_{\text{suff}})$ is the same (PSPACE-complete), the situation is radically different in terms of data complexity: Theorem 4.9 states that evaluation of $\text{CRPQ}(\preceq_{\text{suff}})$ is tractable in data complexity, but we prove next that the data complexity of $\text{CRPQ}(\preceq_{\text{sw}})$ remains PSPACE-complete.

Theorem 4.10. *The evaluation problem for* $\text{CRPQ}(\preceq_{\text{sw}})$ *is* PSPACE*-complete in data complexity.*

The upper bound follows from Theorem 4.4. Due to Lemma 3.3, for hardness we only need to prove this:

Proposition 4.11. *There is an index set* $I \subseteq [m]^2$*, a finite alphabet* Σ*, and a mapping* $\lambda : I \to 2^{\{\preceq_{\text{sw}}\}}$*, such that* $\text{GENINT}_{I,\Sigma,\lambda}(\preceq_{\text{sw}})$ *is* PSPACE*-hard.*

Proof. We use the index set $I_\diamond = \{(1,2),(1,3),(2,4),(3,4)\}$ from Example 4.2. It follows from [16] that there is a finite alphabet Σ such that the following problem is PSPACE-complete: Given regular expressions $L_1, \ldots, L_m$ over Σ such that no L_i accepts the empty word ε, check whether $\bigcap_{1\leq i\leq m} L_i$ is nonempty. We show that this problem can be reduced in polynomial time to $\text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$, where $\Sigma_\$$ denotes the extension of Σ with the fresh symbol \$, and $\lambda_\diamond : I_\diamond \to 2^{\{\preceq_{\text{sw}}\}}$ is such that $\lambda_\diamond((i,j)) = \{\preceq_{\text{sw}}^{\Sigma_\$}\}$, for each $(i,j) \in I_\diamond$.

Given regular expressions $L_1, \ldots, L_m$ over Σ such that no L_i accepts the empty word, we construct an instance (R_1, R_2, R_3, R_4) of $\text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$ such that the R_i's are NFAs over Σ that define the following languages:

1. $R_1 := \$L_1\$L_2\$\ldots\$L_m\$$, i.e., R_1 accepts words of the form $\$w_1\$w_2\$\ldots\$w_m\$$, where each w_i is a (nonempty) word in the language L_i.
2. $R_2 := \Sigma^+(\$\Sigma^*)^m\$$, i.e., R_2 accepts words of the form $w_0\$w_1\$w_2\$\ldots\$w_m\$$, where $w_0, w_1, \ldots, w_m$ are words over Σ and w_0 is required to be nonempty.
3. $R_3 := (\$\Sigma^*)^m\$\Sigma^+$, i.e., R_3 accepts words of the form $\$w_1\$w_2\$\ldots\$w_m\$w_{m+1}$, where $w_1, \ldots, w_m, w_{m+1}$ are words over Σ and w_{m+1} is required to be nonempty.
4. $R_4 := (\$\Sigma^*)^{m+1}\$$, i.e. R_4 accepts words of the form
$$\$w_1\$w_2\$\ldots\$w_m\$w_{m+1}\$,$$
where $w_1, \ldots, w_m, w_{m+1}$ are words over Σ.

Clearly, (R_1, R_2, R_3, R_4) can be constructed in polynomial time from the L_i's.

We claim that $\bigcap_{1\leq i\leq m} L_i$ is nonempty iff (R_1, R_2, R_3, R_4) belongs to $\text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$. Assume first there is word $w \in \bigcap_{1\leq i\leq m} L_i$. Then $w \neq \varepsilon$. Consider the word $\bar{w}$ over $\Sigma_\$$ defined as $\bar{w} := (\$w)^m\$$, and let $w_1 := \bar{w}$, $w_2 := w\bar{w}$, $w_3 := \bar{w}w$ and $w_4 := \$w\bar{w}$. It is clear that $w_i \in R_i$, for $1 \leq i \leq 4$. Furthermore, it is easy to see that $w_i \preceq_{\text{sw}} w_j$, for each $(i,j) \in I_\diamond$. We conclude that (R_1, R_2, R_3, R_4) belongs to $\text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$.

Assume on the other hand that there are words w_1, w_2, w_3 and w_4 such that $w_i \in R_i$, for each $1 \leq i \leq 4$, and $w_i \preceq_{\text{sw}} w_j$, for each $(i,j) \in I_\diamond$. Since $w_1 \in R_1$ it must be the case that w_1 is of the form $\$s_1\$s_2\$\ldots\$s_m\$$, where each s_i is a (nonempty) word in L_i. We prove next that $s_1 = s_j$, for each $2 \leq j \leq m$, and thus that $s_1 \in \bigcap_{1\leq i\leq m} L_i$.

Since $w_1 \preceq_{\text{sw}} w_2$ and $w_2 \in R_2$, the structure of R_2 implies that w_2 must be of the form $s_0\$s_1\$s_2\$\ldots\$s_m\$$, for some nonempty word s_0 over Σ. Similarly, w_3 must be of the form $\$s_1\$s_2\$\ldots\$s_m\$s_{m+1}$, for some nonempty word s_{m+1} over Σ. Now, since $w_2 \preceq_{\text{sw}} w_4$ and $w_4 \in R_4$, the structure of R_4 implies that w_4 must be of the form $\$s_0\$s_1\$s_2\$\ldots\$s_m\$$. Similarly, since $w_3 \preceq_{\text{sw}} w_4$ and $w_4 \in R_4$, the structure of R_4 implies that w_4 must be of the form $\$s_1\$s_2\$\ldots\$s_m\$s_{m+1}\$$. But the only way in which this can happen is if $s_0 = s_1 = s_2 = \cdots = s_m = s_{m+1}$. This concludes the proof. □

An important corollary to the proof of Proposition 4.11 is that there exists a fixed word equation e such that solving e under regular constraints is PSPACE-complete.

Corollary 4.12. *There is a word equation* e *and a finite alphabet* Σ *such that the problem* $\mathsf{Weq}(e)$ *is* PSPACE*-complete, even if restricted to inputs over* Σ*.*

Proof. The word equation e is of the form:

$$w_2\#w_3\#w_4\#w_4 = u_{12}w_1u'_{12}\#u_{13}w_1u'_{13}\#u_{24}w_2u'_{24}\#u_{34}w_3u'_{34},$$

where $\#$ is a constant and all other symbols are variables. In fact, it follows from the proof of Proposition 4.11 that there is a finite alphabet Σ such that $\text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$ is PSPACE-complete. This problem can be reduced in polynomial time to $\mathsf{Weq}(e)$, even if restricted to inputs over $\Sigma_\$ \cup \{\#\}$. □

The logic $\text{CRPQ}(\preceq_{\text{pref}} \cup \preceq_{\text{suff}})$ It is not hard to see that each formula in $\text{CRPQ}(\preceq_{\text{sw}})$ can be expressed in the logic $\text{CRPQ}(\preceq_{\text{pref}} \cup \preceq_{\text{suff}})$. This is because an atom of the form $\chi \preceq_{\text{sw}} \chi'$ can be rewritten as the formula $\exists\chi''(\chi'' \preceq_{\text{pref}} \chi' \wedge \chi \preceq_{\text{suff}} \chi'')$ in $\text{CRPQ}(\preceq_{\text{pref}} \cup \preceq_{\text{suff}})$. From Theorem 4.10 we obtain the following:

Proposition 4.13. *Evaluation of* CRPQ($\preceq_{\text{pref}} \cup \preceq_{\text{suff}}$) *is* PSPACE*-hard in data complexity.*

This result shows how fragile tractability in data complexity is in this context. In fact, extending CRPQs with either $\preceq_{\text{pref}}$ or $\preceq_{\text{suff}}$ preserves tractability in data complexity; in the first case this follows from Theorem 3.1 (since $\preceq_{\text{pref}}$ is in REG_2), and in the second one from Theorem 4.9. But adding both relations at the same time destroys such tractability.

5. The logic CRPQ($\preceq_{\text{ss}}$)

We now study the complexity of evaluation for CRPQ($\preceq_{\text{ss}}$). We prove that the NP and NEXPTIME upper bounds for the data and combined complexity of the logic, respectively, that were obtained in [4] (see Corollary 3.5), cannot be improved further.

Recall that $\preceq_{\text{ss}}$ cannot be defined by word equations, and, therefore, we cannot apply for this logic the techniques used in the previous chapter. Instead, we apply different word combinatorics techniques based on the notion of shuffling to obtain a matching NP lower bound for the data complexity of CRPQ($\preceq_{\text{ss}}$). Using a suitable succinct version of the longest common subsequence problem we also provide a matching NEXPTIME lower bound for its combined complexity.

5.1 The data complexity of CRPQ($\preceq_{\text{ss}}$)

We prove here that evaluation of CRPQ($\preceq_{\text{ss}}$) is NP-complete in data complexity. In order to obtain the lower bound it is convenient to use a reduction from the interesting problem of *unshuffling a square*, which has been recently proved to be NP-complete. We define this problem below.

Let u and v be words over Σ. A *shuffle* of u and v is formed by interleaving the symbols of u and v, keeping the symbols of each word in order. Formally, w is a shuffle of u and v if there are (possibly empty) words u_i and v_i, for $1 \leq i \leq k$, such that $u = u_1 \dots u_k$, $v = v_1 \dots v_k$, and $w = u_1 v_1 \dots u_k v_k$.

A word w is a *square* if there is a word u such that w is a shuffle of u with itself. The problem SQUARE takes as input a word w over Σ, and asks whether w is a square. We write SQUARE_Σ if Σ is fixed. It has been independently proved in [8, 21] that there is a finite alphabet Σ such that SQUARE_Σ is NP-complete. We use a reduction from this problem to pinpoint the data complexity of evaluation for CRPQ($\preceq_{\text{ss}}$).

Theorem 5.1. *The evaluation problem for* CRPQ($\preceq_{\text{ss}}$) *is* NP*-complete in data complexity.*

The upper bound follows from Corollary 3.5. We prove hardness next. Due to Lemma 3.3, we only need to prove the following:

Proposition 5.2. *There is an index set $I \subseteq [m]^2$, a finite alphabet Σ, an and a mapping $\lambda : I \to 2^{\{\preceq_{\text{ss}}\}}$, such that the problem* $\text{GENINT}_{I,\Sigma,\lambda}(\preceq_{\text{ss}})$ *is* NP*-hard.*

Proof. We again use the index set

$$I_\diamond \;=\; \{(1,2),(1,3),(2,4),(3,4)\}$$

from Example 4.2. We know that there is a finite alphabet Σ such that SQUARE_Σ is NP-complete. We show that this problem can be reduced in polynomial time to $\text{GENINT}_{I_\diamond,\Sigma_{\mathsf{b},\mathsf{r}},\lambda_\diamond}(\preceq_{\text{ss}})$, where $\Sigma_{\mathsf{b},\mathsf{r}}$ is the extension of Σ with fresh symbols b and r, and $\lambda_\diamond : I_\diamond \to 2^{\{\preceq_{\text{ss}}\}}$ is such that $\lambda((i,j)) = \preceq_{\text{ss}}^{\Sigma_{\mathsf{b},\mathsf{r}}}$, for each $(i,j) \in I_\diamond$.

Let w be a word in Σ^*. We assume w.l.o.g. that the length of w is $2n$, for $n \geq 0$ (otherwise we immediately declare that w is not a square). Assume $w = a_1 \dots a_{2n}$, where each $a_i \in \Sigma$. We construct an instance (R_1, R_2, R_3, R_4) of $\text{GENINT}_{I_\diamond,\Sigma_{\mathsf{b},\mathsf{r}},\lambda_\diamond}(\preceq_{\text{ss}})$ such that the R_i's are NFAs over Σ that define the following languages:

1. $R_1 := \Sigma^n$, i.e., R_1 are all words of length n over Σ.
2. $R_2 := (\mathsf{b}\,\Sigma\,\mathsf{b})^n$, i.e., R_2 consists of all words of the form $\mathsf{b}\,c_1\,\mathsf{b} \dots \mathsf{b}\,c_n\,\mathsf{b}$ such that $c_1 \dots c_n \in \Sigma^*$.
3. $R_3 := (\mathsf{r}\,\Sigma\,\mathsf{r})^n$, i.e., R_3 consists of all words of the form $\mathsf{r}\,c_1\,\mathsf{r} \dots \mathsf{r}\,c_n\,\mathsf{r}$ such that $c_1 \dots c_n \in \Sigma^*$.
4. $R_4 := (\mathsf{b}\,a_1\,\mathsf{b} \cup \mathsf{r}\,a_1\,\mathsf{r}) \dots (\mathsf{b}\,a_{2n}\,\mathsf{b} \cup \mathsf{r}\,a_{2n}\,\mathsf{r})$, i.e., R consists of all words that are obtained from w by replacing each a_i with either $\mathsf{b}\,a_i\,\mathsf{b}$ or $\mathsf{r}\,a_i\,\mathsf{r}$.

Clearly, (R_1, R_2, R_3, R_4) can be constructed in polynomial time from w.

We explain next why $w \in \text{SQUARE}_\Sigma$ iff (R_1, R_2, R_3, R_4) is in $\text{GENINT}_{I_\diamond,\Sigma_{\mathsf{b},\mathsf{r}},\lambda_\diamond}(\preceq_{\text{ss}})$. Intuitively, words u accepted by R_1 are candidates for squaring w. We then take two copies of u, one accepted by R_2 and the other one accepted by R_3, and we distinguish such copies using the special symbols b and r. Finally, a word v accepted by R_4 makes a choice about which copy of u is synchronizing with. Since both copies of u have to be a subsequence of v, we can ensure that w is a square whenever (R_1, R_2, R_3, R_4) is in $\text{GENINT}_{I_\diamond,\Sigma_{\mathsf{b},\mathsf{r}},\lambda_\diamond}(\preceq_{\text{ss}})$. The other direction is even simpler, since we can easily extract a witness for the fact that (R_1, R_2, R_3, R_4) is in $\text{GENINT}_{I_\diamond,\Sigma_{\mathsf{b},\mathsf{r}},\lambda_\diamond}(\preceq_{\text{ss}})$ if we know that w is a square. □

5.2 The combined complexity of CRPQ($\preceq_{\text{ss}}$)

The evaluation problem for CRPQ($\preceq_{\text{ss}}$) is known to be in NEXPTIME. We prove that such bound is tight.

Theorem 5.3. *The evaluation problem for* CRPQ($\preceq_{\text{ss}}$) *is complete for* NEXPTIME.

Due to Lemma 3.2, for hardness we only need to prove the following:

Proposition 5.4. $\text{GENINT}(\preceq_{\text{ss}})$ *is* NEXPTIME*-hard.*

We start by proving that the following succinct version of the longest common subsequence problem, which we call SUCCINCT-LCS, is NEXPTIME-hard: Given a finite alphabet Σ, regular expressions $L_1, \dots, L_m$ over the alphabet that extends Σ with a fresh symbol \$, and binary integers $k, p \geq 0$, we want to know whether there is a word $w \in \bigcap_{1 \leq i \leq m} L_i$ of the form $\$u_1\$u_2\$ \dots \$u_p\$$, where the u_i's are words over Σ, and a word u over Σ with exactly k symbols, such that $u \preceq_{\text{ss}} u_i$ for each $1 \leq i \leq p$.

Proposition 5.5. SUCCINCT-LCS *is* NEXPTIME*-hard.*

Proof (idea): The original LCS problem takes as input words $w_1, ..., w_p$ over Σ and an integer $\ell \in \mathbb{N}$, and asks whether the w_i's have a common subsequence of length ℓ. It is known that LCS is NP-hard even over a fixed alphabet. In our proof, we will make use of a particular polynomial-time reduction from *independent set* to LCS over a binary alphabet that can be found in [7].

In order to prove that SUCCINCT-LCS is NEXPTIME-hard, we compose three reductions. The first one is the standard reduction from the acceptance problem for a nondeterministic Turing machine M on input x to satisfiability of Cook's formula $\varphi(M, x)$. In our case, M is a nondeterministic machine that works in exponential time, and, thus, $\varphi(M, x)$ is of exponential size. We then use a standard reduction from satisfiability of $\varphi(M, x)$ to the problem of determining if a graph $G(M, x)$ has an independent set of size $k_{M,x} \geq 0$. (In particular, $G(M, x)$ contains a node for each literal in each clause, and there is an edge between nodes q and q' iff q and q' are in the same clause, or the literal represented by q is the negation of the one represented by q'. The size $k_{M,x}$ of the desired independent set corresponds to the number of clauses of $\varphi(M, x)$). Finally, we apply on $(G_M, k_{M,x})$ the reduction to LCS over a binary alphabet that we mentioned in the previous paragraph. In our case, this reduction yields an exponential number of words

$\mathcal{S}$	$\preceq_{\text{suff}}$	$\preceq_{\text{sw}}$	$\preceq_{\text{ss}}$	(REG$_2 \cup \preceq_{\text{suff}}$)	(REG$_2 \cup \preceq_{\text{sw}}$)	(REG$_2 \cup \preceq_{\text{ss}}$)
comb. comp. of CRPQ($\mathcal{S}$)	PSPACE (Thm 4.4)	PSPACE (Thm 4.4)	NEXPTIME (Thm 5.3)	undecidable (Coroll. 5.2 in [4])	undecidable (Coroll. 5.2 in [4])	nonelementary (Coroll. 5.13 in [4])
data comp. of CRPQ($\mathcal{S}$)	NLOGSPACE (Thm 4.9)	PSPACE (Thm 4.10)	NP (Thm 5.1)	undecidable (Coroll. 5.2 in [4])	undecidable (Coroll. 5.2 in [4])	nonelementary (Coroll. 5.13 in [4])

Figure 1: Combined and data complexity of graph logics.

$w_1, \ldots, w_p$, each of exponential size, such that M accepts input x iff $w_1, \ldots, w_p$ share a subsequence of exponential length $\ell \geq 0$.

By looking at the composition of these three reductions, it can be observed that the words $w_1, ..., w_p$ are highly uniform. They are constructed from simple recurring patterns that grow and shrink in a synchronised way. This allows to encode the word $\$w_1\$ \ldots \$w_p\$$ as the unique word accepted by the intersection of polynomially many regular languages $R_1, ..., R_t$ over $\{0, 1, \$\}$. Furtheremore, the R_i's can be constructed in polynomial time from M and x. This encoding uses similar techniques as those used to encode valid sequences of computations performed by a polynomial space Turing machine as intersection of regular expressions [16]. The correctness of the reduction is implied by the composition of the preceding ones. □

We now show that SUCCINCT-LCS problem can be reduced in polynomial time to GENINT($\preceq_{\text{ss}}$), by using techniques developed in the proofs of Theorems 4.10 and 5.1. Consider an input to SUCCINCT-LCS consisting of regular expressions $L_1, ..., L_m$ over the alphabet $\Sigma_\$$ that extends Σ with fresh symbol \$, and binary integers $k, p \geq 0$. Let S_1, S_2, S_3 be the following regular languages over $\Sigma_\$$:

$$S_1 = \{(\$u)^p\$: u \in \Sigma^k\}; \quad S_2 = \bigcap_{1 \leq i \leq m} L_i; \quad S_3 = (\$\Sigma^*)^p\$.$$

It is not hard to see that our input belongs to SUCCINCT-LCS (i.e., there is a word of the form $\$w_1\$ \cdots \$w_p\$$ in S_2, with each w_i being a word over Σ, such that the w_i's have a common subsequence u over Σ of length exactly k) if and only if there are words s_1, s_2, s_3 over $\Sigma_\$$ such that $s_i \in S_i$, for each $i = 1, 2, 3$, and $s_1 \preceq_{\text{ss}} s_2$ and $s_2 \preceq_{\text{ss}} s_3$ (in particular, $s_2 = \$w_1\$ \cdots \$w_p\$$). Although the regular languages S_1, S_2 and S_3 cannot be constructed directly in polynomial time, it is possible to encode each one of them as a generalized intersection problem that can be constructed in polynomial time from our input. We explain this below.

Let us consider $S_1 = \{(\$u)^p\$: u \in \Sigma^k\}$ first. Using standard techniques, it is possible to construct in polynomial time regular expressions $R_1, \ldots, R_t$ over $\Sigma_\$$ such that $\bigcap_{1 \leq i \leq t} R_i = (\$\Sigma^k)^p\$$. In order to force the words over Σ between successive \$ symbols to be equal, we use an index set I_1 constructed as follows: Take the index set $I_\diamond = \{(1,2), (1,3), (2,4), (3,4)\}$ used in the proof of Theorem 4.10, and replace each element $i = 1, 2, 3, 4$ by a directed cycle with $O(t)$ elements. We then define a mapping λ_1 that assigns $\{\preceq_{\text{ss}}\}$ to each pair in I_1. Using the regular expressions $R_1, \ldots, R_t$ and ideas similar to the ones in the proof of Theorem 4.10, it is possible to construct a generalized intersection problem over I_1 and λ_1 such that its solutions encode S_1. Notice, however, that in this case we are using the more loose relation subsequence instead of subword. This causes no trouble, since we already know that words over Σ between succesive \$ symbols are of equal length.

The two other cases are simpler. For $S_2 = \bigcap_{1 \leq i \leq m} L_i$, we just consider the index set $I_2 = \{(1,2), ..., (m-1, m), (m, 1)\}$ and the mapping λ_2 that associates with each edge in I_2 the set $\{\preceq_{\text{ss}}\}$. It is clear then that the solutions to the instance $(L_1, \ldots, L_m, I_2, \lambda_2)$ of GENINT($\preceq_{\text{ss}}$) are precisely the words in S_2. For $S_3 = (\$\Sigma^*)^p\$$, we use the fact that it is possible to construct in polynomial time regular expressions $T_1, ..., T_\ell$ over $\Sigma_\$$ such that $\bigcap_{1 \leq i \leq \ell} T_i = S_3$. In this case we consider the directed cycle $I_3 = \{(1,2), ..., (1,\ell), (\ell, 1)\}$, and the mapping λ_3 that associates the set $\{\preceq_{\text{ss}}\}$ with each edge of I_3. Therefore, solutions to the instance $(T_1, \ldots, T_\ell, I_3, \lambda_3)$ of GENINT($\preceq_{\text{ss}}$) are precisely the words in S_3.

By starting from I_1, I_2, I_3 and $\lambda_1, \lambda_2, \lambda_3$, it is easy to construct in polynomial time an instance $\mathcal{I}$ of the generalized intersection problem such that $\mathcal{I}$ belongs to GENINT($\preceq_{\text{ss}}$) if and only if there are words $s_1 \in S_1, s_2 \in S_2$ and $s_3 \in S_3$ such that $s_1 \preceq_{\text{ss}} s_2$ and $s_2 \preceq_{\text{ss}} s_3$. As we mentioned below, this is equivalent to checking that the original input belongs to SUCCINT-LCS. □

6. Final Remarks

Motivated by applications of graph databases that require comparing labels of paths based on rational relations, we have studied the complexity of evaluation for logics that extend CRPQs with practical relations such as suffix, subword and subsequence. This complements previous results from [4], which established the prohibitive complexity of evaluation for logics that allow, in addition, path comparisons based on arbitrary regular relations. Figure 1 summarizes the precise combined and data complexity of evaluation for the logics we consider in the paper.

Our results show that by disallowing comparisons based on regular relations, but by admitting comparisons based on rational relations from $\{\preceq_{\text{suff}}, \preceq_{\text{sw}}, \preceq_{\text{ss}}\}$, the complexity of evaluation becomes much more reasonable (it is always decidable, and elementary). On the other hand, the data complexity of evaluation for two of these logics (CRPQ($\preceq_{\text{sw}}$) and CRPQ($\preceq_{\text{ss}}$)) continues being intractable, and, therefore, further restrictions need to be imposed on them in order to obtain fragments that can be evaluated in practice.

One such restriction was identified in [4]: data complexity of evaluation becomes tractable when index sets I are acyclic; i.e., when the *undirected* graph defined by the pairs of I is acyclic. Our lower bounds for the data complexity of CRPQ($\preceq_{\text{sw}}$) and CRPQ($\preceq_{\text{ss}}$) show that lifting this restriction immediately leads to intractability. In fact, both lower bounds are proved for the index set $I_\diamond = \{(1,2), (1,3), (2,4), (3,4)\}$, which is a very simple DAG that is not acyclic. Our goal is identifying less restrictive conditions than acyclicity that yield efficient evaluation for our logics.

In the future we plan to study the decidability of the logic CRPQ($\preceq_{\text{suff}}, \preceq_{\text{sw}}, \preceq_{\text{ss}}$), which allows comparing paths based on the three relations we study in the paper. It is also of interest to study whether there are suitable decidable extensions of the logics we have studied in this paper that allow to compare paths based on lenghts or numbers of occurences of labels (in the style of [5]).

Acknowledgments

We are grateful to C. Gutiérrez and V. Diekert for helpful insights on the nature of word equations, and to L. Libkin for his comments on an earlier version of this paper. Barceló is funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004 and Muñoz by CONICYT Ph.D. Scholarship.

References

[1] H. Abdulrab, J.-P- Pécuchet. Solving word equations. *J. Symb. Comput.* 8(5), pages 499-521, 1989.

[2] R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Computing Surveys* 40(1): (2008).

[3] P. Barceló. Querying graph databases (Invited Tutorial). In *Symposium on Principles of Database Systems* (PODS) 2013, pages 175-188.

[4] P. Barceló, D. Figueira, L. Libkin. Graph logics with rational relations. *Logical Methods in Computer Science* 9(3), 2013.

[5] P. Barceló, L. Libkin, A. W. Lin, P. T. Wood. Expressive languages for path queries over graph-structured Data. *ACM Transactions on Database Systems* 37(4), 2012.

[6] J. Berstel. *Transductions and Context-Free Languages*. B. G. Teubner, 1979.

[7] G. Blin, L. Bulteau, M. Jiang, P. J. Tejada, S. Vialette. Hardness of Longest Common Subsequence for Sequences with Bounded Run-Lengths. In *Symposium on Combinatorial Pattern Matching* (CPM) 2012, pages 138-148.

[8] S. Buss, M. Soltys. Unshuffling a square is NP-hard. *Journal of Computer and System Sciences*, 2014.

[9] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *Principles of Knowledge Representation and Reasoning* (KR), pages 176–185.

[10] E. Clarke, O. Grumberg, D. Peled. *Model checking*. MIT Press, 1999.

[11] I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD* 1987, pages 323-330.

[12] V. Diekert, C. Gutiérrez, Ch. Hagenah. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Information and Computation* 202(2), pages 105-140, 2005.

[13] S. Eilenberg, C.C. Elgot, J.C. Shepherdson. Sets recognized by n-tape automata. *Journal of Algebra*, 13:447, 1969.

[14] L. Ilie. Subwords and power-free words are not expressible by word equations. *Fundamenta Informaticae*, 38(1), pages 109-118, 1999.

[15] J. Karhumäki, F. Mignosi, W. Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM* 47(3), pages 483-505, 2000.

[16] D. Kozen. Lower bounds for natural proof systems. In *Symposium on Foundations of Computer Science* (FOCS) 1977, pages 254-266.

[17] A. Lentin. Equations in free monoids. In *Intl. Colloquium on Automata, Languages, and Programming* (ICALP) 1972, pages 67-85.

[18] M. Lothaire. *Combinatorics on words*. Cambridge University Press, 1997.

[19] G.S. Makanin. The problem of solvability of equations in free semigroups. *Math USSR Sbornik* 32, pages 129-198, 1977.

[20] W. Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM* 51(3), pages 483-496, 2004

[21] R. Rizzi, S. Vialette. On recognizing words that are squares for the shuffle product. In *International Computer Science Symposium in Russia* (CSR) 2013 pages 235-245.

[22] M. Y. Vardi. The complexity of relational query languages (Extended abstract). In *Symposium on the Theory of Computing* (STOC) 1982, pages 137-146.

[23] P. T. Wood. Query languages for graph databases. *SIGMOD Record* 41(1), pages 50-60, 2012.