UNIVERSITY OF CHILE
FACULTY OF PHYSICS AND MATHEMATICS
DEPARTMENT OF MATHEMATICAL ENGINEERING

NEW COMPLEXITY BOUNDS FOR EVALUATING CRPQs WITH PATH COMPARISONS

Submitted to the University of Chile in fulfillment of the thesis requirement to obtain the degree of Engineer in Mathematics.

PABLO BENITO MUÑOZ FUENTES

ADVISOR:
PABLO BARCELÓ BAEZA

COMMITTE:
CLAUDIO GUTIERREZ GALLARDO
MARTIN MATAMALA VASQUEZ

SANTIAGO - CHILE
JANUARY 2014

Abstract

For many problems arising in the setting of querying data in graph databases (such as finding semantic associations in RDF graphs, exact and approximate pattern matching, sequence alignment, etc.) it is a common requirement to ask for entities joint by a sequence of relational labels conforming to some regular pattern. For this purpose, the query language CRPQ($\mathcal{S}$) have been proposed to extend the widely studied class of conjunctive regular path queries (CRPQs), which is insufficient for this task, in order to compare paths using relations on words from the class $\mathcal{S}$.

Little is known about the precise computational complexity of evaluating queries from CRPQ($\mathcal{S}$) in large graph databases when $\mathcal{S}$ is a single relation of interest for its natural appearance in database applications, such as *subsequence* ($\preceq_{\text{ss}}$), *suffix* ($\preceq_{\text{suff}}$) and *subword* ($\preceq_{\text{sw}}$). This question is thus studied in this thesis, providing new bounds for the complexity of evaluating queries in CRPQ($\preceq_{\text{ss}}$), CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$). The first one is shown to be impractical, by constructing a query which is NP-complete to evaluate. The evaluation of the later two is shown to be PSPACE-complete by reducing the problem to *word equations with regular constraints.* The class CRPQ($\preceq_{\text{suff}}$) is shown to be practical, by showing the evaluation of its queries to be in NLOGSPACE when these are considered fixed, matching the complexity of standard query languages.

This thesis also raises interesting theoretical questions about word equations with regular constraints. Namely, what is the complexity of solving fixed equations with constraints as input, which to the best of the author's knowledge is an open question in the literature. A result is established for the simplest case, showing that for a class of equations satisfiability can be decided in NLOGSPACE.

Resumen

En muchos problemas que surgen en el contexto de consultar información en bases de datos estructuradas sobre grafos (como encontrar asociaciones semanticas en grafos RDF, encontrar emparejamientos exactos o aproximados the patrones de texto, realizar alineación de secuencias de texto, etc.) es un requerimiento común el buscar entidades unidas por secuencias de etiquetas relacionales de acuerdo a un patrón regular. Para este propósito, el lenguaje de consulta CRPQ($\mathcal{S}$) ha sido propuesto para extender la altamente estudiada clase de consultas conjuntivas por caminos regulares (CRPQs por su sigla en inglés), la cual es insuficiente para esta tarea, realizando comparación de caminos con relaciones en la clase $\mathcal{S}$.

Poco es conocido acerca de la complejidad computacional precisa de la evaluación de consultas en CRPQ($\mathcal{S}$) cuando $\mathcal{S}$ es una relación de interés por aparecer naturalmente en aplicaciones en bases de datos, como lo son *subsecuencia* ($\preceq_{\text{ss}}$), *sufijo* ($\preceq_{\text{suff}}$) y *subpalabra* ($\preceq_{\text{sw}}$). Esta pregunta es consecuentemente estudiada en esta tesis, proporcionando nuevas cotas de complejidad para la evaluación de consultas en los lenguajes CRPQ($\preceq_{\text{ss}}$), CRPQ($\preceq_{\text{suff}}$) y CRPQ($\preceq_{\text{sw}}$). Se muestra que el primer lenguaje es dificil de ser practicable, construyendo una consulta en él cuya complejidad de evaluación es NP-completo. Se muestra también que la evaluación de consultas en los últimos dos lenguajes puede realizarse en PSPACE, mediante la reducción del problema a *ecuaciones de palabras con restricciones regulares.* Adicionalmente, se muestra que la classe CRPQ($\preceq_{\text{suff}}$) es práctica, construyendo un algoritmo de evaluación cuya complejidad, cuando la consulta es considerada una constante, está en NLOGSPACE , la cuál es una complejidad de evaluación estandar en este contexto.

Esta tesis plantea además interesantes preguntas teóricas sobre ecuaciones de palabras con restricciones regulares. Más precisamente, cuál es la complejidad de resolver ecuaciones fijas con restricciones como entrada, la cual es una pregunta abierta en la literatura al leal saber y entendimiento del autor. Un resultado es establecido para el caso más simple, mostrando una clase de ecuaciones cuya satisfacibilidad con restricciones regulares puede ser decidida en NLOGSPACE.

Agradecimientos

Quiero agradecer en primer lugar a mi familia, por su apoyo incondicional durante todo mi proceso de titulación.

Agradezco también al profesor Martín Matamala por haber estado siempre disponible para discutir dificultades que apareciesen durante mi trabajo.

Además, agradezco a mis compañeros del Departamento de Ingeniería Matemática que se mostraron disponibles para discutir problemas relacionados a mi memoria, y para introducirme a ciertas herramientas de programación, en particular a Sebastian Bustamante.

Por último pero no menos importante, quisiera agradecer a mi profesor guía Pablo Barceló por continuamente depositar su confianza en mi, junto con una buena cuota de exigencia y motivación que fueron claves para el desarrollo de mi memoria.

Contents

Chapter 1

Introduction

Graph databases are mathematical objects able to describe real world entities and the relations between them. A standard model for abstracting these is an edge-labelled directed graph, whose vertices represent the entities, and whose edge labels indicate the type of relation between them (see [5] for a survey). Many are the areas that motivate their use and study, starting early in the 80s with hypertext systems [19, 54], semi-structured data [2, 13] and object databases [28] in the 90s, and in the last decades including the semantic web [7, 45], social networks [4, 23, 50, 51], biological networks [37, 38, 40] and transportation networks [11], among many others (see [57] for a survey).

In these areas of application, data is required to be accessed in some particular way in order to retrieve interesting information. Accordingly, querying tools and prototypes are designed to match these requirements. These are named *query languages*, and the main concern from a database point of view is to study at what extent they are expressive enough for their intended purpose, and what is the computational cost of evaluating them in large databases, or its *data complexity* [55].

For this reason, languages for querying graph databases have been developed and studied since the late 80s. They usually query the topology of the graph, often leaving data that might be stored in its vertices to standard database engines. They are designed combining various reachability patterns, according to the applications requirements intended to be addressed.

A common requirement from graph database users, for instance, is to find pairs of entities joint by a sequence of relational labels conforming to some specified pattern. This gave origin to one of the main building blocks for query languages, which are *regular path queries*, or RPQs [21, 20], in which this pattern is specified by a regular language. A natural extension for these are the *conjunctive regular path queries*, or CRPQs [25, 15], which use conjunction of RPQs along with existential quantification to create more complex queries. These have been extensively studied, showing good properties such as their low data complexity, which is in NLogSpace.

Despite such properties, the expressiveness of CRPQs became insufficient in applications such as the Semantic Web or biological networks, due to their inability to *compare* labelled paths joining entities. In these areas, it is a usual requirement to compare paths based for instance on specific semantic association in the case of RDF languages [6], or on similarity, trough their edit distance, for biological networks.

In order to address this limitation, an extension of CRPQs with relations that permit to compare paths was proposed [10]. It used *regular* relations for comparing labelled paths, which include equality, equal length, or fixed edit distance, among others. The extension of CRPQs with this type of relations, or CRPQ(REG), was also shown to have polynomial time data complexity.

The expressiveness of CRPQ(REG), however, was still short in many applications requiring to compare paths based on richer relations. For instance, semantic associations between paths in RDF applications often deal with relations such as *subword* or *subsequence*. Yet this kind of relations are not regular, but *rational*. Nevertheless, the use of rational relations on query languages is to be done with extreme care, for simply changing regular by rational in CRPQ(REG) makes query evaluation an undecidable problem [10].

According to this, several solutions were considered and studied to tackle this expressiveness limitation while keeping an acceptable complexity of query evaluation. The most direct one, extending CRPQ(REG) with rational instead of regular general relations, makes query evaluation undecidable, as it was mentioned before. Another proposed solution was to extend CRPQ(REG) with particular rational relations that are of interest in practice, such as *suffix*, *subword* or *subsequence*. Nevertheless, the evaluation of such queries remained either undecidable, or non-elementary in data complexity, and thus prohibitively high [9]. By additionally restricting the syntactic shape of queries to satisfy certain acyclic forms, the evaluation complexity can be shown to be in PTime. This, however, is too restrictive for some important applications, since relations such as the ones mentioned above do not naturally define acyclic patterns.

A query language that was shown to have a balanced trade-off between expressiveness and complexity was obtained by extending CRPQs directly with specific binary rational relations used in practice, such as *suffix, subsequence* and *subword*. This leads to the following three query languages : CRPQ($\preceq_{\text{suff}}$), CRPQ($\preceq_{\text{ss}}$) and CRPQ($\preceq_{\text{sw}}$). The data complexity of evaluation for the first two was shown to be in NP [9]. Some questions about this query language, however, remained unanswered. They relate to the precise complexity of evaluating these queries, asking if the given bounds were tight or if better evaluation algorithms could be found. Moreover, the question about the decidability of queries in CRPQ($\preceq_{\text{sw}}$) was left open. These questions are studied in detail in this thesis.

In particular, the following questions are studied:

1. What is the precise complexity of evaluation for queries in CRPQ($\preceq_{\text{suff}}$), CRPQ($\preceq_{\text{ss}}$) and CRPQ($\preceq_{\text{sw}}$)?

2. What are the limits of tractability for the different combination of features of the query languages mentioned above?

The following results are established trough this work:

First, by reinterpreting the tools that have been used in the literature to study the problem, it is shown that the evaluation of CRPQ($\preceq_{\text{suff}}$) and CRPQ($\preceq_{\text{sw}}$) queries is linked to the problem of solving word equations with regular constraints [46]. This provides new and strong combinatorial tools to the subject, and shows that the evaluation of CRPQ($\preceq_{\text{sw}}$) queries is indeed decidable, and can be done in PSpace.

Secondly, the complexity of evaluating CRPQ($\preceq_{\text{suff}}$) queries is studied in depth, and a better algorithm is constructed showing that their evaluation is in NLogSpace, hence in PTime.

Finally, in the case of *subsequence*, the previously known NP algorithm for evaluating queries in CRPQ($\preceq_{\text{ss}}$) is shown to be optimal by proving a matching NP lower bound.

These results show, first, that the previously known NP upper bound for evaluating CRPQ($\preceq_{\text{suff}}$) queries can be lowered to PTime. Consequently, it matches the data complexity of standard query languages, such as CRPQs and CRPQ(REG). In contrast to this, CRPQ($\preceq_{\text{ss}}$) queries are unlikely to be practical, since this class contains queries which are NP-hard to evaluate. In the case of CRPQ($\preceq_{\text{sw}}$), the question now is whether the PSpace algorithm given for evaluation is optimal, or if better algorithms can be found.

Additionally, this work raises interesting theoretical questions about the tools used to establish these results. These relate to the complexity of solving fixed word equations with variable regular constraints, which to the best of the author's knowledge is an open question in the literature. A partial answer is given in the simplest case, showing that for a class of equations satisfiability can be efficiently decided in NLogSpace.

1.1 Organization

In Chapter 2, the reader is introduced to the concepts used throughout this work: preliminaries on formal languages, computable string relations, CRPQs and the main tools used in their study, along with previous results concerning them. Chapter 3 is devoted to study CRPQs comparing paths with a vast class of relations including suffix and subword, for which their evaluation is shown to reduce to solving word equations with regular constraints, showing the complexity of their evaluation to be in PSpace. In Chapter 4, the query language CRPQ($\preceq_{\text{suff}}$) is studied in depth, showing its data complexity of evaluation to be in NLogSpace. It is also shown in Chapter 4 that for a class of word equations, satisfiability of them with variable regular constraints is decidable in NLogSpace. In Chapter 5 a query in CRPQ($\preceq_{\text{ss}}$) is constructed such that its data complexity of evaluation is NP-hard. Finally, the results of this thesis are summarized in Chapter 6, which also points out open questions that will be object of study in following research.

Chapter 2

CRPQs with path comparisons

In this chapter, the main problems studied in this thesis are presented, along with previous results obtained for them. After providing basic definitions and notation, the query language of *conjunctive regular path queries with path comparisons* is defined. Its ability to achieve its intended purpose is then put into context by providing some known results about their evaluation when considering different classes of relations for comparing paths. This leads to some open questions about their complexity of evaluation, which are finally explained to establish the goals of this work.

2.1 Preliminaries

Throughout this work, capital greek letters Σ, Γ, Ψ shall be used for denoting finite collections of symbols, or *alphabets*. The free monoid generated by Σ is the set $(\Sigma^*, \cdot, \varepsilon)$ of finite strings with symbols from Σ, along with the associative concatenation operator $\cdot$ and the empty string ε. Given a string $w \in \Sigma^*$, $|w|$ denotes the number of symbols conforming w. For a symbol $a \in \Sigma$, $|w|_a$ counts the number of occurrences of symbol a in w. For $w \in \Sigma^*$, $w[i..j]$ stands for the substring of w bounded by positions i, j, $w[i]$ for $w[i..i]$, and $w[i..]$ for $w[i..|w|]$. For $w \in \Sigma^*$, $alph(w) \subseteq \Sigma$, is the set of symbols appearing in string w. Finally, for $\Sigma_0 \subsetneq \Sigma$ and $w \in \Sigma^*$, $w|_{\Sigma_0}$ stands for the string obtained from w by removing any symbol not in Σ_0. For an integer $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, ..., n\}$.

2.1.1 Computable string relations

This work is inherently related to computable relations over Σ^*, and for complexity purposes, they are classified according to the computing devices needed in order to recognize them. Three classical classes of relations studied in this work are defined: *recognizable* (REC), *regular* (REG), and *rational* (RAT) [12, 17]. When the arity of the relation is relevant, it is added as subscript (*e.g.* the class of binary rational relations is RAT_2).

Recognizable n-ary relations are subsets $R \subseteq (\Sigma^*)^n$ for which a finite monoid M and a morphism $f : (\Sigma^*)^n \to \mathcal{M}$ exist, such that $R = f^{-1}(M_0)$, for some $M_0 \subseteq M$. Recognizable relations are known to coincide with finite unions of sets $L_1 \times \cdots \times L_n$, where each

$L_i \subseteq \Sigma^*$ is a regular language [12].

Regular n-ary relations are subsets $R \subseteq (\Sigma^*)^n$ recognized by a finite automaton $\mathcal{A}$ over a padded alphabet $\Sigma_\perp^n = (\Sigma \cup \{\perp\})^n$, where $\perp$ is symbol not in Σ. A string n-tuple $\bar{w} = (w_1, ..., w_n) \in (\Sigma^*)^n$ can be seen as a string $\otimes\bar{w} \in (\Sigma_\perp^n)^*$, whose i-th symbol $\bar{a}_i = (a_1^i, ..., a_n^i) \in \Sigma_\perp^n$ is a tuple whose j-th element, a_j^i, is the i-th symbol of w_j, if $i \leqslant |w_j|$, and $\perp$ otherwise. Accordingly, $\mathcal{A}$ processes $\bar{w}$ in a *synchronous* fashion (see Figure 2.1).

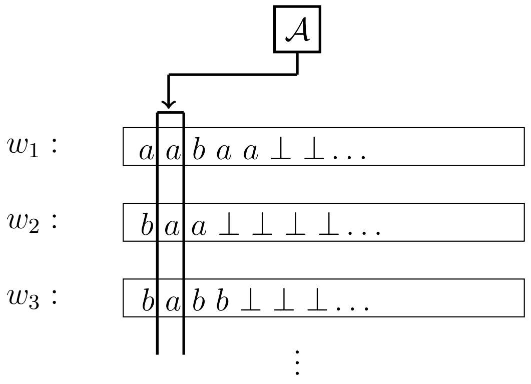
$\mathcal{A}$
w_1 :
$a\ a\ b\ a\ a\ \perp\ \perp \dots$
w_2 :
$b\ a\ a\ \perp\ \perp\ \perp\ \perp \dots$
w_3 :
$b\ a\ b\ b\ \perp\ \perp\ \perp \dots$
⋮

Figure 2.1: A finite automaton over $(\Sigma_\perp^n)^*$ reading a string $\otimes(aabaa, baa, babb, ...)$.

Example 2.1.1. Examples of regular relations prevalent in database applications are *prefix, equal length* and *fixed edit distance.* For instance, in the case of prefix, for a given pair of strings $(w_1, w_2) \in \Sigma^* \times \Sigma^*$ it is easy to recognize w_1 as a prefix of w_2 with a synchronous two-headed automaton, which simultaneously reads every symbol from both strings rejecting whenever they are not equal and accepting if the first string ends.

Rational n-ary relations are subsets $R \subseteq (\Sigma^*)^n$ that can be described by regular expressions with tuple symbols from $(\Sigma \cup \{\varepsilon\})^n$ using the union, concatenation and Kleene star operators. Equivalently, rational relations are languages recognized by n-tape automata, whose read-only-once heads can move independently in each step. In contrast to regular relations, the automata recognizing rational relations processes strings tuples *asynchronously* (see Figure 2.2).

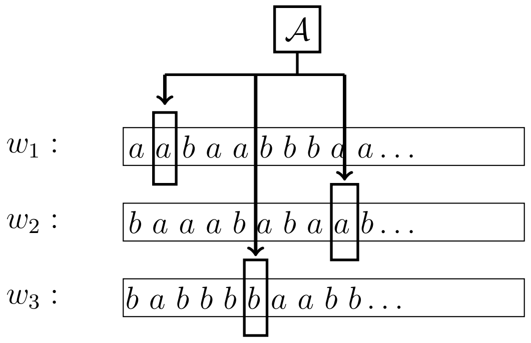
$\mathcal{A}$
w_1 :
$a\ a\ b\ a\ a\ b\ b\ b\ a\ a \dots$
w_2 :
$b\ a\ a\ a\ b\ a\ b\ a\ a\ b \dots$
w_3 :
$b\ a\ b\ b\ b\ b\ a\ a\ b\ b \dots$

Figure 2.2: A finite automaton over $(\Sigma^*)^3$ reading a string $(aabaabbbaa..., baaababaab..., babbbbaabb...)$.

Example 2.1.2. Examples of rational relations naturally appearing in database applications are *suffix* ($\preceq_{\text{suff}}$), *subword* ($\preceq_{\text{sw}}$) and *subsequence* ($\preceq_{\text{ss}}$):

1. $u \preceq_{\text{suff}} v \iff \exists p \in \Sigma^*,\ v = pu$
2. $u \preceq_{\text{sw}} v \iff \exists p, s \in \Sigma^*,\ v = pus$
3. $u \preceq_{\text{ss}} v \iff u$ is obtained by removing some symbols from v.

These relations can be defined by the following regular expressions over $(\Sigma \cup \{\varepsilon\}) \times (\Sigma \cup \{\varepsilon\})$:

1. $\preceq_{\text{suff}} := (\bigcup_{a\in\Sigma}(\varepsilon, a))^* \cdot (\bigcup_{a\in\Sigma}(a, a))^*$
2. $\preceq_{\text{sw}} := (\bigcup_{a\in\Sigma}(\varepsilon, a))^* \cdot (\bigcup_{a\in\Sigma}(a, a))^* \cdot (\bigcup_{a\in\Sigma}(\varepsilon, a))^*$
3. $\preceq_{\text{ss}} := (\bigcup_{a\in\Sigma}\{(\varepsilon, a), (a, a)\})^*$

These three classes of regular relations are known to coincide when unary, that is $\text{REC}_1 = \text{REG}_1 = \text{RAT}_1$, but this is not the case for general arity. Indeed, for $k > 1$, $\text{REC}_k \subsetneq \text{REG}_k \subsetneq \text{RAT}_k$ [12]. For instance, $\preceq_{\text{pref}} \in \text{REG}_2 \backslash \text{REC}_2$ and $\preceq_{\text{suff}} \in \text{RAT}_2 \backslash \text{REG}_2$.

It is typically assumed that relations are given as input for a problem encoded by the automata that defines them. Thus, for instance, a recognizable m-ary relation $R = \bigcup_i L_1^i \times \cdots \times L_m^i$, is given by the automata $\{N_j^i(Q_j^i, \Sigma, \delta_j^i, s_j^i, F_j^i)\}_{i,j}$ (without ε-transitions) each recognizing the regular language L_j^i.

2.1.2 Complexity measures for database querying

The standard model for a graph database (graph-db henceforth) is that of an edge-labelled directed graph $G(V, E)$ with $E \subseteq V \times \Sigma \times V$, for a fixed finite alphabet Σ [5]. A *query* is defined by a logical sentence, and associates with each graph-db $G(V, E)$ a set of tuples of vertices in V satisfying it. *Query languages* are henceforth classes of queries closed under isomorphism. The *query evaluation problem* is: Given a graph-db G and query Q, does Q hold in G? When this is so, it is denoted as $G \models Q$.

From a standard complexity theory point of view, the cost of evaluating a query Q over a graph-db G should be measured according to the size of the given input: $|G| + |Q|$. This approach is not conducive, however, in database applications, since as early noted by theoreticians [55, 29, 16], the scales of the database and query sizes are incomparable; databases in the real world are huge in size when compared to the size of the queries, having different impacts in the complexity of the problem.

This observation led to distinguish the role of the query and the database when measuring computational complexity. Accordingly, two measures of complexity have been studied [55]: *data complexity* and *combined complexity*. For a query language $\mathcal{L}$:

- Data complexity studies the problem $\{\langle G\rangle : G \models Q\}$ for a fixed query $Q \in \mathcal{L}$.

- Combined complexity studies the problem $\{\langle G, Q\rangle : G \models Q,\ G \text{ graph-db and } Q \in \mathcal{L}\}$

2.2 Conjunctive regular path queries with path comparisons

Let $G(V, E)$, with $E \subseteq V \times \Sigma \times V$ and Σ a finite alphabet, be a graph-db. A *path* ρ between vertices u and v in V is a sequence $\rho = v_0 a_1 v_1 a_2 v_2 \dots a_m v_m$, such that $v_0 = u$, $v_m = v$ and for each $i < m$, $(v_i, a_{i+1}, v_{i+1}) \in E$. The *label* of such a path ρ, denoted by $\lambda(\rho)$, is the string $a_1 ... a_m \in \Sigma^*$.

Queries over a graph-db are typically *navigational*, *i.e.* they allow to recursively traverse the edges of a graph-db while checking a regular condition. The main building block for navigational queries over graph-dbs is the class of *regular path queries*, or RPQs [21, 20]. An RPQ is nothing else than a regular language over Σ. Formally, these are expressions of the form $Q(x, y) \ :\ x \xrightarrow{L} y$, for a regular language $L \subseteq \Sigma^*$, whose *evaluation* over a graph-db G is the set

$$Q(G) = \{(u, v) \in V \times V : \text{there exists } \rho \text{ a path between } u \text{ and } v, \text{such that } \lambda(\rho) \in L\}$$

The cost of evaluating RPQs is low, since their data complexity is in NLogSpace. When originally introduced, however, RPQs where argued to be too simple to be useful in practice, lacking features as the possibility to ask for richer patterns through conjunction of atoms [44]. The query language addressing this limitation is the closure under conjunction and existential quantification of RPQs, which forms the class of *conjunctive regular path queries*, or CRPQs [25, 15, 22]. A CRPQ $\phi(\bar{x})$ is formally written as

$$\phi(\bar{x}) \ :\ \exists \bar{y} \bigwedge_{i=1}^{m} Q_i(u_i, u'_i), \quad \text{with} \quad Q_i(x, y) := \left(x \xrightarrow{L_i} y\right) \text{ for each } 1 \leqslant i \leqslant m$$

For a given graph-db G, a vertex tuple $\bar{a}$, and $\phi \in$ CRPQ, $G \models \phi(\bar{a})$ if and only if in G there is a tuple of vertices $\bar{b}$, and an identification v_i, v'_i of vertices from $\bar{a}, \bar{b}$ interpreting the variables u_i, u'_i from ϕ such that for every $i \leqslant m$, $(v_i, v'_i) \in Q_i(G)$. The arity of the query, *i.e.* the number of non-quantified variables in $\bar{x}$, is denoted $ar(\phi)$. Thus for a CRPQ ϕ, its *evaluation* over G is

$$\phi(G) = \{\bar{a} \in V^{ar(\phi)} \ :\ G \models \phi(\bar{a})\}$$

When $ar(\phi) = 0$, meaning that all of its variables are quantified, the query is said to be *Boolean*, and its evaluation $\phi(G)$ equals 1 or 0 according to the truth value of $G \models \phi$.

Example 2.2.1. Let $G_0(V, E_0)$ be a directed graph and s, t two of its vertices. Let $G(V, E)$ be the graph-db obtained by labelling every edge in G with a symbol σ from $\{a, b, c\}$ such that

$$(u, \sigma, v) \in E \ \Leftrightarrow\ (u, v) \in E_0 \ \wedge\ \sigma = \begin{cases} a & \text{if } u = s \\ b & \text{if } u \neq s \ \wedge v \neq t \\ c & \text{if } v = t \end{cases}$$

Consider now the Boolean CRPQ query $\phi \ : \ \exists x, y, \ x \xrightarrow{L} y$ with $L = a \cdot (a + b)^* \cdot c$. It is easy to see from this construction that $\phi(G) = 1$ if and only if t is reachable from s in G_0. From this, it follows that the data complexity of evaluation of CRPQs is NLogSpace hard, since the query ϕ is independent from the reachability instance $\langle G, s, t\rangle$ ..

Similarly, the data complexity of CRPQs is in NLogSpace, and its queries are thus practical to be evaluated. However, in order to be useful in some applications, more features commonly required have to be considered. More precisely, the definition for CRPQs does not allow *comparisons* of labelled paths joining different pairs of vertices (u_i, u_i'), since these paths are not specified in the syntax of its queries. For instance, the CRPQ query $Q(x, z) \ : \ \exists y, \ \left(x \xrightarrow{L_1} y\right) \wedge \left(y \xrightarrow{L_2} z\right)$ asks for pairs of vertices (u, v) joint by a labelled path with a first part conforming to L_1 and the second to L_2. Yet it would not be possible to additionally ask in the query the label in L_2 to be a suffix of the one in L_1, or any other string relation between them.

The class of *conjunctive regular path queries with path comparisons*, or CRPQ($\mathcal{S}$), addresses this limitation by specifying in their syntax pairs of paths to be compared, and a class $\mathcal{S}$ of binary relations over Σ^*, such as REG or RAT, used for this purpose [9]. Formally, $\phi \in$ CRPQ($\mathcal{S}$) if

$$\phi(\bar{x}) \ : \ \exists \bar{y} \ \left[\bigwedge_{i=1}^{m} \left(u_i \xrightarrow{\chi_i \ : \ L_i} u_i'\right) \ \wedge \ \bigwedge_{(i,j) \in I} S_{i,j}(\chi_i, \chi_j) \right]$$

where $I \subseteq [m] \times [m]$ and for each $(i, j) \in I$, $S_{i,j}$ is a relation in $\mathcal{S}$. In this syntax, variables χ_i have the role of representing paths between vertices u_i and u_i' in the graph-db. It is important to note that for CRPQ($\mathcal{S}$) queries, the relations $\{S_{i,j}\}_{(i,j)\in I} \subset \mathcal{S}$ and the index set $I \subseteq [m] \times [m]$ are part of the query. Additionally, if the complexity of evaluation when using a particular relation is to be studied, the class $\mathcal{S}$ could be a singleton, such as $\{\preceq_{\text{suff}}\}$, in which case this class denoted CRPQ($\preceq_{\text{suff}}$).

For a graph-db G, a vertex tuple $\bar{a}$ and a CRPQ($\mathcal{S}$) query ϕ, $G \models \phi(\bar{a})$ is true if and only if in G there is a vertex tuple $\bar{b}$, an identification v_i, v_i' of vertices from $\bar{a}, \bar{b}$ interpreting the variables u_i, u_i', and paths ρ_i between vertices v_i and v_i' interpreting the variables χ_i, such that for every $i \leqslant m$, $\lambda(\rho_i) \in L_i$ and for every $(i, j) \in I$, $(\lambda(\rho_i), \lambda(\rho_j)) \in S_{i,j}$. Likewise, the arity of $\phi \in$ CRPQ($\mathcal{S}$) is the number of its non-quantified vertex variables in $\bar{x}$, and their evaluation over a graph-db $\phi(G)$ is defined in the same way as for CRPQs.

Example 2.2.2. Consider a set $\langle L_1, ...L_m\rangle$ of regular languages over an alphabet Σ. Let G be the trivial graph-db $G(\{u\}, E)$ with $E = \{(u, a, u) \ : \ a \in \Sigma\}$, and the index set $I = \{(1, 2), (2, 4), ..., (m - 1, m), (m, 1)\}$ with the edges of a directed cycle on m

vertices. Consider also the Boolean query in CRPQ($\preceq_{\text{sw}}$) given by

$$\phi : \exists x \left[\bigwedge_{i=1}^{m} \left(x \xrightarrow{\chi_i \,:\, L_i} x \right) \wedge \bigwedge_{(i,j)\in I} \chi_i \preceq_{\text{sw}} \chi_j \right]$$

It is easy to see from this construction that $\phi(G) = 1$ if and only if there is a string in Σ^* present in each language L_i, since $\preceq_{\text{sw}}$ is is transitive and anti-symmetric. It follows from this that evaluation of CRPQ($\preceq_{\text{sw}}$) is PSpace-hard.

Example 2.2.3. Consider a graph-db $G(V, E)$ with $\{a, b\} \in \Sigma$ and the CRPQ($\preceq_{\text{ss}}$) query Q given by:

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

The set $Q(G)$ contains all the vertices u in V such that there are two paths starting from u, one ending with label a, the other ending with label b, and such that every edge label appearing in the second also appears, in the same order, in the first.

The *query evaluation problem*, EvalCRPQ ($\mathcal{S}$), receives as input a tuple $\langle G, \bar{v}, \phi \rangle$ of a graph database $G(V, E)$, a vertex tuple $\bar{v} \in V^{ar(\phi)}$ and a query $\phi \in$ CRPQ($\mathcal{S}$), asking if $\bar{v} \in \phi(G)$. This corresponds to the combined complexity of evaluation. When the query $\phi \in$ CRPQ($\mathcal{S}$) is fixed, the problem is denoted EvalCRPQ ($\mathcal{S}, \phi$). This corresponds to the data complexity of evaluation.

2.3 Languages of the form CRPQ($\mathcal{S}$) and open questions

Languages that extend CRPQs with path comparisons come in different flavours. The simplest such language is the class CRPQ(REG), also known as *extended* CRPQs [10]. Queries in this class use regular relations to compare paths, and were shown to have its data complexity in NLogSpace, matching that of standard query languages previously discussed.

Theorem 2.3.1 ([10]).
The problem EvalCRPQ (REG) *is in* PSpace, *and for each query* $\phi \in$ CRPQ($\mathcal{S}$), *the problem* EvalCRPQ (REG, ϕ) *is in* NLogSpace.

Regular relations do not include, however, the relations such as Subword, Suffix or Subsequence, introduced in Example 2.1.2. In order to overcome this limitation, languages extending CRPQ(REG) with rational features have been introduced. The most direct solution would be extending the previous language with all rational relations, which yields the class CRPQ(RAT). This however, has an immense cost on query evaluation, which becomes undecidable.

Proposition 2.3.2 (Folklore).
EvalCRPQ (RAT) is undecidable.

This situation could perhaps be avoided, if instead of using arbitrary rational relations, CRPQ(REG) was extended with just an extra binary rational relation of practical interest, such as $\preceq_{\text{suff}}$, $\preceq_{\text{sw}}$ and $\preceq_{\text{ss}}$, yielding, for instance, the language CRPQ(REG$\cup \preceq_{\text{suff}}$). However, for $\preceq_{\text{suff}}$ and $\preceq_{\text{sw}}$ the evaluation problem for the resulting language is still undecidable, while for $\preceq_{\text{ss}}$ it is decidable, but with prohibitive complexity.

Theorem 2.3.3 ([9]).

- *There are queries $\phi \in$ CRPQ(REG $\cup \{\preceq_{suff}\}$) and $\phi' \in$ CRPQ(REG $\cup \{\preceq_{sw}\}$), such that* EvalCRPQ (REG $\cup \{\preceq_{suff}\}, \phi$) *and* EvalCRPQ (REG $\cup \{\preceq_{sw}\}, \phi'$) *are undecidable.*
- *There is a query $\phi \in$ CRPQ($\preceq_{ss}$) such that* EvalCRPQ (REG $\cup \preceq_{ss}, \phi$) *is decidable in non-elementary complexity.*

A possible approach to find better complexity bounds is to further restrict the language, by completely disallowing regular relations. This yields the classes CRPQ($\preceq_{\text{ss}}$), CRPQ($\preceq_{\text{sw}}$) and CRPQ($\preceq_{\text{suff}}$). What is known about the complexity of evaluating these languages is the following :

Theorem 2.3.4 ([9]). *The following is true for $S \in \{\preceq_{ss}, \preceq_{suff}\}$:*

- *The problem* EvalCRPQ (S) *is in* NExpTime.
- *For each $\phi \in$ CRPQ(S), the problem* EvalCRPQ (S, ϕ) *is in* NP.

The table in Figure 6.1 summarizes what was previously known about CRPQ(S) query evaluation.

Query	$S = \preceq_{\text{ss}}$	$S = \preceq_{\text{suff}}$	$S = \preceq_{\text{sw}}$	$S \in$ RAT
CRPQ(REG $\cup$ S)	decidable, NEC	undecidable	undecidable	undecidable
CRPQ(S)	NP	NP	?	undecidable
CRPQ(S)	NExpTime	NExpTime	?	undecidable

Figure 2.3: Upper bounds for data and combined complexity, respectively, of CRPQs comparing paths with Subword, Suffix and Subsequence relations. NEC stands for non-elementary complexity.

The questions studied in this thesis are the following:

1. Is NP the optimal upper bound for the complexity of evaluating CRPQ($\preceq_{\text{ss}}$) and CRPQ($\preceq_{\text{suff}}$) queries?

   The known upper bounds for these evaluation problem were achieved with techniques based on NFA-cutting properties allowing to show the existence of polynomially or exponentially sized membership witnesses. Other tools could be incorporated to construct better evaluation algorithms, or instead, look for problems which could be described by means of these relations in order to show the existence of queries NP-hard to evaluate.

2. Is the evaluation of CRPQ($\preceq_{\text{sw}}$) queries decidable? and if so, what is its data complexity?

   The techniques previously used to study this question, which worked for the preceding two query languages, were unfruitful when working with $\preceq_{\text{sw}}$ to find either an upper or lower complexity bound. It would then seem necessary to look elsewhere for tools to study this question.

Chapter 3

A new link between CRPQ($\mathcal{S}$) evaluation and word equations

In this chapter the combined complexity of evaluating CRPQ($\preceq_{\text{sw}}$) and CRPQ($\preceq_{\text{suff}}$) is shown to be PSPACE-complete, by establishing an unprecedented link between the evaluation of these query languages and the problem of solving *word equations with regular constraints.*

First, the idea behind the reduction is suggested for the languages CRPQ($\preceq_{\text{sw}}$) and CRPQ($\preceq_{\text{suff}}$), which is then generalized to the class of CRPQs comparing paths with relations *expressible* by word equations. In order to do so, *word equations with regular constraints* are introduced, followed by the class EQ of relations over Σ^* *expressible* by word equations. Finally, the reduction for the language CRPQ(EQ) is given, along with the consequences on the complexity of evaluating its queries.

3.1 Reducing to word equations

The first step when working with CRPQ($\preceq_{\text{sw}}$) or CRPQ($\preceq_{\text{suff}}$) is an intermediary problem which holds the essence of evaluating their queries. Before giving its definition, consider the following example to introduce the idea.

Let $Q \in$ CRPQ($\preceq_{\text{sw}}$) be a binary query given by

$$Q(x, y) \;:\; \left(x \xrightarrow{\chi\,:\,\Sigma^*\cdot a} y\right) \;\wedge\; \left(y \xrightarrow{\chi'\,:\,\Sigma^*\cdot b} x\right) \;\wedge\; (\chi \preceq_{\text{sw}} \chi')$$

with $\{a, b\} \subset \Sigma$. Let $G(V, E)$ be a graph-db with $E \subseteq V \times \Sigma \times V$, and $(u, v) \in V \times V$ a pair of its vertices. According to the definition, $\langle G, (u, v), \phi\rangle \in$ EVALCRPQ ($\preceq_{\text{sw}}$) if and only if there is in G a path ρ between u and v whose final label is a, and a path ρ' between v and u whose final label is b, such that $\lambda(\rho) \preceq_{\text{sw}} \lambda(\rho')$.

Another way to see this is to interpret the graph-db G as the transition function of an NFA over Σ with states V, without specified starting and accepting states. Formally, this transition function is $\delta \;:\; V \times \Sigma \;\to\; \mathcal{P}(V)$ such that

$$\forall x, y \in V,\;\; a \in \Sigma,\;\; y \in \delta(x, a) \;\Leftrightarrow\; (x, a, y) \in E$$

For instance, $N(V, \Sigma, \delta, u, \{v\})$ is an automaton recognizing a string $w \in \Sigma^*$ if and only if a path exists between u and v in G whose label is precisely w. If in addition to this, A is the NFA recognizing $\Sigma^* \cdot a$, then a string $w \in \Sigma^*$ is recognized by the product automaton $N \times A$ if and only if a path whose final label is a exists between u and v in G. Similarly, one can define the product automaton $N' \times A'$ recognizing the labels of paths existing in G between v and u ending with a b label.

If L and L' are the regular languages defined by $N \times A$ and $N' \times A'$, respectively, the evaluation problem then translates into checking if a pair $(w, w') \in L \times L'$ exist, such that $w \preceq_{\text{sw}} w'$ – *i.e.* $(w, w') \in (L \times L') \bigcap \preceq_{\text{sw}}$. This is a particular instance of a broader problem known as the *generalized intersection problem*, and the extent to which this reduction holds is far more general.

Formally, the *generalized intersection problem*, $\textsc{GenInt}(\text{REC}^*, \mathcal{S})$, for a class $\mathcal{S}$ of binary relations over Σ^*, receives as input a tuple $\langle I, R, S\rangle$ of an index set $I \subseteq [m] \times [m]$, a recognizable m-ary relation $R = L_1 \times ... \times L_m \in \text{REC}^*$ (where $\text{REC}^* \subset \text{REC}$ is the subclass that does not consider unions of products of regular languages), and specified relations $S = \{S_{i,j}\}_{(i,j)\in I}$ such that for each $(i, j) \in I$, $S_{i,j}$ is in $\mathcal{S}$. The problem asks whether strings $w_1, ..., w_m \in \Sigma^*$ exist such that for each $i \in [m]$, $w_i \in L_i$, and for each $(i, j) \in I$, $(w_i, w_j) \in S_{i,j}$. The set of tuples in R satisfying these conditions will be henceforth denoted as $R \bigcap_I S$. As in the case of CRPQs, when a single relation is to be studied $\mathcal{S}$ can be a singleton. When only the recognizable part of the input is considered, and the rest is fixed, this is indicated by subscripts, such as $\textsc{GenInt}_{I,\preceq_{\text{sw}}}(\text{REC}^*)$. The following then holds:

Theorem 3.1.1 ([9]).

- *The problem* $\textsc{EvalCRPQ}(\mathcal{S})$ *reduces in polynomial space to* $\textsc{GenInt}(\text{REC}^*, \mathcal{S})$.
- *For each query* $\phi \in \text{CRPQ}(\mathcal{S})$ *comparing paths specified by* $I \subseteq [m] \times [m]$ *, the problem* $\textsc{EvalCRPQ}(\mathcal{S}, \phi)$ *reduces in logarithmic space to* $\textsc{GenInt}_{I,\mathcal{S}_0}(\text{REC}^*)$.

With this result, upper bounds for the complexity of evaluating $\text{CRPQ}(\preceq_{\text{sw}})$ and $\text{CRPQ}(\preceq_{\text{suff}})$ queries can be found by constructing algorithms deciding $\textsc{GenInt}(\text{REC}^*, \preceq_{\text{sw}})$ and $\textsc{GenInt}(\text{REC}^*, \preceq_{\text{suff}})$ respectively. For these relations, these intersection problems can be reduced to an equation on strings ranging in regular subsets of Σ^*. The following example motivates this idea:

Example 3.1.2. Suppose now an instance is given for $\textsc{GenInt}(\text{REC}^*, \preceq_{\text{suff}})$: a recognizable relation $R = L_1 \times L_2 \times L_3 \times L_4$ over Σ^* and an index set $I = \{(1, 2)$, $(1, 3)$, $(2, 4)$, $(3, 4)\}$. The problem is to find out if strings w_1, w_2, w_3, w_4 exist in Σ^*, w_i belonging to L_i for each $i = 1, 2, 3, 4$, and such that $w_1 \preceq_{\text{suff}} w_2$, $w_1 \preceq_{\text{suff}} w_3$, $\preceq_{\text{suff}} w_4$, $w_3 \preceq_{\text{suff}} w_4$ (see figure 3.1).

$$\begin{tikzcd}
& w_2 \in L_2 \arrow[dr, "\preceq_{\text{suff}}"] & \\
w_1 \in L_1 \arrow[ur, "\preceq_{\text{suff}}"] \arrow[dr, "\preceq_{\text{suff}}"'] & & w_4 \in L_4 \\
& w_3 \in L_3 \arrow[ur, "\preceq_{\text{suff}}"'] &
\end{tikzcd}$$

Figure 3.1: A generalized intersection problem scheme.

According to this, w_1 is not bounded to have as suffix any string. It must be a suffix of w_2 and w_3, and these must be both suffixes of w_4. This means that prefixes $p_{1,2}, p_{1,3}, p_{2,4}, p_{3,4} \in \Sigma^*$ must exist such that

$$\begin{tikzcd}
& w_2 = p_{1,2}w_1 \arrow[dr] & \\
w_1 \arrow[ur] \arrow[dr] & & w_4 = \begin{cases} p_{2,4}w_2 \\ p_{3,4}w_3 \end{cases} \\
& w_3 = p_{1,3}w_3 \arrow[ur] &
\end{tikzcd}$$

Additionally, strings w_i are bounded to regular sets L_i for each $i = 1, 2, 3, 4$. The entire problem can thus be rewritten as the following *equation* on strings:

$$\begin{aligned}
w_2 &= p_{1,2}w_1 \\
w_3 &= p_{1,3}w_1 \\
w_4 &= p_{2,4}w_2 \\
w_4 &= p_{3,4}w_3
\end{aligned}$$

$$\text{s.t.} \quad \begin{aligned} \forall i = 1,2,3,4 &\quad w_i \in L_i \\ \forall (i,j) \in I &\quad p_{i,j} \in \Sigma^* \end{aligned}$$

The response to the original problem is affirmative if and only if the preceding equation has a solution.

This later problem corresponds to solve what is known as a *word equation with regular constraints*. The fact that the reduction for this generalized intersection problem instance holds is precisely because suffix is a relation *expressible* by a word equation, in the sense that $\preceq_{\text{suff}} = \{(x,y) \in \Sigma^* \times \Sigma^* \ : \ \exists p \in \Sigma^*, \ y = px\}$.

As will be shown in the following sections, any instance from GenInt $(\text{REC}^*, \mathcal{S})$ can be reduced to such constraint equations on strings if $\mathcal{S}$ is a class of binary relations over Σ^* expressible by word equations. In order to achieve this, these concepts are formally introduced.

3.2 Word equations with regular constraints

Word equations is the problem of assigning strings to variables in order to satisfy a given pattern. For instance, in an alphabet $\{a, b\}$, the equation

$$XaaXb = aabXY$$

asks to find strings $X,\ Y$ on $\{a, b\}^*$ making the left and right sides of it equal. They could be $X = aab,\ Y = b$. If so, both sides of the equation correspond to the string $aabaaaabb$.

Formally, a *word equation* e consists of two strings $L,\ R \in (\Gamma\ \cup\ \Sigma)^*$, where Γ is a set of variables Γ, and Σ is a set of constants. This is typically written as $\{e\ :\ L = R\}$. The size of $|e|$ is its denotational length, $|L \cdot R|$. A solution, or *unifier*, for the equation is a morphism $\phi : ((\Gamma\ \cup\ \Sigma)^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$ such that $\phi|_\Sigma = id$ and for which $\phi(L) = \phi(R)$. Constraints can be added to the equation to specify subsets of Σ^* where the strings range in. This can be done by means of regular languages $\{L_x\}_{x \in \Gamma}$, asking additionally that for each $x \in \Gamma$, $\phi(x) \in L_x$. These are *word equations with regular constraints.* In this case, the equation may be assumed to be *constant free*, that is, only symbols from Γ appear in L and R, since a subword of LR over Σ can be replaced by variables restricted to the regular language composed of that single subword.

The study of systems of word equations reduces to single word equations, since an infinite system of word equations is equivalent to one of its finite subsystems [3], and every finite system of word equations can be encoded in a single one [31]. One of the most celebrated results in this area is due to Makanin [42], showing decidability of finite systems of word equations. The complexity of his algorithm has currently been shown to be in ExpSpace [52] and non primitive recursive [36]. Another method, using data compression, was later proposed by Plandowski [46], showing that the problem is decidable in PSpace, which is the best currently known bound. Gutiérrez et al. finally developed a non-trivial extension from this, showing that solving word equations with regular constraints is PSpace-complete.

3.3 Relations expressible by word equations - EQ

Word equations can serve as means to express relations on strings. For instance, the equation $\{XY = YX\}$ describes the set of pairs of strings $(X, Y) \in \Sigma^* \times \Sigma^*$ which commute. This use of word equations has been applied by numerous authors [41, 18, 26]. In [35] a general study of the expressive power of word-equations was first given, trying to unify and systematize this topic. It also gives means for disproving relations from being expressed by word equations, by providing pumping-like properties satisfied by them.

An m-ary relation $R \subseteq (\Sigma^*)^m$ is *expressible* by a word equation, if there exists an equation e with $t \geqslant m$ variables over Σ such that R coincides with the projection of the t-tuple solutions of e on a set of m fixed components. The class of finite-arity relations expressible by a word equation will be denoted EQ. When the arity of such a relation is fixed, it is denoted with a subscript (*e.g.* the class of binary relations expressible by word equations is EQ_2).

Some relations with widespread applications on database problems are expressible by word equations. This is the case for the Suffix and Subword relations :

1. $\preceq_{\text{suff}} = \{(X, Y) \in \Sigma^* \times \Sigma^* \ : \ \exists p \in \Sigma^*, \ Y = pX\}$
2. $\preceq_{\text{sw}} = \{(X, Y) \in \Sigma^* \times \Sigma^* \ : \ \exists p, s \in \Sigma^*, \ Y = pXs\}$

Other relations have been negatively proven to be expressible by word equations. In [32], the subsequence relation, was shown to be one of such. They also show that the class of power-free strings (*e.g*, cube-free words on binary alphabets) is neither in EQ. The equal-length relation was proven in [35] to belong outside of EQ as well. A work that completely characterizes the complexity of languages expressed by equations on two variables is [33], and with bounded number of variables is [34].

It is worth noting that RAT $\nsubseteq$ EQ and EQ $\nsubseteq$ RAT, since $\preceq_{\text{ss}}$ is rational yet not expressible by word equations, and EQ contains relations not able to be recognized by memory-limited sequential devices as automata, as $\{(x, y) \in \Sigma^* \ : \ \exists z, \ y = xzx\}$. Also, REG $\nsubseteq$ EQ, since REG contains the equal-length relation. Nevertheless, Plandowski's algorithm ensure that the recognition of EQ relations has PSpace complexity, that is, the problem of deciding, for a given relation $\mathcal{R} \in$ EQ and a string tuple $\bar{w} \in (\Sigma^*)^{ar(R)}$, if $\bar{w} \in \mathcal{R}$, is solvable in polynomial space.

3.4 EvalCRPQ (EQ) is PSpace-complete

Queries from CRPQ(EQ) use binary relations expressible by word equations in order to compare paths joining vertices in a graph-db. Now that the formal requirements have been met, the reduction is given from EvalCRPQ (EQ) to the problem of solving word equations with regular constraints. The result follows from translating general intersection problem instances into word equations.

Proposition 3.4.1. GenInt (REC*, EQ) reduces in logarithmic space to word equations with regular constraints.

The idea behind this proof was already presented in Example 3.1.2. The complete proof is given now:

Proof for Proposition 3.4.1. Consider an input instance for GenInt (REC*, EQ) given by $\langle L, I, S\rangle$, with $L \in$ REC*, $I \subseteq [m] \times [m]$ and $S = \{S_{i,j}\}_{(i,j)\in I}$ such that for each $(i, j) \in I$, $S_{i,j}$ is a binary relation in EQ. Assume L given by $L = L_1 \times \cdots \times L_m$, for regular languages $L_i \subseteq \Sigma^*$. The problem asks for the following : does $w_1, ..., w_m \in \Sigma^*$ exist, such that for each $i \in [m]$, $w_i \in L_i$, and for each $(i, j) \in I$, $(w_i, w_j) \in S_{i,j}$?

For $(i, j) \in I$, let $e^0_{i,j} \ : \ L^0_{i,j} = R^0_{i,j}$ be the word equation describing $S_{i,j}$. It will be assumed that all these equation are written in the same alphabet. Variables $X = \{X_1, \ X_2\}$ are assumed to be those variables projected onto the relations and $Y = \{Y_1, ..., Y_t\}$ those existentially quantified, with t the maximum number of quantified variables in the set

$\{S_{i,j}\}_{(i,j)\in I}$. From this, $L^0_{i,j}$ and $R^0_{i,j}$ are strings from $(X \cup Y \cup \Sigma)^*$. The relations are then described as $S_{i,j} = \{(X_1, X_2) \in \Sigma^* \ : \ \exists Y_1, ..., Y_t \in \Sigma^*, \ L^0_{i,j} = R^0_{i,j}\}$.

A new set of variables is constructed. For each $i \in [m]$ let Z_i be a variable interpreting w_i. For each index $(i,j) \in I$, variables $Y^{i,j}_1, ..., Y^{i,j}_t$ are defined. An unrestricted equation $e_{i,j} \ : \ L_{i,j} = R_{i,j}$ is given for each index $(i,j) \in I$: in order to obtain $e_{i,j}$ from $e^0_{i,j}$, X_1 is replaced by Z_i, X_2 is replaced by Z_j, and for each $l \in [t]$, Y_l is replaced by $Y^{i,j}_l$. The system of word equations $e_{L,I,S}$ is thus given by

$$e_{L,I,S} \begin{cases} & \forall (i,j) \in I \quad L_{i,j} = R_{i,j} \\ \text{s.t} & \forall i \in [m] \qquad Z_i \in L_i \\ & \forall (i,j) \in I, l \in [t] \quad Y^{i,j}_l \in \Sigma^* \end{cases}$$

Claim 3.4.2. There is a solution for $e_{L,I,S}$ if and only if $L \bigcap_I S \neq \varnothing$.

For the *if* part, let $(w_1, ..., w_m) \in L \bigcap_I S$. Since for all $(i,j) \in I$, $(w_i, w_j) \in S_{i,j}$, by definition of $S_{i,j}$, strings $y^{i,j}_1, ..., y^{i,j}_t \in \Sigma^*$ exist such that when replacing $X_1 \to w_i$, $X_2 \to w_j$, and each $Y_l \to y^{i,j}_l$, then the strings obtained $r^{i,j}, l^{i,j}$ are equal. The morphism $\phi : ((Z \cup \Gamma \cup \Sigma)^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$ assigning with each Z_i the string w_i, and with each $Y^{i,j}_l$ the string $y^{i,j}_l$, leaving the symbols of Σ invariant, is clearly a solution for $e_{L,I,S}$, since for each $i \in [m]$, $w_i \in L_i$ and the way the equation is defined. The converse part is analogous.

The variables and equations in $e_{L,I,S}$ depend exclusively on I and S. The number of variables is $m + \sum_{(i,j)\in I} | \langle S_{i,j} \rangle |$, and there are $|I|$ equations of size $| \langle S_{i,j} \rangle |$, where $| \langle S_{i,j} \rangle | = |L^0_{i,j} R^0_{i,j}|$ is the size of its description. There are m restrictions $\{\bigcup_{i=1}^k L^k_i\}_{i\in[m]}$ and thus each of $\mathcal{O}(|L|)$ size, the others being Σ^* and thus of constant size each. It follows that $|e_{L,I,S}| = \mathcal{O}(|I||S||L|)$. Also, it is important to notice that the equations are constructed by consecutively visiting each edge pair $(i,j) \in I$, and this is done independently. Consequently, the procedure and can be carried out in deterministic logarithmic space. This concludes the proof. ☐

Observation 3.4.3. It is important to notice that when the index set I and the relations in S are fixed, the equations and variables are also fixed, and the input part of the reduction is only the set of regular constraints.

As a corollary of this and Proposition 3.4.1, the following result is obtained:

Theorem 3.4.4. *The following is true about* EVALCRPQ (EQ)*:*

- EVALCRPQ (EQ) *many-one reduces in polynomial space to solving word equations with regular constraints.*
- EVALCRPQ (EQ) *is* PSPACE*-complete.*
- *For each query $\phi \in$ CRPQ(EQ), there is an unrestricted equation e_ϕ, for which the complexity of* EVALCRPQ (EQ, ϕ) *is that of solving e_ϕ with variable regular constraints.*

Proof. The first item follows from EvalCRPQ (EQ) reducing to GenInt (REC*, EQ) in polynomial space, since this later problem reduces to word equations with regular constraints in logarithmic space.

The PSpace upper bound in item 2 follows from Plandowski's algorithm for solving word equations with regular constraints in polynomial space. For a lower bound, the non-empty intersection problem for regular languages, which is PSpace-complete, was reduced to the combined complexity of EvalCRPQ (EQ) in Example 2.2.2 using the $\preceq_{\mathrm{sw}}$ relation, which is in EQ. The same result holds when changing $\preceq_{\mathrm{sw}}$ by $\preceq_{\mathrm{suff}}$.

The final item follows directly from the proof of Proposition 3.4.1 and Observation 3.4.3. □

As a consequence of this, the evaluation of queries in CRPQ($\preceq_{\mathrm{sw}}$) is decidable in PSpace, and the combined complexity of EvalCRPQ ($\preceq_{\mathrm{suff}}$) can be lowered from NExpTime to PSpace:

Corollary 3.4.5.
The combined complexity of EvalCRPQ ($\preceq_{\mathrm{sw}}$) and EvalCRPQ ($\preceq_{\mathrm{suff}}$) is PSpace-complete.

It is worth noticing that the data complexity of evaluation of queries in CRPQ(EQ) reduces to the problem of solving a fixed word equation on variable regular constraints. This problem is, to the best of the author's knowledge after personal communications with Claudio Gutierrez and Volker Diekert, an open question in the literature. This question has its own interest, but in this context, it will also serve to study in depth the data complexity of EvalCRPQ ($\preceq_{\mathrm{suff}}$), for which the obtained equations share features that allow to decide satisfiability with variable regular constraints in NLogSpace, as will be shown in the following chapter.

Chapter 4

Evaluating CRPQ($\preceq_{\text{suff}}$) queries

In the preceding chapter it was shown that the data complexity of EvalCRPQ ($\mathcal{S}$) reduces to solving fixed word equations with variable regular constraints when $\mathcal{S}$ is a subset of EQ. In this chapter the equations obtained in the case of EvalCRPQ ($\preceq_{\text{suff}}$) are studied in depth, recognizing shared patterns that allow to solve them in NLogSpace. This result actually extends to any fixed word equation with variable regular constraints admitting a finite number of *minimal* solutions.

4.1 Working with EvalCRPQ ($\preceq_{\text{suff}}$) equations

An equation obtained when working with $\preceq_{\text{suff}}$ is studied in this section to motivate the algorithms further constructed in this chapter. In Example 3.1.2, the following equation reducing EvalCRPQ ($\preceq_{\text{suff}}$) was shown

$$\begin{aligned} w_2 &= p_{1,2}w_1 \\ w_3 &= p_{1,3}w_1 \\ w_4 &= p_{2,4}w_2 \\ w_4 &= p_{3,4}w_3 \end{aligned}$$

$$\begin{aligned} \text{s.t.} \quad \forall i = 1,2,3,4 \quad & w_i \in L_i \\ \forall (i,j) \in I \quad & p_{i,j} \in \Sigma^* \end{aligned}$$

The first thing to be noted, is that through direct substitution of variables, this equation is equivalent to the following

$$p_{2,4}p_{1,2}w_1 = p_{3,4}p_{1,3}w_1 \quad \wedge \quad \begin{aligned} w_2 &= p_{1,2}w_1 \\ w_3 &= p_{1,3}w_1 \\ w_4 &= p_{2,4}p_{1,2}w_1 \end{aligned}$$

$$\begin{aligned} \text{s.t.} \quad \forall i = 1,2,3,4 \quad & w_i \in L_i \\ \forall (i,j) \in I \quad & p_{i,j} \in \Sigma^* \end{aligned}$$

The new system has a non-trivial equation, $\{p_{2,4}p_{1,2}w_1 = p_{3,4}p_{1,3}w_1\}$, and a set of trivial equations whose solutions depend on the preceding one, $\{w_2 = p_{1,2}w_1,\ w_3 = p_{1,3}w_1,\ w_4 = p_{2,4}p_{1,2}w_1\}$. Actually, these are just variable assignations, but they need to be kept in the system for not losing information about the regular constraints.

Let $\Gamma = \{w_i\}_{i\in[4]} \cup \{p_{i,j}\}_{(i,j)\in I}$ be the set of variables for this equation. Recall that a solution for it is a morphism $\phi : (\Gamma^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$ satisfying the patterns in the equation and the regular constraints. An approach to find such a solution is to first find a solution $\phi_1 : (\Gamma^*, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ unifying the unrestricted equation on a intermediary variable alphabet Ψ, and then find the appropriate strings in Σ^*, to associate with each $x \in \Psi$, matching the constraints. The later is a second morphism, $\phi_2 : (\Psi^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$, such that $\phi_2 \circ \phi_1$ is a solution for the constrained equation. For instance, a solution ϕ_1 for the pattern equation $\{p_{2,4}p_{1,2}w_1 = p_{3,4}p_{1,3}w_1\}$ with variables from $\Psi = \{X, Y, Z\}$ can be constructed as follows:

$$\begin{array}{l} \phi_1(w_1) = X \\ \phi_1(p_{1,2}) = YZ \\ \phi_1(p_{1,3}) = BYZ \\ \phi_1(p_{2,4}) = YAB \\ \phi_1(p_{3,4}) = YA \end{array} \quad \Rightarrow \quad \left| \begin{array}{l} p_{2,4} \cdot p_{1,2} \cdot w_1 \\ p_{3,4} \cdot p_{1,3} \cdot w_1 \end{array} \right| \xrightarrow{\phi_1} \left| \begin{array}{l} YAB \cdot YZ \cdot X \\ YA \cdot BYZ \cdot X \end{array} \right|$$

In this setting ϕ_2 should be a morphism such that $\phi_2(X) \in L_1$, $\phi_2(YZX) \in L_2$, $\phi_2(BYZX) \in L_3$ and $\phi_2(YABYZX) \in L_4$. This observation actually extends to any word equation with regular constraints :

Observation 4.1.1. Let e be an equation on variables Γ, and $\{L_x\}_{x\in\Gamma}$ a set of regular languages over Σ. A solution for E with regular constraints $\{L_x\}_{x\in\Gamma}$ can be decomposed in two morphisms $\phi_1 : (\Gamma^*, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ and $\phi_2 : (\Psi^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$ such that $\phi = \phi_2 \circ \phi_1$. This is because regardless of which strings on Σ solve the constraints, they still have to match the patterns present in the unrestricted equation. Thus these patterns, *i.e.* ϕ_1, can be found within the strings. Additionally, ϕ_1 can be supposed to be *minimal*: if $\phi_1 = \alpha \circ \phi_1'$, for a non trivial morphism $\alpha : (\Psi^*, \cdot, \varepsilon) \to (\tilde{\Psi}^*, \cdot, \varepsilon)$, this yields $\phi = \phi_2 \circ \alpha \circ \phi_1$, remaining unchanged by renaming the composition $\tilde{\phi_2} \leftarrow \phi_2 \circ \alpha : (\Psi^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$, producing $\phi = \tilde{\phi_2} \circ \phi_1'$. A *minimal* solution for an unrestricted equation is a solution such that no decomposition $\phi_1 = \alpha \circ \phi_1'$ exists other than those obtained with trivial morphisms α such as the identity or variable renaming.

In the previous example, ϕ_1 is not minimal, since it can be obtained from morphisms ϕ_0 and α defined by

$$\begin{array}{l} \phi_0(w_1) = X \\ \phi_0(p_{1,2}) = C \\ \phi_0(p_{1,3}) = BC \\ \phi_0(p_{2,4}) = AB \\ \phi_0(p_{3,4}) = A \end{array} \quad \Rightarrow \quad \left| \begin{array}{l} p_{2,4} \cdot p_{1,2} \cdot w_1 \\ p_{3,4} \cdot p_{1,3} \cdot w_1 \end{array} \right| \xrightarrow{\phi_0} \left| \begin{array}{l} AB \cdot C \cdot X \\ A \cdot BC \cdot X \end{array} \right|$$

$$\begin{array}{l} \alpha(A) = YA \\ \alpha(B) = B \\ \alpha(C) = YZ \\ \alpha(X) = X \end{array} \quad \Rightarrow \quad \left| \begin{array}{l} AB \cdot C \cdot X \\ A \cdot BC \cdot X \end{array} \right| \xrightarrow{\alpha} \left| \begin{array}{l} YAB \cdot YZ \cdot X \\ YA \cdot BYZ \cdot X \end{array} \right|$$

In other words, the approach to find solutions for the constrained equation is to first construct a minimal pattern solving the unrestricted word equation, and then decide if strings exist matching this pattern and the regular constraints. This approach may be rather naïve since unrestricted word equations may have infinite and arbitrarily large minimal solutions [1]. However, if the set of minimal solutions for an equation is finite, the solutions for it with variable regular constraints can be found by iterating over all the minimal patterns while deciding if strings solve the constrained equation according to them through NFA-reachability analysis.

As a consequence, by affirmatively answering the next questions the result can be concluded:

- Any fixed equation, with a finite number of minimal solutions, can be solved with variable regular constraints in NLogSpace?
- Any equation obtained from EvalCRPQ $(\preceq_{\text{suff}})$ admits a finite number of minimal solutions?

These are answered in the following sections.

4.2 Solving constrained equations with finite minimal solutions

As it was suggested in the previous section, after constructing a string pattern satisfying a fixed word equation, the regular constraints could be met by performing a NFA-reachability analysis. Consider the following example to clarify this idea:

Example 4.2.1. Suppose that an equation on variables $\{X, Y, Z\}$ has been given a solution mapping $X \to ABC$, $Y \to B$ and $Z \to AC$. According to the previous observations, to find a solution of the equation with regular constraints $\{L_x, L_y, L_z\}$ satisfying this pattern the next step would be to associate strings $w_A, w_B, w_C \in \Sigma^*$ with A, B and C such that $w_A w_B w_C \in L_x$, $w_B \in L_y$ and $w_A w_C \in L_z$. If N_x, N_y, N_z are the automata recognizing these regular languages, this means they are needed to be somehow *synchronized* in order to prove, for instance, that w_B is a string in L_y while being a subword of a string in L_x.

One way to do so is to assign *reachability tasks* for each variable A, B, C. Each of the strings to be associated with these variables is to perform an specific reachability task in every automaton it is bounded to be recognized by (see Figure 4.1). More precisely, these strings exist if there are states $q_1^x, q_2^x \in Q_x$, $f_x \in F_x$, $f_y \in F_y$, $q^z \in Q_z$ and $f_z \in F_z$ such that the following NFA-reachability problems are satisfiable

$$\begin{aligned} w_A &: \quad (s_x, s_z) \xrightarrow{N_x \times N_z} (q_1^x, q^z) \\ w_B &: \quad (q_1^x, s_y) \xrightarrow{N_x \times N_y} (q_2^x, f_y) \\ w_C &: \quad (q_2^x, q^z) \xrightarrow{N_x \times N_z} (f_x, f_z) \end{aligned}$$

where $N \times N'$ is the usual product automaton over Σ. .

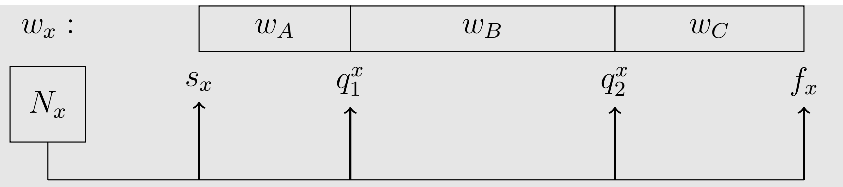

Figure 4.1: States of Automaton N_x when running on input $w_A w_B w_C$.

If the set of minimal solutions for an equations is finite, the procedure can be carried out in non-deterministic logarithmic space by checking all the patterns solving the unrestricted equation and using the standard on-the-fly NFA-reachability algorithm to see if strings exist satisfying the given pattern:

Proposition 4.2.2. Satisfiability of fixed word equations with variable regular constraints is in NLogSpace for equations with finite minimal solutions.

Proof. Let e be an equation whose minimal solutions are finite, and $\langle L\rangle = \{L_x\}_{x\in\Gamma}$ a set of regular constraints. The algorithm picks one by one the minimal solutions $\psi : (\Gamma^*, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ for e from the finite possibilities. Their quantity, range size and images are constants to the algorithm since they depend exclusively on e.

For each minimal pattern solution ψ, a solution exists for the equation E with regular constraints $\langle L\rangle = \{L_x\}_{x\in\Gamma}$ if a mapping $\phi_0 : \Psi \to \Sigma^*$ exists, whose canonical morphism extension ϕ to Ψ^* satisfies $\forall x \in \Gamma$, $\phi(\psi(x)) \in L_x$. For $x \in \Gamma$, let $N_x(Q_x, \Sigma, \delta_x, s_x, F_x)$ be the automaton over Σ recognizing L_x and let $n_x = |\psi(x)|$.

The algorithm non-deterministically chooses $q_0^x, \ldots, q_{n_x}^x \in Q_x$, for each $x \in \Gamma$, such that $q_0^x = s_x$ and $q_{n_x}^x \in F_x$ (see Figure 4.1).

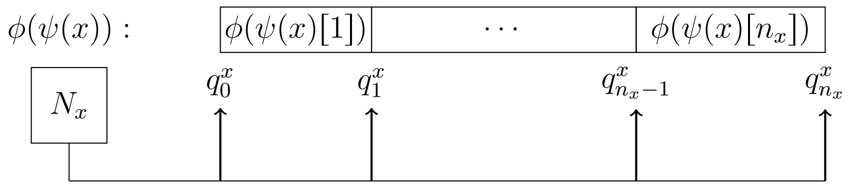

Figure 4.1: Guessing NFA states for each regular constrain.

For coding each one of these states, $\log(|Q_x|)$ bits are needed. Hence, the whole collection needs $\sum_{x\in\Gamma}(n_x+1)\log(|Q_x|)$ bits. Since $n_x = |\psi(x)|$ is considered fixed, this takes $\mathcal{O}(\max_x \log(|Q_x|)) = \mathcal{O}(|\langle L\rangle|)$ bits.

Next, reachability tasks are defined for each symbol $a \in \Psi$. For this, if $a \in \Psi$, let $\sigma(a) = \{(x,i) : \psi(x)[i] = a\} \subseteq \Gamma \times [\max_x n_x]$ be the set of pairs (x,i) signalling variables in $x \in \Gamma$ whose mapping $\psi(x)$ contains symbol a at the signalled position i. The product automaton $Q_{\sigma(a)}$ over Σ where a must perform its reachability task is defined as

$$Q_{\sigma(a)} = \bigtimes_{(x,i)\in\sigma(a)} N_x$$

This definition allows a to perform multiple tasks in one single automaton by allowing it to appear repeatedly in $Q_{\sigma(a)}$. Formally, a *reachability task* for $a \in \Psi$, is a set of two $\sigma(a)$-NFA states $\{\bar{q}_0(a), \bar{q}_f(a)\} \subseteq Q_{\sigma(a)}$. These tuples are *filled* in way that if $\bar{q}_0(a) = \{q_{x,i}\}_{(x,i)\in\sigma(a)}$ and $\bar{q}_f(a) = \{\tilde{q}_{x,i}\}_{(x,i)\in\sigma(a)}$ then $q_{x,i} = q^x_{i-1}$ and $\tilde{q}_{x,i} = q^x_i$ (see Figure 4.2). It is to be noted that since trivially $\sigma(a) \cap \sigma(b) = \varnothing$ for $a, b \in \Psi$, there is no ambiguity in these naming of variables.

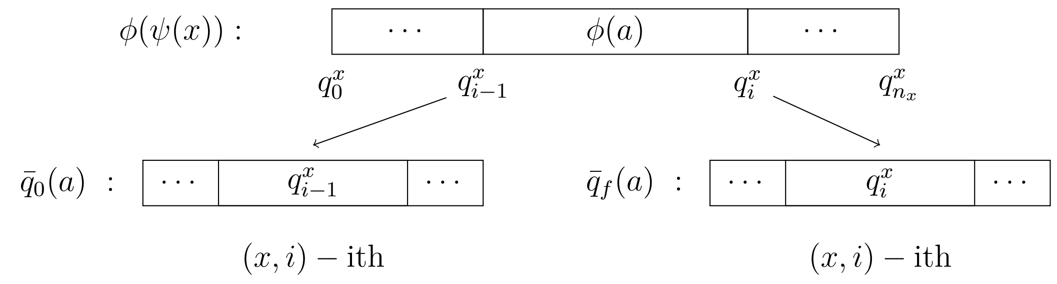

Figure 4.2: Filling the reachability tasks

The algorithm then proceeds to verify if for each $a \in \Psi$ state $\bar{q}_f(a)$ is reachable from state $\bar{q}_i(a)$ in the automaton $Q_{\sigma(a)}$. This can be done with the standard on-the-fly algorithm which requires non-deterministic space logarithmic in $|Q_{\sigma_a}| \leqslant |\sigma(a)| \max_x |Q_x| = \mathcal{O}(\langle L\rangle)$, since $|\sigma(a)|$ equals the number of total apparitions of a in $\{\psi(x)\}_{x\in\Gamma}$, and is thus constant. Also, $|\Psi|$ is of constant size, since it is the number of symbols unifying E. Hence, the entire procedure is carried out in non-deterministic logarithmic space. The algorithm accepts if it gives an affirmative answer for each one of these problems, rejecting otherwise.

Claim 4.2.3. The algorithm accepts if and only if there is a solution for E with regular constraints $\langle L\rangle$.

Suppose for the *only if* part, that all the reachability tasks hold. Let $\phi_0 : \Psi \to \Sigma^*$ be the mapping assigning for each $a \in \Psi$ a string $w_a \in \Sigma^*$ for which $\tilde{q}_{x,i} \in \delta_x(q_{x,i}, w_a)$, for all $(x, i) \in \sigma(a)$. By the way these state-tuples where constructed, this means $q^x_i \in \delta_x(q^x_{i-1}, w_a)$. Let $\phi : (\Psi^*, \cdot, \varepsilon) \to (\Sigma^*, \cdot, \varepsilon)$ to be the canonical morphism extension of ϕ_0. Then for each $x \in \Gamma$, $\phi(\psi(x)) \in L_x$. This is proven by induction in the following invariant: For each $i \in [|\psi(x)|]$, $q^x_i \in \delta_x\left(q^x_0, \phi(\psi(x)[1..i])\right)$.

1. For $i = 1$, this reduces to $q^x_1 \in \delta_x(q^x_0, \phi(a)) = \delta_x(q^x_0, w_a)$ where a is the first symbol of $\psi(x)$. Accordingly, $\bar{q}_0(a)_{x,1} = q^x_1$ and $\bar{q}_f(a)_{x,1} = q^x_0$. String w_a being that which fulfils a's reachability tasks, this is indeed true.

2. For $i > 1$, the i-th symbol of $\psi(x)$ is assumed to be a. Since ϕ is a morphism, then

$$\begin{aligned}
& q^x_i \in \delta_x\left(q^x_0, \phi(\psi(x)[1..i])\right) \\
\Leftrightarrow\quad & q^x_i \in \delta_x\left(q^x_0, \phi(\psi(x)[1..i-1]) \cdot \phi(a)\right) \\
\Leftrightarrow\quad & q^x_i \in \delta_x\left(q^x_0, \phi(\psi(x)[1..i-1]) \cdot w_a\right)
\end{aligned}$$

This, in turn, is true if and only if a state $q^* \in Q_x$ exists, such that $q^* \in \delta_x$ (q^x_0, $\phi(\psi(x)$ $[1..i-1]))$ and $q^x_i \in \delta_x(q^*, w_a)$. By induction $q^x_{i-1} \in \delta_x(q^x_0, \phi(\psi(x)\ [1..i-1]))$, and since w_a is the string fulfilling a's reachability tasks, with a the i-th symbol of $\psi(x)$, $q^x_i \in \delta_x(q^x_{i-1}, w_a)$. The property is hence true.

Consequently, $q^x_{n_x} \in \delta_x(q^x_0, \phi(\psi(x)))$. Since $q^x_0 = s_x$ and $q^x_{n_x} \in F_x$, it follows that $\phi(\psi(x)) \in L_x$. Since $x \in \Gamma$ was arbitrary, $\phi \circ \psi$ solves E with regular constraints $\langle L \rangle$.

Conversely, suppose it is true that a solution exists for E with regular constraints $\langle L \rangle$. By means of Observation 4.1.1, morphisms ϕ_1, ϕ_2 exists such that $\phi_1 : \Gamma^* \to \Psi^*$ minimally unifies E and $\phi_2 : \Psi^* \to \Sigma^*$ solves the constraints over $\phi_1(E)$. Thus, the algorithm picks $\psi \leftarrow \phi_1$ in some iteration. It is straightforward to see from this proof that if ϕ_2 solves the regular constraints according to the pattern ϕ_1, then for each $x \in \Psi$ states $q^x_0, ..., q^x_{n_x}$ must exist in Q_x sequentially reachable by means of the strings $\phi_2(\phi_1(x)[1])$,...,$\phi_2(\phi_1(x)[n_x])$ in automaton N_x. Consequently, there is a non-deterministic branch of the algorithm that successfully guesses these states and gives the thumbs-up for every reachability task. This concludes the proof. □

Remark 4.2.4. It is worth noting that the complexity obtained can only be achieved by considering the equation fixed, for solving word equations with regular constraints remain PSpace-hard on equations with a finite number of minimal solutions. Indeed, an instance of the regular language intersection problem $\langle R_1, ..., R_m \rangle$ is easily coded by the equation $\{x_1 = \cdots = x_m\}$ with constraints $x_i \in R_i$, which admits a finite number of minimal solutions since any variable appears only once.

4.3 Suffix-like equations

In this section a procedure is given for enumerating all the minimal solutions for the class of equations obtained when working with EvalCRPQ $(\preceq_{\text{suff}})$. These equations share a very rigid structure implying its minimal solution set to be always finite.

An equation system $\{E_i : e_{i,1} = \cdots = e_{i,l_i}\}_{i=1}^n \subseteq \Gamma^*$ is said to be *suffix-like*, if it satisfies the following properties:

(1) $\forall a \in \Gamma,\ i \in [l], j \in [l_i],\ |e_{i,j}|_a \leqslant 1$. That is to say, in each line of the system, each variable appears at most once.

(2) $\forall a \in \Gamma,\ i, i' \in [l], j \in [l_i], j' \in [l_{i'}]$, if $a \in \Gamma$ and $p, p', s, s' \in \Gamma^*$ exist such that $e_{i,j} = pas$ and $e_{i',j'} = p'as'$, then $s = s'$. In other words, whenever two lines share a symbol, their suffixes starting from that symbol are equal (see Figure 4.3).

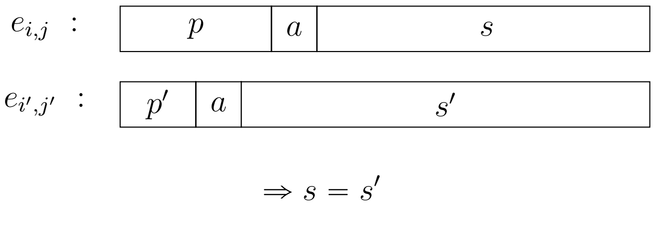

Figure 4.3: Suffix-like equation condition.

The equations obtained when working with EvalCRPQ $(\preceq_{\text{suff}})$ share these properties:

Lemma 4.3.1. The system of word equations obtained from the reduction in 3.4.1 from an instance $\langle G, \bar{a}, \phi \rangle$ for EvalCRPQ $(\preceq_{\text{suff}})$ is equivalent to a system of suffix-like equations.

Before giving the proof for this lemma, a procedure is given to show how this rigid structure can be exploited to enumerate all the minimal solutions with a procedure known as the *pig-pug* algorithm. This corresponds to non-deterministically guessing the length of strings to be associated with variables in order to simplify them in a graphical fashion [1]. First, the result for the base case with $n = 1$ is given:

Lemma 4.3.2. Let $\phi : (\Gamma^*, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ be a minimal solution for the suffix-like equation $\{E : e_1 = \cdots = e_l\} \subset \Gamma^*$. Then, ϕ satisfies the following:

1. $|\Psi| \leqslant |\Gamma|$
2. $\forall a \in \Psi,\ |\phi(E)|_a \leqslant 1$

Consequently, the set of minimal solutions for such an equation is finite, up to isomorphism.

Consider the following example to motivate the algorithm behind this characterization:

Example 4.3.3. Let $\{XYZ = PQRSYZ = ABCXYZ = UVW\}$ be a suffix-like word equation system. Its minimal solutions can be enumerated by applying the *pig-pug* method on :

$$\left|\begin{array}{r} XYZ \\ PQRSYZ \\ ABCXYZ \\ UVW \end{array}\right|$$

These lines can be rearranged while recognizing repeating ending patterns which occur in the system due condition (2).

$$\left|\begin{array}{r} XYZ \\ ABCXYZ \\ PQRSYZ \\ UVW \end{array}\right|$$

In this example, since the variables of line four are not repeated elsewhere, the pig-pug can be first applied in the beginning three lines, and the result obtained combined with line four:

$$\left|\begin{array}{c} \left|\begin{array}{r} XYZ \\ ABCXYZ \\ PQRSYZ \end{array}\right| \\ UVW \end{array}\right|$$

This correspond to solve first the equation $\{XYZ = PQRSYZ = ABCXYZ\}$. This simplifies variables YZ from being considered, since they appear in every line.

$$\left|\begin{array}{c}\left|\begin{array}{r} XYZ \\ ABCXYZ \\ PQRSYZ \end{array}\right| \\ UVW \end{array}\right| \longrightarrow \left|\begin{array}{c}\left|\begin{array}{r} X \\ ABCX \\ PQRS \end{array}\right| YZ \\ UVW \end{array}\right|$$

This process can be applied again by recognizing the repeated pattern X that appears at the end of the first two lines. The following scheme summarizes the sequence of equations to be solved :

$$\left|\begin{array}{r} XYZ \\ ABCXYZ \\ PQRSYZ \\ UVW \end{array}\right| \longrightarrow \left|\begin{array}{c}\left|\begin{array}{r} X \\ ABCX \\ PQRS \end{array}\right| YZ \\ UVW \end{array}\right| \longrightarrow \left|\begin{array}{c}\left|\begin{array}{c}\left|\begin{array}{c} \varepsilon \\ ABC \end{array}\right| X \\ PQRS \end{array}\right| YZ \\ UVW \end{array}\right|$$

The procedure corresponds to a right factorization of words enabling to solve the system by only analysing word equations of variables appearing exactly once, whose sets of minimal solutions are always finite [1, 39].

Observation 4.3.4. In the following proof, strings are to be accessed from right to left. For this purpose, a convention is adopted for indexing strings with negative integers. For a string $w \in \Sigma^*$ and $i \in [|w|]$, $w[-i] = w[n-i+1]$, that is to say, $w[-i]$ is the i-th symbol from right to left. Additionally, for any string w, $w[0] = \varepsilon$, and $w[-i..] = w[n-i+1..]$.

Proof for Lemma 4.3.2. Consider an equation $\{E : e_1 = \cdots = e_l\}$, and let $m = \max |e_i|$, and consider

$$\mathcal{P}(E) = \{i \in [m] \ : \ \exists e, e' \in E,\ i \leqslant \min\{|e|, |e'|\},\ e[-i+1] = e'[-i+1] \ \wedge \ e[-i] \neq e'[-i]\}|$$

Let $T(E) = |\mathcal{P}(E)|$. According to condition (2), $T(E)$ equals the number of times the equation system is able to split as in Example 4.3.3, for it is the number of positions where at least two lines stop being equal from right to left. Accordingly, indexes in $\mathcal{P}(E)$ will be called *split positions*. The proof is by induction on $T \geqslant 0$.

First, if $T = 0$, this means that either all the lines are suffixes of a fixed string $e_0 \in \Gamma^*$ or for each pair $e, e' \in E$, $alph(e) \cap alph(e') = \varnothing$. In the first case, the unique solution for the system is that which trims all the lines, leaving only a quantity of their ending symbols equal to the size of the minimum length line (see Figure 4.4). For this case, the claim is straightforward.

$$\left|\begin{array}{r} XYZ \\ AXYZ \\ YZ \\ STAXYZ \\ TAXYZ \end{array}\right| \rightarrow \left|\begin{array}{r} X\cancel{YZ} \\ AX\cancel{YZ} \\ \cancel{YZ} \\ STAX\cancel{YZ} \\ TAX\cancel{YZ} \end{array}\right| \rightarrow \left|\begin{array}{r} X \\ AX \\ \varepsilon \\ STAX \\ TAX \end{array}\right| \quad \Rightarrow \quad \begin{array}{c} Y, Z \in \Sigma^* \\ S, T, A, X = \varepsilon \end{array}$$

Figure 4.4: Example of equation with $T = 0$, all lines suffixes of a same string

For the second case, this means that every line is conformed of unique symbols. Consequently, from condition (1), every single variable in E appears only once. Accordingly, the minimal solutions for E are a finite set. More precisely, its solutions are obtained by non-deterministically guessing the order in size of the strings to be associated with each variable. The number of minimal solutions is then bounded by the number of ways $s = |\Gamma|$ unknown numbers can be ordered, distinguishing $<$ and $=$, which is at most $(2^{s-1}s!)$. In this case, Ψ corresponds to the partition obtained by projecting the endings of variables onto the unified word (see Figure 4.5). Thus, in the worst case, where no two limits coincide, the number of parts $|\Psi|$ is limited by $\sum_i |e_i| = |\Gamma|$, since there are no repeated variables. Also, in the unified word, every symbol from Γ appears only once since it corresponds to the segment of the partition obtained under its associated string (see Figure 4.5).

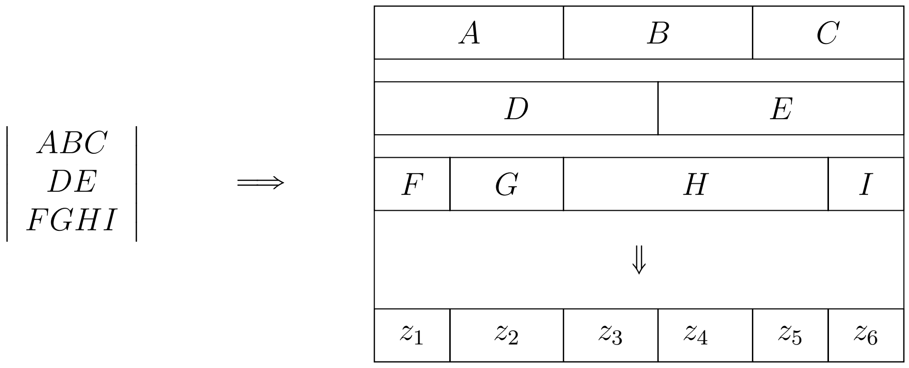

Figure 4.5: A solution for $\{ABC = DE = FGHI\}$ with $\Psi = \{z_1, ..., z_6\}$. For instance, $\phi(B) = z_3z_4$ and $\phi(H) = z_3z_4z_5$.

Now it is assumed the claimed property is true for suffix-like equations E' with $T(E') < T(E)$. The split position $i^* = \arg\min \mathcal{P}(E) \geqslant 1$ is defined to be the first position from right to left where at least two lines have stopped being equal. A common suffix $e_0 \in \Gamma^*$ must exist for all lines, it is the string such that for each $i \in [l]$, $e_0 = e_i[-i^* + 1..]$ (which may be empty, if $i^* = 1$.). Equation $\hat{E} = \{\hat{e}_1 = \cdots = \hat{e}_l\}$ is the one obtained from E by disposing its lines from their shared suffix e_0. In other words, $e_i = \hat{e}_i e_0$ for each $i \in [l]$. A relation on the lines of the trimmed equation system is defined according to their last symbol (which is the symbol at position i^* from right to left in the original equation):

$$\forall \hat{e}, \hat{e}' \in \hat{E}, \quad \hat{e} \sim \hat{e}' \iff \hat{e}[-1] = \hat{e}'[-1]$$

It is straightforward to see that $\sim$ defines an equivalence relation on $\hat{E}$. As such, a partition of subsystems of equations $\hat{E}|_\sim = \{E_1, ..., E_r\}$, $E_i \subseteq \hat{E}$ for $i = 1, ..., r \leqslant l$, is obtained. For a subsystem E_i in $E|_\sim$ let $\Gamma_i \subseteq \Gamma$ be the set of variables appearing in it.

Claim 4.3.5. For each pair of distinct subsystems E_i, E_j in $\hat{E}|_\sim$ it is the case that $\Gamma_i \cap \Gamma_j = \varnothing$.

Proof. Indeed, suppose *ad absurdum* that this does not hold for a distinct pair E_i, E_j and that there is a variable $x^* \in \Gamma$ shared by them. Since these came from the partition generated by $\sim$ on the trimmed equation system $\hat{E}$, this forces the existence of two lines in the original system $e, e' \in E$, with $\hat{e} \in E_i$ and $\hat{e}' \in E_j$, for which a shared symbol x^* appears farther from i^* from right to left. By condition (2), this entails $e[-i^*] = e'[-i^*]$ (see figure 4.6). But if that is so, then the lines $\hat{e}$ and $\hat{e}'$ belong to the same subsystem in $\hat{E}|_\sim$, which is a contradiction.

□

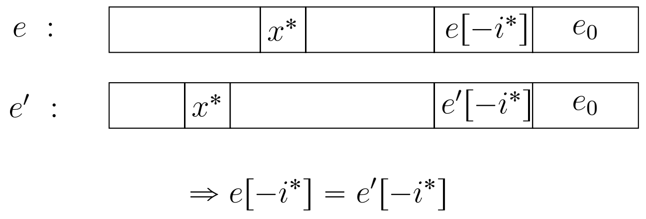

Figure 4.6: Equations systems in $\hat{E}|_\sim$ do not share variables.

From this, it can be concluded that $\Gamma = alph(e_0) \dot\cup \left(\dot\bigcup_{i=1}^{r} \Gamma_i \right)$. Consequently, by right-factorizing e_0, the pig-pug method applied to E is equivalent to:

$$\left\| \begin{array}{c|c|} |E_1| & \\ \vdots & e_0 \\ |E_r| & \end{array} \right.$$

In other words, this gives an order to proceed with the pig-pug method, solving each subsystem E_i independently first, since their variables do not intersect.

It is straightforward to see that for each $i \in [r]$, $T(E_i) \leqslant T(E) - 1$, since their lines are prefixes of lines of E, all of them lacking at least one split position: i^*. Additionally, as prefixes, they inherit its suffix-like properties. Thus by induction, every minimal solution $\phi_i : (\Gamma_i^*, \cdot, \varepsilon) \to (\Psi_i^*, \cdot, \varepsilon)$ for E_i (and thus obtained by the pig-pug method) verifies $|\Psi_i| \leqslant |\Gamma_i|$ and $\forall a \in \Psi_i,\ |\phi(E_i)|_a \leqslant 1$. Additionally, sets Ψ_i can be supposed disjoint: a solution with intersection of them is not minimal since it can be later obtained by a morphism identifying variables to be equal from disjoints Ψ_i's. By substituting the unified words solving each equation E_i in the pig-pug for E the following is obtained:

$$\left\| \begin{array}{c|c|} |E_1| & \\ \vdots & e_0 \\ |E_r| & \end{array} \right. \rightarrow \left| \begin{array}{c} \phi_1(E_1)e_0 \\ \vdots \\ \phi_r(E_r)e_0 \end{array} \right| \rightarrow \left| \begin{array}{c} \phi_1(E_1)\cancel{e_0} \\ \vdots \\ \phi_r(E_r)\cancel{e_0} \end{array} \right| \rightarrow \left| \begin{array}{c} \phi_1(E_1) \\ \vdots \\ \phi_r(E_r) \end{array} \right|$$

Equation E has thus been reduced to solve an r-lines equation on variables $\dot\bigcup_i \Psi_i$, where each line has unique symbols, and each symbol appears at most once in each line. As was previously discussed, a minimal solution for such equation $\phi : (\dot\bigcup_i \Psi_i, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ satisfies

1. $|\Psi| \leqslant \sum_{i=1}^{r} |\phi_i(E_i)| = \sum_{i=1}^{r} |\Psi_i| \leqslant \sum_{i=1}^{r} |\Gamma_i| \leqslant |\Gamma|$
2. $\forall a \in \Psi,\ |\phi(E)|_a \leqslant 1$.

Since any minimal solution for a suffix like equation $\phi : (\Gamma^*, \cdot, \varepsilon) \to (\Psi^*, \cdot, \varepsilon)$ satisfies $|\Psi| \leqslant |\Gamma|$, the number of different minimal solutions, up to isomorphism, is finite. This concludes the proof. $\square$

It easy to extend this base case to the general instance with $n > 1$. The minimal solutions are sought with the pig-pug method applied sequentially in each subsystem E_i, substituting the variables from E_i appearing in E_j with $j > 1$ with the solution obtained. The key is that the solutions for each E_i are so rigid that in each iteration the updated subsystems remain suffix-like:

Lemma 4.3.6. Any suffix-like equation $\{E_i : e_{i,1} = \cdots = e_{i,l_i}\}_{i=1}^n \subset \Gamma^*$ has a finite number of minimal solutions.

Proof. The result can be concluded with the following invariant:

Claim 4.3.7. For each $i \in [n]$, if $\phi_i^0 : (\Gamma_i^*, \cdot, \varepsilon) \to (\Psi, \cdot, \varepsilon)$ is a minimal solution for the subsystem E_i, with $\Psi \cap \bar{\Gamma}_i = \varnothing$, and ϕ_i is its canonical morphism extension to Γ leaving symbols from $\bar{\Gamma}_i$ invariant, then the system $\{\phi_i(E_j) : \phi_i(e_{j,1}) = \cdots = \phi_i(e_{j,l_j})\}_{j \in [n] \setminus \{i\}} \subset (\bar{\Gamma}_i \cup \Psi)^*$ remains suffix-like.

Proof. Fix an index $i \in [n]$, and let $\phi_i : (\Gamma^*, \cdot, \varepsilon) \to (\Psi, \cdot, \varepsilon)$ be the extension to Γ of a minimal solution for $E_i \subseteq \Gamma_i^*$. Accordingly, ϕ_i satisfies the following

$$\forall a \in \Psi,\ |\phi(E_i)|_a \leqslant 1$$

Claim 4.3.8. For each $j \in [n] \setminus \{i\}$, $k \in [l_j]$, and $a \in \bar{\Gamma}_i \dot{\cup} \Psi$, $|\phi_i(e_{j,k})|_a \leqslant 1$.

Proof. Indeed. First, if $e_{j,k} \in \bar{\Gamma}_i^*$, the claim is trivial, since $\phi_i(e_{j,k}) = e_{j,k}$. Accordingly, suppose there is a symbol $x \in \Gamma_i$ such that $|e_{j,k}|_x = 1$. Let x be the first such symbol from left to right appearing in $e_{j,k}$. Since this symbol is in Γ_i because it is shared with a line from equation E_i, then from condition (2) of suffix like systems it follows that there are strings $p \in \bar{\Gamma}_i^*$, $s \in \Gamma_i^*$, such that $e_{j,k} = ps$, with $s[1] = x$, and s the suffix of a line $e_i^* \in E_i$. Consequently, when ϕ_i is applied onto $e_{j,k}$, the resulting string has a first part conformed of symbols in $\bar{\Gamma}_i$, and a second part from symbols in a disjoint alphabet Ψ (see Figure 4.2).

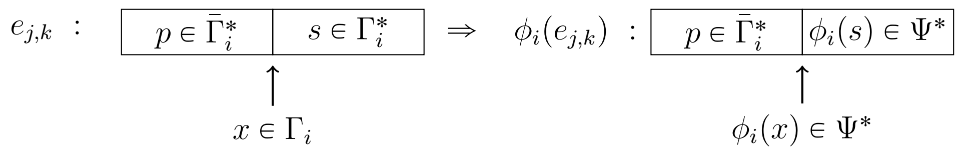

Figure 4.2: Shape of string $e_{j,k} = ps$ with $p \in \bar{\Gamma}_i^*$, $s \in \Gamma_i^*$. Notice that $\phi_i(p) = p$.

Let a be a symbol in $\bar{\Gamma}_i \dot{\cup} \Psi$ appearing in $\phi_i(e_{j,k})$. If $a \in \bar{\Gamma}_i$, then $|p|_a \geqslant 1$ and $|\phi_i(s)|_a = 0$. Since p is a prefix of $e_{j,k}$ and this is a line of a suffix-like equation, then $|p|_a \leqslant |e_{i,j}|_a \leqslant 1$, thus $|\phi_i(e_{j,k})|_a = 1$. Otherwise, if $a \in \Psi$, then $|p|_a = 0$ and $|\phi_i(s)|_a \geqslant 1$. Since ϕ_i is a minimal solution for E_i and s is the suffix of a line $e_i^* \in E_i$, it follows that $|\phi_i(s)|_a \leqslant |\phi_i(e_{i,j})| \leqslant 1$. Consequently, $|\phi_i(e_{j,k})|_a = 1$. This concludes the proof for Claim 4.3.8. $\square$

Claim 4.3.9. For each $j, j' \in [n] \setminus \{i\}$, $k \in [l_j]$, $k' \in [l_{j'}]$ if strings $p, s, p', s' \in (\bar{\Gamma}_i \dot{\cup} \Psi)^*$ and a symbol $a \in \bar{\Gamma}_i \dot{\cup} \Psi$ exist such that $\phi_i(e_{j,k}) = pas$ and $\phi_i(e_{j',k'}) = p'as'$, then $s = s'$.

Proof. Indeed. By using the exact same arguments from the proof for Claim 4.3.8, there are strings $x, x' \in \bar{\Gamma}_i^*$, $y, y' \in \Gamma_i^*$, such that $e_{j,k} = xy$ and $e_{j',k'} = x'y'$. These strings y, y' are suffixes of lines $e_i, e_i' \in E_i$. Consequently, the obtained strings when applying ϕ_i onto these are $\phi_i(e_{j,k}) = x\phi_i(y)$ and $\phi_i(e_{j',k'}) = x'\phi_i(y')$ (see Figure 4.3).

$$\phi_i(e_{j,k}) \ : \ \begin{array}{|c|c|}\hline \quad x \in \bar{\Gamma}_i^* \quad & \phi_i(y) \in \Psi_i^* \\ \hline \end{array} \qquad \phi_i(e_{j',k'}) \ : \ \begin{array}{|c|c|}\hline \quad x' \in \bar{\Gamma}_i^* \quad & \phi_i(y') \in \Psi^* \\ \hline \end{array}$$

Figure 4.3: Shape of strings $\phi_i(e_{j,k})$ and $\phi_i(e_{j',k'})$ according to suffix-like conditions.

Let a be a symbol in $\bar{\Gamma}_i \dot{\cup} \Psi$ appearing both in $\phi_i(e_{j,k})$ and $\phi_i(e_{j',k'})$. If $a \in \bar{\Gamma}_i$, then before applying ϕ_i, these strings have the form

$$e_{j,k} = \underbrace{pas}_{x} \cdot y \quad e_{j',k'} = \underbrace{p'as'}_{x'} \cdot y'$$

for $p, s, p', s' \in \bar{\Gamma}_i$. Consequently, from condition (2) of suffix like systems, $sy = s'y'$, and as a consequence, after applying ϕ_i onto these strings, the suffixes of the resulting strings starting from a are equal. Otherwise, if $a \in \Psi$, then a appears in the portions $\phi_i(y)$ and $\phi_i(y')$ of $\phi_i(e_{j,k})$ and $\phi_i(e_{j',k'})$, respectively. Notice that since y, y' are suffixes of lines $e_i, e_i' \in E_i$ and the fact that ϕ_i being a solution for E_i implies $\phi_i(e_i) = \phi_i(e_i')$, then $\phi_i(y)$ can be supposed to be a suffix of $\phi_i(y')$ without loss of generality. Consequently, when $a \in \Psi$, the claim holds. This concludes the proof Claim 4.3.9.

□

These claims prove that indeed when a system of suffix-like equations $\{E_i : e_{i,1} = \cdots = e_{i,l_i}\}_{i=1}^{n} \subset \Gamma^*$ is updated with a minimal solution for one of its subsystems E_i, each line in the resulting system $\{\phi_i(E_j) : \phi_i(e_{j,1}) = \cdots = \phi_i(e_{j,l_j})\}_{j \in [n] \setminus \{i\}} \subset \Gamma^*$ has at most one appearance of each variable, and whenever two lines share a variable, the suffixes starting from it are equal. The obtained system remains thus suffix-like. This concludes the proof for Claim 4.3.7. □

To conclude the proof of Lemma 4.3.6, the minimal solutions for a suffix like equation $\{E_i : e_{i,1} = \cdots = e_{i,l_i}\}_{i=1}^{n} \subset \Gamma^*$ can be obtained by sequentially applying the pig-pug method on each one of its subsystems, with no particular order, since by doing so the algorithm checks any possible combination of length of strings to be associated with variables. Additionally, the solution for any subsystem E_i can be supposed to range in symbols from Ψ_i disjoint from Γ, since a solution supposing intersection of Γ and Ψ_i is not minimal: it can be obtained by first supposing disjoint sets of variables, and later applying a morphism identifying variables to be equal. Consequently, the iterated minimal solutions conforms to Claim 4.3.7, and in each step the updated system remains suffix-like. The set of minimal solutions for such a system is thus always finite. This concludes the proof.

□

4.4 The data complexity of EvalCRPQ ($\preceq_{\text{suff}}$) is in NLogSpace

Now, in order to conclude this chapter, the proof of Lemma 4.3.1 is given, showing the equations obtained from instances for EvalCRPQ ($\preceq_{\text{suff}}$) are equivalent to a suffix-like equation system trough variable substitution:

Proof of Lemma 4.3.1.

Let $\langle R, I \rangle$ be an instance for GenInt (REC$^*, \preceq_{\text{suff}}$). The index set I can be associated with a directed graph on vertices $[m]$, $G^I([m], I)$, whose edges are given by I. In this context, G^I can be supposed to form a directed acyclic graph without loss of generality, since $\preceq_{\text{suff}}$ is a partial order [9]. Accordingly, let $\{v_1, ..., v_m\} = [m]$ be a topological order for $G^I([m], I)$. Consider $R = L_1 \times \cdots \times L_m$, and $e_{I,R}$ be the system of word equations with regular constraints obtained by reduction 3.4.1 on $\langle R, I, \preceq_{\text{suff}} \rangle$.

According to the proof for Proposition 3.4.1, since $\preceq_{\text{suff}} = \{(X, Y) \in \Sigma^* : \exists p \in \Sigma^*,\ Y = pX\}$, the variables of $e_{I,L}$ are $\Gamma = \{Z_1, ..., Z_m\}$ and $\mathcal{P} = \{P_{i,j}\}_{(i,j) \in I}$, and its constraints are $\{L_i\}_{x \in \Gamma \cup \mathcal{P}}$. Thus $e_{I,R}$ can be seen as $(\Gamma \cup \mathcal{P}, E, \{L_x\}_{x \in \Gamma \cup \mathcal{P}})$, with E a system of unrestricted constant-free equations on variables $\Gamma \cup \mathcal{P}$ and the regular constraints $\{L_x\}_{x \in \Gamma \cup \mathcal{P}}$.

At first, $e_{I,L}$ is not in the correct form for being characterized as suffix-like. A process of variable substitution is shown, following the topological order of G^I, establishing its equivalence to a system of suffix-like equations. Since the index set and the relations are fixed, E is also fixed, and the variable part of this problem is the regular constraints. The handling of these equations has thus no influence in the complexity of the problem.

Let F and $\bar{F}$ be originally the empty set. At the end of the substitution process to be explained in the following, F will be a suffix-like equation system and $\bar{F}$ a set of trivial equations, variable assignations actually, such that E is equivalent to $F \wedge \bar{F}$, in the sense that they share the same set of solutions. The following preamble will clarify the procedure :

The vertices of G^I are to be visited in the topological order. When visiting v_i, a morphism $\phi_i : ((\Gamma \cup \mathcal{P})^*, \cdot, \varepsilon) \to ((\Gamma \backslash \{Z_i\} \cup \mathcal{P})^*, \cdot, \varepsilon)$ will be defined, with $\phi_i(a) = a$ for any symbol other than Z_i, being the variable substitution to be made. According to this, ϕ_i is completely determined by $\phi_i(Z_i)$. Additionally, during this process E is to be applied ϕ_i onto after visiting each vertex, that is to say, $E \leftarrow \phi_i(E)$. For this purpose, $\Phi_i = \phi_{i-1} \circ \cdots \circ \phi_1$ will denote the iterated resulting substitution performed after visiting v_{i-1}. Consequently, if vertex v_j had originally an equation $Z_j = P_{i,j} Z_i$, for $(i,j) \in I$, when it is visited this equation will be $Z_j = P_{i,j} \Phi_j(Z_i)$, since according to the preceding these substitutions always leave symbols from $\mathcal{P}$ invariant, and the symbols changed are always from vertices already visited by the process. Now the substitution ϕ_i for v_i is defined, according to its inner degree in G^I:

1. If $\delta^-_{G^I}(v_i) = 0$, then $\phi_i = id$. That is, no substitution is to be made for this case.
2. If $\delta^-_{G^I}(v_i) = 1$, let v_r be the only vertex in G^I pointing at v_i. As such, the only equation with Z_i at the left side is $Z_i = P_{r,i} \Phi_i(Z_r)$. In this case $\phi_i(Z_i)$ is defined as $P_{r,i} \Phi_i(Z_r)$. The equation $\{Z_i = P_{r,i} \Phi_i(Z_r)\}$ is added to $\bar{F}$ and E is applied ϕ_i onto. No equation is added to F.

3. If $\delta^-_{G^I}(v_i) = d > 1$, let $v_{i_1} < \cdots < v_{i_d}$ be the vertices pointing at v_i in G^I. Accordingly, there are d equations with Z_i at the left side:

$$\{Z_i = P_{i_1,i} Z_{i_1}\}, \cdots, \{Z_i = P_{i_d,i} Z_{i_d}\}$$

The substitution is defined with $\phi_i(Z_i) = P_{i_1,i}\Phi_i(Z_{i_1})$. The equation $\{Z_i = P_{i_1,i}\Phi_i(Z_{i_1})\}$ is added to $\bar{F}$, E is applied ϕ_i onto, and the equation $\{P_{i_1,i}\Phi_i(Z_{i_1}) = \cdots = P_{i_d,i}\Phi_i(Z_{i_d})\}$ is added to F.

Notice that for any $i \in [m]$ such that $\delta^-_{G^I}(v_i) > 0$, with v_r the earliest vertex in the topological order pointing at v_i, $\Phi_j(Z_i)$ is given for any $j \in [m]$ by

$$\Phi_j(Z_i) = \begin{cases} Z_i & j < i \\ P_{r,i}\Phi_i(Z_r) & j \geqslant i \end{cases}$$

This is because each substitution ϕ_i only changes the variable Z_i. Consequently, the iterated substitution Φ_i, only changes symbols indexed by $j \leqslant i$. The following claims will conclude the rest of the proof:

Claim 4.4.1. After the process has ended, E is equivalent to $F \wedge \bar{F}$.

Proof. This is proven by induction on the following invariant : for $i \in [m]$, let E_i to be the subsystem of E with variables Z_j, $P_{m,n}$, with $j, m, n \leqslant i$. After visiting vertex v_i, denote F_i and $\bar{F}_i$ to be the set of equations obtained up to that point. Then E_i is equivalent to $F_i \wedge \bar{F}_i$.

1. When $i = 1$, v_1 is a minimal vertex of G^I. As such, $\delta^-_{G^I}(v_1) = 0$. There are no equations in E_1 and F and $\bar{F}$ are empty, so the claim is true.

2. Suppose now that E_{i-1} is equivalent to $F_{i-1} \wedge \bar{F}_{i-1}$, and let v_i be the vertex being visited.

   (a) If $\delta^-_{G^I}(v_i) = 0$, then $E_i = E_{i-1}$, since no new equations are to be considered. Also $F_i = F_{i-1}$ and $\bar{F}_i = \bar{F}_{i-1}$, so the claim holds.

   (b) If $\delta^-_{G^I}(v_i) = 1$, then there is just one extra equation in E_i not in E_{i-1}, which is $Z_i = P_{r,i}Z_r$, with $(r, i) \in I$ and $r < i$. Accordingly, since Z_r has already been visited, there is an equation in $\bar{F}_{i-1}$ stated as $Z_r = P_{t,r}\Phi_r(Z_t)$ for some $t < r$. By induction, the equations in E_{i-1} are consistent with this, and in the system E_i, the equation $Z_i = P_{r,i}Z_r$ can be replaced by $Z_i = P_{r,i}P_{t,r}\Phi_r(Z_t)$. But according to the variable substitution process, $\Phi_r(Z_r)$ is precisely $P_{t,r}\Phi_r(Z_t)$. Thus in E_i, $Z_i = P_{r,i}P_{t,r}\Phi_r(Z_t) = P_{r,i}\Phi_r(Z_r)$. Since the symbols appearing in $\Phi_r(Z_r)$ are all of the form $Z_j, P_{m,n}$ with $j, m, n \leqslant r$ and the substitutions later than r only change symbols with greater indexes, $\Phi_r(Z_r) = \Phi_i(Z_r)$. Putting all of this together, in E_i, $Z_i = P_{r,i}Z_r$ is equivalent to $Z_i = P_{r,i}\Phi_i(Z_r)$. Conversely, this is precisely the new equation to be put in $\bar{F}_i$ after visiting v_i. Since in this case $F_i = F_{i-1}$, it follows that E_i is equivalent to $\bar{F}_i \wedge F_i$.

   (c) If $\delta^I_{G^I}(v_i) = d > 1$, let $v_{i_1}, ..., v_{i_d}$ be the vertices pointing at v_i. There are exactly d new equations in E_i which are

$$\{Z_i = P_{i_1,i} Z_{i_1}\}, \cdots, \{Z_i = P_{i_d,i} Z_{i_d}\}$$

For the proof, one of these is picked arbitrarily, say $Z_i = P_{i_j,i}Z_{i_j}$ with $j \in [d]$. By using the exact same arguments, in the system E_i this equation can be replaced by $Z_i = P_{i_j,i}P_{t,i_j}\Phi_{i_j}(Z_t)$ for some $t < i_j$ and consequently by $Z_i = P_{i_j,i}\Phi_{i_j}(Z_{i_j})$. Which in turn, since $i_j < i$, is equal to $Z_i = P_{i_j,i}\Phi_i(Z_{i_j})$. This holds for any $j \in [d]$, thus the new equation in F_i, $\{P_{i_1,i}\Phi_i(Z_{i_1}) = \cdots = P_{i_d,i}\Phi_i(Z_{i_d})\}$, and in particular $Z_i = P_{i_1,i}\Phi_i(Z_{i_1})$ which is the equation to be put in $\bar{F}_i$ after visiting v_i, are implied by E_i.

Conversely, for $j \in [d]$, $Z_i = P_{i_j,i}\Phi_i(Z_{i_j})$ will not appear in $\bar{F}_i$, but $Z_i = P_{i_1,i}\Phi_i(Z_{i_1})$ will. And additionally, in this case the equation $\{P_{i_1,i}\Phi_i(Z_{i_1}) = \cdots = P_{i_d,i}\Phi_i(Z_{i_d})\}$ is added to F_i. Thus, in E_i, $Z_i = P_{i_1,j}\Phi_i(Z_{i_j})$ is implied by $F_i \wedge \bar{F}_i$ for any $j \in [d]$. Thus E_i is equivalent to $F_i \wedge \bar{F}_i$.

□

Claim 4.4.2. F is a suffix like equation system.

Proof. Let $X = \{v_i \in [m] : \delta^-_{G^I}(v_i) > 1\}$ be the set of vertices of G^I associated with equations to be put in F. Accordingly, the system F is given by $F = \{F_i \ : \ P_{i_1,i}\Phi_i(Z_{i_1}) = \cdots = P_{i_1,d}\Phi_i(Z_{i_d})\}_{v_i \in X}$. For each $v_i \in X$, let $\rho(v_i) = \{(x_1, x_0), (x_2, x_1), (x_3, x_2), ..., (x_{n_i}, x_{n_i-1})\} \subseteq I$ with $x_0 = v_i$, be a sequence of edges of G^I such that :

1. $\delta^-_{G^I}(x_{n_i}) = 0$.
2. For each $j \in [n_i - 1]$, $x_{j+1} = \arg\min N^-_{G^I}(x_j)$.

In other words, for $v_i \in [m]$, $\rho(v_i)$ is the set of edges $(x_{i+1}, x_i) \in I$ joining a sink x_{n_i} with v_i, such that when traversing its edges backwards, the preceding vertex in the path for any intermediary vertex is its earlier back-neighbour in the topological order. Notice that this definition is correct, since there is only one such path. The following then holds:

Claim 4.4.3. For any $i \in [m]$ such that $\delta^-_{G^I}(v_i) > 0$, with $\rho(v_i) = \{(x_1, x_0), (x_2, x_1), (x_3, x_2), ..., (x_{n_i}, x_{n_i-1})\}$, it is the case that $\Phi_l(Z_i) = P_{x_1,x_0}P_{x_2,x_1}\cdots P_{x_{n_i-1},x_{n_i-2}}Z_{n_i}$ for any $l \geqslant i$.

Proof. Let $v_i \in [m]$ such that $\delta^-_{G^I}(v_i) > 0$. This is proven by induction on $|\rho(v_i)| \geqslant 1$.

First, if $|\rho(v_i)| = 1$, this means the earlier vertex in the topological order pointing at v_i in G^I is one of its sinks $s \in [n]$. If $\delta^-_{G^I}(v_i) = 1$, then s is the only vertex pointing at v_i. Consequently, $\phi_i(Z_i) = P_{s,i}Z_s$, so the claim holds. Otherwise, if $\delta^-_{G^I}(v_i) = d > 1$, vertices $v_{i_1} < \cdots < v_{i_d}$ point at v_i. If $|\rho(v_i)| = 1$, then v_{i_1} is a sink of G^I. In this case, $\phi_i(Z_i) = P_{i_1,i}Z_{i_1}$, and so the claim holds.

Now let $\rho(v_i) = \{(x_1, x_0), (x_2, x_1), (x_3, x_2), ..., (x_{n_i}, x_{n_i-1})\}$ with $v_i = x_0$ and $n_i > 1$. Since x_1 is the earliest vertex in the topological order pointing at $x_0 = v_i$, it is the case that $\phi_i(Z_i) = P_{x_1,x_0}\Phi_{x_0}(Z_{x_1})$, either if the inner degree of v_i is 1 or greater. It is straightforward to see that $\rho(x_1) = \{(x_2, x_1), (x_3, x_2), ..., (x_{n_i}, x_{n_i-1})\}$. Consequently, $|\rho(x_1)| < |\rho(v_i)|$, so by induction, since $x_0 > x_1$, $\Phi_{x_0}(Z_{x_1}) = P_{x_2,x_1}\cdots P_{x_{n_i-1},x_{n_i-2}}Z_{n_i}$. As a result, $\phi_i(Z_i) = P_{x_1,x_0}\Phi_{x_1}(Z_{x_1}) = P_{x_1,x_0}P_{x_2,x_1}\cdots P_{x_{n_i-1},x_{n_i-2}}Z_{n_i}$. This concludes the proof for Claim 4.4.3. □

Since all the lines of equations in F are of the form $P_{j,i}\Phi_i(Z_j)$ for $(j, i) \in I$, from Claim 4.4.3 they represent the unique path in I given by $\rho(v_i)$ between a sink node and the vertex v_i. Since in $\rho(v_i)$ no edge is repeated, each variable in $P_{j,i}\Phi_i(Z_j)$ appears at most once. Additionally, if two lines $P_{j,i}\Phi_i(Z_j)$ and $P_{j',i'}\Phi_{i'}(Z_{j'})$ share a variable, is because the sets $\rho(v_i)$ and $\rho(v_{i'})$ intersect at some edge. Since these sets represent the paths described earlier, once an edge is shared, all the previous edges used in the path must coincide, and consequently, the suffixes of both lines starting at the shared variable must do as well. In other words, F is a suffix like equation system. □

Claim 4.4.4. The minimal solutions for $\{F \wedge \bar{F}\}$ are determined by those for F.

Proof. It is straightforward to see from the variable substitution process that each equation in $\bar{F}$ has the form $Z_i = P_{r,i}\Phi_i(Z_r)$ for $(r, i) \in I$ and a vertex v_i not a sink. Additionally, every variable Z_i associated with a vertex v_i of inner degree greater than 0 is substituted from E. Consequently, since there are no equations at all for variables associated with sink vertices, the set of variables appearing at the left side of the equations in $\bar{F}$ is disjoint from those appearing at the right side. It is also disjoint for this same reason from the set of variables in F. Since for each left-side variable Z_i there is just one equation containing it in $\bar{F}$, it follows that the solutions for $\bar{F}$ are completely determined by F.

□

Summarizing, the system of word equations E obtained from an instance $\langle R, I\rangle$ for GENINT (REC*, $\preceq_{\text{suff}}$) is equivalent trough variable substitution to the system $F \wedge \bar{F}$. The minimal solutions for the later are completely determined by F, which in turn is a suffix-like equation system. This concludes the proof.

□

As a consequence of this, the following result is obtained

Theorem 4.4.5. *For each query* $\phi \in$ CRPQ($\preceq_{\text{suff}}$)*, the problem* EVALCRPQ ($\preceq_{\text{suff}}, \phi$) *is in* NLOGSPACE.

Proof. An instance $\langle G, \bar{a}\rangle$ for the evaluation problem EVALCRPQ ($\preceq_{\text{suff}}, \phi$) reduces in logarithmic space, considering the query ϕ fixed, to the satisfiability problem of a fixed word equation with variable regular constraints. From the preceding lemma, this word equation is equivalent to a fixed system of suffix-like equations which admits a finite set of minimal solutions. Consequently, satisfiability of it with variable regular constraints can be decided in NLOGSPACE, since NLOGSPACE functions are closed under composition.

□

Chapter 5

A CRPQ($\preceq_{\mathrm{ss}}$) query that is NP-hard to evaluate

In this chapter it is shown that there exist CRPQ($\preceq_{\mathrm{ss}}$) queries which are NP-hard to evaluate. The conclusion follows from a recent result from [14] and [49] , in which NP-hardness is proven for the problem of *unshuffling a square string*, which can be reduced to EvalCRPQ $((\,)\preceq_{\mathrm{ss}}, \phi)$ for a fixed query $\phi \in$ CRPQ($\preceq_{\mathrm{ss}}$).

5.1 Unshuffling a square

Let Σ be a fixed finite alphabet. If u, v, w are strings over Σ, for which strings $x_1, ..., x_k$, $y_1, ..., y_k \in \Sigma^*$ exist such that

$$v = x_1 x_2 \cdots x_k \quad u = y_1 y_2 \cdots y_k \quad \text{and} \quad w = x_1 y_1 \cdots x_k y_k$$

then w is called a *shuffle* of u and v, written $w = u \odot v$ (see Figure 5.1). If $w = u \odot u$, then w is called a *square* string. The set of all square strings is denoted Square $= \{w \in \Sigma^* \ : \ \exists u \in \Sigma^*,\ w = u \odot u\}$.

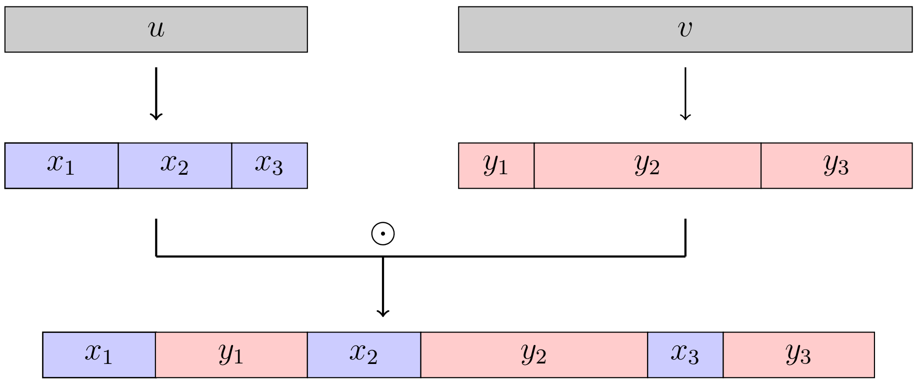

Figure 5.1: A scheme for the shuffle $u \odot v$.

The first appearances of the shuffle operator are from abstract formal languages in [27]. From there, its applications on sequential execution of concurrent process made it object of study by numerous authors [47, 53, 48]. Other studies focused on the shuffle operator itself in a general fashion [43, 56]. Recently, the problem of deciding if an input string is a square was received some attention [24, 30] . The problem is clearly in NP, since a witness $u \in \Sigma^*$ and its partition $u = x_1...x_k = y_1...y_k$ exhibiting $w = u \odot u$ is of linear size on $|w|$. After some partial answers [8], a proof yielding NP-hardness for the problem by a many-one reduction from 3-PARTITION was given last year:

Theorem 5.1.1 ([14, 49])**.** *The set* SQUARE *is NP-complete. This is true even for sufficiently large fixed alphabets.*

This result shows NP-hardness for alphabets with at least 7 symbols, but the authors conjecture it is valid even in binary alphabets (for unary alphabets SQUARE equals the set of even length strings, and can thus be solved in polynomial time).

5.2 Coding Square as an EvalCRPQ ($\preceq_{ss}$) instance

Before giving a fixed query that is NP-hard to evaluate on graph-dbs, the following intermediary result is given for the generalized intersection problem GENINT (REC*, $\preceq_{ss}$), which was presented in Section 3.1 when working with equations for EVALCRPQ (EQ) as a problem holding the essence of CRPQ($\mathcal{S}$) query evaluation. Let $I_d = \{(1, 2), (1, 3), (2, 4), (3, 4)\}$.

Lemma 5.2.1. SQUARE many-one reduces in polynomial time to $\text{GENINT}_{I_d,\preceq_{ss}}$ (REC*).

Before giving the proof, an example is shown to illustrate the main idea behind it.

Example 5.2.2. Consider $u = abaabca \in \{a, b, c\}^*$. There are several ways in which u can be segmented to create square shuffles. For instance the following partition

$$u = abaabca$$

$$x_1 = ab \quad x_2 = aab \quad x_3 = ca$$

$$y_1 = a \quad y_2 = baa \quad y_3 = bca$$

This setting engenders the shuffle $w = abaaabbaacabca$. If given w, it would seem to require some work to understand how w is build up from two partitions of u, even while having seen u before. If one knew, however, from which partition each symbol came from, then the problem would become trivial. By assigning the color blue for each symbol coming from partition $x_1x_2x_3$, and color red for those from $y_1y_2y_3$, it becomes evident that w is a square string, since by concatenating only the blue symbols one would obtain the same as when concatenating the red ones:

$$w = abaaabbaacabca$$

The notion of *colouring symbols* can be formalised by adding two new symbols to the alphabet, say σ_b, σ_r. Each symbol from Σ in w is preceded and followed by σ_b if its colour is blue, or by σ_r, if it is red. The string $\hat{w}$, in an extended alphabet $\Sigma \cup \{\sigma_b, \sigma_r\}$, would be obtained from w:

$$\hat{w} = (\sigma_b a \sigma_b)(\sigma_b b \sigma_b)(\sigma_r a \sigma_r)(\sigma_b a \sigma_b) \cdots (\sigma_r a \sigma_r)$$

The key behind the proof is that all the different two-colourings of w can be encoded by a regular language of linear size on w easily constructed. By using the structure given by I_d and the subsequence relation, since σ_b, σ_r are symbols not in Σ, u can be enforced to appear twice in w with two different colourings (see Figure 5.1).

$$\begin{tikzcd}
 & \text{"}\textit{Blue}\text{" } \textit{candidates} \arrow[dr, "\preceq_{\text{ss}}"] & \\
\textit{Candidates} \arrow[ur, "\preceq_{\text{ss}}"] \arrow[dr, "\preceq_{\text{ss}}"'] & & \textit{Input colourings} \\
 & \text{"}\textit{Red}\text{" } \textit{candidates} \arrow[ur, "\preceq_{\text{ss}}"'] &
\end{tikzcd}$$

Figure 5.1: Coding Square in a GenInt$_{I_d, \preceq_{\text{ss}}}$ (REC*) instance.

Proof of Lemma 5.2.1. Let $w \in \Sigma^*$ be an instance input for Square. Assume, without loss of generality, that $|w| = 2n$ (odd length inputs can always be ruled out, since a square is always of even length). Consider new symbols $\sigma_1, \sigma_2 \notin \Sigma$. Four regular languages are to be defined: L_1, the language describing the root candidates for w; L_2 and L_3, the "blue" and "red" colourings of the candidates described by L_1; and L_4, describing all the ways in which w can be two-coloured. Formally, they are defined as follows:

1. $L_1 = alph(w)^n$, the set of strings of exactly n symbols from those appearing in w. $|L_1| = n|alph(w)| = \mathcal{O}(n^2)$.

2. $L_2 = (\sigma_1 \cdot alph(w) \cdot \sigma_1)^n$, the set of strings of exactly $3n$ symbols, whose every symbol from $alph(w)$ is preceded and followed by σ_1. $|L_2| = \mathcal{O}(n(2 + |alph(w)|)) = \mathcal{O}(n^2)$

3. $L_3 = (\sigma_2 \cdot alph(w) \cdot \sigma_2)^n$, the set of strings of exactly $3n$ symbols, whose every symbol from $alph(w)$ is preceded and followed by σ_2. $|L_3| = \mathcal{O}(n(2 + |alph(w)|)) = \mathcal{O}(n^2)$

4. $L_4 = [(\sigma_1 \cdot w_1 \cdot \sigma_1) + (\sigma_2 \cdot w_1 \cdot \sigma_2)] \cdots [(\sigma_1 \cdot w_{2n} \cdot \sigma_1) + (\sigma_2 \cdot w_{2n} \cdot \sigma_2)]$ the set of strings $\hat{w}$ of exactly $6n$ symbols from $\Sigma \bigcup \{\sigma_1, \sigma_2\}$ such that $\hat{u}|_{\Sigma} = w$, and whose each symbol from Σ is preceded and followed by either σ_1 or σ_2. $|L_4| = 2n(13) = \mathcal{O}(n)$.

Let $R = L_1 \times L_2 \times L_3 \times L_4$. The notational length of R is polynomially sized on $|w|$, thus the automata coding R are also of polynomial size on $|w|$.

Additionally, for the proof consider $\phi_1, \phi_2 : (\Sigma^*, \cdot, \varepsilon) \to ((\Sigma \cup \{\sigma_1, \sigma_2\})^*, \cdot, \varepsilon)$ to be the canonical morphisms obtained by extending to Σ^* the mappings

$$\phi_i(a) = \sigma_i a \sigma_i \quad \forall a \in \Sigma, \; i = 1, 2$$

Observation 5.2.3. The following straightforward properties about these morphisms and languages are to be noted:

1. For each $x \in \Sigma^*$, $\phi(x)|_\Sigma = x$. Hence, if $u \preceq_{\text{ss}} v$ then $u \preceq_{\text{ss}} \phi_i(v)$.
2. For each $x \in \Sigma^*$, $|\phi_i(x)| = 3|x|$
3. $L_2 = \{\phi_1(x) : x \in L_1\}$ and $L_3 = \{\phi_2(x) : x \in L_1\}$

Claim 5.2.4. $R\bigcap_{I_d} \preceq_{\text{ss}} \neq \varnothing$ if and only if $w \in$ SQUARE.

Indeed, for the *if* part, suppose $w \in$ SQUARE. Hence, strings $x_1, ..., x_k, y_1, ..., y_k \in \Sigma^*$ exist such that $u := x_1...x_k = y_1...y_k$ and $w = x_1y_1...x_ky_k$. Let $w_1 = u$, $w_2 = \phi_1(u)$, $w_3 = \phi_2(u)$ and $w_4 = \phi_1(x_1)\phi_2(y_1)\cdots\phi_1(x_k)\phi_1(y_k)$. The following then holds

1. $w_1 \in L_1$: If $w_1 = u$ and $w = u \odot u$, then $|w_1| = |u| = \frac{1}{2}|w| = n$. Also, since w is a square shuffle of u, they share the same symbols. Thus, $alph(w_1) \subseteq alph(w)$.
2. $w_2 \in L_2$: Since $w_2 = \phi_1(w_1)$ and $w_1 \in L_1$, then $w_2 \in L_2$ from the previous observation. Equivalently, $w_3 \in L_3$.
3. $w_4 \in L_4$: Since for each $x \in \Sigma^*$, $\phi_i(x)|_\Sigma = x$, then $w_4|_\Sigma = x_1y_1...x_ky_k = w$. Also, every symbol from Σ in w_4 is either preceded and followed by σ_1 or by σ_2, since $w_4 = \phi_1(x_1)\phi_2(y_1)\cdots\phi_1(x_k)\phi_1(y_k)$ and the way morphisms ϕ_i are defined.
4. $w_1 \preceq_{\text{ss}} w_2 \;\wedge\; w_1 \preceq_{\text{ss}} w_3$: Since $w_2 = \phi_1(w_1)$, then $w_2|_\Sigma = w_1$. Thus, w_1 can be obtained from w_2 by removing the σ_1 symbols. The same idea holds for $w_1 \preceq_{\text{ss}} w_3$.
5. $w_2 \preceq_{\text{ss}} w_4 \;\wedge\; w_3 \preceq_{\text{ss}} w_4$: If every $\phi_2(y_i)$ part from w_4 is removed, then one obtains $\phi_1(x_1)...\phi_1(x_k) = \phi_1(u) = w_2$. By removing the $\phi_1(x_i)$ parts, one obtains w_3.

From this, it follows that $(w_1, w_2, w_3, w_4) \in R\bigcap_{I_d} \preceq_{\text{ss}}$.

Assume now, for the *only if* part that $R\bigcap_{I_d} \preceq_{\text{ss}} \neq \varnothing$ and let $(w_1, w_2, w_3, w_4) \in R\bigcap_{I_d} \preceq_{\text{ss}}$. First, it is always the case that $w_2 = \phi_1(w_1)$ and $w_3 = \phi_2(w_1)$. Suppose *ad absurdum* this does not hold, and $w_2 = \phi_1(x)$ for a string $x \in L_1$ other than w_1. Since $w_1 \preceq_{\text{ss}} w_2$, all of its n symbols must appear somewhere in w_2. However, from the $3n$ symbols occurring in w_2, only n of them are from $alph(w)$, the others being σ_1. Thus, if $w_2 = \phi(x)$ and $x \neq w_1$, this leaves no place for at least one symbol from w_1 to appear in w_2, which is a contradiction. Analogously , $w_3 = \phi_2(w_1)$.

Second, $|w_4|_{\sigma_1} = |w_4|_{\sigma_2} = 2n$. Indeed, from the $6n$ symbols in w_4, $2n$ are symbols from $alph(w_4)$, and $4n$ are symbols ranging in $\{\sigma_1, \sigma_2\}$. Strings w_2 and w_3 contain exactly $2n$ symbols σ_1 and σ_2 respectively. Since $w_2 \preceq_{\text{ss}} w_4$ and $w_3 \preceq_{\text{ss}} w_4$, it must be the case that $|w_4|_{\sigma_1} \geqslant 2n$ and $|w_4|_{\sigma_2} \geqslant 2n$. Consequently, since $|w_4|_{\sigma_1,\, \sigma_2} = 4n$, $|w_4|_{\sigma_1} = |w_4|_{\sigma_2} = 2n$.

Let $x_1, ..., x_k$ be maximal subwords from w_4 containing σ_1 but no σ_2 symbols, and $y_1, ..., y_k$ maximal subwords from w_4 containing σ_2 but no σ_1 symbols, yielding, according to the form of L_4, $w_4 = x_1y_1...x_ky_k$ (some of these may be empty strings, if w_4 starts and ends with σ_1 for instance). Let $X = x_1...x_k$, $Y = y_1...y_k$. Then, it must be the case that $X = w_2$. First, the σ_1 symbols of w_2 contained in w_4 must lie within X, since $|Y|_{\sigma_1} = 0$. From this, since $w_2 \preceq_{\text{ss}} w_4$, w_2 is a subsequence of X: all the symbols from Σ in w_2 are preceded and followed by σ_1, thus, they must form part of the maximal subwords $x_1, ..., x_k$ due the definition of L_4. Finally, $|w_4|_{\sigma_1} = 2n$, thus $|X|_{\sigma_1} = 2n$, and those maximal subwords must contain the symbols from Σ between consecutive σ_1's, enforcing $|X| = 3n$. Since w_2 is a subsequence of X, and they have equal lengths, they must be equal. Equivalently, $Y = w_2$.

This shows that $w_4 = w_2 \odot w_3$. Thus, $w_4|_\Sigma = w_2|_\Sigma \odot w_3|_\Sigma$. But $w_4|_\Sigma = w$ and $w_2|_\Sigma = w_3|_\Sigma = w_1$. It follows that $w = w_1 \odot w_1 \in$ SQUARE, which concludes the proof. □

As a consequence of this result a fixed query Q in CRPQ($\preceq_{\text{ss}}$) can be constructed such that for a given SQUARE instance string w there is a graph-db G satisfying Q if and only if $w \in$ SQUARE:

Theorem 5.2.5. *There is a query $\phi \in$ CRPQ($\preceq_{ss}$) such that* EVALCRPQ ($\preceq_{ss}, \phi$) *is* NP*−hard.*

Proof. Consider a fixed alphabet Σ_0 with at least 7 symbols and $w \in \Sigma_0^*$ an instance for SQUARE with $|w| = 2n$. Let σ_1, σ_2 be symbols not in Σ_0 and the regular languages L_i over $\Sigma = \Sigma_0 \cup \{\sigma_1, \sigma_2\}$ for $i = 1, 2, 3, 4$ defined in the proof of Lemma 5.2.1

1. $L_1 = alph(w)^n$
2. $L_2 = (\sigma_1 \cdot alph(w) \cdot \sigma_1)^n$
3. $L_3 = (\sigma_2 \cdot alph(w) \cdot \sigma_2)^n$
4. $L_4 = [(\sigma_1 \cdot w_1 \cdot \sigma_1) + (\sigma_2 \cdot w_1 \cdot \sigma_2)] \cdots [(\sigma_1 \cdot w_{2n} \cdot \sigma_1) + (\sigma_2 \cdot w_{2n} \cdot \sigma_2)]$

For $i = 1, 2, 3, 4$, let $N_i(Q_i, \Sigma, \delta_i, s_i, F_i)$ be the NFA over Σ recognizing the language L_i. Also, consider the Σ-edge labelled graph $G_i(Q_i, E_i)$ interpreting NFA N_i as a graph-db, that is to say, whose edges are given by

$$\forall q, q' \in Q_i,\ \sigma \in \Sigma, \quad (q, \sigma, q') \in E_i \ \Leftrightarrow\ q' \in \delta_i(q, \sigma)$$

Let $\{\#, \bar{\#}\}$ be additional symbols not in Σ, and let $\Psi = \Sigma \cup \{\#, \bar{\#}\}$. A new graph-db $G(V, E)$ is constructed linking those given by G_i (see Figure 5.2). To that end, let v_0, v_1, v_2, v_3, v_4 and v_5 be new vertices, and consider $V = \left(\bigcup_{i=1}^4 Q_i\right) \cup \{v_0, ..., v_4\}$ to be the vertices of the new graph-db G. Its edges are given by $E \subseteq V \times \Psi \times V$ such that

- For each $i \in [4]$ and each edge (q, σ, q') in E_i, $(q, \sigma, q') \in E$.
- For each $i \in [4]$, $(v_{i-1}, \#, s_i) \in E$
- For each $i \in [4]$ and each final state $f \in F_i$, $(f, \bar{\#}, v_i) \in E$.

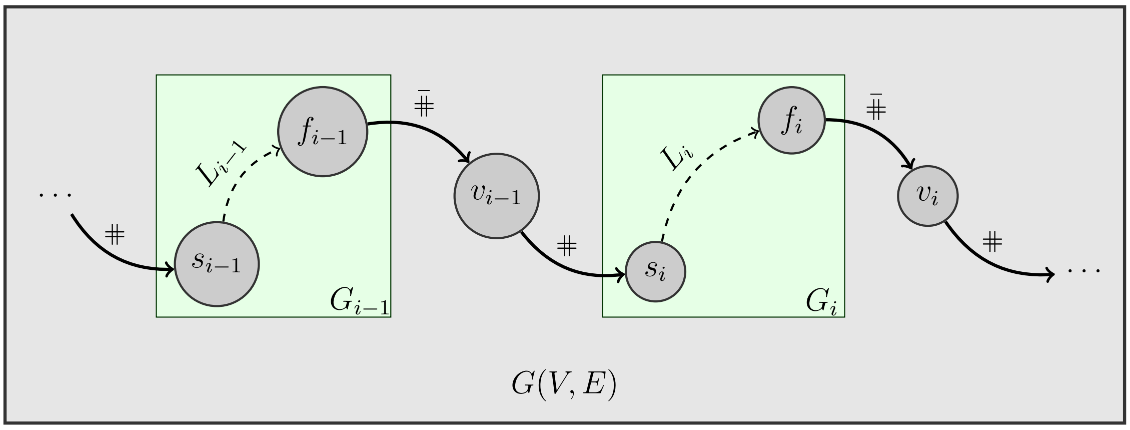

Figure 5.2: Linking NFAs as graph-dbs.

Observation 5.2.6. According to this construction, the only paths existing in G between vertices v_{i-1} and v_i are of the form $\#w\bar{\#}$ with $w \in L_i \subseteq \Sigma^*$, for any $i \in [4]$. Conversely, a path labelled with a string of the form $\#w\bar{\#}$ with $w \in \Sigma^*$ must join two vertices v_{i-1} and v_i for some $i \in [4]$ with $w \in L_i$, since $\#$ and $\bar{\#}$ are symbols not in Σ.

The fixed query in CRPQ($\preceq_{\text{ss}}$) with alphabet Ψ, which compares paths specified by $I_d = \{(1,2),\ (1,3),\ (2,4),\ (3,4)\}$, is the following:

$$Q(x_0, x_1, x_3, x_3, x_4) \ : \ \left[\bigwedge_{i=1}^{4} x_{i-1} \xrightarrow{\chi_i \ : \ L_0} x_i\right] \ \wedge \ \left[\bigwedge_{(i,j)\in I_d} \chi_i \preceq_{\text{ss}} \chi_j\right]$$

$$\text{with } L_0 = \# \cdot \Sigma^* \cdot \bar{\#}$$

Notice that $\Psi = \Sigma_0 \cup \{\sigma_1, \sigma_2, \#, \bar{\#}\}$ is a fixed alphabet, thus this query is independent from w. Consider now the recognizable relation $L = L_1 \times L_2 \times L_3 \times L_4$.

Claim 5.2.7. $G \models Q(v_0, v_1, v_2, v_3, v_4)$ if and only if $L \bigcap_{I_d} \preceq_{\text{ss}} \neq \varnothing$.

Proof. Suppose for the *if* part that strings $w_i \in L_i$ exist for $i = 1, 2, 3, 4$ such that for each $(i,j) \in I_d$, $w_i \preceq_{\text{ss}} w_j$. According to this, for each $i \in [4]$ there is an accepting state $f_i \in F_i$ such that $f_i \in \delta_i(s_i, w_i)$. Equivalently, there is a path between s_i and f_i in the graph-db G whose label is the string w_i. It follows from the construction of G that the string $\hat{w}_i = \#w\bar{\#}$ is the label of a path ρ_i between vertices v_{i-1} and v_i in G. In addition to this, for each $(i,j) \in I_d$, $w_i \preceq_{\text{ss}} w_j$, thus $\lambda(\rho_i) = \#w_i\bar{\#} \preceq_{\text{ss}} \#w_j\bar{\#} = \lambda(\rho_j)$. Consequently, $G \models Q(v_0, v_1, v_2, v_3, v_4)$.

For the *only if* part, suppose for each $i \in [4]$ there is a path ρ_i between vertices v_{i-1} and v_i in G, such that $\lambda(\rho_i) = \#w_i\bar{\#}$, with $w_i \in \Sigma^*$, and for each $(i,j) \in I_d$, $\lambda(\rho_i) \preceq_{\text{ss}} \lambda(\rho_j)$. It follows from Observation 5.2.6 that w_i is the label of a path between s_i and some accepting state $f_i \in F_i$, and consequently, that $w_i \in L_i$. Since $\lambda(\rho_i) \preceq_{\text{ss}} \lambda(\rho_j)$ means that $\#w_i\bar{\#} \preceq_{\text{ss}} \#w_j\bar{\#}$, and the $\#, \bar{\#}$ symbols cannot appear neither in w_i nor w_j, it follows that $w_i \preceq_{\text{ss}} w_j$. Consequently, $(w_1, w_2, w_3, w_4) \in L \bigcap_{I_d} \preceq_{\text{ss}}$. □

Finally, it is easy to see that G an the tuple $\bar{v}(v_0, v_1, v_2, v_3, v_4)$ can be constructed in polynomial time from the regular languages $\{L_1, L_2, L_3, L_4\}$, and these in turn are also constructed in polynomial time from $|w|$ according to the proof of Lemma 5.2.1. Since $L \bigcap_{I_d} \preceq_{\text{ss}} \neq \varnothing$ if and only if $w \in$ Square, this concludes the proof.

□

Chapter 6

Conclusions and Further Research

This work has been primarily focused in answering questions about the complexity of evaluating CRPQs comparing paths with rational relations of interest for their natural appearance in database applications: Subword, Suffix and Subsequence. The results obtained here complete those presented in Section 2.3 and are summarized in the following table.

Query	$S = \preceq_{\text{ss}}$	$S = \preceq_{\text{suff}}$	$S = \preceq_{\text{sw}}$
CRPQ(S)	NP−complete	NLogSpace	PSpace
CRPQ(S)	NExpTime	PSpace − complete	PSpace − complete

Figure 6.1: Upper bounds for data and combined complexity, respectively, of evaluating CRPQs comparing paths with Subword, Suffix and Subsequence relations.

With respect to the subsequence relation, it was proven that there are queries in CRPQ($\preceq_{\text{ss}}$) which are NP-hard to evaluate. In this sense, the NP evaluation algorithm previously provided in [9] is optimal. This means that this query language is unlikely to be practical.

Secondly, the combined complexity of evaluating CRPQs comparing paths with relations such as suffix and subword was proven to be PSpace-complete. The extent to which this holds is more general, including combinations of any relation expressible by word equations, such as prefix, commutativity, and other relations not recognizable by memory-limited sequential devices as automata.

In addition to this, the data complexity of evaluating CRPQs comparing paths with suffix was shown to be in NLogSpace, proving it to be practical, since it matches the complexity of standard query languages such as CRPQs and ECRPQs.

The means trough which these results were proven pointed out unanswered questions in the literature about word equations with regular constraints, namely, what is the complexity of solving them when the equation is fixed and the input to the problem are the regular constraints. The simplest case of this question was answered, showing that satisfiability is decidable in NLogSpace when the fixed equation has a finite number of minimal solutions.

This work leaves some open questions that will be studied in following research. These are:

- Is the NExpTime upper bound for the combined complexity of evaluating CRPQ ($\preceq_{\text{ss}}$) queries optimal?

  This would be a generalisation of the NP-completeness result obtained for its data complexity, and it would not be surprising, as this behaviour is common in database problems [55].

- What is the data complexity of evaluating CRPQ($\preceq_{\text{sw}}$) queries?

  An upper PSpace bound for it follows directly from its combined complexity. The equations obtained when working with instances for this problem relate to the equation $\{SXT = UXV\}$, asking for pairs of strings sharing a common subword. The minimal solutions for this equation are not finite, and so a different strategy needs to be used for studying this question.

- Are there fixed word equations PSpace-hard to solve with variable regular constraints?

  This relates to the preceding question. The minimal solutions for the equation $\{SXT = UXV\}$ can grow arbitrarily, presenting repeated patterns when the common subword overlaps. This feature hints that it may be possible to code the intersection of arbitrary regular languages in the solutions of this equation with the correct regular constraints.

Bibliography

[1] Habib Abdulrab and Jean-Pierre Pécuchet. Solving word equations. *Journal of Symbolic Computation*, 8(5):499–521, 1989.

[2] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L Wiener. The lorel query language for semistructured data. *International journal on digital libraries*, 1(1):68–88, 1997.

[3] Michael H. Albert and J Lawrence. A proof of ehrenfeucht's conjecture. *Theoretical Computer Science*, 41:121–123, 1985.

[4] Sihem Amer-Yahia, Laks Lakshmanan, and Cong Yu. Socialscope: Enabling information discovery on social content sites. *arXiv preprint arXiv:0909.2058*, 2009.

[5] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1, 2008.

[6] Kemafor Anyanwu and Amit Sheth. P-queries: enabling querying for semantic associations on the semantic web. In *Proceedings of the 12th international conference on World Wide Web*, pages 690–699. ACM, 2003.

[7] Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. An extension of sparql for rdfs. In *Semantic Web, Ontologies and Databases*, pages 1–20. Springer, 2008.

[8] P. Austrin. How hard is unshuffling a string (reply). *Theoretical Computer Science, http://cstheory. stackexchange. com/q/34 (version: 2010-12-01)*, 2010.

[9] Pablo Barcelo, Diego Figueira, and Leonid Libkin. Graph logics with rational relations and the generalized intersection problem. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, pages 115–124. IEEE Computer Society, 2012.

[10] Pablo Barcelo, Leonid Libkin, Anthony W Lin, and Peter T Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems (TODS)*, 37(4):31, 2012.

[11] Chris Barrett, Riko Jacob, and Madhav Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

[12] Jean Berstel. *Transductions and context-free languages*, volume 4. Teubner Stuttgart, 1979.

[13] Peter Buneman. Semistructured data. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 117–121. ACM, 1997.

[14] Sam Buss and Michael Soltys. Unshuffling a square is np-hard. *arXiv preprint arXiv:1211.7161*, 2012.

[15] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y Vardi. Containment of conjunctive regular path queries with inverse. In *KR*, pages 176–185, 2000.

[16] Ashok Chandra and David Harel. Structure and complexity of relational queries. *Journal of Computer and system Sciences*, 25(1):99–128, 1982.

[17] Christian Choffrut. Relations over words and logic: a chronology. 2006.

[18] Christian Choffrut and Juhani Karhumäki. Combinatorics of words. In *Handbook of formal languages*, pages 329–438. Springer, 1997.

[19] Mariano P Consens and Alberto O Mendelzon. Expressing structural hypertext queries in graphlog. In *Proceedings of the second annual ACM conference on Hypertext*, pages 269–292. ACM, 1989.

[20] Mariano P Consens and Alberto O Mendelzon. Graphlog: a visual formalism for real life recursion. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 404–416. ACM, 1990.

[21] Isabel F Cruz, Alberto O Mendelzon, and Peter T Wood. A graphical query language supporting recursion. In *ACM SIGMOD Record*, volume 16, pages 323–330. ACM, 1987.

[22] Alin Deutsch and Val Tannen. Optimization properties for classes of conjunctive regular path queries. In *Database Programming Languages*, pages 21–39. Springer, 2002.

[23] Anton Dries, Siegfried Nijssen, and Luc De Raedt. A query language for analyzing networks. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 485–494. ACM, 2009.

[24] J Erickson. How hard is unshuffling a string. *Theoretical Computer Science, http://cstheory. stackexchange. com/q/34 (version: 2010-12-01)*, 2010.

[25] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 139–148. ACM, 1998.

[26] Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin Rinard. What is decidable about strings? 2011.

[27] Seymour Ginsburg and Edwin H Spanier. Mappings of languages by two-tape devices. *Journal of the ACM (JACM)*, 12(3):423–434, 1965.

[28] Marc Gyssens, Jan Paredaens, and Dirk Van Gucht. A graph-oriented object database model. In *Proceedings of the ninth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 417–424. ACM, 1990.

[29] David Harel et al. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[30] Dane Henshall, Narad Rampersad, and Jeffrey Shallit. Shuffling and unshuffling. *arXiv preprint arXiv:1106.5767*, 2011.

[31] Hmelevskiĭ. Systems of equations in a free group. i.

[32] Lucian Ilie. Subwords and power-free words are not expressible by word equations. *Fundamenta Informaticae*, 38(1):109–118, 1999.

[33] Lucian Ilie and Wojciech Plandowski. Two-variable word equations. *RAIRO-Theoretical Informatics and Applications*, 34(06):467–501, 2000.

[34] Juhani Karhumäki and Wojciech Plandowski. On the expressibility of languages by word equations with a bounded number of variables. 2001.

[35] Juhani Karhumäki, Wojciech Plandowski, and Filippo Mignosi. *The expressibility of languages and relations by word equations.* Springer, 1997.

[36] Antoni Kościelski and Leszek Pacholski. Makanin's algorithm is not primitive recursive. *Theoretical Computer Science*, 191(1):145–156, 1998.

[37] Zoé Lacroix, Hyma Murthy, Felix Naumann, and Louiqa Raschid. Links and paths through life sciences data sources. In *Data Integration in the Life Sciences*, pages 203–211. Springer, 2004.

[38] Woei-Jyh Lee, Louiqa Raschid, Padmini Srinivasan, Nigam Shah, Daniel Rubin, and Natasha Noy. Using annotations from controlled vocabularies to find meaningful associations. In *Data Integration in the Life Sciences*, pages 247–263. Springer, 2007.

[39] André Lentin. *Equations dans les monoïdes libres*, volume 16. Mouton, 1972.

[40] Ulf Leser. A query language for biological networks. *Bioinformatics*, 21(suppl 2):ii33–ii39, 2005.

[41] M_ Lothaire. *Combinatorics on words.* Cambridge University Press, 1997.

[42] Gennady S Makanin. The problem of solvability of equations in a free semigroup. *Sbornik: Mathematics*, 32(2):129–198, 1977.

[43] Anthony Mansfield. An algorithm for a merge recognition problem. *Discrete Applied Mathematics*, 4(3):193–197, 1982.

[44] Tova Milo and Dan Suciu. Index structures for path expressions. In *Database Theory—ICDT'99*, pages 277–295. Springer, 1999.

[45] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. nsparql: A navigational language for rdf. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):255–270, 2010.

[46] Wojciech Plandowski. Satisfiability of word equations with constants is in pspace. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 495–500. IEEE, 1999.

[47] William E Riddle. A method for the description and analysis of complex software systems. In *ACM SIGPLAN Notices*, volume 8, pages 133–136. ACM, 1973.

[48] William E Riddle. An approach to software system modelling and analysis. *Computer Languages*, 4(1):49–66, 1979.

[49] Romeo Rizzi and Stéphane Vialette. On recognizing words that are squares for the shufle product. In *Computer Science–Theory and Applications*, pages 235–245. Springer, 2013.

[50] Royi Ronen and Oded Shmueli. Soql: A language for querying and creating data in social networks. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1595–1602. IEEE, 2009.

[51] Mauro San Martın, Claudio Gutierrez, and Peter T Wood. Snql: A social networks query and transformation language. *cities*, 5:r5, 2011.

[52] Klaus U Schulz. Makanin's algorithm for word equations-two improvements and a generalization. In *Word equations and related topics*, pages 85–150. Springer, 1992.

[53] Alan C. Shaw. Software descriptions with flow expressions. *Software Engineering, IEEE Transactions on*, (3):242–254, 1978.

[54] Frank Wm Tompa. A data model for flexible hypertext database systems. *ACM Transactions on Information Systems (TOIS)*, 7(1):85–100, 1989.

[55] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 137–146. ACM, 1982.

[56] Manfred K Warmuth and David Haussler. On the complexity of iterated shuffle. *Journal of Computer and System Sciences*, 28(3):345–358, 1984.

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