A

Graph Logics with Rational Relations: The Role of Word Combinatorics

PABLO BARCELÓ and PABLO MUÑOZ, Center for Semantic Web Research & Dept. of Computer Science, University of Chile

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 longest common subsequence.

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

General Terms: Algorithms, Languages, Theory

Additional Key Words and Phrases: complexity of evaluation, logics for graphs, rational relations, regular path queries, shuffle, word equations

ACM Reference Format:
Pablo Barceló and Pablo Muñoz. 2015. Graph logics with rational relations: the role of word combinatorics. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 41 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Graph databases are important for applications in which the topology of 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. This include, e.g., the semantic web, social networks and biological networks. Some surveys

Barceló and Muñoz are funded by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. Muñoz is also funded by CONICYT Ph.D. Scholarship. We are grateful to Claudio Gutiérrez and Volker Diekert for helpful insights on the nature of word equations, and to Diego Figueira for suggesting us the simple proof of Proposition 5.4 we present in the paper.
Authors' addresses: Pablo Barceló and Pablo Muñoz, Department of Computer Science, University of Chile, Beaucheff 851, Santiago, Chile.

DOI: http://dx.doi.org/10.1145/0000000.0000000

on graph database models and graph logical languages have been published as of late [Angles and Gutiérrez 2008; Wood 2012; Barceló 2013].

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 [Clarke et al. 1979]. 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* [Cruz et al. 1987], or RPQs, that define pairs of nodes 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 [Calvanese et al. 2000]. For instance,

$$\exists y \exists z (x \xrightarrow{\rho_1 : L_1} y \wedge x \xrightarrow{\rho_2 : L_2} z)$$

is a CRPQ that defines the set of nodes x that are the origin of paths ρ_1 and ρ_2 labeled by words matching regular expressions L_1 and L_2, respectively. Evaluation of CRPQs is NP-complete, but its *data complexity* – i.e., the complexity when the formula is assumed to be fixed – is tractable (NLOGSPACE). The latter is considered to be acceptable in the database context, in which a typically *small* formula is evaluated over a *large* dataset [Vardi 1982].

CRPQs fall short of expressive power for modern applications of graph databases due to their inability to compare paths [Barceló et al. 2012]. 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 [Barceló 2013]. 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) [Barceló et al. 2012], where REG is the class of *regular relations* on words [Eilenberg et al. 1969], 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. As an example, assume that $\preceq_{\text{pref}}$ is the binary relation that consists of all pairs (w_1, w_2) of words such that w_1 is a prefix of w_2. Then the CRPQ(REG) formula

$$\exists y \exists z \left(x \xrightarrow{\rho_1 : L_1} y \wedge x \xrightarrow{\rho_2 : L_2} z \wedge \rho_1 \preceq_{\text{pref}} \rho_2 \right)$$

defines the set of nodes in a graph database that are the origin of paths ρ_1 and ρ_2 labeled in L_1 and L_2, respectively, and such that the label of ρ_1 is a prefix of the label of ρ_2. Using automata tools it can be shown that CRPQ(REG) preserves the good data complexity properties of CRPQs (i.e., evaluation of CRPQ(REG) formulas 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 (see, e.g., [Anyanwu et al. 2007; Gusfield 1997]), but these relations are *not* regular. They are *rational*; i.e., they are still defined by automata, but those whose heads move asynchronously [Berstel 1979].

Adding rational relations to CRPQs has to be done carefully since the evaluation problem for CRPQ(RAT) is undecidable. However, 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 [Barceló et al. 2013] was studying to what extent rational relations such as subword $\preceq_{\text{sw}}$, suffix $\preceq_{\text{suff}}$ or subse-

quence $\preceq_{\mathrm{ss}}$, 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 both 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, i.e., not bounded by any stack of exponentials). In conclusion, these languages are completely 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 $\preceq_{\mathrm{ss}}$ – which 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, obtained using automata techniques, show that this restriction dramatically reduces the complexity of evaluation for CRPQ($\preceq_{\mathrm{ss}}$): the problem is in NEXPTIME, and even in NP in data complexity [Barceló et al. 2013]. On the other hand, such techniques were insufficient for determining the precise complexity of this problem and for establishing the decidability of evaluation for the logic CRPQ($\preceq_{\mathrm{sw}}$). In particular, they provide no answer to the question if any of these logics have good behavior in terms of data complexity.

This shows that some important problems – of both theoretical and practical impact – remain unanswered regarding the complexity of evaluation for these languages:

(1) Is the evaluation for the logic CRPQ($\preceq_{\mathrm{sw}}$) decidable? If so, what is its computational cost?
(2) What is the precise complexity of evaluating formulas in CRPQ($\preceq_{\mathrm{ss}}$), for which so far we only have an NEXPTIME upper bound and an NP upper bound in data complexity?
(3) Very much related to the previous questions, we would like to know if the complexity of evaluation for any of these languages is well-suited for practical applications. In complexity-theoretical terms, this asks whether any of them can be evaluated efficiently in 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 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 both $\preceq_{\mathrm{suff}}$ and $\preceq_{\mathrm{sw}}$ are definable by *word equations*, which implies 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 decidable in PSPACE [Schulz 1990; Diekert et al. 2005], based on ideas generated by the sophisticated Makanin's algorithm [Makanin 1977]. This immediately answers one of the questions left open by Barceló, Figueira, and Libkin [Barceló et al. 2013]: 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 [Barceló et al. 2013]; therefore, both problems are PSPACE-complete (matching the complexity of evaluation for first-order logic).

We then move to study whether any of these logics can be evaluated in polynomial time in data complexity (i.e., assuming the formula to be fixed). We start by proving the quite surprising result that evaluation of CRPQ($\preceq_{\rm sw}$) is PSPACE-complete even in data complexity (i.e., there is a *fixed* CRPQ($\preceq_{\rm sw}$) formula for which evaluation is PSPACE-complete). 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. In other words, solving word equations with regular constraints is not simpler when the equation is fixed in advance. This was not previoulsy known in the literature.

In striking contrast, we show that CRPQ($\preceq_{\rm suff}$) can be efficiently evaluated in data complexity. This follows by an easy reduction to the problem of evaluating CRPQ(REG) formulas in data complexity, which is known to be tractable [Barceló et al. 2013]. However simple, this reduction does not provide us with a general criterion for identifying when a (syntactic fragment of a) logic of the form CRPQ(S), for S a relation definable by word equations (such as suffix or subword), can be efficiently evaluated in data complexity. We provide here one such criterion based on a semantic condition. This criterion establishes that the word equation with regular constraints to which the evaluation of CRPQ(S) formulas in the fragment reduces only allows for a finite number of *principal* solutions. We then show that not only CRPQ($\preceq_{\rm suff}$) satisfies this criterion (which provides an alternative proof to the fact that this logic can be efficiently evaluated in data complexity), but also a natural syntactic restriction of a general family of logics of the form CRPQ(S), where S lies beyond the class of rational relations.

In the second part of the paper we study the complexity of evaluation for CRPQ($\preceq_{\rm ss}$). 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 for this problem in [Barceló et al. 2013] are sharp. We start by showing that the evaluation problem for CRPQ($\preceq_{\rm ss}$) is NP-complete in data complexity. The lower bound is obtained by a reduction from longest common subsequence (LCS). We also prove that, in general, evaluation for CRPQ($\preceq_{\rm ss}$) is NEXPTIME-complete. In this case, we use a more cumbersome reduction from a suitable succinct version of LCS. This proves that, in its full generality, the language CRPQ($\preceq_{\rm ss}$) is impractical.

Inverses. It is practically convenient to extend CRPQs with the ability to traverse edges in both directions. This gives rise to the well-studied class of CRPQs *with inverses*, or C2RPQs [Calvanese et al. 2000; 2002]. In order to simplify the presentation we only deal with CRPQs in the paper. Nevertheless, it is easy to see that all our complexity bounds for the evaluation problem continue to hold when basic CRPQs are extended with inverses. In order to explain this we introduce the notion of "completion" $G^{\pm}$ of a graph database G. This completion is obtained from G by adding the "inverse" $u \xrightarrow{a^-} v$ of every edge $v \xrightarrow{a} u$ in G (for a a symbol in the alphabet). Then clearly evaluation of a C2RPQ(S) formula ϕ over G corresponds to evaluation of the same ϕ, seen now as a formula in CRPQ(S), over $G^{\pm}$. Since $G^{\pm}$ can be constructed in LOGSPACE from G, and all complexity classes considered in this paper are closed under LOGSPACE reductions, the latter shows that all our upper bounds continue to hold in the presence of inverses. Lower bounds, on the other hand, follow directly from the ones for CRPQs.

Organization. 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($\preceq_{\rm suff}$) and CRPQ($\preceq_{\rm sw}$) are presented in Section 4, and those on the data complexity in Section 5. Results on the complexity of evaluation for the logic CRPQ($\preceq_{\rm ss}$) can be found in Section 6. We finish with our final remarks in Section 7.

2. PRELIMINARIES AND BASIC CONCEPTS

Relations on words. For a finite alphabet Σ, we study practically motivated binary relations over Σ^* such as prefix, subword, suffix and subsequence, which are defined as follows. Consider words $w, p, x, s \in \Sigma^*$ such that $w = pxs$. Then:

(1) p is a *preffix* of w (written as $p \preceq_{\text{pref}} w$).
(2) x is a *subword* of w (written as $x \preceq_{\text{sw}} w$).
(3) s is a *suffix* of w (written as $s \preceq_{\text{suff}} w$).

Furthermore, for $w, w' \in \Sigma^*$ we call w' a *subsequence* of w (written as $w' \preceq_{\text{ss}} w$) if w' is obtained from w by removing some letters (perhaps none) from w; that is, $w = a_1 \dots a_n$, and $w' = a_{i_1} a_{i_2} \dots a_{i_k}$, where $1 \leq i_1 < i_2 < \dots < i_k \leq n$.

Regular and rational relations. Recall that a *nondeterministic finite automata* (NFA) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$, where (i) Σ is a finite alphabet, (ii) Q is a finite set of *states*, (iii) $q_0 \in Q$ is the *initial* state, (iv) $F \subseteq Q$ is the set of *final* states, and (v) $\delta \subseteq Q \times \Sigma \times Q$ is the *transition relation*. A *run* of $\mathcal{A}$ over a word $w = a_1 a_2 \dots a_n$ in Σ^* is a sequence $q_0 q_1 q_2 \dots q_n$ of states of $\mathcal{A}$ (starting in the initial state q_0) such that $(q_{i-1}, a_i, q_i) \in \delta$, for each $1 \leq i \leq n$. The run is *accepting* if $q_n \in F$. We denote by $L(\mathcal{A})$ the *language* defined by $\mathcal{A}$, that is, the set of words $w \in \Sigma^*$ such that $\mathcal{A}$ has an accepting run over w. A language $L \subseteq \Sigma^*$ is *regular* if it is of the form $L(\mathcal{A})$ for some NFA $\mathcal{A}$ over Σ. A classical result in automata theory establishes that the regular languages are precisely the ones defined by regular expressions.

We now define the class of 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 $\Sigma_\perp^n$ that defines $\{w_1 \otimes \dots \otimes w_n \mid (w_1, \dots, w_n) \in R\}$. 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. (It is worth remarking that NFAs recognising regular relations are known by many different names in the literature, e.g., synchronised automata [Blumensath and Grädel 2000; Frougny and Sakarovitch 1993], letter-to-letter transducers [Abdulla et al. 2004; Benedikt et al. 2003], synchronised rational transducers [To and Libkin 2010], and aligned multi-track automata [Yu et al. 2011]).

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

There are two equivalent ways to define rational relations over Σ (e.g. see [Berstel 1979; Sakarovitch 2009]). One uses regular expressions, which are now built from tuples $\bar{a} \in (\Sigma \cup \{\varepsilon\})^n$, where ε is the empty word, applying the operations of union, concatenation, and Kleene star. Alternatively, rational relations can be defined by means of n-tape automata, that have n heads for the tapes and one additional finite 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.

More formally, an n-tape NFA over the alphabet Σ is a tuple $\mathcal{A} = (\Gamma, Q, q_0, F, \delta)$, where $\Gamma := (\Sigma \cup \{\varepsilon\})^n$, such that $\mathcal{A}$ is syntactically an NFA over Γ. A tuple $\bar{w} \in (\Sigma^*)^n$ is *accepted* by $\mathcal{A}$ if there exists a word $\sigma_1\sigma_2 \dots \sigma_n \in \Gamma^*$ for which the following holds: (a) if we treat $\mathcal{A}$ as an NFA, then $\sigma_1\sigma_2 \dots \sigma_n$ belongs to $L(\mathcal{A})$, and (b) $\bar{w} = \sigma_1 \circ \sigma_2 \circ \dots \circ \sigma_n$, where $\circ$ denotes an operator that first acts like the word concatenation operator extended to tuples over words component-wise (i.e., $(v_1, v_2) \circ (w_1, w_2) = (v_1w_1, v_2w_2)$), and then deletes every apperance of the empty word ε from the resulting tuple of words (e.g., $(a\varepsilon, b) \circ (cd, \varepsilon e) = (acd, be)$).

An n-ary relation R of words over Σ (i.e., $R \subseteq (\Sigma^*)^n$) is said to be *rational* if there is an n-tape NFA which accepts precisely the tuples in R. We denote by RAT the class of rational relations, and write RAT_n to denote the restriction of RAT to relations of arity n. Notice that it is immediate from the definition that the class of regular relations is a subset of the rational relations, that is, $\mathsf{REG}_n \subseteq \mathsf{RAT}_n$ for each $n \geq 1$.

Example 2.2. For each Σ, the relations $\preceq^{\Sigma}_{\text{suff}}$, $\preceq^{\Sigma}_{\text{sw}}$, and $\preceq^{\Sigma}_{\text{ss}}$ are 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:

$$\Big(\bigcup_{a\in\Sigma}(\varepsilon, a)\Big)^* \cdot \Big(\bigcup_{a\in\Sigma}(a, a)\Big)^* \cdot \Big(\bigcup_{a\in\Sigma}(\varepsilon, a)\Big)^*.$$

— 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, $\mathsf{REG}_n \subsetneq \mathsf{RAT}_n$ for each $n > 1$ (see, e.g., [Berstel 1979]). For instance, 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 $\preceq_{\text{ss}}$.

In the rest of the paper we do not distinguish between an NFA (resp., regular expression) S and the set of tuples of words it defines; e.g., we write $\bar{w} \in S$ to denote that the 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. Equivalently for $\preceq_{\text{suff}}$ and $\preceq_{\text{ss}}$.

The generalized intersection problem. The evaluation problem for logics of the form CRPQ($\mathcal{S}$) (for $\mathcal{S} \subseteq \mathsf{RAT}$) can be stated in language-theoretical terms [Barceló et al. 2013]. 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 $\{1, \dots, m\}$. 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 $(L_1, \dots, L_m, I, \lambda)$ 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 $1 \leq i \leq m$, 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.

Example 2.3. In several of our proofs we make use of the index set:

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

In particular, if S is a binary relation on Σ^* then the inputs to GENINT$_{I_\diamond}(S)$ are 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 $\lambda(i,j) = \{S\}$ for each $(i,j) \in I_\diamond$. □

In the case when $\mathcal{S} = (\mathsf{REG}_2 \cup \preceq)$, for $\preceq$ one of $\preceq_{\mathrm{sw}}$, $\preceq_{\mathrm{suff}}$ or $\preceq_{\mathrm{ss}}$, 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 [Barceló et al. 2013] 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 $\preceq_{\mathrm{sw}}$, $\preceq_{\mathrm{suff}}$ or $\preceq_{\mathrm{ss}}$. In this case the problem becomes either undecidable or highly intractable:

PROPOSITION 2.4. [Barceló et al. 2013] *The following holds:*

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

We consider now the cases when $\mathcal{S} = \preceq_{\mathrm{ss}}$. This restriction allows us to reduce the complexity of GENINT($\mathcal{S}$).

PROPOSITION 2.5. [Barceló et al. 2013] *The problem* GENINT$(\preceq_{\mathrm{ss}})$ *is in* NEXPTIME. *For each fixed I, the problem* GENINT$(\preceq_{\mathrm{ss}})$ *is in* NP.

The case of GENINT$(\preceq_{\mathrm{sw}})$ GENINT$(\preceq_{\mathrm{suff}})$ were left open in [Barceló et al. 2013].

3. LOGICS FOR GRAPH DATABASES

Graph databases. The standard abstraction of graph databases [Angles and Gutiérrez 2008] 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

$$(v_0, a_0, v_1)(v_1, a_1, v_2) \ldots (v_{m-1}, a_{m-1}, v_m)$$

of edges from E, for some $m \geq 0$. 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 $v \in V$.

Each graph database $G = (V, E)$ can be naturally interpreted as an NFA $\mathcal{A}(G)$ over Σ without initial and final states. The states of $\mathcal{A}(G)$ are the nodes in V, and there is a transition labeled a from state u to v in $\mathcal{A}(G)$ if and only if $(u, a, v) \in E$. In our proofs, we typically do not distinguish between G and $\mathcal{A}(G)$.

Graph logics. The main building block for graph logics are *regular path queries*, or *RPQ*s [Cruz et al. 1987]; they are expressions of the form:

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

where L is a regular language (given as an NFA)[1] over Σ. Given a Σ-labeled graph $G = (V,E)$, an RPQ $\phi(x,y)$ of the form above, and v, v' nodes of G, we have that $G \models \phi(v,v')$ if and only if there is a path ρ from v to v' in G with $\kappa(\rho) \in L$.

Conjunctive RPQs, or *CRPQs* [Calvanese et al. 2000], are the closure of RPQs under conjunction and existential quantification. Formally, they are expressions of the form

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

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

CRPQs can further be extended to *compare* labels of 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:

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

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 $G \models \phi(\bar{a})$ 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 such that there are two paths starting from v, the first one ending with an a-edge and the second one with a b-edge, and the label of the first path is a subsequence of the label of the second one. □

The evaluation problem. For a logic CRPQ($\mathcal{S}$) this is the problem of, given a graph database G, a tuple $\bar{a}$ of nodes, and a formula $\phi(\bar{x})$ in CRPQ($\mathcal{S}$), determine whether $G \models \phi(\bar{a})$. This corresponds to the *combined complexity* of evaluation. In the context of databases, one is also interested in *data complexity*, which fixes the (typically small) formula ϕ and defines the input as consisting solely of the (large) graph database G and the tuple $\bar{a}$. More specifically, let ϕ be a formula in CRPQ($\mathcal{S}$). The *evaluation problem for* ϕ takes as input $(G, \bar{a})$, and asks whether $G \models \phi(\bar{a})$. Let $\mathcal{C}$ be a complexity class. We say that the evaluation of CRPQ($\mathcal{S}$) is in $\mathcal{C}$ in data complexity, if the evaluation problem for each formula in CRPQ($\mathcal{S}$) is in $\mathcal{C}$. The evaluation of CRPQ($\mathcal{S}$) is $\mathcal{C}$-hard in

[1]In examples and proofs we typically use regular expressions to specify regular languages, as this helps readability. Each regular expression can be easily translated into an equivalent NFA in polynomial time, and, therefore, this does not have any effect on our results.

data complexity, if there is a formula in CRPQ($\mathcal{S}$) for which the evaluation problem is $\mathcal{C}$-hard. Also, 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 using standard automata tools. In particular, this logic is tractable in data complexity.

PROPOSITION 3.2. (see [Barceló et al. 2012; Barceló 2013]) *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($\mathcal{S}$) if the latter belongs to a class that is closed under PSPACE reductions. This is stated in the next lemma, which uses techniques from [Barceló et al. 2013].

LEMMA 3.3. *Let $\mathcal{C}$ be a complexity class closed under* PSPACE *reductions. Then:*

(*1*) *If* GENINT($\mathcal{S}$) *is in $\mathcal{C}$, then evaluation of CRPQ($\mathcal{S}$) is in $\mathcal{C}$.*
(*2*) *If* GENINT($\mathcal{S}$) *is $\mathcal{C}$-hard, then evaluation of CRPQ($\mathcal{S}$) is $\mathcal{C}$-hard.*

PROOF. For the first statement, it is sufficient to prove that there is a PSPACE reduction from the evaluation problem for CRPQ($\mathcal{S}$) to GENINT($\mathcal{S}$) (since $\mathcal{C}$ is closed under PSPACE reductions). Consider an input to the evaluation problem for CRPQ($\mathcal{S}$) given by a graph database $G = (V, E)$ over Σ, a tuple $\bar{a}$ of nodes, and a formula

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

We construct a PSPACE algorithm that solves the evaluation problem using GENINT($\mathcal{S}$) as an oracle. The algorithm cycles through all tuples of nodes $\bar{b}$ from G such that $|\bar{b}| = |\bar{y}|$. Notice that each such tuple $\bar{b}$ can be stored using $\log(|V|) \cdot |\bar{y}|$ bits, which is linear in the size of the input. For each $1 \leq i \leq m$, let v_i and v'_i be the nodes in $\{\bar{a}, \bar{b}\}$ interpreting the variables u_i and u'_i, respectively, and assume that $\mathcal{A}_i$ is the NFA over Σ which is obtained from G by setting v_i and v'_i as the unique initial and final state, respectively. Recall that L_i, for $1 \leq i \leq m$, is an NFA over Σ. It is easy to see that the NFA $\mathcal{A}_i \times L_i$ which defines the product of $\mathcal{A}_i$ and L_i accepts precisely those words $w \in \Sigma^*$ such that there is a path ρ from u_i to v_i in G for which $\kappa(\rho) \in L_i$. Clearly then, $G \models \phi(\bar{a})$ if and only if $(\mathcal{A}_1 \times L_1, ..., \mathcal{A}_m \times L_m, I, \lambda)$ belongs to GENINT($\mathcal{S}$). Further, $(\mathcal{A}_1 \times L_1, ..., \mathcal{A}_m \times L_m, I, \lambda)$ can be constructed in polynomial time from G and ϕ.

For the second part, let $(L_1, ..., L_m, I, \lambda)$ be an input instance to GENINT($\mathcal{S}$). We construct in PTIME a graph database $G = (V, E)$, a tuple $\bar{a}$ of nodes in G, and a CRPQ($\mathcal{S}$) formula $\phi(\bar{x})$, such that $(L_1, ..., L_m, I, \lambda)$ belongs to GENINT($\mathcal{S}$) if and only if $G \models \phi(\bar{a})$. Let $\{f_1, ..., f_m\}$ be fresh nodes and $\#$ a symbol not in Σ. We define a graph database G by first taking the disjoint union of the NFAs/graph databases L_i, for $1 \leq i \leq m$, and then inserting fresh nodes $\{t_1, ..., t_m\}$ and edges $(f_i, \#, t_i)$, for each $1 \leq i \leq m$ and final state f_i in $\mathcal{A}_i$. According to this construction, for each $1 \leq i \leq m$ all paths in G from q_i to t_i, where q_i is the initial state of the NFA L_i, are of the form $w\#$ with $w \in L_i$. Conversely, for each $1 \leq i \leq m$ and path in G from q_i to t_i which is labeled with a word of the form $\tilde{w} = w\#$, it is the case that $w_i \in L_i$ (since $\#$ is not in Σ and t_i has no outcoming edges). Notice that G can be constructed in polynomial time (actually, in logarithmic space) from $L_1, \ldots, L_m$.

The formula $\phi(\bar{x}) \in$ CRPQ($\mathcal{S}$) we consider is the following:

$$\exists x'_1, \ldots, x'_m \bigwedge_{i \in [m]} \Big(x_i \xrightarrow{\chi_i : \Sigma^*} x'_i \wedge x'_i \xrightarrow{\chi'_i : \#} y_i \Big) \wedge \bigwedge_{(i,j)\in I} \bigwedge_{S \in \lambda(i,j)} S(\chi_i, \chi_j), \qquad (2)$$

where $\bar{x} = (x_1, y_1, \ldots, x_m, y_m)$. Clearly, ϕ can be constructed in polynomial time from I, Σ, and λ. From our previous remarks it follows that $G \models \phi(q_1, t_1, \ldots, q_m, t_m)$ if and only if there are words $w_1, ..., w_m$ in $L_1, ..., L_m$, respectively, such that $(w_i, w_j) \in S$ for each $(i, j) \in I$ and $S \in \lambda(i, j)$. This concludes the proof. □

Again applying techniques from [Barceló et al. 2013] we can prove that the data complexity of evaluation for CRPQ($\mathcal{S}$) can be studied in terms of suitable restrictions of GENINT($\mathcal{S}$). Recall that GENINT$_I(\mathcal{S})$ corresponds to the restriction of GENINT($\mathcal{S}$) in which the index set I is fixed. We also denote by 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 mapping $\lambda : I \to 2^{\mathcal{S}}$ are fixed. Thus, the input to this problem consists only of regular expressions L_i, for $1 \leq i \leq m$, over the fixed alphabet Σ.

LEMMA 3.4. *Let $\mathcal{C}$ be a complexity class closed under* LOGSPACE *reductions.*

(*1*) *If for each $I \subseteq [m]^2$ it is the case that* 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* GENINT$_{I,\Sigma,\lambda}(\mathcal{S})$ *is $\mathcal{C}$-hard, then evaluation of CRPQ($\mathcal{S}$) is $\mathcal{C}$-hard in data complexity.*

PROOF. The proof of this lemma follows directly from that of Lemma 3.3. For the first item, notice that if the formula is fixed the space needed to store each tuple $\bar{b}$ of nodes such that $|\bar{b}| = |\bar{y}|$ is $O(\log |V|)$. Furthermore, it is not hard to see that for each $1 \leq i \leq m$ the NFA $\mathcal{A}_i \times L_i$ can be constructed in LOGSPACE. Therefore, since the formula is fixed, the tuple $(\mathcal{A}_1 \times L_1, ..., \mathcal{A}_m \times L_m, I, \lambda)$ can also be constructed in LOGSPACE. Notice that I is fixed in this case, and hence this tuple is an input to GENINT$_I(\mathcal{S})$. Now the result follows from the facts that LOGSPACE computable functions are closed under composition and $\mathcal{C}$ is closed under LOGSPACE reductions.

For the second item, notice that the reduction used in the second part of the proof of Lemma 3.3 constructs a formula $\phi(\bar{x})$ (in Equation (2)) which only depends on $I \subseteq [m]^2$, the alphabet Σ, and $\lambda : I \to 2^{\mathcal{S}}$. These parameters are all fixed, and, therefore, the formula ϕ itself is fixed. Furthermore, the graph database G and the tuple $\bar{a} = (q_1, t_1, \ldots, q_m, t_m)$ of nodes used in such reduction can be constructed in LOGSPACE from $L_1, \ldots, L_m$. This implies our result. □

Applying these two lemmas, together with Propositions 2.4 and 2.5, 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 3.5. [Barceló et al. 2013] *The following holds:*

(*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 3.3 and Proposition 2.4. On the other hand, the lower bounds follow from Lemma 3.4 and Proposition 2.4. 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 GENINT$_{I,\Sigma',\lambda}(\mathsf{REG}_2 \cup \preceq)$, for some I, Σ' and λ [Barceló et al. 2013].

The next corollary deals with the logic CRPQ($\preceq_{\text{ss}}$). It is shown that this restriction reduces dramatically the complexity of evaluation. The proof follows directly from Proposition 2.5 and Lemmas 3.3 and 3.4.

COROLLARY 3.6. [Barceló et al. 2013] *The evaluation problem for CRPQ*($\preceq_{\text{ss}}$) *can be solved in* NEXPTIME*, and in* NP *in data complexity.*

The case of CRPQ($\preceq_{\text{sw}}$) was left open in previous work [Barceló et al. 2013]. 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 admits a clean presentation, we concentrate on studying the complexity of such problem and then transfer the results to the complexity of evaluation for the corresponding logics using Lemmas 3.3 and 3.4.

4. COMBINED COMPLEXITY OF LOGICS CRPQ($\preceq_{\text{SUFF}}$) AND CRPQ($\preceq_{\text{SW}}$)

We start by studying the combined complexity of evaluation for logics CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$). The relations $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ share an important property: 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 known to be in PSPACE [Diekert et al. 2005]. It follows that evaluation of both CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$) is PSPACE-complete.

4.1. Word equations and the generalized intersection problem

Let X be a countably infinite set of variables. A *word equation* over Σ [Makanin 1977] is an expression e of the form $\phi = \psi$, 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., $h(\phi) = h(\psi)$, assuming that $h(a) = a$ for each symbol $a \in \Sigma$.[2] A *word equation with regular constraints* [Schulz 1990] 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.

In order to simplify notation, we sometimes specify properties using *systems of word equations*. These are nothing but finite sets of word equations. A solution for any such system E is a mapping h from the variables that appear in E to the set of words over a specified alphabet Σ, such that h is a solution for each word equation in E. Allowing for systems of word equations come at no cost, as E can always be represented as an equivalent word equation e. In fact, assume that E is of the form $\{\phi_i = \psi_i \mid 1 \leq i \leq n\}$. Then E can be represented as

$$\phi_1 \# \dots \# \phi_n \;=\; \psi_1 \# \dots \# \psi_n,$$

where $\#$ is a fresh symbol that does not appear in E. Notice that the solutions for E coincide with the solutions for e. Moreover, e can be constructed in LOGSPACE from E. Therefore, from now on we do not distinguish between a system of word equations and the single word equation that is equivalent to it.

A deep result due to Makanin states that the problem of existence of solutions for word equations is decidable [Makanin 1977]. By applying somewhat different techniques, Plandowski proved that the problem is in PSPACE [Plandowski 2004]. Then Schulz developed an extension of those techniques to prove that the latter holds even for word equations with regular constraints:

[2]We assume, as usual, that if $\phi = t_1 \dots t_n$ then $h(\phi) = h(t_1) \dots h(t_n)$.

THEOREM 4.1. [Schulz 1990; Diekert et al. 2005] *The problem of existence of solutions for word equations with regular constraints is* PSPACE*-complete.*

Word equations can be used to define relations on words (see, e.g., [Lothaire 1997; Karhumäki et al. 2000]). Formally, an n-ary relation R over Σ^* is *definable by word equations*, if there is a word equation e over Σ and an ordered tuple $(x_1, \ldots, x_n)$ of variables 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 which are definable by word equations. We assume that each such binary relation is specified as a word equation that defines it.

Example 4.2. We provide examples of definability in EQ and the relationship of the latter with REG and RAT:

(1) Both $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ are in EQ. More precisely, $\preceq_{\text{suff}}$ is the set of pairs (x, y) that satisfy the word equation $y = px$, and $\preceq_{\text{sw}}$ is the set of pairs (x, y) that satisfy the word equation $y = pxs$. On the other hand, $\preceq_{\text{ss}}$ is not in EQ [Ilie 1999].
(2) REG $\not\subset$ EQ: the *equal length relation* (the set of pairs (w_1, w_2) of words of the same length) is regular, but not definable by word equations [Karhumäki et al. 2000].
(3) EQ $\not\subset$ RAT: the *conjugacy relation* $\{(ww', w'w) \mid w, w' \in \Sigma^*\}$ is in EQ, as witnessed by the system of word equation $x = yz$ and $x' = zy$, by taking projection over (x, x'). On the other hand, this relation is not rational. □

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 idea first with an example.

Example 4.3. Let us consider the index set $I_\diamond = \{(1,2), (1,3), (2,4), (3,4)\}$ from Example 2.3. Furthermore, let $(L_1, L_2, L_3, L_4, \lambda)$ be an instance of the problem $\text{GENINT}_{I_\diamond}(\preceq_{\text{suff}})$, such that the L_i's are NFAs over Σ for $1 \leq i \leq 4$, and $\lambda(i,j) = \{\preceq_{\text{suff}}^{\Sigma}\}$ 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 $\text{GENINT}_{I_\diamond}(\preceq_{\text{suff}})$ if and only if the word equation with regular constraints (e, ν) has a solution, where e and ν are as follows: First, e is the system of word equations:

$$\begin{aligned} w_2 &= u_{12} w_1, \\ w_3 &= u_{13} w_1, \\ w_4 &= u_{24} w_2, \\ w_4 &= u_{34} w_3. \end{aligned}$$

Second, $\nu(w_i) = L_i$ for each $1 \leq i \leq 4$, and (iii) $\nu(u_{i,j}) = \Sigma^*$ for each $(i,j) \in I_\diamond$. Clearly, (e, ν) can be constructed in polynomial time from $(L_1, L_2, L_3, L_4, \lambda)$. Note that e only uses variables and its form depends exclusively on I and λ (i.e., it is independent of the L_i's and Σ). □

Generalizing from the idea presented in the previous example, we can prove the following proposition:

PROPOSITION 4.4. *There is a polynomial time reduction 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 Σ such that $(L_1, \ldots, L_m, , I, \lambda) \in$ GENINT(EQ) if and only if (e, ν) has a solution.*

Also, the form of e depends exclusively on I and λ. We refer to this equation as $e(I, \lambda)$.

PROOF. A binary relations $S \in \mathsf{EQ}$ is defined by the word equation $e_S := (\phi_S = \psi_S)$, by taking projection over the ordered pair of variables (x_1, x_2). We call *existential* the variables in e_S which are different from x_1 and x_2. We assume without loss of generality that for each $(i, j) \in I$, each binary relation $S \in \lambda(i, j)$ is defined by the word equation $e_S^{i,j} := (\phi_S^{i,j} = \psi_S^{i,j})$ obtained from e_S by replacing variables x_1, x_2 by x_i, x_j, and each existential variable z by $z_{i,j}$.

Let $e_1, \ldots, e_\ell$, with $e_i := (\phi_i = \psi_i)$ for each $1 \leq i \leq \ell$, be an enumeration of the equations defining all relations in the sets of the form $\lambda(i, j)$, for $(i, j) \in I$. This enumeration exists since λ is of finite range. Let us now define a word equation with regular constraints (e, ν) over $\Sigma_\#$ such that:

— The word equation e is defined by the set:

$$\{\phi_i \ = \ \psi_i \mid 1 \leq i \leq \ell\}.$$

Clearly, e is completely determined by I and λ. We can thus denote it as $e(I, \lambda)$.

— The mapping ν satisfies that $\nu(x_i) = L_i$ for each $1 \leq i \leq m$, and $\nu(y) = \Sigma^*$ for each other variable y mentioned in $e(I, \lambda)$

Clearly, $(e(I, \lambda), \nu)$ can be constructed in polynomial time from $(L_1, \ldots, L_m, , I, \lambda)$. Furthermore, it is easy to see from our previous remarks that $(L_1, \ldots, L_m, I, \lambda) \in$ GENINT(EQ) if and only if $(e(I, \lambda), \nu)$ has a solution. □

As a corollary to Proposition 4.4 and Theorem 4.1, we obtain that GENINT(EQ), and, thus, GENINT($\preceq_{\text{suff}}$) and 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 NFAs/regular expressions $L_1, \ldots, L_m$, which is known to be PSPACE-hard [Kozen 1977], can be efficiently reduced to 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* [Barceló et al. 2013], i.e., when the undirected graph defined by I over $[m]$ is acyclic. Summing up:

COROLLARY 4.5. *The problem* GENINT(EQ) *is in* PSPACE.

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

PROOF. The upper bound follows from Proposition 4.4 and Theorem 4.1. The lower bounds are obtained by a reduction from the problem of checking for nonemptiness the language defined by the intersection of m NFAs $L_1, ..., L_m$ over an alphabet Σ (described above). Let $\preceq$ be either $\preceq_{\text{suff}}$ or $\preceq_{\text{sw}}$ and assume that $\#$ is a symbol not in Σ. Further, for each $1 \leq i \leq m$ let $\tilde{L}_i$ be the NFA which accepts the language $\#L_i\#$. We also define (a) an index set $I = \{(i, i+1) \mid 1 \leq i \leq m-1\}$, and (b) a mapping $\lambda : I \to 2^{\preceq}$ that associates the relation $\preceq$ with each pair $(i, i+1) \in I$. Notice that I is acyclic.

We claim that $(\tilde{L}_1, ..., \tilde{L}_m, I, \lambda) \in$ GENINT(EQ) if and only if $\bigcap_{1 \leq i \leq m} L_i$ is nonempty. In fact, assume first that there is a word $w \in \bigcap_{1 \leq i \leq m} L_i$. Then for each $1 \leq i \leq m$ the word $\tilde{w} = \#w\#$ belongs to $\tilde{L}_i$. Further, since $\preceq$ is a reflexive relation, all the constraints imposed by the mapping λ are satisfied. Assume, on the other hand, that there are words $\#w_1\#, \ldots, \#w_m\#$ such that (a) $\#w_i\# \in \tilde{L}_i$ for each $1 \leq i \leq m$, and (b) $\#w_i\# \preceq \#w_{i+1}\#$ for each $1 \leq i \leq m-1$. Then it must be the case that $w_i \in L_i$ for each

$1 \leq i \leq m$. Further, since $\# \notin \Sigma$, the $\#$ symbols at the beginning and end of $\#w_i\#$ and $\#w_{i+1}\#$, respectively, have to match in any witness of $\#w_i\# \preceq \#w_{i+1}\#$. Therefore $w_1 = w_2 = \cdots = w_m$, which implies that $w_1 \in \bigcap_{1 \leq i \leq m} L_i$. □

4.2. Complexity of CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$)

We can now apply Lemma 3.3, and make use of the results in Corollary 4.5, to determine the precise complexity of evaluation for the logics CRPQ(EQ), CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$):

THEOREM 4.6. *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 in which the index set* I *is acyclic.*

5. DATA COMPLEXITY OF LOGICS CRPQ($\preceq_{\text{SUFF}}$) AND CRPQ($\preceq_{\text{SW}}$)

Although the logics CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$) exhibit similar properties in combined complexity, we show in this section a remarkable difference in their data complexity: CRPQ($\preceq_{\text{suff}}$) allows for tractable evaluation in data complexity (in particular, in NLOGSPACE), whereas there are queries in CRPQ($\preceq_{\text{sw}}$) which are still PSPACE-hard to evaluate. While the proof of the former is simple, it does not provide us with a uniform condition for explaining when the evaluation of (a fragment of) a logic of the form CRPQ(S), for $S \in$ EQ, is tractable in data complexity. We provide here one such condition based on the techniques developed in Section 4 and use it to obtain tractability in data complexity for relevant fragments of a family of logics of the form CRPQ(S), where $S \in$ EQ.

5.1. Data complexity of CRPQ($\preceq_{\text{sw}}$)

We start by studying CRPQ($\preceq_{\text{sw}}$), for which we prove the following:

THEOREM 5.1. *The evaluation problem for* CRPQ($\preceq_{\text{sw}}$) *is complete for* PSPACE *in data complexity.*

The upper bound follows from Theorem 4.6. Due to Lemma 3.4, for hardness we only need to prove the following:

PROPOSITION 5.2. *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* 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 4.3. It follows from [Kozen 1977] that there is a finite alphabet Σ such that the following problem is PSPACE-complete: Given NFAs/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 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 NFAs $L_1, \ldots, L_m$ over Σ such that no L_i accepts the empty word, we construct an instance (R_1, R_2, R_3, R_4) of GENINT$_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$ such that the R_i's are NFAs over $\Sigma_\$$ 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 $\bigcap_{1\leq i\leq m} L_i \neq \emptyset$ if and only if $(R_1, R_2, R_3, R_4) \in \text{GENINT}_{I_\diamond,\Sigma_\$,\lambda_\diamond}(\preceq_{\text{sw}})$. Assume first there is word $w \in \bigcap_{1\leq i\leq m} L_i$. Then $w \neq \varepsilon$. Consider the word u over $\Sigma_\$$ defined as $u := (\$w)^m\$$, and let $w_1 := u$, $w_2 := wu$, $w_3 := uw$ and $w_4 := \$wu$. It is clear that $w_i \in R_i$ for each $1 \leq i \leq 4$. Furthermore, it is easy to see that $w_i \preceq_{\text{sw}} w_j$ for each $(i, j) \in I_\diamond$ (in fact, w_2 is obtained from w_4 by removing the first symbol \$, and w_3 is obtained from w_4 by removing the last symbol \$). 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 interesting corollary to the proof of Proposition 5.2 is that there exists a *fixed* word equation e such that solving e under regular constraints is PSPACE-complete. Formally, we denote by $\mathsf{WE}(e)$ the problem of evaluating the fixed word equation e under regular constraints. This problem 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.

COROLLARY 5.3. *There is a word equation e and a finite alphabet Σ such that the problem $\mathsf{WE}(e)$ is* PSPACE*-complete, even if restricted to inputs over Σ.*

PROOF. The word equation e corresponds to the system:

$$\begin{aligned} w_2 &= u_{12}w_1u'_{12} \\ w_3 &= u_{13}w_1u'_{13} \\ w_4 &= u_{24}w_2u'_{24} \\ w_4 &= u_{34}w_3u'_{34}, \end{aligned}$$

where all symbols are variables. In fact, it follows from the proof of Proposition 5.2 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 $\mathsf{WE}(e)$, even for inputs over $\Sigma_\$$. □

5.2. Data complexity of CRPQ($\preceq_{\text{suff}}$)

In sharp contrast to CRPQ($\preceq_{\text{sw}}$), the logic CRPQ($\preceq_{\text{suff}}$) can be evaluated in polynomial time in data complexity. This follows by an easy reduction to the problem of evaluating CRPQ($\preceq_{\text{pref}}$) formulas in data complexity, which is in polynomial time (actually, in NLOGSPACE) from Proposition 3.2 (since $\preceq_{\text{pref}}$ is a binary regular relation).

PROPOSITION 5.4. *The evaluation problem for* CRPQ($\preceq_{\text{suff}}$) *is in* NLOGSPACE *in data complexity.*

PROOF. Let us start by noticing that for any pair of words $u, v \in \Sigma^*$ it is the case that $u \preceq_{\text{suff}} v$ if and only if $u^{-1} \preceq_{\text{pref}} v^{-1}$, where $(\cdot)^{-1}$ denotes the *word reversal operation*. This allows us to construct a LOGSPACE translation from CRPQ($\preceq_{\text{suff}}$) evaluation to CRPQ($\preceq_{\text{pref}}$) evaluation. The result then follows from the fact that CRPQ($\preceq_{\text{pref}}$) evaluation is in NLOGSPACE in data complexity (since NLOGSPACE-computable functions are closed under composition).

Let $\phi(\bar{x})$ be a fixed CRPQ($\preceq_{\text{suff}}$) formula of the following form:

$$\exists \bar{y} \, \big(\bigwedge_{i=1}^{m} u_i \xrightarrow{\rho_i : L_i} u'_i \;\; \wedge \bigwedge_{(i,j) \in I} \rho_i \preceq_{\text{suff}} \rho_j \big).$$

It is known that the *reversal* of a regular language L (i.e., the set of words of the form w^{-1}, for $w \in L$) is also regular. We can assume then that there are NFAs $L_1^{-1}, \ldots, L_m^{-1}$ that define the reversals of $L_1, \ldots, L_m$, respectively. From this we define a CRPQ($\preceq_{\text{pref}}$) formula $\phi'(\bar{x})$ as follows:

$$\exists \bar{y} \, \big(\bigwedge_{i=1}^{m} u'_i \xrightarrow{\rho'_i : L_i^{-1}} u_i \;\; \wedge \bigwedge_{(i,j) \in I} \rho'_i \preceq_{\text{pref}} \rho'_j \big).$$

Notice that $\phi'(\bar{x})$ only depends on $\phi(\bar{x})$, and thus it is fixed.

Given a graph database $G = (V, E)$ we construct the graph database G^{-1} by *reversing* the edges of G. Formally, G^{-1} is constructed from G by replacing each edge $(u, a, v) \in E$ with (v, a, u). From our previous remarks it is clear that $G \models \phi(\bar{a})$, for a tuple $\bar{a}$ of nodes in V, if and only if $G^{-1} \models \phi'(\bar{a})$. Further, $(G^{-1}, \bar{a})$ can be constructed in LOGSPACE from $(G, \bar{a})$. This concludes the proof. □

It is not hard to see that each formula in CRPQ($\preceq_{\text{sw}}$) can be expressed in the logic 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 CRPQ($\preceq_{\text{pref}} \cup \preceq_{\text{suff}}$). From Theorem 5.1 we obtain the following:

PROPOSITION 5.5. *Evaluation for* 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 Proposition 3.2 (since $\preceq_{\text{pref}}$ is in REG_2), and in the second one from Proposition 5.4. But adding both relations at the same time destroys such tractability.

5.3. A general condition for tractability in CRPQ(EQ)

While the proof of Proposition 5.4 is simple, it does not provide us with a uniform condition for explaining when the evaluation (a fragment of) a logic of the form CRPQ($\mathcal{S}$), for $\mathcal{S} \subseteq \mathsf{EQ}$, is tractable in data complexity. We provide here one possible such condition. Our condition imposes that the fragment of CRPQ(S) we consider only includes formulas of the form

$$\exists \bar{y} \big(\bigwedge_{i=1}^{m} (u_i \xrightarrow{\chi_i : L_i} u'_i) \wedge \bigwedge_{(i,j) \in I} S(\chi_i, \chi_j) \big)$$

for which the following holds: Let $e(I, \lambda)$ be the word equation that represents I and λ, for λ the mapping that assigns relation S to each element in I, as constructed in the proof of Proposition 4.4. Then $e(I, \lambda)$ only allows for a finite number of principal

solutions. Moreover, when this holds for each index set $I \subseteq [m]^2$ we can conclude that the whole logic $\mathrm{CRPQ}(S)$ is tractable in data complexity. We explain this below.

5.3.1. Finite number of principal solutions. For the sake of convenience, we assume from now on that word equations e are expressions of the form $\phi = \psi$, 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 interpreted, is part of the definition of e. In fact, we freely interpret e over any alphabet extending the set of constants that are mentioned in the equation. This is convenient for defining and working with *principal solutions* of word equations. In a precise formal sense (stated in Claim 5.9), such principal solutions can be seen as the "generators" of all particular solutions of a word equation with regular constraints over every given alphabet Σ.

Let e be a word equation and h_1, h_2 two solutions for e over Σ_1 and Σ_2, respectively. We say that 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 $\alpha(w) = \alpha(a_1) \dots \alpha(a_n)$. (We can thus identify the morphism α with its restriction to Σ_1). The morphism α is continuous if it doesn't delete symbols, i.e., $\alpha(a) \neq \varepsilon$ for each $a \in \Sigma$. A solution h for e is said to be *principal* [Abdulrab and Pécuchet 1989] if it is divided by no other solution but itself (up to isomorphism). It is known that each word equation that has a solution has a principal solution [Lothaire 1997]. We denote by $\mathcal{E}_{\mathsf{fin}}$ the class of word equations e with only a finite number of principal solutions up to isomorphism.

Example 5.6. The word equation $x = yz$ (which defines the suffix relation on the pair (x, z)) is in $\mathcal{E}_{\mathsf{fin}}$. In fact, its only principal solution h is the one that satisfies $h(x) = ab$, $h(y) = a$ and $h(z) = b$. The same holds for the word equation $x = yzw$ (which defines the subword relation on (x, z)).

Let us consider the set of word equations $\{x = uv, y = vu\}$, which defines on (x, y) the conjugacy relation $\sim$ introduced in Example 4.2. This word equation is also in $\mathcal{E}_{\mathsf{fin}}$. In fact, its only principal solution is $h(x) = ab$, $h(y) = ba$, $h(u) = a$ and $h(v) = b$.

On the other hand, the *commutativity* equation $xy = yx$ does not belong to $\mathcal{E}_{\mathsf{fin}}$; 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 $h_{m,n}(y) = a^n$. □

The fact that the word equations $x = yz$ and $x = yzw$ (which define the suffix and subword relations, respectively) have only finitely many principal solutions is part of a more general phenomenon: Word equations without constants and that do not repeat variables (such as $x = yz$ and $x = yzw$) always have a finite number of principal solutions. This is formalized in the following lemma, which will be useful for the rest of our proof. While the lemma uses standard techniques, we have not been able to find it in the literature.

LEMMA 5.7. *Let e be a word equation of the form $\phi = \psi$ that contains neither constants nor repeated variables. Then: (1) e has only finitely many principal solutions, and (2) for each principal solution of the form $h : X \to \Delta^*$, where Δ is a finite alphabet, it is the case that $h(\phi) = h(\psi)$ contains no repeated symbols.*

PROOF. It is known that by iteratively guessing the relative lengths of words to be associated with variables, and solving an equation in a left-to-right fashion by eliminating common prefixes (also known as Lentin's *pig-pug procedure* [Lentin 1972]), one can produce all principal solutions to a word equation without constants. When the equation admits a finite number of principal solutions, the procedure terminates and generates them all [Lentin 1972; Abdulrab and Pécuchet 1989].

We follow this method and exploit the fact that the word equations in question have no repetitions of variables to achieve the desired properties of the lemma. Let e be a word equation of the form $\psi = \phi$ such that $\psi \cdot \phi$ contains neither constants nor repeated variables. Let us notice first that it suffices to prove the claim for the class of principal solutions that can only map variables to non-empty words (the so-called *non-erasing* principal solutions). For general principal solutions, one can first choose a subset Y of the variables in e and replace all occurrences of variables from Y in e by the empty word, thus obtaining a new word equation e' whose non-erasing principal solutions correspond to the general principal solutions of e in which exactly the variables in Y are assigned the empty word.

The procedure to obtain the non-erasing principal solutions of e is as follows:

(1) Let $\phi = x\phi'$, $\psi = y\psi'$, for variables x, y.
(2) Let z be a fresh variable. Then apply one of the following transformations to e:
    (a) $x \leftarrow yz$,
    (b) $x \leftarrow y$, or
    (c) $y \leftarrow xz$.
(3) After applying this transformation, e can be reduced to a word equation e' by removing the common prefixes of both its sides. According to which transformation was used, the following are possible:
    (a) e' is $z\phi' = \psi'$,
    (b) e' is $\phi' = \psi'$, or
    (c) e' is $\phi' = z\psi'$.
(4) If both sides of e' are non-empty, repeat this procedure for e'. If only one of the sides is the empty word, then reject. If both sides are the empty word, accept.

Since e has no repetition of symbols, the word equation e' obtained in step (3) has no repeated symbols either, and, furthermore, $|e'| < |e|$. Indeed, in step (2) any chosen transformation will only be applied once. Two of these transformations introduce a new variable z to the equation, hence enlarging its length by 1. However, all of them remove both x and y from e. Overall, we have that $|e'| \leq |e| - 1$. We conclude inductively that this process always ends in at most $|e|$ steps, regardless of the transformation chosen in step (2).

Principal solutions can finally be obtained from the sequence of transformations used in an accepting run of this procedure. Let Y be the set of variables mentioned in e and h the identity over the variables in Y. For a variable $y \in Y$, iteratively replace symbols in $h(y)$ with the assignments they receive after applying the transformations described in step (2), in the order these transformations where chosen in the accepting run, until no further replacements can be applied. When this is done for all variables, the resulting mapping h is a principal solution for e. Notice that since none of the word equations generated by the process has repeated variables, each variable can be transformed at most once. Furthermore, when a variable is transformed yielding a fresh variable z, then this z appears always as a suffix of one of the words in the equation. If z is later transformed, it can only be replaced by variables that are disjoint from the ones that have been used in the assignments to previous variables. It follows that $h(\phi) = h(\psi)$ has no repeated symbols. □

Next we establish the good behavior of $\mathcal{E}_{\mathsf{fin}}$ in our context. Recall that $\mathsf{WE}(e)$ is the problem of evaluating the fixed word equation e under given regular constraints. We prove that $\mathsf{WE}(e)$ can be solved in NLOGSPACE every time e belongs to $\mathcal{E}_{\mathsf{fin}}$.

THEOREM 5.8. $\mathsf{WE}(e)$ is in NLOGSPACE, for each $e \in \mathcal{E}_{\mathsf{fin}}$.

PROOF. Consider an input to $\mathsf{WE}(e)$ 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. Let Y and Σ_0 be the set of variables and constants mentioned in e, respectively. We start by proving the following claim, which states that checking for the existence of a solution for (e, ν) over Σ can be reduced to checking for the existence of a principal solution for e and of a particular morphism from such principal solution to Σ^*:

CLAIM 5.9. *It is the case that (e, ν) has a solution over Σ if and only if it has a solution of the form $h = \alpha \circ h'$, where $h' : Y \to \Delta^*$ is a principal solution for e and $\alpha : \Delta^* \to \Sigma^*$ is a morphism satisfying $\alpha(h'(y)) \in \nu(y)$ for each $y \in Y$.*

PROOF. The *if* direction follows immediately since $\alpha \circ h' : Y \to \Sigma^*$ solves both e and the regular constraints given by ν. For the *only if* direction, let $h : Y \to \Sigma^*$ be a solution for (e, ν) over Σ. If h is already a principal solution for e then we are done, since then it suffices to consider $h' = h$ and α equal to the identity. Assume otherwise that h is not principal. Since any solution of e is divided by some principal solution, there is a principal solution $h' : Y \to \Delta^*$ for e and a continuous morphism $\alpha : \Delta^* \to \Sigma^*$ such that $h = \alpha \circ h'$. Further, since h is a solution for (e, ν) it is the case that $h(y) = \alpha(h'(y)) \in \nu(y)$ for each $y \in Y$. This concludes the proof of the claim. □

We now continue with the proof of Theorem 5.8. Claim 5.9 states that our problem can be reduced to checking whether some principal solution $h : Y \to \Delta^*$ for e, where Δ is a finite alphabet, admits a morphism $\alpha : \Delta^* \to \Sigma^*$ such that $\alpha(h(y)) \in \nu(y)$ for each $y \in Y$. The set of principal solutions for e is not only finite but fixed (since e itself is fixed).[3] It remains to show that there is an NLOGSPACE procedure that, given a fixed principal solution $h : Y \to \Delta^*$ for e, checks whether there is a morphism $\alpha : \Delta^* \to \Sigma^*$ such that $\alpha(h(y)) \in \nu(y)$ for each $y \in Y$. We illustrate this procedure with an example.

Example 5.10. Suppose that $Y = \{x, y, z\}$, $\Delta = \{a, b, c\}$, and h satisfies the following: (1) $h(x) = abc$, (2) $h(y) = b$, and (3) $h(z) = ac$. Let us assume 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 (1) q_0, r_0, s_0 are initial states of $\nu(x)$, $\nu(y)$, and $\nu(z)$, respectively, (2) q_3, r_1, s_2 are final states of $\nu(x)$, $\nu(y)$, and $\nu(z)$, respectively, and (3) 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 $\alpha \circ h$ over Σ, where $\alpha : \Delta^* \to \Sigma^*$ is the morphism that satisfies the following: (1) $h'(x) = w_1 w_2 w_3$, (2) $h'(y) = w_2$, and (3) $h'(z) = w_1 w_3$. 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. We develop these ideas in the rest of the proof. □

Fix a principal solution $h : Y \to \Delta^*$ for e, where Δ is a finite alphabet. Observe that it may be the case that $\Delta \cap \Sigma_0 \neq \emptyset$. Let m be the maximum length of a word of the form $h(y)$, for $y \in Y$. We define a set $\sigma(d) \subseteq Y \times \{1, \ldots, m\}$, for each $d \in \Delta$, such that $(y, i) \in \sigma(d)$ if and only if the i-th symbol of $h(y)$ is d. We then define for each $d \in \Delta \setminus \Sigma_0$

[3] Furthermore, the principal solutions can be generated in this case by applying, e.g., Lentin's *pig-pug* procedure [Lentin 1972]. But this is not needed for our proof.

an NFA L_d over Σ such that:

$$L_d \;=\; \prod_{(y,i)\in\sigma(d)} \nu(y),$$

i.e., L_d is the product of all NFAs of the form $\nu(y)$ such that $\sigma(d)$ contains a pair of the form (y,i). Notice that each state in L_d has as many *components* as the number $|\sigma(d)|$ of appearances of the symbol $d \in \Delta$ in the set of words $\{h(y) \mid y \in Y\}$. Since e, and thus $|\sigma(d)|$, is fixed, this implies the following: (a) each L_d is of polynomial size in the input, (b) each state in L_d can be encoded using only logarithmic space, and (c) checking whether there is a transition from state q to state q' in L_d can be done in logarithmic space.

We need to decide whether the symbols $d \in \Delta \setminus \Sigma_0$ can be mapped to words in Σ^* by a morphism α, with $\alpha(x) = x$ for each $x \in \Sigma_0$, such that $\alpha \circ h$ solves each constraint imposed by ν. The basic idea of the algorithm is to assign a *reachability task* to each symbol $d \in \Delta \setminus \Sigma_0$. Such reachability task amounts simply to checking whether in L_d a certain state q' can be reached from another state q (i.e., there is a word $w \in \Sigma^*$ such that L_d can read w from state q to q').

Formally, the algorithm proceeds as follows. For each variable $y \in Y$ it guesses a sequence

$$q_0^y, \ldots, q_{|h(y)|}^y$$

of states in $\nu(y)$ such that q_0^y is the initial state of $\nu(y)$ and $q_{|h(y)|}^y$ is a final state of $\nu(y)$. For each $d \in \Delta$ we then assign the following reachability task to L_d. Let $(y_1,i_1),(y_2,i_2),\ldots,(y_p,i_p)$ be an enumeration of all pairs $(y,i) \in \sigma(d)$. Then do the following for each $d \in \Delta$:

(1) if $d \in \Sigma_0$, for each $(y_j,i_j) \in \sigma(d)$, check whether there is a transition $(q_{i_j-1}^{y_j}, d, q_{i_j}^{y_j})$ in $\nu(y_j)$
(2) if $d \in \Delta \setminus \Sigma_0$, check whether in L_d the state

$$\big(\, q_{i_1}^{y_1},\, q_{i_2}^{y_2},\, \ldots,\, q_{i_p}^{y_p} \,\big)$$

is reachable from the state

$$\big(\, q_{i_1-1}^{y_1},\, q_{i_2-1}^{y_2},\, \ldots,\, q_{i_p-1}^{y_p} \,\big)$$

(In other words, check if there is $w \in \Sigma^*$ such that for each $1 \le j \le p$ the word w can be read in $\nu(y_j)$ from $q_{i_j-1}^{y_j}$ to $q_{i_j}^{y_j}$).

The algorithm then accepts if and only if each such reachability task holds.

It is not hard to see that the algorithm runs in NLOGSPACE. In fact, notice that the reachability task assigned to L_d is no other than the usual reachability over the "directed graph" defined by L_d. Since reachability can be solved in NLOGSPACE over directed graphs, our remarks (a)-(b)-(c) stated above show that the reachability task over L_d can be solved in NLOGSPACE. Further, since Δ is fixed, all reachability tasks can be performed in NLOGSPACE. Furthermore, each guessed sequence $q_0^y, \ldots, q_{|h(y)|}^y \in \nu(y)$ of states is of length bounded by the fixed parameter m, and thus can be represented using logarithmic space.

To finish the proof we only need to show that the previous procedure accepts the principal solution $h : Y \to \Delta^*$ if and only if there is a morphism $\alpha : \Delta^* \to \Sigma^*$ such that $\alpha(h(y)) \in \nu(y)$ for each $y \in Y$. For the *if* direction, let $w(d) := \alpha(d)$ for each $d \in \Delta$, and assume that $h(y) = d_1^y \ldots d_{|h(y)|}^y \in \Delta^*$, for each $y \in Y$. Since $\alpha(h(y)) \in \nu(y)$, there is an accepting run ρ of $\nu(y)$ over $w(d_1^y) \ldots w(d_{|h(y)|}^y) \in \Sigma^*$. Let us assume that $q_0^y, q_1^y, \ldots, q_{|h(y)|}^y$

are the states assigned by ρ to positions $0, |w(d_1^y)|, \ldots, |w(d_{|h(y)|}^y)|$, respectively. Notice that if $(y,i) \in \sigma(d)$ then there is a run of $\nu(y)$ over $w(d)$ from state q_{i-1}^y to q_i^y.

We claim that the algorithm accepts the guess that assigns the sequence $q_0^y, q_1^y, \ldots, q_{|h(y)|}^y$ of states in $\nu(y)$ to each $y \in Y$. We only need to show then that the reachability task assigned to each $d \in \Delta$ is satisfied. If $d \in \Delta \setminus \Sigma_0$, it is not hard to see from the way in which L_d is defined that such reachability task not only holds but can be witnessed by the word $w(d)$. In fact, let $(y_1,i_1),(y_2,i_2),\ldots,(y_p,i_p)$ be an enumeration of all pairs $(y,i) \in \sigma(d)$. Then we need to check whether in L_d the state $(q_{i_1}^{y_1}, q_{i_2}^{y_2}, \ldots, q_{i_p}^{y_p})$ is reachable from the state $(q_{i_1-1}^{y_1}, q_{i_2-1}^{y_2}, \ldots, q_{i_p-1}^{y_p})$. By definition of L_d, this boils down to showing that there is a word $w \in \Sigma^*$ such that for each $1 \leq j \leq p$ the state $q_{i_j}^{y_j}$ is reachable from $q_{i_j-1}^{y_j}$ in $\nu(y_j)$ by reading word w. But this clearly holds since for each $1 \leq j \leq p$ it is the case that $q_{i_j}^{y_j}$ is reachable from $q_{i_j-1}^{y_j}$ in $\nu(y_j)$ by reading the word $w(d)$. On the other hand, if $d \in \Sigma_0$, the accepting run ρ must have used a transition labeled by d when jumping from state q_{i-1}^y to q_i^y, as in this case we have that $w(d) = d$. Hence, in any case the reachability tasks are satisfied.

For the *only if* direction, let us assume that the algorithm accepts, i.e., there are sequences $q_0^y, \ldots, q_{|h(y)|}^y$ of states in $\nu(y)$, for $y \in Y$, such that the reachability task holds for each $d \in \Delta$. Thus, for each $d \in \Delta$ there is a word $w(d) \in \Sigma^*$ ($w(d) = d$ if $d \in \Sigma_0$) which can be read in $\nu(y)$ from q_{i-1}^y to q_i^y for each pair $(y,i) \in \sigma(d)$. Let us define a morphism $\alpha : \Delta \to \Sigma^*$ such that $\alpha(d) = w(d)$ for each $d \in \Delta$. We prove next that $\alpha(h(y)) \in \nu(y)$ for each $y \in Y$. Let $h(y)$ be the word $d_1 \ldots d_{|h(y)|} \in \Delta^*$ for an arbitrary $y \in Y$. Then $\alpha(h(y)) = w(d_1) \ldots w(d_{|h(y)|}) \in \Sigma^*$. Since $(y,i) \in \sigma(d_i)$ for each $1 \leq i \leq |h(y)|$, the word $w(d_i)$ can be read in $\nu(y)$ from q_{i-1}^y to q_i^y. Since by definition q_0^y and $q_{|h(y)|}^y$ are an initial and final state of $\nu(y)$, respectively, there is an accepting run of $\nu(y)$ over $\alpha(h(y)) = w(d_1) \ldots w(d_{|h(y)|})$. □

5.3.2. The class $\mathcal{E}_{fin}$ and the tractable evaluation of logics. Let us recall from Proposition 4.4 that there is a polynomial time reduction 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 $(e(I,\lambda),\nu)$ over Σ such that $(L_1,\ldots,L_m,,I,\lambda) \in \text{GENINT}(\mathsf{EQ})$ if and only if $(e(I,\lambda),\nu)$ has a solution. Interestingly, this result can be strengthened. In fact, by inspecting the proof of Proposition 4.4 it is clear that the reduction from $(L_1,\ldots,L_m,,I,\lambda)$ to $(e(I,\lambda),\nu)$ can be computed in LOGSPACE.

Let S be a binary relation in EQ, i.e., S is definable by a word equation by taking the projection over an ordered pair (x,y) of variables. Let $I \subseteq [m]^2$ be an index set. Consider an input $(L_1,\ldots,L_m,\lambda)$ to $\text{GENINT}_I(S)$. Notice that we can assume without loss of generality that $\lambda(i,j) = \{S\}$ for each $(i,j) \in I$, and thus λ is also fixed in this case. From our previous remarks, there is a LOGSPACE translation that constructs a word equation with regular constraints $(e(I,\lambda),\nu)$ such that $(L_1,\ldots,L_m,\lambda) \in \text{GENINT}_I(\mathsf{EQ})$ if and only if $(e(I,\lambda),\nu)$ has a solution. Since $e(I,\lambda)$ only depends on I and λ, it can be considered to be fixed. From now on we denote such equation by $e(I,S)$, in order to be explicit about its dependence on I and S only (since λ is determined by I and S). By combining this with Therorem 4.1 we immediately obtain the following:

PROPOSITION 5.11. *Let S be a relation in* EQ *and $I \subseteq [m]^2$ an index set such that $e(I,S) \in \mathcal{E}_{fin}$. Then* $\text{GENINT}_I(S)$ *can be solved in* NLOGSPACE.

By combining Proposition 5.11 and the proof of Lemma 3.4 we obtain the following important corollary:

COROLLARY 5.12. *Let S be a relation in* EQ *and $\mathcal{K}$ a class of* $\mathrm{CRPQ}(S)$ *formulas of the form*

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

such that $e(I,S) \in \mathcal{E}_{\mathsf{fin}}$. Then the evaluation problem for the formulas in $\mathcal{K}$ is in NLOGSPACE *in data complexity (i.e., each fixed formula in $\mathcal{K}$ can be evaluated in* NLOGSPACE*). In particular, if $e(I,S)$ is in $\mathcal{E}_{\mathsf{fin}}$ for each index set $I \subseteq [m]^2$ the evaluation for* $\mathrm{CRPQ}(S)$ *is in* NLOGSPACE *in data complexity.*

PROOF. Consider a formula $\phi(\bar{x})$ in $\mathcal{C}$ of the form $\exists \bar{y}(\bigwedge_{i=1}^{m}(u_i \xrightarrow{\chi_i:L_i} u_i') \wedge \bigwedge_{(i,j)\in I} S(\chi_i,\chi_j))$. Proposition 5.11 states that $\textsc{GenInt}_I(S)$ is in NLOGSPACE. By applying the proof of the first item of Lemma 3.4 we obtain that the evaluation problem for $\phi(\bar{x})$ is in NLOGSPACE. □

5.3.3. Tractable relations in EQ. Corollary 5.12 provides us with a semantic criterion for showing that a logic of the form CRPQ(S), for $S \in$ EQ, is tractable in data complexity: We only need to prove that $e(I,S)$ is in $\mathcal{E}_{\mathsf{fin}}$ for each index set $I \subseteq [m]^2$. We establish next the robustness of this notion by showing that it holds for $S = \preceq_{\mathrm{suff}}$ (which provides an alternative proof, based on our semantic criterion, to Proposition 5.4).

PROPOSITION 5.13. *For each index set $I \subseteq [m] \times [m]$ the word equation $e(I, \preceq_{\mathrm{suff}})$ is in $\mathcal{E}_{\mathsf{fin}}$. Hence, evaluation for* $\mathrm{CRPQ}(\preceq_{\mathrm{suff}})$ *is in* NLOGSPACE *in data complexity.*

PROOF. Recall that the suffix relation $x_1 \preceq_{\mathrm{suff}} x_2$ is definable by the word equation $e_{\preceq_{\mathrm{suff}}} := (x_2 = px_1)$. Remember from the proof of Proposition 4.4 that $e(I, \preceq_{\mathrm{suff}})$ is defined by the system of word equations $E = \{(x_j = p_{i,j}x_i) : (i,j) \in I\}$. It is convenient to represent E as a single word equation. This corresponds to the concatenation of the left- and right-hand sides of the word equations in E, respectively, properly separated by a constant delimiter $\#$, in any ordering of the index set I. Consider one such ordering $\{(i_1,j_1),\ldots,(i_\ell,j_\ell)\}$ of I. The equation defining $e(I, \preceq_{\mathrm{suff}})$ corresponds then to:

$$\phi_1\# \cdots \#\phi_\ell \quad = \quad \psi_1\# \cdots \#\psi_\ell \tag{3}$$

where $\phi_k = x_{j_k}$ and $\psi_k = p_k x_{i_k}$, for all $1 \leq k \leq \ell$. We assume from now, without loss of generality, that $\#$ is only a distinguished delimiter that does not appear in the solutions of $\phi_1\# \cdots \#\phi_\ell = \psi_1\# \cdots \#\psi_\ell$ (recall that such solutions are only defined on the variables of such equation, and that we assume that such solutions correspond to the identity on constants, e.g., over $\#$).

Let us define e_0 as the trivial word equation $\epsilon = \epsilon$, and e_k (for $1 \leq k \leq \ell$) as the word equation:

$$\phi_1\# \cdots \#\phi_k \quad = \quad \psi_1\# \cdots \#\psi_k.$$

Further, let Y_k be the set of variables in e_k. The next lemma states an important property regarding the form of the principal solutions of e_k, for $1 \leq k \leq \ell$.

LEMMA 5.14. *Let $h : Y_k \to \Delta^*$ be a principal solution of e_k, for $1 \leq k \leq \ell$. (In particular, $\#$ does not belong to Δ). Then $h = g \circ h_0$, where $h_0 : Y_{k-1} \to \Delta_0^*$ is a principal solution of e_{k-1} and $g : (Y_k \setminus Y_{k-1}) \cup \Delta_0 \to \Delta^*$ is a principal solution of $h_0(\phi_k) = h_0(\psi_k)$ (assuming h_0 to be the identity on $Y_k \setminus Y_{k-1}$).*

PROOF. We prove the lemma inductively for $1 \leq k \leq \ell$. We start with the basis case $k = 1$. Since e_0 is the trivial word equation $\epsilon = \epsilon$, its only principal solution is the empty mapping h_0. The property thus trivially follows.

We now prove for $k+1$ assuming that the property holds for $1 \leq k < \ell$ by inductive hypothesis. Let $h : Y_{k+1} \to \Delta^*$ be a principal solution of e_{k+1}. It is then easy to see that h is also a solution to the equation e_k. If h is a principal solution of e_k, then it suffices to define (1) $h_0 : Y_k \to \Delta^*$ by letting h_0 to be h over Y_k and the identity over $Y_{k+1} \setminus Y_k$, and (2) $g : (Y_{k+1} \setminus Y_k) \cup \Delta \to \Delta^*$ by letting $g(x) = h(x)$ if $x \in (Y_{k+1} \setminus Y_k)$, and $g(x) = x$ if $x \in \Delta$. Then it follows that $h = g \circ h_0$. Recall that we have assumed that $\#$ is a special delimiter not used by solutions, and thus that $h(x)$ contains the symbol $\#$ iff $x = \#$. Along with the fact that h is a solution for e_{k+1}, this implies that g is a solution of the equation $h_0(\phi_{k+1}) = h_0(\psi_{k+1})$. It is also a principal solution because otherwise h would not be a principal solution of e_{k+1}.

So now let us assume that h is not a principal solution of e_k. It follows then that there is a principal solution $h_0 : Y_k \to \Delta_0^*$ and a continuous morphism $\alpha : \Delta_0^* \to \Delta^*$ such that h corresponds to $\alpha \circ h_0$ over Y_k. We can assume without loss of generality that $(Y_{k+1} \setminus Y_k) \cap \Delta_0 = \emptyset$, since symbols from $Y_{k+1} \setminus Y_k$ are not mentioned in the equation e_k. Consider now the morphism $g : (Y_{k+1} \setminus Y_k) \cup \Delta_0 \to \Delta^*$ such that g corresponds to h over $Y_{k+1} \setminus Y_k$ and to α over Δ_0. Notice that g is well-defined since $(Y_{k+1} \setminus Y_k) \cap \Delta_0 = \emptyset$. Furthermore, $g \circ h_0$ is precisely h. In fact, if $y \in Y_{k+1} \setminus Y_k$ then $g(h_0(y)) = g(y) = h(y)$. If $y \in Y_k$ then $h_0(y) \in \Delta_0^*$, and therefore $g(h_0(y)) = \alpha(h_0(y)) = h(y)$. Thus, $g \circ h_0$ is a solution of e_{k+1} since h is a solution of e_{k+1}. Again, since $h(x)$ contains the symbol $\#$ iff $x = \#$ and h is a solution for e_{k+1} we have that g is a solution of the equation $h_0(\phi_{k+1}) = h_0(\psi_{k+1})$. It is also a principal solution because otherwise h would not be a principal solution of e_{k+1}. □

An important corollary of this lemma is that in order to prove that $e(I, \preceq_{\text{suff}})$ has finitely many principal solutions, it is sufficient to inductively prove the following:

(†) For every $1 \leq k < \ell$ and principal solution $h : Y_k \to \Delta^*$ of e_k, there are only finitely many principal solutions of the word equation $h(\phi_{k+1}) = h(\psi_{k+1})$ (assuming that h is the identity on $Y_{k+1} \setminus Y_k$).

In fact, since e_0 only has one principal solution (the empty mapping), by iteratively applying Lemma 5.14 together with property (†) we obtain that $e(I, \preceq_{\text{suff}}) = e_\ell$ has only finitely many principal solutions.

We finish by proving property (†). Take an arbitrary principal solution $h : Y_k \to \Delta^*$ of e_k, for $1 \leq k < \ell$. By definition, $\phi_{k+1} = \psi_{k+1}$ corresponds to the word equation $x_{j_{k+1}} = p_{k+1} x_{i_{k+1}}$. Furthermore, p_{k+1} does not appear in e_k and therefore $h(p_{k+1}) = p_{k+1}$. Thus, $h(\phi_{k+1}) = h(\psi_{k+1})$ corresponds to the word equation $h(x_{j_{k+1}}) = h(p_{k+1} x_{i_{k+1}})$, which in turn corresponds to the word equation:

$$h(x_{j_{k+1}}) \;=\; p_{k+1} h(x_{i_{k+1}}).$$

We prove next that this word equation has only finitely many principal solutions. Solutions of this equation are mappings from the set of symbols that appear in it (i.e., $(Y_{k+1} \setminus Y_k) \cup \Delta$) to a fresh alphabet Δ'. This means that all symbols in $h(x_{j_{k+1}}) = p_{k+1} h(x_{i_{k+1}})$ can be regarded as variables and, thus, the equation contains no constants. Therefore, in virtue of Lemma 5.7 we only need to consider the case when there are repeated variables in the equation. The proof in this case relies on the following important property:

(††) Neither $h(x_{j_{k+1}})$ nor $h(x_{i_{k+1}})$ has repeated variables. On the other hand, assume that $h(x_{j_{k+1}})$ and $h(x_{i_{k+1}})$ share at least one variable, i.e., for some variable z it is the case that:

$$h(x_{j_{k+1}}) = \alpha z \beta \;\text{ and }\; h(x_{i_{k+1}}) = \alpha' z \beta',$$

where $\alpha, \beta, \alpha', \beta'$ are words over $(Y_{k+1} \setminus Y_k) \cup \Delta$. Then $\beta = \beta'$.

What the property intuitively states is the following: Repeated variables in $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$ do not appear on the same side of the equation. Moreover, if a variable appears both in $h(x_{j_{k+1}})$ and $h(x_{i_{k+1}})$ then it completely determines its suffix in such words.

Before proving property (††), we show how it implies that $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$ has only finitely many principal solutions in case the equation contains repeated variables. In fact, from (††) any such repeated variable z appears both in $h(x_{j_{k+1}})$ and $h(x_{i_{k+1}})$ and completely determines its suffix in such words. This means that $h(x_{j_{k+1}}) = \alpha z \beta$ and $h(x_{i_{k+1}}) = \alpha' z \beta$, for words α, α', β over $(Y_{k+1} \setminus Y_k) \cup \Delta$. We can then "simplify" this equation by getting rid of the common suffixes (i.e., we convert it to $\alpha = \alpha'$). If we do this with the leftmost variable that is shared by both words we will end up with a word equation e which has no repeated variables. This word equation e has only finitely many principal solutions due to Lemma 5.7. Any principal solution h' of $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$ is then obtained from a principal solution h'' of e as follows: (1) $h' = h''$ on the variables of e, and (2) h' is a fresh renaming of the variables in $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$ that do not appear in e. It follows that $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$ has finitely many principal solutions, as there is only one way (up to renaming) of extending a principal solution of e to obtain a principal solution of $h(x_{j_{k+1}}) = p_{k+1}h(x_{i_{k+1}})$.

We finish by proving property (††) inductively. We actually need to prove something stronger for the inductive hypothesis to hold. This is stated in the following claim:

CLAIM 5.15. *Let $h : Y_k \to \Delta^*$ be a principal solution of e_k for $0 \leq k < \ell$ and S_k^h the set of words defined as $\bigcup_{k<k'\leq\ell}\{h(\phi_{k'}), h(\psi_{k'})\}$. The following holds:*

(1) *No word $s \in S_k^h$ has repeated variables.*
(2) *For all words $s, s' \in S_k^h$, if there is a variable z and words $\alpha, \beta, \alpha', \beta'$ such that $s = \alpha z \beta$ and $s' = \alpha' z \beta'$, then $\beta = \beta'$ (that is, suffixes of words in S_k^h are completely determined by their first symbol).*

We now prove Claim 5.15 by induction on $0 \leq k \leq \ell$. For $k = 0$ we have e_0 to be the trivial word equation $\epsilon = \epsilon$, whose only principal solution h is the empty mapping. Consider the set $S_0^h = \bigcup_{1\leq k\leq\ell}\{\phi_k, \psi_k\}$. Condition (1) holds trivially since neither ϕ_k nor ψ_k, for $1 \leq k \leq \ell$, has repeated variables. Condition (2) also holds trivially since if a variable appears in two words in the set $S_0^h = \bigcup_{1\leq k\leq\ell}\{\phi_k, \psi_k\}$ then it appears in the last position of such words. This is because $\phi_k = x_{j_k}$ and $\psi_k = p_k x_{i_k}$ for each $1 \leq k \leq \ell$.

We now prove for $1 \leq k \leq \ell$ assuming that the claim holds by induction hypothesis for $k-1$. Take an arbitrary principal solution $h : Y_k \to \Delta^*$ for e_k. From Lemma 5.14 it follows that h is of the form $g \circ h_0$, where $h_0 : Y_{k-1} \to \Delta_0^*$ is a principal solution for e_{k-1} and $g : (Y_k \setminus Y_{k-1}) \cup \Delta_0 \to \Delta^*$ is a principal solution for $h_0(\phi_k) = h_0(\psi_k)$. We prove next that the conditions of the claim are satisfied.

Consider the set $S_k^h = \bigcup_{k<k'\leq\ell}\{g(h_0(\phi_k)), g(h_0(\psi_k)\}$. We start by proving condition (1). Take an arbitrary $k' > k$ and let us consider the word $h(\phi_{k'})$ (the analysis for the word $h(\psi_{k'})$ is analogous). We prove next that this word has no repeated variables. By definition, $h(\phi_{k'}) = g(h_0(\phi_{k'}))$. Since h_0 is a principal solution of e_{k-1} and $h_0(\phi_{k'}) \in S_{k-1}^{h_0}$, we have by induction hypothesis that $h_0(\phi_{k'})$ has no repeated variables. Assume first that no variable in $h_0(\phi_{k'})$ appears in $h_0(\phi_k) = h_0(\psi_k)$. Then g is the identity on each variable of $h_0(\phi_{k'})$, from which we conclude that $h(\phi_{k'}) = g(h_0(\phi_{k'}))$ has no repeated symbols. Assume otherwise, and let z be the leftmost variable in $h_0(\phi_{k'})$ that appears in $h_0(\phi_k) = h_0(\psi_k)$. Then by inductive hypothesis (using condition (2)), we have that $h_0(\phi_{k'})$ is of the form $\alpha z \beta$, where α does not mention variables from the equation $h_0(\phi_k) = h_0(\psi_k)$ and β is a suffix of either $h_0(\phi_k)$ or $h_0(\psi_k)$ (because $\{h_0(\phi_{k'}), h_0(\phi_k), h_0(\psi_k)\} \subseteq S_{k-1}^{h_0}$ and shared variables between words in $S_{k-1}^{h_0}$ uniquely

determine the suffixes of such words). It follows that $g(h_0(\phi_{k'})) = g(\alpha)g(z\beta) = \alpha g(z\beta)$. This word has no repeated symbols. In fact, α is a prefix of $h_0(\phi_{k'})$, which by induction has no repeated symbols. On the other end, $g(z\beta)$ is a suffix of $g(h_0(\phi_k)) = g(h_0(\psi_k))$, which has no repeated variables for the reasons we explain next. We have that g is a principal solution of the equation $h_0(\phi_k) = h_0(\psi_k)$. According to the inductive hypothesis, these words can be written as $h_0(\phi_k) = \alpha\beta$ and $h_0(\psi_k) = \alpha'\beta$, respectively, where netiher α, α', nor β repeats symbols, α does not share symbols with α', and β is the largest common suffix of $h_0(\phi_k)$ and $h_0(\psi_k)$. This means the word equation $h_0(\phi_k) = h_0(\psi_k)$ corresponds (after simplification) to $\alpha = \alpha'$, which is an equation without repetition of variables and variables from the left hand side are disjoint from those on the right hand side. Therefore, the principal solution g of $h_0(\phi_k) = h_0(\psi_k)$ corresponds to a principal solution g' of $\alpha = \alpha'$ by assigning fresh renaiming to those symbols mentioned in $h_0(\phi_k) = h_0(\psi_k)$ but not in $\alpha = \alpha'$. From Lemma 5.7, we have that $g'(\alpha)$ does not repeat symbols. Since $g(h_0(\phi_k)) = g(\alpha\beta) = g(\alpha)\hat{\beta}$ we have that $g(h_0(\phi_k))$ does not repeat symbols as claimed.

It remains to prove condition (2). Consider words $s, s' \in S^h_k$ and assume that they share a symbol z. Since $s, s' \in S^h_k$, we know that there are words $s_0, s'_0 \in \bigcup_{k'>k}\{h_0(\phi_{k'}), h_0(\psi_{k'})\}$ such that $s = g(s_0)$ and $s' = g(s'_0)$. Also, since $s_0 \in S^{h_0}_{k-1}$ we have by induction that there are words α_0, β_0 such that $s_0 = \alpha_0\beta_0$, where α_0 is a word only composed by variables that do not appear in $h_0(\phi_k) = h_0(\psi_k)$ and β_0 is a (possibly empty) suffix of either $h_0(\phi_k)$ or $h_0(\psi_k)$. This suffix β_0 is empty if and only if the word s_0 mentions no variable from the equation $h_0(\phi_k) = h_0(\psi_k)$. Analogously, s'_0 can be decomposed as $\alpha'_0\beta'_0$ under the same conditions. From this we conclude that $s = g(s_0) = g(\alpha_0\beta_0) = \alpha_0 g(\beta_0)$ and $s' = g(s'_0) = \alpha'_0 g(\beta'_0)$. We need to make a case analysis relative to where in the words s and s' the shared symbol z appears. Since g ranges over fresh symbols (i.e., symbols outside $h_0(\phi_1\# \cdots \#\phi_\ell) = h_0(\psi_1\# \cdots \#\psi_\ell)$) there are only two possibilities:

(1) Symbol z appears in α_0 and α'_0, that is, $\alpha_0 = uzv$ and $\alpha'_0 = u'zv'$ for some words u, v, u', v'. Since (a) $s_0 = \alpha_0\beta_0 = uzv\beta_0$, (b) $s'_0 = u'zv'\beta'_0$, and (c) $s_0, s'_0 \in S^{h_0}_{k-1}$, by induction we obtain that $v\beta_0 = v'\beta'_0$. It follows that $vg(\beta_0) = v'g(\beta'_0)$, since v does not mention symbols from the equation $h_0(\phi_k) = h_0(\psi_k)$ which g solves. Plugging this into the expressions we had for s and s' we obtain that $s = uzvg(\beta_0)$ and $s_0 = u'zv'g(\beta'_0)$. Hence the suffixes $s = g(s_0)$ and $s' = g(s'_0)$ starting at symbol z, which correspond to $zvg(\beta_0)$ and $zv'g(\beta'_0)$, respectively, are equal as claimed.
(2) Symbol z appears in $g(\beta_0)$ and $g(\beta'_0)$. Since both β_0 and β'_0 are suffixes of either $h_0(\phi_k)$ or $h_0(\psi_k)$, and g is a principal solution of the equation $h_0(\phi_k) = h_0(\psi_k)$, it follows that one of the words $g(\beta_0)$ or $g(\beta'_0)$ is a suffix of the other. Furthermore, for the same reasons it is the case that neither $g(\beta_0)$ nor $g(\beta'_0)$ repeat symbols, and hence in each the symbol z appears once. It follows that the suffixes of these words starting at symbol z must be equal.

This concludes the proof of Claim 5.15. □

It is worth noticing the difference between $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ in this context: As mentioned in Example 5.6, both relations can be defined by equations in $\mathcal{E}_{\text{fin}}$. The difference is that $e(I, \preceq_{\text{suff}})$ is in $\mathcal{E}_{\text{fin}}$ for each index set $I \subseteq [m]^2$ (Proposition 5.13), while there is an index set I such that $e(I, \preceq_{\text{sw}})$ is not in $\mathcal{E}_{\text{fin}}$ (namely, $I_\diamond$, as obtained by combining the proof of Proposition 5.2 and Theorem 5.8).

5.3.4. Tractable fragments of logics of the form CRPQ(S). Corollary 5.12 also provides us with a semantic criterion for showing that a fragment $\mathcal{K}$ of a logic of the form CRPQ(S), for $S \in \mathsf{EQ}$, is tractable in data complexity: We only need to prove that $e(I, S)$ is in $\mathcal{E}_{\text{fin}}$

for each index set $I \subseteq [m]^2$ that is mentioned in a formula in $\mathcal{K}$. We use this to show that there is an interesting family of logics of the form $\mathrm{CRPQ}(S)$, for $S \in \mathsf{EQ}$, whose *acyclic fragment* is tractable in data complexity. The acyclic fragment of a logic of the form $\mathrm{CRPQ}(S)$ is the one defined by the formulas of the form

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

where I is acyclic. Recall that an index set $I \subseteq [m]^2$ is acyclic if the undirected graph defined by I on $\{1, \ldots, m\}$ is acyclic.

We define a restriction $\mathcal{E}^r_{\mathsf{fin}}$ of the class $\mathcal{E}_{\mathsf{fin}}$ that is composed by those word equations $e \in \mathcal{E}_{\mathsf{fin}}$ that satisfy the following: for each principal solution $h : X \to \Delta^*$ of e it is the case that no word of the form $h(x)$, for $x \in X$, has repeated symbols. For instance, it follows from Example 5.6 that the word equations that define the subword ($\preceq_{\mathrm{sw}}$) and conjugacy ($\sim$) relations are in $\mathcal{E}^r_{\mathsf{fin}}$. We then prove the following:

PROPOSITION 5.16. *Assume $S \in \mathsf{EQ}$ is definable by a word equation e in $\mathcal{E}^r_{\mathsf{fin}}$ over a pair (x, y) of distinct distinguished variables. Then:*

(*1*) *For each acyclic index set $I \subseteq [m] \times [m]$ the word equation $e(I, S)$ is in $\mathcal{E}_{\mathsf{fin}}$.*
(*2*) *In particular, evaluation for the acyclic fragment of $\mathrm{CRPQ}(S)$ is in NLOGSPACE in data complexity.*

Therefore, as a corollary to Example 5.6 and Proposition 5.16 we obtain that:

COROLLARY 5.17. *Evaluation for the acyclic fragments of $\mathrm{CRPQ}(\preceq_{\mathrm{sw}})$ and $\mathrm{CRPQ}(\sim)$ are in NLOGSPACE in data complexity.*

Some important remarks are in order regarding this corollary:

(1) The tractability in data complexity of the acyclic fragment of $\mathrm{CRPQ}(\preceq_{\mathrm{sw}})$ follows from [Barceló et al. 2013]. In fact, it is proved there that the acyclic fragment of any logic of the form $\mathrm{CRPQ}(R)$, where $R \in \mathsf{RAT}$, is tractable in data complexity. Proposition 5.16 can be seen then as a reformulation of this general result in the context of relations defined by word equations. The difference though is that in this context it is not possible to obtain the result in full generality (as it was the case for RAT), which justifies the syntactic restriction imposed on relations from EQ that we use in the statement of such Proposition.
(2) On the other hand, the tractability in data complexity of the acyclic fragment of $\mathrm{CRPQ}(\sim)$ does not follow from existing results (recall that $\sim$ is in $\mathsf{EQ} \setminus \mathsf{RAT}$).
(3) The acyclicity restriction is, in a sense, optimal. Indeed, recall that $\mathrm{CRPQ}(\preceq_{\mathrm{sw}})$ is intractable in data complexity even if restricted to formulas that are defined over the simple DAG index set $I_\diamond = \{(1,2),(1,3),(2,4),(3,4)\}$. We conjecture that a similar behaviour holds for $\mathrm{CRPQ}(\sim)$.

We finish by proving Proposition 5.16:

PROOF OF PROPOSITION 5.16. Assume that S is definable by the word equation $e := (\phi = \psi)$ in $\mathcal{E}^r_{\mathsf{fin}}$ over the distinguished pair (x, y) of distinct variables. Let $z_1, \ldots, z_p$ be an enumeration of the variables in e that are not distinguished (i.e., those that are existentially quantified). Take an acyclic index set $I \subseteq [m]^2$ of the form $\{(i_1, j_1), \ldots, (i_\ell, j_\ell)\} \subseteq [m] \times [m]$. Let $\{x_{i_k}, x_{j_k}, z^1_{i_k,j_k}, \ldots, z^p_{i_k,j_k} \mid 1 \leq k \leq \ell\}$ be a set of fresh variables (i.e., variables not mentioned in e). Then the word equation $e(I, S)$ is of the form:

$$\phi_1 \# \cdots \# \phi_\ell \;=\; \psi_1 \# \cdots \# \psi_\ell,$$

where for each $1 \leq k \leq \ell$ the words ϕ_k and ψ_k are respectively obtained from ϕ and ψ by (1) replacing the distinguished variables x and y by x_{i_k} and x_{j_k}, respectively, and (2) replacing existential variables $z_1, \ldots, z_p$ with $z^1_{i_k,j_k}, \ldots, z^p_{i_k,j_k}$, respectively.

For $0 \leq k \leq \ell$, let e_k be the word equation $\phi_1 \# \cdots \# \phi_k = \psi_1 \# \cdots \# \psi_k$ and Y_k the set of variables mentioned in e_k (we assume e_0 to be the trivial equation $\epsilon = \epsilon$ and, thus, $Y_0 = \emptyset$). Lemma 5.14 also applies in this case, which implies that for each $1 \leq k \leq \ell$ the principal solutions of e_k are of the form $h = g \circ h_0$, where h_0 is a principal solution of e_{k-1} and g is a principal solution of the equation $h_0(\phi_k) = h_0(\psi_k)$. In order to prove the proposition, it is then sufficient to prove the following:

(†) For every $1 \leq k \leq \ell$ and principal solution $h : Y_{k-1} \to \Delta^*$ of e_{k-1}, there are only finitely many principal solutions of the word equation $h(\phi_k) = h(\psi_k)$ (assuming that h is the identity on $Y_k \setminus Y_{k-1}$).

In order to prove this, we prove by induction the following stronger claim:

CLAIM 5.18. *For every $1 \leq k \leq \ell$ and principal solution $h : Y_{k-1} \to \Delta^*$ of e_{k-1}, it is the case that:*

(*1*) *For every existential variable z_s, where $1 \leq s \leq p$, we have that $h(z^s_{i_k,j_k}) = z^s_{i_k,j_k}$.*
(*2*) *For each t such that $k - 1 < t \leq \ell$, it is the case that $h(x_{i_t})$ and $h(x_{j_t})$ share a symbol only if there is a path from i_t to j_t in the undirected graph induced by $I_{k-1} = \{(i_1, j_1), \ldots, (i_{k-1}, j_{k-1})\}$ (assuming that $I_0 = \emptyset$).*
(*3*) *The word $h(x_{i_k})$ does not share symbols with $h(x_{j_k})$.*
(*4*) *For no variable y in $e(I, S)$ it is the case $h(y)$ has repeated symbols.*
(*5*) *There are only finitely many principal solutions of the equation $h(\phi_k) = h(\psi_k)$.*

For the base case $k = 1$, we have that h is a principal solution of the trivial equation $\epsilon = \epsilon$, and therefore it is the identity over all variables in $e(I, S)$. Thus, the first and third items hold. The second item holds since I is acyclic, and therefore $i_t \neq j_t$ for each $1 \leq t \leq \ell$. In fact, $h(x_{i_t}) = x_{i_t}$ and $h(x_{j_t}) = x_{j_t}$, and therefore $h(x_{i_t})$ and $h(x_{j_t})$ are different variables for each such t. The fourth item follows from a similar argument. Finally, the equation $h(\phi_1) = h(\psi_1)$ corresponds to $\phi_1 = \psi_1$, which is the original equation e. Therefore, it has only finitely many principal solutions by hypothesis.

For the inductive step we consider $k + 1$, where $1 \leq k < \ell$, and a principal solution $h : Y_k \to \Delta^*$ of the word equation e_k. We proceed to prove each one of the items of the claim:

(1) For every existentially quantified variable z_s, for $1 \leq s \leq p$, it is the case that $h(z^s_{i_{k+1},j_{k+1}}) = z^s_{i_{k+1},j_{k+1}}$: This is because $z^s_{i_{k+1},j_{k+1}}$ does not appear in e_k, and therefore h is the identity over it.

(2) For each t such that $k < t \leq \ell$, it is the case that $h(x_{i_t})$ and $h(x_{j_t})$ share a symbol only if there is a path from x_{i_t} to x_{j_t} in the undirected graph induced by I_k. First of all, Lemma 5.14 also holds in this case, i.e., h is of the form $h = g \circ h_0$, where h_0 is a principal solution of e_{k-1} and g is a principal solution of $h_0(\phi_k) = h_0(\psi_k)$. Consider an arbitrary t such that $k < t \leq \ell$. We consider two cases:

— Suppose first that $h_0(x_{i_t})$ shares symbols with $h_0(x_{j_t})$. Since h_0 is a principal solution of e_{k-1}, we have by induction hypothesis then that there is a path in I_{k-1} from x_{i_t} to x_{j_t}. This path also exists in I_k.
— Suppose now that $h_0(x_{i_t})$ does not share symbols with $h_0(x_{j_t})$, yet $h(x_{i_t}) = g(h_0(x_{i_t}))$ shares some with $h(x_{i_t}) = g(h_0(x_{i_t}))$. Since g is a principal solution, it can only have effect on the variables appearing in the equation it solves, which is $h_0(\phi_k) = h_0(\psi_k)$. Therefore, in order for the strings $g(h_0(x_{i_t}))$ and $g(h_0(x_{j_t}))$ to share symbols while

this not being the case for $h_0(x_{i_t})$ and $h_0(x_{j_t})$, it must be the case that both $h_0(x_{i_t})$ and $h_0(x_{j_t})$ mention symbols from the equation $h_0(\phi_k) = h_0(\psi_k)$. Among the symbols mentioned in the equation $h_0(\phi_k) = h_0(\psi_k)$ we have those coming from $h_0(x_{i_k})$ and $h_0(x_{j_k})$, and those coming from $h_0(z^s_{i_k,j_k})$ (for existentially quantified variables z_s in $e : \phi = \psi$). For the latter we have that $h_0(z_{i_k,j_k}) = z_{i_k,j_k}$, since they only appear in the strings ϕ_k or ψ_k and so are left untouched by g. From this it follows that (1) $h_0(x_{i_t})$ shares variables with either $h_0(x_{i_k})$ or $h_0(x_{j_k})$, and (2) $h_0(x_{j_t})$ shares variables with either $h_0(x_{i_k})$ or $h_0(x_{j_k})$. By induction hypothesis we can then conclude that both i_t and j_t are connected to either i_k or j_k in I_{k-1}. Since I_k is precisely I_{k-1} plus the undirected edge $\{i_k, j_k\}$, it follows that, in any case, i_t is connected to j_t in I_k.

(3) There are no symbols appearing both in $h(x_{i_{k+1}})$ and in $h(x_{j_{k+1}})$: Assume otherwise, i.e., $h(x_{i_{k+1}})$ shares symbols with $h(x_{j_{k+1}})$. It follows from (2) that i_{k+1} is connected to j_{k+1} in I_k. However, the undirected graph induced by $I_\ell = I$ contains both I_k (which does not contain the undirected edge $\{i_{k+1}, j_{k+1}\}$) and the edge $\{i_{k+1}, j_{k+1}\}$ (which is in $I_{k+1} \setminus I_k$). This contradicts the acyclicity of I.

(4 and 5) The word equation $h(\phi_{k+1}) = h(\psi_{k+1})$ corresponds to the equation $\phi_{k+1} = \psi_{k+1}$ transformed by h. The only symbols in $\phi_{k+1} = \psi_{k+1}$ that can actually be transformed by h are $x_{i_{k+1}}$ and $x_{j_{k+1}}$, since the rest are associated with the existential variables of $e : \phi = \psi$ and only appear in the strings ϕ_{k+1}, ψ_{k+1}. Therefore, solving $h(\phi_{k+1}) = h(\psi_{k+1})$ is equivalent to solving the system of equations $\phi_{k+1} = \psi_{k+1} \wedge x_{i_{k+1}} = h(x_{i_{k+1}}) \wedge x_{j_{k+1}} = h(x_{i_{k+1}})$. We can rewrite this system into one word equation as follows:

$$\phi_{k+1}\#x_{i_{k+1}}\#x_{j_{k+1}} \;=\; \psi_{k+1}\#h(x_{i_{k+1}})\#h(x_{j_{k+1}}),$$

This word equation is hence equivalent to $h(\phi_{k+1}) = h(\psi_{k+1})$, i.e., they have the same solutions when restricted to those variables that are neither $x_{i_{k+1}}$ nor $x_{j_{k+1}}$. Once again applying Lemma 5.14 we obtain that the principal solutions g of the latter equation are of the form $g = g_3 \circ g_2 \circ g_1$ where (a) g_1 is a principal solution of $\phi_{k+1} = \psi_{k+1}$, (b) g_2 is a principal solution of $g_1(x_{i_{k+1}}) = g_1(h(x_{i_{k+1}}))$, and g_3 is a principal solution of $g_2(g_1(x_{j_{k+1}})) = g_2(g_1(h(x_{j_{k+1}})))$. We now prove there can only be finitely many such principal solutions.

— First, the equation $\phi_{k+1} = \psi_{k+1}$ is just a relabelling of the word equation $e : \phi = \psi$ defining S, and has only finitely many principal solutions by hypothesis. So let us pick any such principal solution $g_1 : Y_{k+1} \setminus Y_k \to \Delta_1^*$. Since g_1 is a principal solution for equation $e : \phi = \psi$ with variables renamed, we have that for any variable $y \in Y_{k+1} \setminus Y_k$ $g_1(y)$ does not have repeated symbols.
— Now we need to prove that there are only finitely many principal solutions for the equation $g_1(x_{i_{k+1}}) = g_1(h(x_{i_{k+1}}))$. Recall that $h : Y_k \to \Delta^*$ is a principal solution of the equation $e_k : \phi_1\#\cdots\#\phi_k = \psi_1\#\cdots\#\psi_k$, with Δ a fresh alphabet. On the other hand, g_1 is a principal solution of $\phi_{k+1} = \psi_{k+1}$ and can only change symbols mentioned in it. Hence g_1 is the identity over Δ. It thus follows that $g_1(h(x_{i_{k+1}})) = h(x_{i_{k+1}})$ and, consequently, the equation $g_1(x_{i_{k+1}}) = g_1(h(x_{i_{k+1}}))$ is reduced to $g_1(x_{i_{k+1}}) = h(x_{i_{k+1}})$. The latter equation has the following properties: (a) the left hand side does not share symbols with the right hand side (since they are from disjoint alphabets Δ_1 and Δ respectively), (b) $g_1(x_{i_{k+1}})$ does not repeat symbols by hypothesis because it is a principal solution of the original equation e with variables renamed, and (c) $h(x_{i_{k+1}})$ does not repeat symbols by item (4) in the inductive hypothesis. Putting all this together and using Lemma 5.7, we have that the equation $g_1(x_{i_{k+1}}) = g_1(h(x_{i_{k+1}}))$ has finitely many principal solutions. Moreover, for

each such solution g_2 it is the case that $g_2(y)$ has no repeated symbols, where y is an arbitrary variable mentioned in $e(I, S)$.

— Let us pick an arbitrary principal solution $g_2 : (\Delta \cup \Delta_1) \to \Delta_2^*$ to the equation $g_1(x_{i_{k+1}}) = g_1(h(x_{i_{k+1}}))$. We finally need to prove that the equation $g_2(g_1(x_{j_{k+1}})) = g_2(g_1(h(x_{j_{k+1}})))$ has finitely many principal solutions. The proof goes along the lines of the preceding item. First, as argued above, we have that $g_1(h(x_{j_{k+1}})) = h(x_{j_{k+1}})$. Furthermore, it is also the case that $g_2(h(x_{j_{k+1}})) = h(x_{j_{k+1}})$. The reason is that by item (3) of the inductive hypothesis, we have that $h(x_{j_{k+1}})$ does not share symbols with $h(x_{i_{k+1}})$. This means that no symbol from $h(x_{j_{k+1}})$ occurs in the equation which g_2 solves, so $g_2(h(x_{j_{k+1}})) = h(x_{j_{k+1}})$ as claimed. The equation $g_2(g_1(x_{j_{k+1}})) = g_2(g_1(h(x_{j_{k+1}})))$ consequently reduces to $g_2(g_1(x_{j_{k+1}})) = h(x_{j_{k+1}})$. Once again we can use Lemma 5.7 to this equation since: (a) the left hand side does not share symbols with the right hand side (since they are from disjoint alphabets Δ_2 and Δ respectively), (b) $g_2(g_1(x_{j_k}))$ does not repeat symbols, since for each variable x neither $g_1(x)$ nor $g_2(x)$ repeats symbols, and (c) $h(x_{j_{k+1}})$ does not repeat symbols by item (4) in the inductive hypothesis. Putting all this together, we have from Lemma 5.7 that there are finitely many principal solutions to the equation $g_2(g_1(x_{j_{k+1}})) = g_2(g_1(h(x_{j_{k+1}})))$, and that for each such solution g_3 it is the case that for each variable y mentioned in $e(I, S)$ the word $g(y)$ has no repeated symbols.

We conclude that the equation $h(\phi_k) = h(\psi_k)$ has only finitely many principal solutions. Moreover, for each such solution g it is the case that $g(y)$ has no repeated symbols, where y is an arbitrary variable mentioned in $e(I, S)$. This finishes the proofs of the claim and the proposition. □

6. THE LOGIC CRPQ($\preceq_{\text{SS}}$)

We now switch to study the complexity of evaluation for the logic CRPQ($\preceq_{\text{ss}}$), for which upper bounds were already obtained: NEXPTIME and NP in combined and data complexity, respectively (see Corollary 3.6). We prove here that both bounds are actually optimal. For the data complexity of CRPQ($\preceq_{\text{ss}}$), we give a reduction from the longest common subsequence problem obtaining an NP lower bound. For the combined complexity, we use a suitable succinct version of the same problem to provide a matching NEXPTIME lower bound.

6.1. 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 we use a reduction from the *longest common subsequence problem* LCS: given a finite alphabet Σ, words $u_1, \ldots, u_n \in \Sigma^*$, and $\ell > 0$, decide whether there is a word $u \in \Sigma^*$ such that (1) $u \preceq_{\text{ss}} u_i$ for each $i = 1, \ldots, n$ and (2) $|u| = \ell$. When the alphabet Σ is fixed, we denot this problem as LCS_Σ.

THEOREM 6.1. *Evaluation for* CRPQ($\preceq_{\text{ss}}$) *is* NP*-complete in data complexity.*

The upper bound follows from Corollary 3.6. We prove hardness next. Due to Lemma 3.4, we only need to prove the following:

PROPOSITION 6.2. *There is an index set $I \subseteq [m]^2$, a finite alphabet Σ, 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 the proof of Proposition 5.2, but now extend it with the pair $(1,0)$ to define $I := I_\diamond \cup \{(1,0)\}$. We know that there is a finite alphabet Σ such that LCS_Σ is NP-hard

[Maier 1978; Blin et al. 2012]. We show that there is a polynomial time reduction from this problem to $\text{GENINT}_{I,\Sigma_\$,\lambda}(\preceq_{\text{ss}})$, where $\Sigma_\$$ is the extension of Σ with a fresh symbol $\$$, and $\lambda : I \to 2^{\preceq_{\text{ss}}}$ is such that $\lambda(i,j) = \{\preceq_{\text{ss}}^{\Sigma_\$}\}$ for each $(i,j) \in I$.

Let $u_1, \ldots, u_n$ be words over Σ and let $\ell \geq 0$. We want to determine whether these words have a common subsequence $u \in \Sigma^*$ of length $|u| = \ell$. To this end, consider the regular languages $L_0 = \{\$u_1\$ \cdots \$u_n\$\}$ and $L_1 = \{(\$u)^n\$ \mid u \in \Sigma^\ell\}$ over $\Sigma_\$$. Since each word in these two languages has exactly $n+1$ occurrences of the symbol $\$ \notin \Sigma$, and all the words $u, u_1, \ldots, u_n$ lie in Σ^*, it follows that $u_1, \ldots, u_n$ have a common subsequence of length ℓ if and only if there are words $w_0 \in L_0$ and $w_1 \in L_1$ such that $w_1 \preceq_{\text{ss}} w_0$. Clearly, though, L_1 cannot be constructed in polynomial time from the input. Instead, we make use of $I_\diamond$ to encode L_1 as a polynomial size generalized intersection instance. The construction is similar to the one used in the proof of Proposition 5.2.

Consider the following regular expressions over $\Sigma_\$$:

(1) $R_1 = (\$\Sigma^\ell)^n\$$,
(2) $R_2 = \$(\$\Sigma^\ell)^n\$$,
(3) $R_3 = (\$\Sigma^\ell)^n\$\$$, and
(4) $R_4 = (\$\Sigma^\ell)^{n+1}\$$.

These expressions can clearly be constructed in polynomial from $u_1, \ldots, u_n$ and $\ell \geq 0$. The following claim is fundamental for our proof:

CLAIM 6.3. *For every words w_i ($i = 1,2,3,4$) over $\Sigma_\$$, the following are equivalent:*

(a) $w_i \in R_i$ for each $i = 1,2,3,4$ and $w_i \preceq_{\text{ss}} w_j$ for each $(i,j) \in I_\diamond$.
(b) There is a word $u \in \Sigma^\ell$ such that:

$$w_1 = (\$u)^n\$, \quad w_2 = \$(\$u)^n\$, \quad w_3 = (\$u)^n\$\$, \quad \textit{and} \quad w_4 = (\$u)^{n+1}\$.$$

PROOF. Clearly, (b) implies (a). In fact, each such w_i belongs to R_i (for $i = 1,2,3,4$). Moreover, it is easy to see that $w_i \preceq_{\text{ss}} w_j$ for each $(i,j) \in I_\diamond$. To prove that (a) implies (b), consider words $w_i \in R_i$, for $i = 1,2,3,4$, such that $w_i \preceq_{\text{ss}} w_j$ for each $(i,j) \in I_\diamond$. Let (i) $\alpha_1, \ldots, \alpha_n$, (ii) $\beta_1, \ldots, \beta_n$, (iii) $\gamma_1, \ldots \gamma_n$, and (iv) $\eta_0, \ldots, \eta_n$ be the words from Σ^ℓ that appear between succesive $\$$ symbols in w_1, w_2, w_3, and w_4, respectively. More precisely,

(1) $w_1 = \$\alpha_1\$ \cdots \$\alpha_n\$$.
(2) $w_2 = \$\$\beta_1\$ \cdots \$\beta_n\$$.
(3) $w_3 = \$\gamma_1\$ \cdots \$\gamma_n\$\$$.
(4) $w_4 = \$\eta_0\$ \cdots \$\eta_n\$$.

We prove next that:

$$\alpha_i \;=\; \beta_i \;=\; \gamma_i \;=\; \eta_i \;=\; \eta_{i-1} \quad \text{(for each } 1 \leq i \leq n\text{)}.$$

This directly implies the claim by choosing $u = \alpha_1$.

First, we have that $w_1 \preceq_{\text{ss}} w_2$ and the number of occurrences of $\$$ in these words is $n+1$ and $n+2$, respectively. Since the symbol $\$$ is not in Σ, it follows that $\alpha_i \preceq_{\text{ss}} \beta_i$ for each $1 \leq i \leq n$. But both α_i and β_i are of length ℓ, and thus $\alpha_i = \beta_i$ for each $1 \leq i \leq n$. Similarly, from $w_1 \preceq_{\text{ss}} w_3$ we conclude that $\alpha_i = \gamma_i$ for each $1 \leq i \leq n$.

A similar analysis can be carried out for $w_2 \preceq_{\text{ss}} w_4$. Since both words have exactly $n+2$ occurrences of the symbol $\$$, we conclude that $\beta_i = \eta_i$ for each $1 \leq i \leq n$. Moreover, $w_3 \preceq_{\text{ss}} w_4$, which implies that $\gamma_i = \eta_{i-1}$ for each $1 \leq i \leq n$. Now, since $\alpha_i = \beta_i = \gamma_i$ for each $1 \leq i \leq n$, we obtain on the one hand that $\eta_1 = \alpha_1, \ldots, \eta_n = \alpha_n$, and on the other hand that $\eta_0 = \alpha_1, \ldots, \eta_{n-1} = \alpha_n$. We conclude that:

$$\alpha_1 \;=\; \ldots \;=\; \alpha_n \;=\; \eta_0 \;=\; \eta_1 \;=\; \ldots \;=\; \eta_n.$$

This proves the claim. □

Finally, we claim that the words $u_1, \cdots, u_n$ have a common subsequence of length ℓ if and only if $(R_0, R_1, R_2, R_3, R_4)$ is accepted by $\text{GENINT}_{I,\Sigma_\$,\lambda}(\preceq_{\text{ss}})$, where $R_0 = \{\$u_1\$\cdots\$u_n\$\}$ and R_1, R_2, R_3, R_4 are as defined above. This finishes the proof of the theorem since $(R_0, R_1, R_2, R_3, R_4)$ can be constructed in polynomial time from $u_1, \cdots, u_n$ and $\ell \geq 0$.

Assume first that $(R_0, R_1, R_2, R_3, R_4)$ is accepted by $\text{GENINT}_{I,\Sigma_\$,\lambda}(\preceq_{\text{ss}})$. Then there are words $w_i \in R_i$ for each $i = 0, 1, 2, 3, 4$ and $w_i \preceq_{\text{ss}} w_j$ for each $(i, j) \in I$. Thus, from Claim 6.3 we have that w_1 is of the form $(\$u)^n\$$ for $u \in \Sigma^\ell$. Since $w_1 \preceq_{\text{ss}} w_0$ and $w_1 \in L_1$, where L_1 is the regular language defined at the beginning of the proof, we conclude that $u_1, \cdots, u_n$ have a common subsequence of length ℓ (namely, the word u). Assume, on the other hand, that $u_1, \cdots, u_n$ have a common subsequence u of length ℓ. It is clear then that the words:

$$w_0 = \$u_1\$\cdots\$u_n\$,\ w_1 = (\$u)^n\$,\ w_2 = \$(\$u)^n\$,\ w_3 = (\$u)^n\$\$,\ \text{and}\ w_4 = (\$u)^{n+1}\$$$

are a witness for the fact that $(R_0, R_1, R_2, R_3, R_4)$ is accepted by $\text{GENINT}_{I,\Sigma_\$,\lambda}(\preceq_{\text{ss}})$.

6.2. 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 this bound is tight:

THEOREM 6.4. *The evaluation problem for* CRPQ($\preceq_{\text{ss}}$) *is* NEXPTIME*-complete.*

Due to Lemma 3.3 we only need to prove the following:

PROPOSITION 6.5. *The problem* GENINT($\preceq_{\text{ss}}$) *is* NEXPTIME*-hard.*

PROOF. We use a reduction from a suitable succinct version of LCS, which we prove to be NEXPTIME-hard. This succinct version SUCCINCT-LCS is defined as follows: We are given binary n-bit integers $\ell, q \geq 0$ and regular languages $R_1, \ldots, R_m$ over $(\Sigma \cup \{\$\})^2$, where $\$$ is a delimiter, such that:

$$\bigcap_{i=1}^{m} R_i = \{(\$,\$)\, w_1\, (\$,\$)\ \ldots\ (\$,\$)\, w_q\, (\$,\$)\},$$

for $w_1, \ldots, w_q$ words over $\Sigma \times \Sigma$. Let us assume that $w_i = u_i \otimes u_i'$ for each $1 \leq i \leq q$ (where the operation $\otimes$ is as defined in Section 2). We want to determine whether there is a word $u \in \Sigma^*$ such that (1) $u \preceq_{\text{ss}} u_i$ for each $1 \leq i \leq q$, and (2) $|u| = \ell$.

Using standard techniques based on succinct reductions, it is possible to prove that this succinct version of LCS is NEXPTIME-hard. The proof of this result can be found in the appendix:

PROPOSITION 6.6. *The problem* SUCCINCT-LCS *is* NEXPTIME*-hard.*

We now show that the problem SUCCINCT-LCS can be reduced in polynomial time to GENINT($\preceq_{\text{ss}}$). The basic idea is to construct a succinct version of the proof of Theorem 6.1. Consider an input instance to SUCCINCT-LCS as defined above. We want to determine whether the words $u_1, \ldots, u_q$ have a common subsequence of length ℓ.

Let us first explain the idea of the proof. Consider the regular language:

$$\mathcal{L}_0 = \bigcup_{u \in \Sigma^\ell} (\$u)^q\$.$$

Observe that since $\$$ does not belong to Σ, the given instance to SUCCINCT-LCS is positive if and only if there is a word $w \in \mathcal{L}_0$ such that

$$w \preceq_{\text{ss}} \$u_1\$\ldots\$u_q\$.$$

As in the proof of Proposition 6.2, we build a generalized intersection instance of polynomial size that encodes this constraint. However, this is a bit harder than in Proposition 6.2 since now $\ell, q > 0$ are both exponential in n. On the other hand, we now have the benefit that neither the index set I nor the alphabet needs to be fixed.

First we deal with $\mathcal{L}_0$. Since ℓ, q are exponential in n, we need to keep binary counters that ensure us that if w is an arbitrary word in $\mathcal{L}_0$, then the length of each subword of w between consecutive \$'s is exactly ℓ and the number of such subwords is exactly q. Thus, it is convenient not to encode $\mathcal{L}_0$ but a "padded" version of it, denoted $\mathcal{L}'_0$, which is defined as the union over each word $u = a_1 a_2 \dots a_\ell \in \Sigma^\ell$ of the language:

$$\begin{aligned}
&\$ \left(a_1\#[1]\#[1] \star a_2\#[1]\#[2] \star \cdots \star a_\ell\#[1]\#[\ell]\right) \dots \\
&\qquad \$ \left(a_1\#[i]\#[1] \star a_2\#[i]\#[2] \star \cdots \star a_\ell\#[i]\#[\ell]\right) \dots \\
&\qquad\qquad \$ \left(a_1\#[q]\#[1] \star a_2\#[q]\#[2] \star \cdots \star a_\ell\#[q]\#[\ell]\right) \$,
\end{aligned}$$

where $[i]$ is the binary $n+1$ bit representation of the integer $1 \leq i \leq 2^n$, and $\#$ and $\star$ are fresh delimiters. We assume, of course, that the binary integers 0 and 1 are not in $\Sigma \cup \{\$\}$. Intuitively, $a_1\#[i]\#[1] \star a_2\#[i]\#[2] \star \cdots \star a_\ell\#[i]\#[\ell]$ represents the i-th copy of the word $u = a_1, \dots, a_\ell$ in a word in $\mathcal{L}_0$, for $1 \leq i \leq q$.

Next we show that it is possible to construct in polynomial time an instance of the generalised intersection problem over binary relation $\preceq_{\text{ss}}$, such that $\mathcal{L}'_0$ corresponds to the projection of the corresponding index set over one of its components. To do this, it is convenient to start by showing that there is an exponential size instance of $\textsc{GenInt}(I_\diamond, \preceq_{\text{ss}})$, where $I_\diamond = \{(1,2),(1,3),(2,4),(3,4)\}$ is the index set used in the proof of Proposition 6.2, such that $\mathcal{L}'_0$ is defined by projecting the solutions of this instance over its first component.

Let us define a regular language A_1 as follows:

$$\begin{aligned}
&\$ \left(\Sigma\#[1]\#[1] \star \Sigma\#[1]\#[2] \star \cdots \star \Sigma\#[1]\#[\ell]\right) \dots \\
&\qquad \$ \left(\Sigma\#[i]\#[1] \star \Sigma\#[i]\#[2] \star \cdots \star \Sigma\#[i]\#[\ell]\right) \dots \\
&\qquad\qquad \$ \left(\Sigma\#[q]\#[1] \star \Sigma\#[q]\#[2] \star \cdots \star \Sigma\#[q]\#[\ell]\right) \$.
\end{aligned}$$

That is, A_1 encodes the language of words that are obtained by taking the concatenation of q words of length ℓ over Σ, separated by delimiters. Furthermore, we define A_2 as $\$A_1$, A_3 as $A_1\$$, and A_4 as the following language:

$$\begin{aligned}
&\$ \left(\Sigma\#[0]\#[1]\#[1] \star \Sigma\#[0]\#[1]\#[2] \star \cdots \star \Sigma\#[0]\#[1]\#[\ell]\right) \dots \\
&\qquad \$ \left(\Sigma\#[1]\#[2]\#[1] \star \Sigma\#[1]\#[2]\#[2]\#[2] \star \cdots \star \Sigma\#[1]\#[2]\#[\ell]\right) \dots \\
&\qquad \$ \left(\Sigma\#[i-1]\#[i]\#[1] \star \Sigma\#[i-1]\#[i]\#[2] \star \cdots \star \Sigma\#[i-1]\#[i]\#[\ell]\right) \dots \\
&\qquad\qquad \$ \left(\Sigma\#[q]\#[q+1]\#[1] \star \Sigma\#[q]\#[q+1]\#[2] \star \cdots \star \Sigma\#[q]\#[q+1]\#[\ell]\right) \$.
\end{aligned}$$

Thus, compared to A_1, in A_4 we have added a new binary counter that goes from 0 to q, while our previous first counter now goes from 1 to $q+1$. Notice that $n+1$ bits are still enough to represent each integer in A_4.

Further, for each word $u = a_1 a_2 \dots a_\ell \in \Sigma_\ell$ we define $A_1(u)$ as:

$$\begin{aligned}
&\$ \left(a_1\#[1]\#[1] \star a_2\#[1]\#[2] \star \cdots \star a_\ell\#[1]\#[\ell]\right) \dots \\
&\qquad \$ \left(a_1\#[i]\#[1] \star a_2\#[i]\#[2] \star \cdots \star a_\ell\#[i]\#[\ell]\right) \dots \\
&\qquad\qquad \$ \left(a_1\#[q]\#[1] \star a_2\#[q]\#[2] \star \cdots \star a_\ell\#[q]\#[\ell]\right) \$,
\end{aligned}$$

and $A_4(u)$ as:

$$\$\left(a_1\#[0]\#[1]\#[1]\star a_2\#[0]\#[1]\#[2]\star\cdots\star a_\ell\#[0]\#[1]\#[\ell]\right)\ldots$$
$$\$\left(a_1\#[1]\#[2]\#[1]\star a_2\#[1]\#[2]\#[2]\star\cdots\star a_\ell\#[1]\#[2]\#[\ell]\right)\ldots$$
$$\$\left(a_1\#[i-1]\#[i]\#[1]\star a_2\#[i-1]\#[i]\#[2]\star\cdots\star a_\ell\#[i-1]\#[i]\#[\ell]\right)\ldots$$
$$\$\left(a_1\#[q]\#[q+1]\#[1]\star a_2\#[q]\#[q+1]\#[2]\star\cdots\star a_\ell\#[q]\#[q+1]\#[\ell]\right)\$.$$

Notice that $A_1(u)\in A_1$ and $A_4(u)\in A_4$, for each $u=a_1a_2\ldots a_\ell\in\Sigma_\ell$.

It is then possible to establish the following crucial claim:

CLAIM 6.7. *For every words w_i ($i=1,2,3,4$) over $\Sigma\cup\{\$,\#,\star\}\cup\{0,1\}$, the following are equivalent:*

(a) $w_i\in A_i$ for each $i=1,2,3,4$ and $w_i\preceq_{\rm ss} w_j$ for each $(i,j)\in I_\diamond$.
(b) There is a word $u\in\Sigma^\ell$ such that:

$$w_1=A_1(u),\quad w_2=\$w_1,\quad w_3=w_1\$,\quad \textit{and}\quad w_4=A_4(u).$$

PROOF. We mimick the proof of Claim 6.3. The reason why we need to include an extra counter in A_4 is explained next. To prove that that (a) implies (b), assume that w_4 is of the form

$$\$\left(\Sigma\#[0]\#[1]\#[1]\star\Sigma\#[0]\#[1]\#[2]\star\cdots\star\Sigma\#[0]\#[1]\#[\ell]\right)w_4'$$

and of the form

$$w_4''\left(\Sigma\#[q]\#[q+1]\#[1]\star\Sigma\#[q]\#[q+1]\#[2]\star\cdots\star\Sigma\#[q]\#[q+1]\#[\ell]\right)\$.$$

Then the proof requires w_1 to be a subsequence of both w_4' and w_4''. In the first case, we match the two counters of w_1 with the first and third counters of w_4', respectively, while in the second case we do it with the second and third counters of w_4'', respectively. □

That is, the projection over the first component of the solutions to the instance of GENINT$(I_\diamond,\preceq_{\rm ss})$ given by (A_1,A_2,A_3,A_4) defines the language $\mathcal{L}_0'$. Of course, the problem is that the A_i's, for $i=1,2,3,4$, are of exponential size. Nonetheless, we show next that each one of them can be represented as an intersection of a set of regular expressions that is constructible in polynomial time:

LEMMA 6.8. *For each $i=1,2,3,4$, it is possible to construct in polynomial time regular languages $S_1,\ldots,S_t$ such that $A_i=\bigcap_{1\leq j\leq t}S_j$.*

PROOF. We only provide a sketch, since the full proof is standard but rather technical. We concentrate on A_1 because the other cases are similar. It is known that we can construct in polynomial time a set of regular expressions whose intersection defines a $(2n+2)$-bit counter, where every pair of consecutive addresses is separated by a fresh delimiter [Börger et al. 1997]. In our case, we assume such delimiter to be a word of the form $\star\Sigma\#$. We slightly modify such construction using standard techniques, so that the first $n+1$ bits of the counter are separated by delimiter $\#$ from the last $(n+1)$ bits. Furthermore, the counter defined by the first $n+1$ bits resets every time it reaches the integer ℓ (in binary), and the counter defined by the last $n+1$ bits stops once it reaches the integer q (in binary). Furthermore, every time the counter defined by the first $n+1$ bits resets (i.e., reaches the integer ℓ), we force a symbol $\$$ to appear immediately afterwards. □

We would like to replace then each A_i, for $i=1,2,3,4$, with the polynomial set $S_1,\ldots,S_t$ of regular expressions that defines A_i. However, it is not possible to do this

directly since each component of an index set can be constrained by at most one regular expression. Instead, we replace component i in $I_\diamond$ with t new components $c_1, \ldots, c_t$, constraint component c_j with regular expression S_j, for each $1 \leq j \leq t$, and add subsequence constraints between every pair $(c_j, c_{j'})$ of such components, for $1 \leq j, j' \leq t$. This implies that all these components are witnessed by the same word in a solution. The subsequence constraints of the form $(i, j) \in I_\diamond$ can then be established from an arbitrary component representing i to an arbitrary component representing j. Furthermore, the projection over an arbitrary component representing the index 1 of the solutions of this extended instance of the generalised intersection problem defines the language $\mathcal{L}_0'$. It is easy to see that this extended instance can be constructed in polynomial time from our input.

We now continue with the proof of the theorem. Let us recall that we need to construct an instance of the generalized intersection problem that checks whether there is a word $w \in \mathcal{L}_0$ such that:

$$w \preceq_{\text{ss}} \$u_1\$ \ldots \$u_q\$.$$

Recall, however, that we only have access to $\mathcal{L}_0$ through its padded version $\mathcal{L}_0'$. Furthermore, we only have access to the words $u_1, \ldots, u_q$ through $\bigcap_{1 \leq i \leq m} R_i$, where $\bigcap_i R_i$ is a single word over $(\Sigma \cup \{\$\})^2$ of the form $(\$, \$)w_1(\$, \$) \ldots (\$, \$)w_q(\$, \$)$ such that $w_i = u_i \otimes u_i'$ for each $1 \leq i \leq q$. In other words, u_i corresponds to the projection of w_i over its first component. Thus, in order to check whether there is a word $w \in \mathcal{L}_0$ such that $w \preceq_{\text{ss}} \$u_1\$ \ldots \$u_q\$$ we use the following construction:

(1) Instead of defining $\mathcal{L}_0'$, we define a language $\mathcal{L}_0''$ that accepts precisely those words of the form $w \otimes w'$ over $(\Sigma \cup \{\$, \#, \star\} \cup \{0, 1\})^2$ such that $w \in \mathcal{L}_0'$ and w' is an arbitrary word in $\Sigma \cup \{\$, \#, \star\} \cup \{0, 1\}$ of the same length than w. Using a slight extension of our previous arguments, it is is easy to see that an instance of the generalised intersection problem that defines $\mathcal{L}_0''$ (over one of its components) can be constructed in polynomial time from the input to our problem.
(2) We then construct in polynomial time an instance of the generalized intersection problem that defines (over one of its components) a suitable padding $(\bigcap_i R_i)^{\text{pad}}$ of $\bigcap_i R_i$ in such a way that the following are equivalent:
— There is a word $w \in \mathcal{L}_0$ such that $w \preceq_{\text{ss}} \$u_1\$ \ldots \$u_q\$$.
— There is a word $w' \in \mathcal{L}_0''$ such that $w' \preceq_{\text{ss}} (\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$.
(3) We finish by adding a subsequence constraint from the component that defines $\mathcal{L}_0''$ to the one that defines $(\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$.

It only remains to explain how $(\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$ is defined. Let us assume then that each word w_j, for $1 \leq j \leq q$, is of the form $(a_j^1, b_j^1) \ldots (a_j^{t_j}, b_j^{t_j}) \in (\Sigma \times \Sigma)^{t_j}$. Furthermore, let us define w_j^{pad} as the following padded version of w_j:

$$(a_j^1, b_j^1)(\#, \#)[j]^{\otimes}(\#, \#)[1]^{\otimes}(\#, \#)[j]^{\otimes}(\#, \#)[2]^{\otimes}(\#, \#) \ldots (\#, \#)[j]^{\otimes}(\#, \#)[\ell]^{\otimes}\ (\star, \star) \ldots$$
$$(\star, \star)\,(a_j^{t_j}, b_j^{t_j})(\#, \#)[j]^{\otimes}(\#, \#)[1]^{\otimes}(\#, \#)[j]^{\otimes}(\#, \#)[2]^{\otimes}(\#, \#) \ldots (\#, \#)[j]^{\otimes}(\#, \#)[\ell]^{\otimes},$$

where $[k]^{\otimes}$ denotes the word $[k] \otimes [k]$ over $\{0, 1\} \times \{0, 1\}$. Notice that w_j^{pad} extends w_j by adding after each symbol a_j^k, for $1 \leq k \leq t_j$, all possible values of the counters that appear in a word in $\mathcal{L}_0'$ between the j-th and the $(j+1)$-th delimiter in $\$$. Finally, let us define $(\bigcap_i R_i)^{\text{pad}}$ as the following padded version of $\bigcap_i R_i$:

$$(\$, \$)\, w_1^{\text{pad}}\, (\$, \$)\ \ldots\ (\$, \$)\, u_q^{\text{pad}}\, (\$, \$).$$

We now prove that $(\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$ satisfies our desired property:

$\mathcal{S}$	$\preceq_{\text{suff}}$	$\preceq_{\text{sw}}$	$\preceq_{\text{ss}}$
comb. comp. of CRPQ($\mathcal{S}$)	PSPACE-complete (Thm 4.6)	PSPACE-complete (Thm 4.6)	NEXPTIME-complete (Thm 6.4)
data comp. of CRPQ($\mathcal{S}$)	NLOGSPACE (Prop 5.4)	PSPACE-complete (Thm 5.1)	NP-complete (Thm 6.1)

Fig. 1. Combined and data complexity of graph logics CRPQ($\preceq_{\text{suff}}$), CRPQ($\preceq_{\text{sw}}$), and CRPQ($\preceq_{\text{ss}}$).

CLAIM 6.9. *The following are equivalent:*

— *There is a word* $w \in \mathcal{L}_0$ *such that* $w \preceq_{\text{ss}} \$u_1\$ \dots \$u_q\$$.
— *There is a word* $w' \in \mathcal{L}_0''$ *such that* $w' \preceq_{\text{ss}} (\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$.

PROOF. Let u be the natural padding of w that fits the form of $\mathcal{L}_0'$. By definition of $(\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$, there is word u' of the same length than u such that $u \otimes u' \preceq_{\text{ss}} (\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$. We then define $w' = u \otimes u'$. Analogously, assume that w' is of the form $u \otimes u'$. Then w is obtained from u by removing symbols in $\{\star, \#\}$ and the counters. □

Based on Claim 6.9, to finish the proof it is sufficient to show that it is possible to construct in polynomial time a set of regular expressions that defines the language $(\bigcap_i R_i)^{\text{pad}}$. In fact, if this was the case we could build an instance of the generalized intersection problem that represents this language (in the same way we did above to represent each regular language A_i, for $i = 1, 2, 3, 4$), and then add a subsequence constraint from the component that represents $\mathcal{L}_0''$ to an arbitrary component in the new index set. We prove this below:

LEMMA 6.10. *It is possible to construct in polynomial time regular languages* $S_1, \dots, S_t$ *such that* $\bigcap_{1 \leq j \leq t} S_j = (\bigcap_{1 \leq i \leq m} R_i)^{\text{pad}}$.

PROOF. First, we modify the regular languages R_i, for $1 \leq i \leq m$, so that they now check that the restriction of the padded language to the alphabet $(\Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})$ corresponds precisely to $\bigcap_{1 \leq i \leq m} R_i$. This is simple, as we only need to add to each such R_i a condition that forces not to do anything (i.e., not to change state) when scanning a symbol outside $(\Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})$. Afterwards, we implement the desired counters and delimiters between symbols from $(\Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})$. This is done again by using standard techniques. □

This finishes the proof of Proposition 6.5. □

7. 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 extends and complements previous results from [Barceló et al. 2013], which established the prohibitive complexity of evaluation for logics that allow, in addition, path comparisons based on arbitrary regular relations. Figures 1 and 2 summarize the 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 re-

$\mathcal{S}$	$(\mathsf{Reg}_2 \cup \preceq_{\text{suff}})$	$(\mathsf{Reg}_2 \cup \preceq_{\text{sw}})$	$(\mathsf{Reg}_2 \cup \preceq_{\text{ss}})$
comb. comp. of CRPQ($\mathcal{S}$)	undecidable (Coroll. 5.2 in [Barceló et al. 2013])	undecidable (Coroll. 5.2 in [Barceló et al. 2013])	nonelementary (Coroll. 5.13 in [Barceló et al. 2013])
data comp. of CRPQ($\mathcal{S}$)	undecidable (Coroll. 5.2 in [Barceló et al. 2013])	undecidable (Coroll. 5.2 in [Barceló et al. 2013])	nonelementary (Coroll. 5.13 in [Barceló et al. 2013])

Fig. 2. Combined and data complexity of logics CRPQ($\mathsf{Reg}_2 \cup \preceq_{\text{suff}}$), CRPQ($\mathsf{Reg}_2 \cup \preceq_{\text{sw}}$), and CRPQ($\mathsf{Reg}_2 \cup \preceq_{\text{ss}}$).

strictions need to be imposed on them in order to obtain fragments that can be evaluated in practice.

One such restriction was identified: 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 [Barceló et al. 2013]. 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. It would be interesting to understand whether less restrictive conditions than acyclicity yield efficient evaluation for our logics. Another recent restriction proposes to replace rational relations in formulas with counting overapproximations based on Parikh automata [Figueira and Libkin 2015]. This restriction leads to tractability of evaluation in data complexity for several classes of formulas of interest, although the approximation provided by it might be a bit rough in some scenarios.

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 [Barceló et al. 2012]).

Logics that combine the relations $\{\preceq_{\text{suff}}, \preceq_{\text{sw}}, \preceq_{\text{ss}}\}$. Let us finish with a few words regarding the decidability status of the logics that combine two of the relations we study in this paper, i.e., logics of the form CRPQ($\preceq_1$,$\preceq_2$) for $\preceq_1, \preceq_2 \in \{\preceq_{\text{suff}}, \preceq_{\text{sw}}, \preceq_{\text{ss}}\}$:

(1) The logic CRPQ($\preceq_{\text{suff}}$,$\preceq_{\text{sw}}$) is PSPACE-complete from Theorem 4.6, since both $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ are in EQ.
(2) The logic CRPQ($\preceq_{\text{suff}}$,$\preceq_{\text{ss}}$) is decidable. In fact, a similar argument to the one used in the proof of Proposition 5.4 shows that there is a polynomial time translation from the evaluation problem for CRPQ($\preceq_{\text{suff}}$,$\preceq_{\text{ss}}$) to the evaluation problem for CRPQ($\preceq_{\text{pref}}$,$\preceq_{\text{ss}}$). The latter is contained in CRPQ($\mathsf{Reg}_2 \cup \preceq_{\text{ss}}$) for which the evaluation problem is decidable (see Corollary 3.5). We do not know the exact complexity of evaluation for CRPQ($\preceq_{\text{pref}}$,$\preceq_{\text{ss}}$) at this point.
(3) It is open whether evaluation for the logic CRPQ($\preceq_{\text{sw}}$,$\preceq_{\text{ss}}$) is decidable. This problem is directly related to the decidability of word equations with regular and subsequence constraints (i.e., word equations with regular constraints in which some pairs (x, y) of variables are forced to be replaced by words (w_x, w_y) such that $w_x \preceq_{\text{ss}} w_y$)), which seems very challenging.

REFERENCES

Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. 2004. A Survey of Regular Model Checking. In *Proc. CONCUR 2004-15th Int. Conf. on Concurrency Theory (Lecture Notes in Computer Science)*, Vol. 3170. 348–360.

Habib Abdulrab and Jean-Pierre Pécuchet. 1989. Solving Word Equations. *J. Symb. Comput.* 8, 5 (1989), 499–521.

Renzo Angles and Claudio Gutiérrez. 2008. Survey of graph database models. *ACM Comput. Surv.* 40, 1 (2008).

Kemafor Anyanwu, Angela Maduko, and Amit P. Sheth. 2007. SPARQ2L: towards support for subgraph extraction queries in rdf databases. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*. 797–806.

Pablo Barceló. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013*. 175–188.

Pablo Barceló, Diego Figueira, and Leonid Libkin. 2013. Graph Logics with Rational Relations. *Logical Methods in Computer Science* 9, 3 (2013).

Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. 2012. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Trans. Database Syst.* 37, 4 (2012), 31.

Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. 2003. Definable relations and first-order query languages over strings. *J. ACM* 50, 5 (2003), 694–751.

Jean Berstel. 1979. *Transductions and Context-Free Languages*. Teubner Verlag.

Guillaume Blin, Laurent Bulteau, Minghui Jiang, Pedro J. Tejada, and Stéphane Vialette. 2012. Hardness of Longest Common Subsequence for Sequences with Bounded Run-Lengths. In *Combinatorial Pattern Matching - 23rd Annual Symposium, CPM 2012*. 138–148.

Achim Blumensath and Erich Grädel. 2000. Automatic Structures. In *15th Annual IEEE Symposium on Logic in Computer Science, (LICS) 2000*. 51–62.

Egon Börger, Erich Grädel, and Yuri Gurevich. 1997. *The Classical Decision Problem*. Springer.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning, KR 2000*. 176–185.

Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2002. Rewriting of Regular Expressions and Regular Path Queries. *J. Comput. Syst. Sci.* 64, 3 (2002), 443–465.

Edmund Clarke, Orna Grumberg, and Doron Peled. 1979. *Model Checking*. MIT Press.

Isabel F. Cruz, Alberto O. Mendelzon, and Peter T. Wood. 1987. A Graphical Query Language Supporting Recursion. In *Proceedings of SIGMOD 1987*. 323–330.

Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. 2005. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.* 202, 2 (2005), 105–140.

S Eilenberg, C.C Elgot, and J.C Shepherdson. 1969. Sets recognized by n-tape automata. *Journal of Algebra* 13, 4 (1969), 447 – 464.

Diego Figueira and Leonid Libkin. 2015. Path Logics for Querying Graphs: Combining Expressiveness and Efficiency. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015*. 329–340.

Christiane Frougny and Jacques Sakarovitch. 1993. Synchronized Rational Relations of Finite and Infinite Words. *Theor. Comput. Sci.* 108, 1 (1993), 45–82.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

Lucian Ilie. 1999. Subwords and Power-Free Words are not Expressible by Word Equations. *Fundam. Inform.* 38, 1-2 (1999), 109–118.

Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. 2000. The expressibility of languages and relations by word equations. *J. ACM* 47, 3 (2000), 483–505.

Dexter Kozen. 1977. Lower Bounds for Natural Proof Systems. In *18th Annual Symposium on Foundations of Computer Science, FOCS 1977*. 254–266.

André Lentin. 1972. Equations in Free Monoids. In *ICALP*. 67–85.

M. Lothaire. 1997. *Combinatorics on Words*. Cambridge University Press.

David Maier. 1978. The Complexity of Some Problems on Subsequences and Supersequences. *J. ACM* 25, 2 (1978), 322–336.

G.S. Makanin. 1977. The Problem of Solvability of Equations in Free Semigroups. *Math USSR SBornik* 32 (1977), 129–198.

Wojciech Plandowski. 2004. Satisfiability of word equations with constants is in PSPACE. *J. ACM* 51, 3 (2004), 483–496.

Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Cambridge University Press.

Klaus U. Schulz. 1990. Makanin's Algorithm for Word Equations - Two Improvements and a Generalization. In *Word Equations and Related Topics, First International Workshop, IWWERT '90*. 85–150.

Anthony Widjaja To and Leonid Libkin. 2010. Algorithmic Metatheorems for Decidable LTL Model Checking over Infinite Systems. In *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010*. 221–236.

Moshe Y. Vardi. 1982. The Complexity of Relational Query Languages (Extended Abstract). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, STOC 1982*. 137–146.

Peter T. Wood. 2012. Query languages for graph databases. *SIGMOD Record* 41, 1 (2012), 50–60.

Fang Yu, Tevfik Bultan, and Oscar H. Ibarra. 2011. Relational String Verification Using Multi-Track Automata. *Int. J. Found. Comput. Sci.* 22, 8 (2011), 1909–1924.

Apprendix

Proof of Proposition 6.6:

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 of a nondeterministic Turing machine M on input x to the satisfiability of Cook's formula $\phi_{M,x}$. In our case, M is a nondeterministic machine that works in exponential time, and thus $\phi_{M,x}$ is of exponential size. We then use a standard reduction from satisfiability of $\phi_{M,x}$ 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 $\phi_{M,x}$). Finally, we apply on the indepdent set instance given by $(G_M, k_{M,x})$ a reduction to LCS over a binary alphabet that can be found in [Blin et al. 2012, Proposition 1]. In this case, the reduction yields an exponential number of words $w_1, \ldots, w_n$, each of exponential size, such that M accepts input x iff $w_1, \ldots, w_n$ 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 $w_1, \ldots, w_n$ are highly uniform. They are constructed from simple recurring patterns that grow and shrink in a synchronised way. This allows to encode the word $\$w_1\$ \ldots \$w_n\$$ as the unique word accepted by the intersection of polynomially many regular languages R_i. Furthermore, the R_i's can be constructed in polynomial time given the input x for M. This encoding uses similar techniques as those used to encode valid sequences of computations of polynomial space Turing machines in intersections of regular expressions [Kozen 1977, Lemma 3.2.3]. The correctness of the reduction thus obtained will be implied by the composition of the preceding ones.

Let $L \subseteq \Sigma^*$ be an arbitrary NEXPTIME language, and $M = (\Gamma, \Sigma, Q, \delta)$ a Turing machine deciding L with the following parameters:

— a tape alphabet $\Gamma = \{s_1 = \bot, s_2, ..., s_\gamma\}$, where $\bot$ is a blank symbol, and $\gamma > 0$,
— an input alphabet $\Sigma \subseteq \Gamma$,
— a set of states $Q = \{q_1, q_2 = q_y, q_3 = q_n, q_4, ..., q_r\}$, where q_1, q_y, q_n are respectively the start, accepting and rejecting states of M, and $r > 0$, and
— a transition function $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{-1,1\}}$.

We assume without loss of generality that M runs in time $t(n) = 2^n$ on inputs of size n. Let $x \in \Sigma^*$ be a input instance for L of size $n = |x|$. The first step in the reduction is to construct an exponential size formula $\phi_{M,x}$ (Cook's formula) which is satisfiable iff M accepts x. Recall that $\phi_{M,x}$ has variables:

$$\{Q_{i,k}, H_{i,j}, S_{i,j,\ell} : 1 \leq i \leq t(n), 1 \leq k \leq r, -t(n) \leq j \leq t(n), 1 \leq \ell \leq \gamma\}$$

whose interpetation is as follows. When M runs on input x we have that:

— $Q_{i,k}$ is true iff M is in state q_k in the i-th step of computation.
— $H_{i,j}$ is true iff M's head is in the j-th cell of the tape in the i-th step of computation.
— $S_{i,j,\ell}$ is true iff M's tape contains the symbol s_ℓ in its j-th cell in the i-th step of computation.

The formula $\phi_{M,x}$ then corresponds to the conjunction of six sub-formulas $\phi^1_{M,x}, \ldots, \phi^6_{M,x}$ that together encode valid computations of M on input x. These sub-formulas, with their intended meaning, are:

— *at each step, M is in one and only one state:*

$$\phi^1_{M,x} \;=\; \bigwedge_{1\leq i\leq t(n)} (Q_{i,1} \vee \cdots \vee Q_{i,r}) \wedge \bigwedge_{\substack{1\leq i\leq t(n)\\ 1\leq j<j'\leq r}} (\neg Q_{i,j} \vee \neg Q_{i,j'})$$

— *at each step, M's head is in one and only one tape position:*

$$\phi^2_{M,x} \;=\; \bigwedge_{1\leq i\leq t(n)} (H_{i,-t(n)} \vee \cdots \vee H_{i,t(n)}) \wedge \bigwedge_{\substack{1\leq i\leq t(n)\\ -t(n)\leq j<j'\leq t(n)}} (\neg H_{i,j} \vee \neg H_{i,j'})$$

— *at every step, each position of M's tape has one and only one symbol:*

$$\phi^3_{M,x} \;=\; \bigwedge_{\substack{1\leq i\leq t(n)\\ -t(n)\leq j\leq t(n)}} (S_{i,j,1} \vee \cdots \vee S_{i,j,\gamma}) \wedge \bigwedge_{\substack{1\leq i\leq t(n)\\ -t(n)\leq j\leq t(n)\\ 1\leq r<r'\leq \gamma}} (\neg S_{i,j,r} \vee \neg S_{i,j,r'})$$

— *the computation encoded by the variable assignments is valid regarding the transition function:*

$$\psi^Q_{i,j,\ell} \;=\; \left(\neg Q_{i,k} \vee \neg H_{i,j} \vee \neg S_{i,j,\ell} \vee \left\{ \bigvee_{(q_{k'},s,d)\in\delta(q_k,s_\ell)} Q_{i+1,k'} \right\}\right)$$

$$\psi^S_{i,j,\ell} \;=\; \left(\neg Q_{i,k} \vee \neg H_{i,j} \vee \neg S_{i,j,\ell} \vee \left\{ \bigvee_{(q,s,d)\in\delta(q_k,s_\ell)} H_{i+1,j+d} \right\}\right)$$

$$\psi^H_{i,j,\ell} \;=\; \left(\neg Q_{i,k} \vee \neg H_{i,j} \vee \neg S_{i,j,\ell} \vee \left\{ \bigvee_{(q,s_{\ell'},d)\in\delta(q_k,s_\ell)} S_{i+1,j,\ell'} \right\}\right)$$

$$\phi^4_{M,x} \;=\; \bigwedge_{\substack{1\leq i\leq t(n)\\ -t(n)\leq j\leq t(n)\\ 1\leq \ell\leq \gamma}} \psi^Q_{i,j,\ell} \wedge \psi^H_{i,j,\ell} \wedge \psi^S_{i,j,\ell}$$

— *the initial conditions are according to the given input:* if $x = s_{k_1} \cdots s_{k_n}$, then:

$$\phi^5_{M,x} \;=\; Q_{1,1} \wedge H_{1,1} \wedge S_{1,1,k_1} \wedge \cdots \wedge S_{1,n,k_n} \wedge S_{1,n+1,1} \wedge \cdots \wedge S_{1,t(n),1}$$

— *the final step of computation has an accepting state:*

$$\phi^6_{M,x} \;=\; Q_{t(n),2}$$

Given that $t(n) = 2^n$, the number of clauses in $\phi_{M,x}$ is

$$k_{M,x} = 2^{3n-1} + 2^{2n-1}[3 + \gamma(\gamma+7)] + 2^{n-1}[6 + r(r+1)] + 3$$

whose binary encoding $[k_{M,x}]$ requires $O(n)$ bits. The binary encoding $[k_{M,x}]$ can clearly be constructed in polynomial time given $n \in \mathbb{N}$ and constants $\gamma, r > 0$.

The second step in this reduction is to construct an input instance $(G_{M,x}, k_{M,x})$ for INDSET such that $G_{M,x}$ is a graph with a large independent set if and only if $\phi_{M,x}$ is satisfiable. We use the standard reduction from SAT to INDSET. That is, the set $V(G_{M,x})$ of nodes of $G_{M,x}$ corresponds to the set of literals associated with $\phi_{M,x}$, i.e.,

$$V(G_{M,x}) \;=\; \{Q_{i,j}, H_{i,j}, S_{i,j,\ell}, \neg Q_{i,k}, \neg H_{i,j}, \neg S_{i,j,\ell} \mid \\ 1 \leq i \leq t(n),\; -t(n) \leq j \leq t(n),\; 1 \leq \ell \leq \gamma,\; 1 \leq k \leq r\}.$$

The set $E(G_{M,x})$ of edges of $G_{M,x}$ contains all those pairs of literals that either (1) appear in the same clause together, or (2) one is the negation of the other. From this construction it is easy to see that $\phi_{M,x}$ is satisfiable if and only if $G_{M,x}$ has an independent set of size $k_{M,x}$ (recall that this is the number of clauses in $\phi_{M,x}$).

Finally, we describe the construction of the words such that they have a long common subsequence iff $G_{M,x}$ has an independent set of size $k_{M,x}$. To this end, we use a reduction from [Blin et al. 2012, Proposition 1] which is explained next. Given an undirected graph G with nodes $\{1, ..., m\}$, construct for each edge $(i, j) \in G$ (with $1 \leq i < j \leq m$) a word $w_{i,j}$ over the alphabet $\{a, b\}$ as follows:

$$w_{i,j} \;=\; (a^m b)^{i-1} a^m (a^m b)^{j-i} a^m (a^m b)^{m-j}.$$

Further, consider the word $w_0 = (a^m b)^m$. It follows that the words in the set

$$W(G) \;=\; \{w_0\} \cup \{w_{i,j} \mid (i, j) \text{ is an edge of } G\}$$

have a common subsequence of length $m^2 + k$ if and only if G has an independent set of size k. In our case, $k = k_{M,x}$ is the number of clauses in Cook's formula $\phi_{M,x}$ and $m = O(k)$ is the number of literals this formula mentions. Thus, $m^2 + k$ can be represented using $O(n)$ bits. Even more, it is is possible to construct in polynomial time from the input the binary representation of k, m, and $m^2 + k$.

Now, the question is whether starting from the input x it is possible to construct in polynomial time a set of regular expressions that "encodes" (in the form $(\$, \$)w_0(\$, \$)w_1(\$, \$) \ldots (\$, \$)w_q(\$, \$)$) the input to LCS that is obtained by composing the three reductions we explained above. We explain next why this is possible.

First of all, since both the number of words in the set $W(G_{M,x})$ and their length is exponential in the size of the input, we need to use counters that ensure consistency (as we do in the proof of Proposition 6.5). For instance, since the word w_0 is in our case of the form $(a^m b)^m$, and the binary representation of m can be constructed in polynomial time from the input, it is possible to construct in polynomial time a set of regular expressions that defines precisely a "padded" version of w_0 with counters that ensure that the word is of the required form and length (for more details, see the proof of Lemma 6.8). Similarly, it is possible to construct in polynomial time a set of regular expressions that defines the "padded" version of the word $w_{i,j}$, given binary representations of integers $1 \leq i, j \leq m$.

Following in this way, we can construct in polynomial time a set $S_{M,x}$ of regular expressions that defines a padded version of the concatenation of all the words in the set $W(G_{M,x})$ separated by delimiter $\$$. In this new padding we add information that allows to enumerate the pairs that correspond to edges in the graph $G_{M,x}$. Recall that such edges appear between elements in the same clause of $\phi_{M,x}$, or between complementary literals. Since $\phi_{M,x}$ is generated in a very uniform way, it is possible to do this with special counters. In particular, we should have a counter that specifies the kind of clause, as well as the indexes of the literals being taken into consideration at the time. For instance, when specifying that literals $Q_{i,j}$ and $Q_{i,j'}$, for $1 \leq i \leq t(n)$ and $1 \leq j < j' \leq r$, are adjacent due to the first type of clauses in $\phi_{M,x}$ (i.e., the ones of the form $Q_{i,1} \vee \cdots \vee Q_{i,r}$), we will have a counter c_1 set to 1 stating that we are codifying

adjacency produced by clauses of the first sort, another counter c_2 set to i (in binary) stating that we are codifying adjacency with respect to literals of the form $Q_{i,\ell}$, and two counters c_3 and c_4 set to j and j', respectively, stating that it is $Q_{i,j}$ and $Q_{i,j'}$ that are adjacent. Following this, we will have the word

$$(a^m b)^{\ell-1} a^m (a^m b)^{\ell'-\ell} a^m (a^m b)^{m-\ell'}$$

that codifies the existence of this edge in the reduction to LCS. Here, ℓ and ℓ' are suitable binary encodings for the literals $Q_{i,j}$ and $Q_{i,j'}$, respectively. If $j' < r$, we increase c_4 by 1. If $j' = r$ and $j < r$, we increase c_3 by 1. If $j = j' = r$, we increase c_2 by 1 if $i < t(n)$. Otherwise, we switch to clauses of the second sort by increasing c_1 by 1.

Recall that our goal is to check whether the exponential-length words in the exponential-size set $W(G_{M,x})$ have a common subsequence of length $m^2 + k$, where m and k are represented in binary. Further, we would like to do this over a suitable succinct representation of $W(G_{M,x})$ given by the intersection of polynomially many regular expressions. The obvious candidate for this is the set $S_{M,x}$ of regular expressions defined above. The problem is that $S_{M,x}$ represents a padded version of $W(G_{M,x})$, and the presence of the padding counters difficults the detection of a common subsequence of the desired length for the words in $W(G_{M,x})$. This can be solved as follows. Assume that the regular expressions in $S_{M,x}$ are defined over alphabet $\Sigma \cup \{\$\}$, where $\$$ is the delimiter that separates the different words in $W(G_{M,x})$ and Σ is the alphabet that is used to represent the padded versions of such words. Then we define a new set $S'_{M,x}$ of regular expressions over $(\Sigma \cup \{\$\}) \times (\Sigma \cup \{\$\})$ in such a way that its intersection accepts the word $u \times u'$, for $u, u' \in (\Sigma \cup \{\$\})^*$, such that u' is the word accepted by the intersection of the regular languages in $S_{M,x}$ and u is the word obtained from u' by replacing each symbol not in $\{\$, a, b\}$ by a fresh dummy symbol $\&$. In other words, u is obtained from u' by replacing any information about the counters with this dummy symbol and keeping all the information about the words in $W(G_{M,x})$ and the delimiter $\$$ that separates them. As before, the set $S'_{M,x}$ can be constructed in polynomial time. Furthermore, it is possible to construct in polynomial time a binary integer m' such that the following are equivalent:

(1) The words in $W(G_{M,x})$ have a common subsequence of length $m^2 + k$.
(2) The words $u_1, \ldots, u_q$ have a common subsequence of length m', where $u_1, \ldots, u_q$ are all the words over Σ that can be found in u among consecutive appearances of the delimiter $\$$.

It is then possible to reduce in polynomial time the problem of acceptance of x by M to the instance of SUCCINCT-LCS given by the regular expressions in $S'_{M,x}$ and the binary integer m'. This finishes the proof of Proposition 6.6.