Synchronous relations and complexity of query evaluation

Varun Ramanathan

LABORATOIRE BORDELAIS DE RECHERCHE EN INFORMATIQUE

THÈSE PRÉSENTÉE

POUR OBTENIR LE GRADE DE

DOCTEUR
DE L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE

SPÉCIALITÉ INFORMATIQUE

Par **Varun RAMANATHAN**

Synchronous relations and complexity of query evaluation

Sous la direction de : **Pascal WEIL** et **Diego FIGUEIRA**

Soutenue le **17 Mai 2022** devant un jury composé de

Mme S. N. KRISHNA	Professeure	Indian Institute of Technology, Bombay	Rapporteure
M. Jean-Marc TALBOT	Professeur	Aix-Marseille Université	Rapporteur
Mme Meghyn BIENVENU	Directrice de Recherche	CNRS, LaBRI	Examinatrice
Mme Claire DAVID	Maître de Conférences	Université Paris-Est Marne-la-Vallée	Examinatrice
M. Pascal WEIL	Directeur de Recherche	CNRS, LaBRI, Université de Bordeaux	Co-directeur
M. Diego FIGUEIRA	Chargé de Recherche	CNRS, LaBRI, Université de Bordeaux	Co-directeur

Contents

Acknowledgements

I have had the privilege and good fortune of being acquainted with many wonderful people in my life, whose knowledge, experience, and understanding helped me immensely along my journey. Though I know I will never be able to thank them enough, I wish to write a few words about some of them.

It was my parents who fostered my curiosity about scientific topics. They encouraged me to pursue higher studies in mathematics and computer science. My parents have never stopped believing in me. Their words of encouragement kept me going through tough times. I thank them for their guidance, advice, love, and support. I also thank my extended family, especially my relatives who are scientists and engineers. They have ignited a passion for learning about the natural world in my heart.

During middle and high school, I was taught by excellent teachers who stimulated my interest in science, mathematics, and literature. I express my deep reverence and gratitude to them.

I owe a huge debt of gratitude to Pascal Weil, my Ph.D. co-advisor. I want to thank him for the internship opportunities at LaBRI, which gave me my first experience in research. He has been a wonderful host for the duration of my stay in France, always making sure that I have no administrative or academic hassles. His keen eye for the written word has been immensely valuable in sharpening my academic language. During the course of our work together, he has allowed me to freely pursue my ideas and encouraged clarity of thought above all else.

I would also like to express my heartfelt gratitude to my co-advisor Diego Figueira. I was first acquainted with him during my second internship at LaBRI, which formed the basis for my Masters' project. His ideas and dynamism have been a source of inspiration for me. We have had many interesting technical discussions which I will never forget. I could not have asked for better advisors than Pascal and Diego, both academically and personally. I am honored and privileged that I got the opportunity to work with them.

I am proud to call myself an alumnus of Chennai Mathematical Institute, where I was educated by a set of highly distinguished professors in the field of mathematics and computer science. Despite their high academic caliber, they taught their subjects clearly and effectively, ensuring that beginners like me could understand these topics.

In the math department, I would like to thank professors BV Rao and Shiva Shankar for their excellent foundational courses in real analysis and algebra respectively. In the computer science department, I would like to thank professors M Praveen, Aiswarya Cyriac, K Narayan Kumar, Madhavan Mukund, K V Subrahmanyan, Amaldev Manuel, and S P Suresh. Special thanks to professor R Ramanujam at IMSc for offering me an insightful reading course in axiomatic

set theory.

Finally, I thank my close friends Mohit, Shraiyance, and Mama for their reassuring virtual company during the difficult Covid lockdown period. Kind regards to my friends at LaBRI, including Govind, Sougata, Soumyajit, Tidiane, Sidoine, Attila, Pierre-Etienne, Karim (both of them), Rupayan, Meghna, and many more, for the special memories. Special thanks to Bala, Ahad, Thejaswini, Aalok, Pranshu, Theo, and Nathan for the interesting technical discussions. Your knowledge has enriched my mind.

Apologies to the scores of people whose names are not listed here. All of you have made me the person I am today, and I am privileged beyond measure to count you as well-wishers.

Then even nothingness was not, nor existence.
There was no air then, nor the heavens beyond it.
What covered it? Where was it? In whose keeping?
Was there then cosmic water, in depths unfathomed?

But after all, who knows, and who can say,
Whence it all came, and how creation happened?
The gods themselves are later than creation,
so who truly knows whence it has arisen?

Whence all creation had its origin,
He, whether He fashioned or whether He did not,
He, who surveys it all from the highest heaven,
He knows, or maybe even He does not know.

–Excerpt from *Nasadiya Sukta*,
an ancient hymn from the *Rig Veda*

Technical abstract

Over the past few decades, several models of databases have emerged to address commercial and consumer needs. Data models such as relational databases, graph databases, and object-oriented databases each have their own database management systems as well as associated query formalisms. An age-old problem in database theory is that of query evaluation, which, given an input pair of query and database, asks for the set of answers of the query evaluated on the database. Query evaluation has been studied with respect to various measures of complexity; the *combined complexity* takes as input the query and the database, while the *data complexity* treats the query size as fixed and the database as the input [117].

In this thesis, we focus on studying path queries on graph databases. Broadly speaking, a path query operates on an edge-labeled graph – we call the labels *letters*. Thus, a path between two nodes in a labeled graph is labeled by the *word* formed by concatenating the edge labels along those paths. A path query, then, is a condition on a labeled graph that describes a set of nodes connected by paths whose labels belong to certain sets, called *languages*. Among the various sub-formalisms of path queries that we consider, we only use subsets of the class of *regular languages* – these are languages specifying patterns in words defined by simple machines called *finite automata*. Regular conditions on path labels can be generalized to *tuples* of paths. In this case, the query specifies that the resulting tuple of labels belongs to a *relation* over words. A language is a special case of a relation, of *arity* 1. We study two topics in this thesis: (i) the query evaluation problem for the formalism of path queries, and (ii) a language-theoretic and logical characterization of the relations used in them.

Relational databases are modeled mathematically using tuples of elements, sets, and relations (which are sets of tuples). This data model was introduced by E.F. Codd in 1970 [33]. Under set semantics, query evaluation and other database operations take on the form of algebraic operators on relations (relational algebra) or standard set-theoretic transformations (relational calculus). Associated with these are Conjunctive Queries (CQs), a well-known class of queries that are modeled as first order formulæ with (i) free as well as existentially quantified variables (like $\exists x_1 \exists x_2 \dots$), representing elements in the database, and (ii) relation symbols (like $R(y_1, \dots, y_n)$), representing relations in the database signature.

Evaluation of CQs was shown to be NP-complete by Chandra and Merlin [28]. Yannakakis suggested that *parameterized complexity* (where query size is parameterized and the database is the input) might be a better measure to study the evaluation problem [120]. However, Papadimitriou and Yannakakis showed that the parameterized complexity of CQ evaluation is W[1]-hard [91], which

is the parameterized analogue of NP-hardness. Naturally, the question arises as to which subclasses of CQs might have tractable evaluation. An early result of Yannakakis shows that acyclic CQs can be evaluated in PTIME [121]. This relies on the notion of *shape* of a query, which is given by its Gaifman graph or Gaifman abstraction. The *tree-width* of a query refers to the tree-width of its Gaifman graph; Chekuri and Rajaraman showed that bounded tree-width queries admit tractable evaluation [29]. An equivalent formalism of this result was given by Kolaitis and Vardi [70], and it was made more precise by Gottlob et al. who showed that for each fixed tree-width k, the evaluation of CQs of tree-width k is LOGCFL-complete [54]. Finally, the converse to Chekuri and Rajaraman's result was given by Grohe et al., who showed that a subclass of CQs abstracted by some class of graphs $\mathcal{C}$ is tractable if, and only if $\mathcal{C}$ has bounded tree-width [57].

Following the relational model, several other models were developed to address a wide variety of data modeling challenges. In this thesis, we study a prominent data model called the graph data model, which employs databases called graph databases. These databases are used in contexts where the interrelationships between real-world objects are prioritized and queried at the same level as those objects themselves. Mathematically, these databases are abstracted as labeled graphs. Simply put, a graph database is a graph $D = (G, \eta)$, where $G = (V, E)$ is a graph and $\eta\colon E \to \mathbb{A}$ is a labeling function which assigns to every edge e a label $\eta(e)$ belonging to a finite alphabet $\mathbb{A}$. The most ubiquitous query formalism on these databases is that of a *path query*, which specifies a reachability condition on nodes and paths along with an additional condition on the path labels. The simplest kind of path queries are called *regular path queries*, or RPQs. An RPQ is of the form

$$x \xrightarrow{\pi} y \wedge L(\pi)$$

which is interpreted as "find all pairs of nodes (x, y) such that there is a path π from x to y whose label satisfies some regular expression L". Conjunctions of RPQs, called CRPQs, constitute an important formalism that finds many applications associated with the Semantic Web [60, 93], biological networks [75, 86] and social networks [101, 102]. In practice, however, it is found that a statistical majority of CRPQs do not utilize the full power of regular expressions, instead relying on a small fraction of them in their usage [18]. These subclasses of regular languages correspond to "small fragments" of the first-order logic FO($<$) which characterizes the set of aperiodic languages [84].

CRPQs have been extended by allowing path labels to conform to relations as opposed to languages. These *extended* CRPQ, abbreviated ECRPQ, use *synchronous relations*, a class of word relations which is analogous to regular languages in higher dimensions. A simple ECRPQ (which uses only binary relations) is of the form

$$x_1 \xrightarrow{\pi_1} y_1 \wedge x_2 \xrightarrow{\pi_2} y_2 \wedge R(\pi_1, \pi_2)$$

interpreted as "find all tuples (x_1, y_1, x_2, y_2) such that there are paths π_1, π_2 connecting x_1 to y_1 and x_2 and y_2 respectively, and the labels on π_1, π_2 conform to the synchronous relation R" (for example, they have the same length).

While ECRPQs enjoy greater expressive power than CRPQs, they are also significantly harder to evaluate. The combined complexity of CRPQ evaluation

is NP-complete, and that of ECRPQs is PSPACE-complete, which was proved by Libkin et al. [15]. In the present work, we study the ECRPQ evaluation problem in the same spirit as was done for CRPQ and CQ evaluation. In other words, our goal is to find a graph-based abstraction for ECRPQs, similar to the Gaifman graph abstraction for CQs, and introduce measures on it (similar to the tree-width measure for CQs).

The resulting abstraction is called a *two-level-multi-hypergraph* or a *2L-graph*; the first level abstracts the reachability subquery (nodes and their connecting paths) using a multi-graph, and the second level abstracts the relational subquery (relations on the paths) using a multi-hypergraph. In addition to the tree-width measure on the first level graph (which plays a similar role to CQ evaluation), we introduce connectivity measures on the second level hypergraph. By putting a bound on these measures, we obtain a theorem similar to CQ evaluation, where we give precise conditions on ECRPQ subclasses such that their evaluation is in PSPACE, NP or PTIME. Our results generalize prior results that were only known for CRPQs and CQs.

The second part of this thesis is devoted to studying synchronous relations in greater depth. In particular, we consider a result of Eilenberg, Elgot, and Shepherdson [43], which gives a logical characterization of synchronous relations. This is the first order theory of finite words with the prefix, equal length, and last letter predicates, which we denote by $\mathrm{FO}[\sigma]$. Recognized by finite multi-tape automata with synchronous movement of heads, this class of relations has robust closure properties that are reflected in its first order characterization. We study the quantifier alternation hierarchy of this logic, which is a standard way to stratify first order formulæ into classes defined by their syntactic complexity.

An FO formula written in first order logic over any signature can be re-written in its *prenex normal form*, which is produced by moving all its quantifiers to the left-hand side, leaving a quantifier-free formula to the right. The quantifiers form alternating blocks of $\exists^*$ and $\forall^*$ respectively, and when a formula admits a prenex normal form with at most i alternating blocks beginning with a (possibly empty) $\exists^*$ block, we call it a $\mathbf{\Sigma}_i$ formula (with $\mathbf{\Pi}_i$ being the set of negations of $\mathbf{\Sigma}_i$). In the case of $\mathrm{FO}(<)$, the first order logic characterizing aperiodic languages, the lower levels of the hierarchy $\mathbf{\Sigma}_1(<), \mathbf{\Sigma}_2(<), \mathbf{\Sigma}_3(<)$ as well as the Boolean closures $\mathcal{B}\mathbf{\Sigma}_1(<)$ and $\mathcal{B}\mathbf{\Sigma}_2(<)$ correspond to classes of regular languages that have interesting combinatorial and algebraic properties. Moreover, the membership problem – which asks whether a given regular language is definable by a formula in a subset $\mathcal{F}$ of $\mathrm{FO}(<)$ – is decidable for these lower levels. It remains open for higher levels.

Coming back to synchronous relations, a closer look at the proof of Eilenberg et al's theorem reveals that every synchronous relation is definable in $\mathbf{\Sigma}_3$, the third level in the first order quantifier alternation hierarchy. Thus the expressive power of $\mathrm{FO}[\sigma]$ collapses to $\mathbf{\Sigma}_3[\sigma]$. The natural question that arises is: what are the relations definable in the lower levels $\mathbf{\Sigma}_1[\sigma]$ and $\mathbf{\Sigma}_2[\sigma]$, as well as their Boolean closures $\mathcal{B}\mathbf{\Sigma}_1[\sigma], \mathcal{B}\mathbf{\Sigma}_2[\sigma]$? Moreover, we investigate the relational membership problem – given a synchronous relation R, is it decidable to check whether R is definable by a formula in $\mathcal{F}$, where $\mathcal{F} \in \{\mathbf{\Sigma}_1[\sigma], \mathcal{B}\mathbf{\Sigma}_1[\sigma], \mathbf{\Sigma}_2[\sigma], \mathcal{B}\mathbf{\Sigma}_2[\sigma]\}$?

In this thesis, we provide an effective characterization of $\mathbf{\Sigma}_1[\sigma]$-definable relations that parallels the characterization of $\mathbf{\Sigma}_1(<)$-definable languages in its description; the latter relies on the *subword* relation while the former uses *syn-*

chronized subword, an extension of subword that we introduce specifically to characterize these relations. This parallel extends to the characterization of $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable relations and $\mathcal{B}\mathbf{\Sigma}_1(<)$-definable languages as well. The second levels $\mathbf{\Sigma}_2[\sigma]$ and $\mathbf{\Sigma}_2(<)$ are linked even more closely: we show that a relation is $\mathbf{\Sigma}_2[\sigma]$-definable if and only if its synchronized language is $\mathbf{\Sigma}_2(<)$-definable. This result directly extends to $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$-definable relations as well. Thus we obtain the decidability of the relational membership problem for every level of the FO$[\sigma]$ quantifier alternation hierarchy.

Chapter 1

Introduction

1.1 Technical preliminaries

1.1.1 Notation

Let S be any set. For any $k \in \mathbb{N}$, we denote by S^k the set of all length k sequences of elements in S. S^0 is defined to be $\{\varepsilon\}$, where ε is the empty sequence. We define the set

$$S^* = \bigcup_{k=0}^{\infty} S^k$$

We denote by $\mathscr{P}(S)$ the powerset of S, and for any $n \in \mathbb{N}$ we denote by $\mathscr{P}_n(S)$ the set of elements of $\mathscr{P}(S)$ of size n. An *alphabet* $\mathbb{A}$ is a finite set comprising of elements $\{a_1, \ldots, a_m\}$ called *letters*. A *word* w over $\mathbb{A}$ is any element in $\mathbb{A}^*$, written as $w = a_1 a_2 \ldots a_n$, where $n \geq 0$ (with the empty word ε having length 0 since it contains no letters). We denote by $|w|$ the length of w, and for all non-empty w and $1 \leq i \leq |w|$ we denote by $w[i]$ the letter at position i in w. A *language* L over $\mathbb{A}$ is any subset of $\mathbb{A}^*$.

1.1.2 Complexity classes

A *decision problem* is a function that takes an input (in some domain) and returns a YES or NO value as the output. A good example is SAT, which takes as input a Boolean formula and decides whether it is satisfiable, or in other words, there exists some assignment of variables that makes the formula evaluate to TRUE. SAT is known to be NP-complete (see definition below).

The complexity of a decision problem yields a measure of its computational hardness by placing it in the hierarchy of complexity classes. Several decision problems related to queries are studied in the field of database theory, including query evaluation, query containment, and query satisfiability. In our study of the query evaluation problems, we deal with the standard complexity classes PTIME, NP, PSPACE. Additionally, we recollect some definitions from the world of parameterized complexity.

A problem is said to be in *polynomial time* (or PTIME), if there exists some $k \in \mathbb{N}$ such that a Turing machine decides it using $O(n^k)$ (of the order of n^k) units of time, where n is the size of the input. Similarly, we say that a problem

is in *polynomial space* (or PSPACE), if there exists some $k \in \mathbb{N}$ such that a Turing machine decides it using $O(n^k)$ units of memory in its execution.

By NP we refer to the class of algorithms/problems which are solvable using a non-deterministic Turing machine in running time $O(n^k)$, for some $k \in \mathbb{N}$. By NPSPACE we refer to the class of algorithms/problems which are solvable using a non-deterministic Turing machine that uses $O(n^k)$ units of space, for some $k \in \mathbb{N}$. It is a well-known result that NPSPACE is equivalent to PSPACE (Savitch, 1970 [103]).

Given decision problems P, P', a *polynomial time (Karp) reduction* from P to P' refers to an algorithm that takes as input an instance x of P and produces in polynomial time an "*equivalent*" instance x' of P' – that is, $x \in P$ if and only if $x' \in P'$. A problem P' is said to be NP-complete if $P' \in$ NP and for every problem P $\in$ NP there exists a polynomial time reduction from P to P'. Similarly, a problem P' is said to be PSPACE-complete if $P' \in$ PSPACE and for every problem P $\in$ PSPACE there exists a polynomial time reduction from P to P'.

A *parameterized problem* is a special kind of decision problem which takes an input along with a *parameter*, which is typically an integer. The complexity of a parameterized problem, called its *parameterized complexity*, is evaluated in terms of the input as well as its parameter. A parameterized problem is said to be *fixed parameter tractable* (FPT for short) if and only if there exists a computable function f and a constant c such that every instance $(x, k) \in \mathbb{A}^* \times \mathbb{N}$ of the problem can be solved in time $f(k) \cdot |x|^c$.

Let P, P' be parameterized problems. A (many-to-one) FPT reduction from P to P' (denoted by $P \leq^{\text{fpt}} P'$) is an algorithm along with computable functions $f, g \colon \mathbb{N} \to \mathbb{N}$ and a constant c, which, given an instance (x, k) of P, computes an instance (x', k') of P' such that

- $(x, k) \in P$ if and only if $(x', k') \in P'$
- $k' \leq g(k)$
- The algorithm runs in time $f(k) \cdot |x|^c$

We also consider the complexity class W[1] which is the parameterized analogue of NP. It contains a lot of natural computational problems. Although this class is originally defined using Boolean circuits, we define it here using the Parameterized Independent Set problem. A detailed treatment of parameterized complexity classes can be found in the textbook of Downey and Fellows [41].

In graph theory, an *independent set* refers to a set of nodes in a graph such that no two nodes in the set are connected by an edge. A *clique* refers to a set of graph nodes such that any two elements in the set are connected by an edge. Now, PARAMETERIZED INDEPENDENT SET (P-IS) is the parameterized problem of checking whether an input graph contains an independent set of size (parameter) k. We define the class W[1] to be the set of all parameterized problems that are fixed-parameter reducible to P-IS.

Another W[1]-complete problem is deciding whether a given graph contains a clique of size (parameter) k. In this thesis, we show that for a specific subclass of path queries on graph databases, the parameterized query evaluation problem is W[1]-complete (see Chapter 4: Propositions 2 and 3, Corollary 1 and (2) of

Theorem 2).

Finally, we recollect that the parameterized complexity class XNL, introduced in [40] and studied in [31] as [Uniform-XNL]$^{\text{FPT}}$, is the closure under FPT-reductions of the class of parameterized problems P such that there exists a computable function that assigns to each parameter k an NL algorithm for the decision problem

$$\{x \colon (x, k) \in P\}$$

Parameterized Intersection Emptiness (p-IE) is an example of a XNL-complete problem (see Lemma 8 in Chapter 4) [118].

For the above mentioned parameterized complexity classes we have

$$\text{FPT} \subseteq \text{W[1]} \subseteq \text{XNL}$$

with the containments conjectured to be strict.

1.2 Thesis overview

We live in times of unprecedented technological progress. Each day civilization benefits more and more from the development of artificial intelligence. Energy, transportation, and logistics rely crucially on a network of automated systems, hardware devices, and Internet servers running simultaneously to ensure our cities and countries keep running productively. 5G, satellite internet, and other more efficient incarnations of the Internet are evolving as we speak, providing greater accessibility to millions more every day. This influx of new users, amplified by the burgeoning mobile internet industry, brings with it a massive outpouring of data. The collection, processing, maintenance, and analysis of this data poses a continuous challenge to programmers and computer scientists.

Machine learning and data mining are disciplines that have traditionally dealt with classifying, analyzing, and generating useful knowledge from a collection of data points. To enable the meaningful classification of any data, it has to be arranged and stored in a database of some form. Simply put, a database is a set of data entries equipped with a set of operations. The most commonly used operations are searching (querying), and the option to update or modify the data in a particular entry.

Example 1. Consider an online dictionary comprised of a set of words and their definitions. The dictionary allows a "search" feature, by which one can enter a word w and obtain a list of all words which have w as their prefix, as well as their synonyms. Observe that this is exactly how we look up words in a regular dictionary, by prefix-searching our required word letter-by-letter. ■

For commercial applications, data is stored and queried in software called *database management systems* (DBMSs). Users can create database *instances* using a DBMS, and various operations like querying, updating, and sorting can be carried out on the data entries in an instance. DBMSs were in development as early as 1973 [1]. Since then, their ubiquity has grown. OracleDB, MySQL, and Neo4j are some well-known DBMSs in use today.

[1] see System R and INGRES

When modeling data in a real-world scenario, a one-size-fits-all approach does not work. Several *data models* exist to address the various needs of commercial and technological applications. These models can be distinguished based on the data structures and querying mechanisms they allow. For example, in the context of administration, relational DBMSs are used in offices, banks, and universities because the raw data needs to be processed or viewed as tables. Therefore it is convenient to model these tables as relations containing tuples of objects, and relational DBMSs like Oracle DB or MySQL are well-suited for this purpose. When dealing with a large number of highly interconnected data values, the use of graph databases might be more appropriate. Thus spatial and engineering applications may need a graph DMBS like Neo4j compared to, say, a relational DBMS.

The graph data model represents data via graph-related structures. The popularity of this model peaked in the early 90s before fading away with the appearance of XML, semantic, and object-oriented databases. However, graph databases have made a resurgence in the modern era in applications related to social networks, big-data and computational biology. We use labeled graphs as a theoretical representation of graph databases. We discuss these applications in more detail in Section 2.3 of Chapter 2. We discuss various data models in Section 1.3 of this chapter.

An important feature of DBMSs is the ability to *query* database instances. A *query* is a command or instruction which fetches a set of *answers* when it is *evaluated* on a database instance. These queries are written in special languages called *query languages*. However, here we do not deal with query languages directly. Instead, we deal with mathematical representations of these queries; these are given as first-order logic formulæ using free variables and relation symbols to specify the querying conditions. We discuss these *query formalisms* in more detail in Section 1.4 of this chapter.

In this thesis, we study the *query evaluation problem* corresponding to a family of graph database querying formalisms called *path queries*. The essential function of a path query is to extract a set of nodes and paths that are connected by a reachability condition, such that the labels on the paths conform to a given pattern. Every atom in this reachability condition is of the form

$$x \xrightarrow{\pi} y$$

which states that a path π connects node x to node y. *Regular path queries* (RPQs) restrict these reachability conditions to paths whose labels conform to a given regular expression (see Definition 1). Allowing conjunctions of RPQ atoms with shared node variables produces an expressive formalism analogous to Conjunctive Queries (CQs) over relational databases. In fact, these *conjunctive regular path queries* (CRPQs) have good expressive power while maintaining the computational complexity of CQ evaluation. CRPQs have been further extended by adding relational conditions on path labels, that is, only allowing tuples of paths whose labels conform to a given word relation. In order to guarantee good algorithmic properties, we restrict our attention to the extension of CRPQs that use so-called *synchronous relations*. These relations generalize regular languages in higher dimensions as they are recognized by multi-tape finite automata with synchronous movement of tape heads [17]. We define synchronous relations and illustrate their properties in Section 1.5.5 of this chapter.

CRPQs extended with synchronous relations produce a query formalism called *extended conjunctive regular path queries*, or ECRPQs [15]. This thesis studies the query evaluation problem for the ECRPQ formalism. We define CQs and path queries in Sections 2.2.1 and 2.3.2 of Chapter 2 respectively. A common theme in the study of CQs and other first-order logic-based formalisms is to characterize the evaluation problem based on the underlying structure of queries. This structure or 'shape' of a query is often defined in terms of the Gaifman graph of its formula. Restricting queries by their shape produces subclasses of queries whose evaluation is tractable. The paradigmatic example is the tree-width of the Gaifman graph of a CQ; the complexity of CQ evaluation falls from NP to PTIME when restricted to CQs of bounded tree-width, a result which is true for CRPQs as well [58]. The general ECRPQ evaluation problem is known to be PSPACE-complete [15]. Naturally, we may wonder about the underlying structure of an ECRPQ. Can such a result – which rigorously specifies the kind of queries that are evaluated in lower complexity – be given for ECRPQs as well? This thesis tackles these questions and answers them in the affirmative.

While restricting path queries by their shape proves to be beneficial for lowering the complexity of their evaluation, restricting them by only allowing a small subset of regular expressions (as opposed to the full power of regular languages) is useful in real-world scenarios. Consider the implementation of CRPQs in SPARQL, a widely used graph query language. An analysis of SPARQL query logs over Wikidata archives [18] revealed that among organic[2] queries, 35% use regular languages of the form AB^* and 25% use A^* (where A, B are finite alphabets). Among robotic queries, the same analysis revealed that 67 % of them used the regular languages a^*, a^+ and ab^* (where a, b are letters). The languages used in the majority of robotic and organic queries are *polynomials*, which have the general form

$$\bigcup A_1^* a_1 A_2^* a_2 \dots A_n^* a_n$$

Polynomials constitute a small subclass of aperiodic regular languages. They are logically characterized by the fragment $\mathbf{\Sigma}_2(<)$, the second level of the quantifier alternation hierarchy of FO($<$) on finite words. We formally define this hierarchy in Chapter 5 (see Definition 5.2.3). The significant presence of polynomials in SPARQL instances necessitates a deeper study of languages belonging to "small" fragments.

Anticipating that the phenomenon of less expressive classes of languages being overrepresented in real-life use-cases holds for ECRPQs as well, we broaden our focus from languages to relations. In particular, we are interested in finding "small" subclasses of synchronous relations, analogous to polynomials within regular languages. We ask if there are subclasses of synchronous relations that have desirable theoretical properties and are defined by small fragments of some logical structure. Synchronous relations are not captured by any extension of FO($<$). Therefore we must turn to other logics for finding relational equivalents of polynomials. Here we consider a paper by Eilenberg, Elgot, and Shepherdson that introduces a logic that characterizes synchronous relations [43].

[2]organic queries are asked by human users, while robotic queries are data crawls generated by scripts or bots (see [82])

Over any finite alphabet $\mathbb{A}$, consider the first order theory of $\mathbb{A}^*$ words equipped with a set of predicates σ containing prefix ($\preceq$), equal length eq and last letter $\{\ell_a\}_{a\in\mathbb{A}}$. We denote this logic by FO[σ] here. A simple formula in this logic, over the free variables x, y, z, is

$$\forall x'((x' \preceq x) \wedge (x' \preceq y)) \iff (x' \preceq z)$$

Note that this formula naturally defines the relation

$$\{(x, y, z) \in (\mathbb{A}^*)^3 \colon z \text{ is the longest common prefix of } x \text{ and } y\}$$

In fact, up to permutation of the entries in a tuple, each formula with k free variables defines a k-ary relation. We define the syntax and semantics of this logic formally in Section 5.3 of Chapter 5 (see Definition 31).

Eilenberg et al. showed that a relation is definable in FO[σ] if and only if it is synchronous [43]. To obtain the polynomial-like "small fragments" of synchronous relations, we exploit the quantifier alternation hierarchy of this logic. As a result, we obtain certain well-behaved subsets of relations that correspond to the lower levels of this hierarchy. In fact, we prove that FO[σ] is equal in expressive power to $\mathbf{\Sigma}_3[\sigma]$, the third level in the hierarchy. Moreover, we characterize the relations definable in all levels of this hierarchy and establish a profound link with the regular languages definable in the lower levels of the quantifier alternation hierarchy of FO($<$).

Organization. We continue this introduction with a brief look at some data models in Section 1.3. Following that, in Section 1.4, we introduce the query evaluation problem for querying formalisms on relational databases and graph databases. Then, in Section 1.5, we define regular languages, finite automata, and word relations definable using automata. We state our research goals in Section 1.6. Finally, the plan of the thesis is outlined in Section 1.7, which contains details on the contents of each subsequent chapter.

1.3 A history of data models

A *database model*, or simply data model, is comprised of a set of conceptual tools used to model entities in the real world and their interrelationships [108]. Notable data models include the relational model, the graph model, semantic and object-oriented databases, and semi-structured databases. To understand the evolution of database theory, let us go back in time to the early '60s and look at how the earliest computers operated.

Computers sequentially carried out instructions on input stored in punch cards and magnetic tapes. The first databases in the world used sequential file technology. Each record in the file was associated with a primary key: a unique identifier like social security number or purchase order number. Searching was carried out as follows: to find a record, run the whole file through the core memory until a record with a matching or higher key value is found. Although sequential file technology was suitable for most tasks in its time, its drawbacks were apparent: (i) spontaneous record retrieval[3] was impossible because each

[3] i.e. random access

search operation required the computer to run the whole file again, and (ii) primary data keys had to be unique IDs, so natural attributes like names of people or places could not be used.

Physical models. Two early models for managing large collections of data were the *hierarchical database model* (Tsichritzis and Lochovsky, 1976) [116] and the *network model* (Taylor and Frank, 1976) [113]. They were called physical models, because they directly operated on the level of data records and pointers, making it hard to distinguish the data model from its implementation. Further, database navigation was done through low-level operations on the records.

As storage technology evolved, magnetic tapes gave way to magnetic disks. These removed the limitations of sequential access and allowed programmers to mitigate the problems of sequential file technology. This new kind of database was called a *navigational database*, a term coined by Charles Bachman in his seminal paper that introduced this data model [12] (originally published in 1973). Bachman was given the Turing award for his outstanding contributions to database theory. He had also created Integrated Data Store (IDS), one of the world's first navigational databases, developed for General Electric in the 1960s. Programmers using navigational databases, equipped with the techniques of randomized access (aka hashing) and index sequential access, could retrieve data more efficiently. Index sequential access allowed the programmer to extract a record using its primary key, which was more efficient than sifting through every single record in the input. This caused a shift in the mindset of data programmers: input data was no longer seen as being fed into the computer, but rather into the database. The database was seen as an independent entity that could be independently accessed by multiple programmers, updated, and queried whenever necessary. Data became independent from the computer applications that used it.

The need then emerged for commercial database management systems, that would not only handle data retrieval but also update the database whenever required. Programs running on navigational databases maintained *currency indicators*, which was a set of global variables, serving as the "current" pointer which moved back and forth in the database. Executing commands like [GET NEXT] and [GET PREVIOUS] would move this pointer to the desired record that was being queried. This imposed, either implicitly or explicitly, a procedural structure on navigational databases.

Ultimately, the procedural structure of navigational interfaces proved to be a limitation for the paradigm, as more declarative paradigms like SQL emerged in the early 80s. Arguably the most extensively used query language paradigm to date, Structured Query Language or SQL was developed for relational databases. The ubiquity of SQL and its variants has made it an indispensable part of any discussion on relational databases. The *relational model* was developed by E.F. Codd in an attempt to formalize tabular data collection via logical abstraction [33, 34]. This model separated the physical and logical representation of data, both in terms of storage and database operations.

Relational databases. The relational data model is one of the most widely used frameworks underlying database management systems, to date. Mathematically, a relational database consists of a set of tables that are represented using relations over sets of attributes. Each table consists of rows interlinking attributes via tuples, and each column represents a particular kind of attribute. The logical representation of data ensures that its access, retrieval, and manipulation are independent of changes in its physical storage mechanism. The relational model ensures that arbitrarily structured input data is represented and queried in a consistent fashion, making it highly suitable for the majority of commercial applications.

In the relational model, data manipulation and query evaluation rest on two equivalent formalisms: relational algebra and relational calculus. The former uses a set of operators which transform one or more input relations into an output relation, whereas the latter uses symbolic logic, in particular predicate calculus, in order to compute the results of queries.

The early 90s saw the emergence of new types of databases like spatial and engineering databases. Relational calculus was not sufficiently expressive to query these complex databases. A new approach was needed, which allowed for a simpler design, easier scalability to large clusters of machines, and greater control over availability[4]. Two models emerged in the eighties and nineties to address the limitations of the relational model: the semantic data model and the object-oriented data model. The former emerged because there was a need to improve database design by allowing a richer set of semantics. Relational databases of that time were efficient for certain tasks, but there were a lot of real-world models too complex for a relational model to be mapped onto.

Semantic databases. The *semantic data model* addressed this problem by assigning semantics, or meanings, to objects and relationships in the database. Some examples of this data model are the *entity-relationship model* (Chen, 1976) [30], *IFO* (Abiteboul and Hull, 1984) [3] and the eponymous Semantic Data Model (Hammer and McLeod, 1978) [64]. This data model is tied to the development of graph databases, as the structure generated by the interrelationships of the objects in a semantic database is fundamentally graph-based.

Another limitation of relational databases was the inability to handle data-intensive domains like knowledge bases and engineering applications. This necessitated the development of the *object-oriented data model*.

[4]Simply put, availability of a system is the degree with which it will reliably run at any arbitrary time. It is the extent to which a system is "battle-ready".

Object-oriented databases. Object-oriented databases (also known as O-O databases), used in computer graphics, CAD/CAM software, information retrieval, and various other applications, represent data as a collection of objects organized into classes, along with complex associated values and methods. This model also allows manipulation of local data values via methods, not unlike object-oriented programming languages. The object-oriented perspective of the world is, as one would expect, a set of complex objects possessing a certain set of states, with data interaction being done via method passing. This is in contrast with the graph database view of the world, where the domain is a network of objects and relations with an emphasis on their interconnections.

Moreover, the development of non-relational database paradigms like Not Only SQL (NoSQL) was underway. These query languages were designed to support SQL-like operations, while also allowing more complex algebraic operations like transitive closure to be carried out. NoSQL contained procedural elements within it, reminiscent of the navigational databases of the early 60s. Navigational interfaces found a re-emergence in problems relating to big-data and online networks via the development of graph databases.

Graph databases. The graph data model is characterized by the usage of graph-based structures containing nodes and edges equipped with properties. Graph databases are suited for applications involving real-world objects with complex interrelationships; edges are considered as "first-class citizens" in the graph model. Queries on graph databases take on the form of reachability conditions on nodes along with specifications of path properties. Query answering involves graph traversal, covering nodes relevant to the query and ignoring other parts of the graph. These databases are sometimes implemented using tables, which introduces an additional layer of abstraction via relational structures. Neo4j, Oracle Spatial and Graph, and OrientDB are some of the more popular graph database management software.

1.4 Querying formalisms

A *querying formalism* is a mathematical framework to represent queries. Studying queries within this formal framework allows us to quantify their expressive power and investigate several algorithmic problems on them, independent of the query language in which they are implemented. We deal with two broad query formalisms in this thesis. They are: (i) Conjunctive Queries (CQs) on relational databases and (ii) Path Queries (PQs) on graph databases, comprising of Regular Path Queries and their extensions.

Conjunctive Queries

Conjunctive Queries are widely used to query relational databases, owing to their succinct mathematical representation and efficient evaluation algorithms.

A CQ is of the form

$$\exists x_1 \exists x_2 \ldots \exists x_n \, R_1(\bar{x}_i) \wedge R_2(\bar{x}_2) \cdots \wedge R_m(\bar{x}_m)$$

where each $\bar{x}_i$ is a tuple of variables, some of them among $x_1, \ldots, x_n$. These are evaluated over instances of relational databases, with each relation symbol in the CQ being associated with a relation of elements in the database. We work with *set semantics*, treating relations as sets of *tuples* of elements of fixed size, called the *arity* of the relation. For example, the query "is there a person whose father is a teacher?" can be encoded as

$$\exists x \, \exists y \, \texttt{Father}(x, y) \wedge \texttt{Teacher}(y)$$

We formally define relational databases and CQs in Section 2.2 and Section 2.2.1 of Chapter 2 respectively.

Path Queries

We abstract graph databases with labeled graphs; these are graphs whose edges are labeled by letters in a given alphabet $\mathbb{A}$. As paths are sequences of edges, their labels are simply defined as the concatenation of labels of edges occurring along them. Therefore, paths are labeled by elements of $\mathbb{A}^*$. As stated in Section 1.2, a regular path query (RPQ) takes the form

$$x \xrightarrow{\pi} y \wedge L(\pi)$$

and searches for pairs of nodes (x, y) in a graph database such that there is a path π going from x to y whose label belongs to a regular language $L \subseteq \mathbb{A}^*$. A common extension to CQs and RPQs is the Conjunctive Regular Path Query (CRPQ) formalism. A CRPQ (without any quantified node variables) takes the form

$$x_1 \xrightarrow{\pi_1} y_1 \wedge \cdots \wedge x_m \xrightarrow{\pi_m} y_m \wedge L_1(\pi_1) \wedge \cdots \wedge L_m(\pi_m)$$

where $x_1, y_1, \ldots, x_m, y_m$ are (not necessarily pairwise distinct) node variables, $\pi_1, \ldots, \pi_m$ are pairwise distinct path variables, and $L_1, \ldots, L_m$ are regular languages. This is interpreted as follows:

Do there exist pairs of nodes $\{(x_i, y_i)\}_{1 \leq i \leq m}$ such that for every $i \in \{1, \ldots, m\}$, there exists a path π_i from x_i to y_i that is labeled by a word in L_i?

We formally define regular languages in Section 1.5 of this chapter, and CRPQs in Section 2.3.2 of Chapter 2 (see Definition 11). The CRPQ formalism can be further extended to produce ECRPQs, or extended CRPQs, which contain relational atoms of the form $R(\bar{\pi})$, where R is a synchronous relation over words and $\bar{\pi}$ is a tuple of path variables. These relational atoms are interpreted naturally – a tuple of paths $\bar{\pi}_i$ in a graph database satisfies a relational atom if its labels belong to the relation. Now, an ECRPQ (without any quantified node variables) takes the form

$$x_1 \xrightarrow{\pi_1} y_1 \wedge \cdots \wedge x_m \xrightarrow{\pi_m} y_m \wedge R_1(\bar{\pi}_1) \wedge \cdots \wedge R_m(\bar{\pi}_n)$$

which is interpreted as follows:

Do there exist triples (x_i, y_i, π_i) (for every $1 \leq i \leq m$) where π_i is a path going from x_i to y_i, such that for every $1 \leq j \leq n$, the tuple of labels of the paths $\bar{\pi}_j$ conform to the relation R_j?

In addition to regular languages, Section 1.5 of this chapter also contains the formal definition of synchronous relations. Meanwhile, ECRPQs are defined in 2.3.4 of Chapter 2 (see Definition 12). In the same section, we also re-define CRPQs as a subclass of ECRPQs (see Definition 13).

Algorithmic problems on querying formalisms

A query q evaluated on a database D returns a set of answers, denoted by $ans(q, D)$. Based on this formal structure, several algorithmic problems associated with query formalisms have been studied in the literature. The prominent ones are as follows:

- Query evaluation: Given a query q and a database D, the query evaluation problem involves the computation of the answer set $ans(q, D)$.
- Query satisfiability: Given a query q, the query satisfiability problem asks if there exists a database D such that q evaluated on D produces a non-empty set of answers (in other words, D *satisfies* q).
- Query containment: Given queries q_1, q_2 (belonging to the same formalism and operating over the same set of variables), we say that q_1 is contained in q_2 (denoted by $q_1 \subseteq q_2$) if for every database D, $ans(q_1, D) \subseteq ans(q_2, D)$. The query containment problem for a given input (q_1, q_2) asks whether $q_1 \subseteq q_2$.

We focus on the query evaluation problem in this thesis. The query satisfiability problem is a special case of the query evaluation problem since a query is satisfiable if and only if it can be evaluated on the trivial database. Although studying the query containment problem is beyond the scope of the present work, we include a discussion on it in Section 4.4.2 of Chapter 4. Now, given a query formalism $\mathcal{Q}$, we formally state the decision problem underlying query evaluation as follows:

QUERY EVALUATION (EVAL-($\mathcal{Q}$)); $\mathcal{Q}$ is a query formalism	
Input:	A query $q \in \mathcal{Q}$, a database D, and a tuple $\bar{v}$ of database elements.
Question:	Does $\bar{v}$ belong to $ans(q, D)$?

The complexity of query evaluation, especially in the context of CQs and CRPQs, has been studied with respect to four measures. If all the node variables of a query q are quantified, then it is said to be a *Boolean query*. We study the complexity of the query evaluation problem under varying conditions on what is considered to be an input or parameter.

The complexity of query evaluation when the input is the query and database pair (q, D), is referred to as *combined complexity*. Using this measure of complexity, EVAL-CQ and EVAL-CRPQ are NP-complete [28] problems and EVAL-ECRPQ is a PSPACE-complete problem [15].

The combined complexity serves well as a general theoretical measure for query evaluation. However, in practice, it is observed that arbitrarily large databases are often queried with "short" queries whose size is minuscule compared to the size of the database they are evaluated on. This motivates the study of the *data complexity* of the algorithm, where the size of the query is taken to be a constant, and the input is the database. A related measure is the *parameterized complexity*, where the query size is the parameter and the input is, again, the database. It is known that EVAL-CQ, and consequently EVAL-CRPQ, is W[1]-complete in parameterized complexity [58] (see Propositions 2, 3 and Corollary 1).

Additionally, by *query complexity* we mean the complexity measure when the database size is considered to be fixed. It is known that EVAL-CQ and EVAL-CRPQ are both NP-complete in query complexity; however, the data complexity jumps from LOGSPACE to NL-complete when going from CQ evaluation to CRPQ evaluation [28].

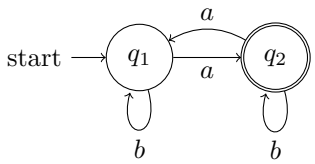
1.5 Languages and relations on words

In this section, we present a short introduction to automata, regular languages, and word relations. Regular languages are expressible via many formalisms and we briefly discuss some of them.

1.5.1 Finite automata

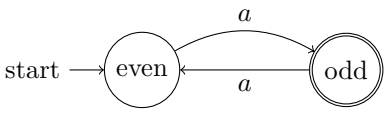
DFAs are a class of simple machines that recognize regularities in words. A DFA contains a set of states Q and a transition function $\delta\colon Q \times \mathbb{A} \to Q$. It also contains a starting state s and a set of final states F. When a DFA reads a word, it starts at s and reads each letter, moving its states as dictated by the transition function. When the word ends, the DFA ends up at some state q. If $q \in F$, then the word is said to be *recognized* by the DFA, if not it is rejected. The language of words recognized by a DFA is called a *regular language*. The set of regular languages is denoted by REG.

Example 2. The following DFA accepts all words in $\{a, b\}^*$ which contain an odd number of a's. The initial state is marked by an incoming edge from "start", and the final state is marked by a double ring.



NFAs, or nondeterministic finite automata, are another class of machines in which we have a finite set of states, and $\delta \subseteq Q \times A \times Q$ is a relation (as opposed to a function). NFAs and DFAs have equal expressive power, despite NFAs being very succinct.

Example 3. Here is an NFA over $\{a, b\}$ which accepts the same language as the DFA above. We use the same convention as Example 2 to denote initial and final states.



1.5.2 Regular expressions

Regular expressions are an alternative formalism to defining regular languages. A regular expression (also known as a rational expression) is a string written in a given syntactic format that defines a pattern on words. These expressions are implemented as `Regex` strings and are most commonly used in algorithms to search, find and replace words in documents. Search engines, query languages, and text editors implement regular expressions in various syntactic formats. The origin of regular expressions lies in the work of Kleene (1956) [69] who formalized the notion of a regular language. In formal language theory, regular expressions are defined as follows:

Definition 1. A **regular expression** over $\mathbb{A}$ is a string constructed recursively using the operators $+, \cdot$ and $*$. To every regular expression r we associate a language $L(r) \subseteq \mathbb{A}^*$.

- $\emptyset$ is a regular expression and $L(\emptyset) = \emptyset$.
- For every letter $a \in A$, a is a regular expression with $L(a) = \{a\}$.
- If r_1 and r_2 are regular expressions, then $r_1 + r_2$ is a regular expression and $L(r_1 + r_2) = L(r_1) \cup L(r_2)$.
- If r_1 and r_2 are regular expressions, then $r_1 \cdot r_2$ is a regular expression and $L(r_1 \cdot r_2) = L(r_1) \cdot L(r_2)$.
- If r is a regular expression, then r^* is a regular expression and $L(r^*) = (L(r))^*$.

While using regular expressions in queries, we use the negation operator $\neg$ for succinctness (we are allowed to do so as Reg is closed under complement). Its semantics are defined as follows (as an addition to Definition 1): $\neg r$ is a regular expression and $L(\neg r) = \mathbb{A}^* \setminus L(r)$.

Regular expressions which do not use the Kleene star ($*$), but use complement ($\neg$) and intersection ($\wedge$) are called *star-free* regular expressions. The regular languages defined by these expressions are called star-free languages.

1.5.3 Monadic Second Order (MSO($<$)) logic

In addition to finite automata and regular expressions, regular languages are also characterized via Monadic Second Order logic over words (MSO($<$)). It was shown by Büchi in 1960 that the set of sentences written in Monadic Second Order logic on finite words, denoted by MSO($<$), precisely characterizes Reg [20]. MSO formulæ are composed using:

- first order variables written in lowercase ($x, y, z, \dots$) and second order variables written in uppercase ($X, Y, Z, \dots$);

- the universal and existential quantifiers ($\forall$ and $\exists$ respectively), used to write formulæ of the form $\exists x\varphi, \exists X\varphi, \forall x\varphi, \forall X\varphi$;
- atomic formulæ of the form $x < y$ and $\mathbf{a}(x)$, for all first order variables x, y and letters $a \in \mathbb{A}$;
- Boolean combinations of formulæ written as $\varphi_1 \vee \varphi_2$(disjunction), $\varphi_1 \wedge \varphi_2$ (conjunction), and $\neg\varphi$ (negation).

An MSO sentence is a formula without free variables. A MSO sentence φ is interpreted over a word $w \in \mathbb{A}^*$ using semantic rules that assign to first order variables positions within w (i.e. elements of $\{1, \ldots, |w|\}$), and to second order variables subsets of $\{1, \ldots, |w|\}$. Every MSO sentence φ defines a set of words $|\varphi| \subseteq \mathbb{A}^*$, called the language of φ. For example, the set of all words that begin with the letter a is defined by the sentence

$$\exists x(\mathbf{a}(x) \wedge \forall y(x < y \vee x = y))$$

Büchi's theorem states that a language $L \subseteq \mathbb{A}^*$ is regular if and only if L is defined by an MSO sentence. We formally define the syntax and semantics of MSO in Chapter 5, Section 5.2.1.

1.5.4 Algebraic characterization of regular languages

Regular languages have been studied extensively from an algebraic point of view [44, 95, 111, 53, 68]. Below we provide a brief introduction of this field of study, popularly referred to as *algebraic automata theory*. First, we lay down some definitions.

A *semigroup* is a structure $(S, \circ)$, where S is a set equipped with an binary operation

$$\circ\colon S \times S \to S$$

such that for all elements $p, q, r \in S$, we have $(p \circ q) \circ r = p \circ (q \circ r)$. A *monoid* is a semigroup $(M, \circ)$ containing an identity element $\mathbf{1} \in M$ such that for every element $p \in M$, $\mathbf{1} \circ p = p \circ \mathbf{1} = p$. When it is clear what the underlying operation $\circ$ is, we abuse notation and refer to the semigroup (respectively monoid) as its underlying set S (respectively M).

Example 4. Some examples of infinite semigroups are:

- The set of all positive integers $\mathbb{N}\backslash\{0\}$ forms a semigroup under the addition operation $+$.
- A finite alphabet $\mathbb{A}$ generates the monoid $\mathbb{A}^*$, called a *free monoid* (where the monoid operation is word concatenation, and the identity element is the empty word ε).

Some examples of finite semigroups are:

- The set $\{-1, 0, 1\}$ forms a monoid under multiplication.
- For any $n \in \mathbb{N}$ such that $n > 1$, we denote by $\mathbb{Z}/n\mathbb{Z}$ the set $\{0, 1 \ldots, (n-1)\}$, which forms a monoid under the operation $(p, q) \mapsto (p + q) \bmod n$. ■

A *monoid homomorphism* between two monoids $(M_1, \circ_1)$ and $(M_2, \circ_2)$ is a function $h\colon M_1 \to M_2$ such that (1) h maps the identity element of M_1 to the identity element of M_2, and (2) for all $p, q \in M_1$, we have $h(p \circ_1 q) = h(p) \circ_2 h(q)$.

Languages recognized by monoids

Let $\mathbb{A}$ be a finite alphabet. For any language $L \subseteq \mathbb{A}^*$, the relation $\approx_L \subseteq \mathbb{A}^* \times \mathbb{A}^*$, defined as

$$x \approx_L y \text{ if and only if for all } u, v \in \mathbb{A}^*,\ uxv \in L \iff uyv \in L$$

is an equivalence relation. This relation is called the *syntactic congruence* of L. A well-known theorem of Myhill and Nerode states that the syntactic congruence of a language has finite index (that is, it has finitely many equivalence classes) if and only if the language is regular [42].

For any language L, the set of equivalence classes $\{[w]\}_{w \in \mathbb{A}^*}$ of $\approx_L$ constitutes a monoid under the operation $([x], [y]) \mapsto [xy]$ (with identity element $[\varepsilon]$). This is known as the *syntactic monoid* of L. The syntactic monoid was first conceptualized by Schützenberger [104], and it was independently established by Rabin and Scott (who credited it to Myhill) that a language is regular if and only if its syntactic monoid is finite [100]. This fundamental result opened up a bridge between the world of finite semigroups and regular languages, beginning with the notion of word languages recognized by monoids. We formally define this below.

Definition 2. Given a finite monoid M, we say that M *recognizes* a language L if there exists a homomorphism $\eta\colon \mathbb{A}^* \to M$ and a subset $X \subseteq M$ such that $L = \eta^{-1}(X)$.

For example, the language of even length words is recognized by $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$ as the inverse image of $\{0\}$ under the homomorphism

$$w \mapsto \begin{cases} 0 \text{ if } |w| \text{ is even} \\ 1 \text{ otherwise} \end{cases}$$

If a finite monoid M recognizes a regular language $L \subseteq \mathbb{A}^*$ via a homomorphism h and recognizing set $X \subseteq M$, it also functions as a DFA for L. The states of this DFA are the elements of M, the starting state is $h(\varepsilon)$, the set of final states is X, and the transition function is given by $\delta(q, a) = q \circ h(a)$.

Combinatorial properties of languages sometimes translate to algebraic properties of the monoids recognizing them. One of the earliest non-trivial instances of such a result was Schützenberger's characterization of star-free languages by the class of aperiodic monoids. We say that a monoid M is aperiodic if there exists an integer n such that for every $p \in M$, $p^n = p^{n+1}$. Schützenberger showed in 1965 that a language is star-free if and only if its syntactic monoid is finite and aperiodic [105].

Regular expressions over monoids

Regular expressions can be defined more generally over monoids. Let $(M, \cdot)$ be a monoid and X_1, X_2 be arbitrary subsets of it. We define $X_1 \cdot X_2$ to be $\{x_1 \cdot x_2 \colon x_1 \in X_1, x_2 \in X_2\}$. Therefore, $(\mathcal{P}(M), \cdot)$ is a monoid whose identity element is $\{\mathbf{1}_M\}$. We now inductively define the set of regular expressions over M such that each expression r generates a subset of $X(r)$ of M, as follows:

- $\emptyset$ is a regular expression and $X(\emptyset) = \emptyset$.

- For every element $m \in M$, m is a regular expression with $X(m) = \{m\}$.
- If r_1 and r_2 are regular expressions, then $r_1 + r_2$ is a regular expression and $X(r_1 + r_2) = X(r_1) \cup X(r_2)$.
- If r_1 and r_2 are regular expressions, then $r_1 \cdot r_2$ is a regular expression and $L(r_1 \cdot r_2) = X(r_1) \cdot X(r_2)$.
- If r is a regular expression, then r^* is a regular expression and $X(r^*)$ is the submonoid generated by the elements of $X(r)$.

We observe that given a finite alphabet $\mathbb{A}$, the set of classical regular expressions over $\mathbb{A}$ define the same languages as the set of regular expressions over the free monoid $\mathbb{A}^*$.

1.5.5 Automata-definable word relations

A *k-tuple* of elements of S is an element $\bar{s} \in S^k$, that is $\bar{s} = (s_1, \ldots, s_k)$ with $s_1, \ldots, s_k \in S$. For each $1 \leq i \leq k$, we denote by $\bar{s}(i)$ the element s_i. A *relation* over S is a subset of S^k for some $k \in \mathbb{N}$. A *word relation* is a relation over $\mathbb{A}^*$.

Going beyond languages, the formalisms of finite automata and regular expressions can be modified to define relations over words. However, multiple models of relations are definable using automata. We illustrate a few of them here, using a survey on word relations by Choffrut as our reference [32].

Recognizable relations

Let $k \in \mathbb{N}$. Consider a morphism η from the monoid $(\mathbb{A}^*)^k$ (whose operation is $(u_1, \ldots, u_k) \circ (v_1, \ldots, v_k) \mapsto (u_1 v_1, \ldots, u_k v_k)$) to a finite monoid M. Let X be a subset of M. Now, the set $\eta^{-1}(X)$ is a k-ary relation over words in $\mathbb{A}^*$. We see that defining relations using monoids generalizes the algebraic characterization of regular languages. These relations are called *recognizable*, and we denote the set of all recognizable relations (of all arities) by Rec. An unpublished result attributed to Mezei by Eilenberg (see [42], p. 75) characterizes recognizable relations as finite unions of Cartesian products of regular languages (given in [42], Proposition 12.2).

Recognizable relations are quite limited in terms of expressivity. Even the equality relation $\text{Eq} = \{(u, u) \colon u \in \mathbb{A}^*\}$ is not recognizable, as can be seen from the following argument: if Eq is a union of products $L_i \times L_i'$, then for each i, L_i, L_i' are singleton languages, forcing the union to be infinite.

Rational relations

Let $\mathbb{A}$ be a finite alphabet. For all $k \in \mathbb{N}$, consider the monoid $((\mathbb{A}^*)^k, \cdot)$ whose operation is defined as

$$(u_1, \ldots, u_k) \cdot (v_1, \ldots, v_k) = (u_1 v_1, \ldots, u_k v_k)$$

for all $u_i, v_i \in \mathbb{A}^*$. We define $\text{Rat}_k(\mathbb{A})$ to be the set of k-ary relations over $\mathbb{A}^*$ given by regular expressions over $(\mathbb{A}^*)^k$. We call these k-ary *rational relations*. The class of all rational relations is denoted by

$$\text{Rat} = \bigcup_{k \in \mathbb{N}} \text{Rat}_k$$

The class RAT has been studied extensively in the literature, albeit characterized by different models of automata. It was characterized by Rabin and Scott (1959) as the set of relations recognized by one-way single-tape automata accepting tuples of words [100]. A study of the properties of these relations was carried out by Elgot and Mezei (1965) [45]. The main decision problems were addressed by Fischer and Rosenberg (1968) [49].

Every recognizable relation is rational. To see why, consider the following argument: given regular languages $L_1, \ldots, L_k \subseteq \mathbb{A}^*$ defined by regular expressions $r_1, \ldots, r_k$, a regular expression defining $L_1 \times \cdots \times L_k$ may be constructed as follows:

- For every $i \in \{1, \ldots, k\}$, we produce a regular expression r'_i that defines
$$(\mathbb{A}^*)^{i-1} \times L_i \times (\mathbb{A}^*)^{k-i}$$
by replacing, for every $a \in \mathbb{A}$, each instance of a in r_i with
$$\bigcup (a_1, \ldots, a_{i-1}, a, a_{i+1}, \ldots, a_k)$$
where the union ranges over all tuples $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k) \in \mathbb{A}^{k-1}$.
- Now $r = r'_1 \wedge \cdots \wedge r'_k$ defines the set
$$\{(w_1, \ldots, w_k) \colon w_i \in L_i \text{ for every } i \in \{1, \ldots, k\}\}$$
In other words r is a regular expression that defines $L_1 \times \cdots \times L_k$.

Since regular expressions are closed under finite union, the above observation shows that every recognizable relation can be defined by a regular expression over $(\mathbb{A}^*)^k$. Therefore REC $\subseteq$ RAT. A more general result can be stated in terms of *finitely generated* monoids. We say that N is a generator for a monoid M if N contains the identity element of M and every element in M can be written as a monoid product of elements in N. Thus $(\mathbb{A}^*)^k$ is finitely generated for every $k \in \mathbb{N}$. McKnight's theorem states that recognizable subsets of finitely generated monoids are rational (see Theorem 2.14 on p. 81 in [95]).

Further examples of rational relations over the two letter alphabet $\mathbb{A} = \{a, b\}$ include the suffix relation $\{(u, vu) \colon u, v \in \mathbb{A}^*\}$, given by
$$((\varepsilon, a) + (\varepsilon, b))^* \cdot ((a, a) + (b, b))^*$$
over the alphabet $\mathbb{A} = \{a, b\}$, and the subword relation
$$\{(a_1 \ldots a_n, u_1 a_1 \ldots u_n a_n u_{n+1}) \colon a_1, \ldots, a_n \in \mathbb{A}, u_1, \ldots, u_{n+1} \in \mathbb{A}^*\}$$
given by the expression
$$((\varepsilon, a) + (\varepsilon, b) + (a, a) + (b, b))^*$$

It is known that RAT is not closed under intersection [16]. Furthermore, several key problems on rational relations are undecidable, such as checking whether two rational relations have a non-empty intersection [16], or if a given rational relation has a non-empty intersection with a given synchronous relation [14]. However, the class of synchronous relations – which is a strict subset of RAT (see Lemma 2) – proves to be a robust class from a theoretical as well as application point of view. It covers REC and can be seen as a natural extension of REG in higher dimensions, owing to its equivalence to the relations recognized by synchronous automata.

Synchronous relations

Synchronous relations are defined by synchronous automata. These are multi-tape DFAs with synchronous movement of tape heads. These automata can be conceptualized as DFAs over *synchronized words* – originally presented by Blumensath and Grädel (2000) in their work on *automatic structures* [17].

A synchronized word is a way of representing a tuple of words $(w_1, \ldots, w_k)$ as a single word, written as $w_1 \otimes \cdots \otimes w_k$. This is sometimes called the *convolution of* $w_1 \otimes \cdots \otimes w_k$. We formally define this as follows:

Definition 3. Let $\mathbb{B}$ be a finite alphabet. For any word $w = b_1 \ldots b_n \in \mathbb{B}^*$, and $i \in \{1, \ldots, n\}$ we denote by $w[i]$ the letter b_i. Further for all $1 \leq i < j \leq n$ let $w[i \ldots j]$ denote the word $b_i \ldots b_j$. Given a tuple $\bar{s} = (s_1, \ldots, s_k)$, for each $1 \leq i \leq k$, we denote by $\bar{s}(i)$ the element s_i.

Let $\mathbb{A}$ be a finite alphabet and $\perp$ be a symbol not in $\mathbb{A}$. We denote by $\mathbb{A}_\perp^k$ the set $(\mathbb{A} \dot\cup \{\perp\})^k \setminus \{(\underbrace{\perp, \ldots, \perp}_{k \text{ times}})\}$. Every element $\bar{a} = (a_1, \ldots, a_k)$ of $\mathbb{A}_\perp^k$ is called a *k-synchronized letter*. For every $i \in \{1, \ldots, k\}$, we let $\bar{a}(i) = a_i$.

For each k-tuple $(w_1, \ldots, w_k) \in (\mathbb{A}^*)^k$ the **synchronized word** of $(w_1, \ldots, w_k)$, written $\bar{w} = w_1 \otimes \cdots \otimes w_k$, is the unique word $\bar{w} \in (\mathbb{A}_\perp^k)^*$ such that:

- $|\bar{w}| = \max_{1 \leq i \leq k} |w_i|$
- For every $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, |w|\}$,

$$\bar{w}[j](i) = \begin{cases} w_i[j] \text{ if } j \leq |w_i| \\ \perp \text{ otherwise} \end{cases}$$

Let us consider some examples of synchronized words over the alphabet $\mathbb{A} = \{a, b, c\}$.

- $abaa \otimes bab = (a, b)(b, b)(a, b)(a, \perp)$.
- Let $n \in \mathbb{N}$, and let $\bar{u}_n = a^n \otimes b^{2n} \otimes c^{3n+1} = (a, b, c)^n (\perp, b, c)^n (\perp, \perp, c)^{n+1}$. Then
  - $\bar{u}_n[n](1) = a$
  - $\bar{u}_n[2n](2) = b$
  - $\bar{u}_n[3n](3) = c$.

We denote by SW_k the set of all k-synchronized words. Formally,

$$\mathrm{SW}_k = \{w_1 \otimes \cdots \otimes w_k \colon (w_1, \ldots, w_k) \in (\mathbb{A}^*)^k\}$$

For each k-ary relation $R \subseteq (\mathbb{A}^*)^k$ we denote by L_R the SW_k subset

$$\{w_1 \otimes \cdots \otimes w_k \colon (w_1, \ldots, w_k) \in R\}$$

called the synchronized language of R. Then, R is called a **synchronous relation** if and only if L_R is a regular language. We denote by SYNC the set of all synchronous relations.

Example 5. Let $k \in \mathbb{N}$ and $\mathbb{A}$ be an alphabet. The universal k-ary relation

$$\underbrace{\mathbb{A}^* \times \cdots \times \mathbb{A}^*}_{k \text{ times}}$$

denoted by $\mathbb{U}_k(\mathbb{A})$, is a synchronous relation given by the synchronized language SW_k.

Example 6. Fix an alphabet $\mathbb{A}$, consider the alphabet $\mathbb{A}_\perp^2$. Let $\mathbb{A}_1 = \mathbb{A} \times \{\perp\}, \mathbb{A}_2 = \{\perp\} \times \mathbb{A}, \mathbb{A}_\Delta = \{(a, a) \colon a \in \mathbb{A}\}$. Consider the equal length and prefix binary relations:

$$\begin{aligned} R_{\text{eqlen}} &= \{(w_1, w_2) \in \mathbb{A}^* \times \mathbb{A}^* \colon |w_1| = |w_2|\} \\ R_{\text{pref}} &= \{(w_1, w_2) \in \mathbb{A}^* \times \mathbb{A}^* \colon w_1 \text{ is a prefix of } w_2\} \end{aligned}$$

Let $L_{\text{eqlen}}, L_{\text{pref}}$ be the synchronized languages of these relations. They are represented by the regular expressions

$$\begin{aligned} L_{\text{eqlen}} &= (\mathbb{A} \times \mathbb{A})^* \\ L_{\text{pref}} &= \mathbb{A}_\Delta^* \cdot (\mathbb{A}_1^* + \mathbb{A}_2^*) \end{aligned}$$

So, the prefix and equal length relations are synchronous relations.

Due to the closure properties of regular languages, SYNC is closed under Boolean operations like union, intersection and complement. Moreover, synchronous relations are also closed under *projections*, which we define below:

Definition 4. For any subset $\{s_1, \ldots, s_n\} \subseteq \{1, \ldots, k\}$, where $s_1 < s_2 < \cdots < s_n$, we define the S-projection on SW_k as

$$\begin{aligned} \pi_S \colon \mathrm{SW}_k &\to \mathrm{SW}_n \\ w_1 \otimes \cdots \otimes w_k &\mapsto w_{s_1} \otimes \cdots \otimes w_{s_n} \end{aligned}$$

and when $S = \{i\}$, we simply denote π_S as π_i. This notion is lifted to sets of synchronized words as:

$$\pi_S(L) = \{\pi_S(\bar{w}) \colon \bar{w} \in L\}$$

Lemma 1. *Regular synchronized languages are closed under projections.*

Proof. Let $\mathcal{A}$ be a DFA recognizing a k-synchronized language L, and let $S = \{s_1, \ldots, s_n\}$ be a subset of $\{1, \ldots, k\}$. To show that $\pi_S(L)$ is also a synchronized language, we construct a DFA $\mathcal{A}'$ for it. Let Q be the set of states of $\mathcal{A}$, with q_0 as its initial state and F the set of its final states. Further, let δ be the transition function of $\mathcal{A}$. Now, we define $\mathcal{A}'$ to be the DFA which has the same set of states as Q, including its initial state and final states, with the transition function

$$\delta' = \{(q, \pi_S(\bar{a}), q') \colon (q, \bar{a}, q') \in \delta\}$$

By definition, $\mathcal{A}'$ accepts exactly those synchronized words $\bar{u}$ such that $\bar{u} = \pi_S(\bar{w})$ for some $\bar{w}$ accepted by $\mathcal{A}$. In other words, $\mathcal{A}'$ accepts the language $\pi_S(L)$, and therefore $\pi_S(L)$ is a regular n-synchronized language. □

An example of a rational but not synchronous relation is the subword relation.

Definition 5. Let $u = a_1 \ldots a_n, v = b_1 \ldots b_m \in \mathbb{A}^*$. We say u is a (scattered) **subword** of v, denoted by $u \sqsubseteq v$, if and only if there exists an increasing function $p \colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ (called the **witness function**) such that for every $1 \leq i \leq n$, $b_{p(i)} = a_i$. For any $S \subseteq \mathbb{A}^*$, we let

$$\uparrow S = \{v \in \mathbb{A}^* \colon \text{ there exists } u \in S \text{ such that } u \sqsubseteq v\}$$

Lemma 2. *The subword relation is in* Rat \ Sync.

Proof. The subword relation $\sqsubseteq$ is given by the regular expression

$$\left(\bigcup_{a \in \mathbb{A}} (\varepsilon, a) + (a, a) \right)^*$$

Therefore, $\sqsubseteq$ is a rational relation. To show that $\sqsubseteq$ relation is not synchronous, we proceed by contradiction and assume that the synchronized language of $\sqsubseteq$ is regular. We use the well-known *pumping lemma* of regular languages, first proved by Rabin and Scott [100]. This is stated below.

For any regular language L, there exists an integer n (called its *pumping length*) such that for all words $x \in L$ of length at least n, then there exist $u, v, w \in \mathbb{A}^*$ such that $x = uvw$, and :

- $|uv| \leq n$,
- $|v| \geq 1$, and
- for all $i \in \mathbb{N}$, $uv^i w \in L$.

Going back to the subword relation, let us assume that the synchronized language of $\sqsubseteq$ is a regular language L whose pumping length is n. The word $a^n \otimes c^n a^n = (a, c)^n (\bot, a)^n$ is in L as a^n is a subword of $c^n a^n$. By the pumping lemma, there exist synchronized words u, v, w such that $uvw = (a, c)^n (\bot, a)^n$ and

- $|uv| \leq n$,

- $|v| \geq 1$, and
- for all $i \in \mathbb{N}$, $uv^i w \in L$.

Let $m = |v|$. By the pumping lemma, $uv^i w \in L$ for each $i \in \mathbb{N}$. In particular,

$$uv^2 w = uv.v.w = (a, c)^{m+n}(\bot, a)^n \in L$$

Therefore, $a^{m+n} \otimes c^{m+n}a^n \in L$. However, a^{m+n} is not a subword of $c^{m+n}a^n$. This leads to a contradiction and we conclude that $\sqsubseteq$ is not in SYNC. □

1.6 Research goals

We pursue research goals on two topics: evaluation of ECRPQs, and the logical characterization of synchronous relations. We now outline these goals.

1.6.1 ECRPQ evaluation

A common theme in the study of query evaluation is to find subclasses of queries where the evaluation problem has lower complexity than in the general case. It was shown by Chandra and Merlin (1997) that EVAL-CQ is NP-complete [28]. The problem was simplified by Chekuri and Rajaraman (1997) and Kolaitis and Vardi (1998) who showed that by restricting *the shape* of CQs (that is, the tree-width of its Gaifman graph), we obtain subclasses whose evaluation is in polynomial time (that is, bounded tree-width CQs). The graph abstraction-based approach was formalized by Grohe et al. (2001) [58], who completely characterized CQ evaluation using Gaifman abstractions.

The tree-width of a graph is a number associated with it that determines its degree of connectivity. It is a metric that plays a fundamental role in many algorithmic problems on graphs and graph-related structures. The Gaifman graph of a Conjunctive Query is an undirected graph over its variables, with two variables sharing an edge if and only if they occur in some relational atom. Thus it is an abstraction of a CQ, a rough approximation of its shape, with all its relation atoms stripped away.

The tractability of the evaluation problem for bounded tree-width query subclasses extends to CRPQs as well, with a similar notion of the Gaifman graph and its tree-width. As stated earlier, CRPQs use reachability atoms of the form $x \xrightarrow{\pi} y$ in place of relation atoms, and regular expressions to specify conditions on path labels. Therefore, their Gaifman graphs abstract away the regular languages to produce a rough approximation of their shape.

What can be said about ECRPQ evaluation? We know that the general problem is PSPACE-complete [15]. The goal, then, is to simplify the problem using the techniques used in CQ and CRPQ evaluation. This entails that we identify a good abstraction for ECRPQs, in the form of a graph-based structure. Moreover, we need to identify measures on this abstraction (similar to tree-width in the case of CRPQ and CQ evaluation) such that when these measures are bounded, they yield subclasses that have tractable evaluation.

We analyze the underlying structure of ECRPQs and formalize it using a graph-based abstraction which we call *two-level multi-hypergraphs*, or *2L-graphs* for short. A 2L-graph abstracts an ECRPQ via an underlying labeled graph,

which models the reachability subquery, as well as a second level hypergraph over the edges of the graph, which models the relations used in the ECRPQ. By defining measures or quantities over a 2L-graph, we can talk about classes of ECRPQs with specific structural properties. Based on these properties, we can talk about the complexity of the evaluation problem for these subclasses, and stratify them according to various levels of hardness. We study the combined complexity as well as the parameterized complexity of ECRPQ evaluation in this way, building upon the results of CRPQ evaluation. The 2L-graph-based analysis reveals the role that relations play in making the general ECRPQ evaluation problem significantly harder than CRPQ evaluation.

We have formalized our results on ECRPQ evaluation in a paper submitted to the Principles of Database Systems (PODS) conference, which was subsequently accepted to be published in the 2022 proceedings. A pre-print can be found in Figueira and Ramanathan[47].

1.6.2 Logical characterization of synchronous relations

The class of regular languages has been characterized in many different ways, including regular expressions, finite automata, finite monoids, and Monadic Second Order (MSO) logic over finite words. As discussed earlier, regular expressions are used in path queries to specify conditions on path labels. The set of first order formulæ of MSO(<), denoted as FO(<), defines a subclass of REG which coincides with the star-free languages (McNaughton and Papert, 1971) [84]. How is this connected to regular path queries? As mentioned earlier, the practical implementation of CRPQs in SPARQL reveals a pattern that a majority of human and script-generated queries only use regular languages definable in "small fragments" of FO(<).

These small fragments are defined in terms of levels of the quantifier alternation hierarchy of FO(<). Simply put, any first order formula can be re-written by putting all of its quantifiers $(\exists, \forall)$ to the left, leaving a quantifier-free formula prefixed by alternating blocks of quantifiers of the form $\exists^*$ and $\forall^*$. This is called its *prenex normal form*. Σ_i is the set of formulæ whose prenex normal form is in

$$\underbrace{\exists^*\forall^*\ldots}_{i \text{ alternating blocks}} \quad \text{(a quantifier free formula)}$$

and Π_i is the set of formulæ whose prenex normal form is in

$$\underbrace{\forall^*\exists^*\ldots}_{i \text{ alternating blocks}} \quad \text{(a quantifier free formula)}$$

Since ECRPQs extend CRPQs with word relations, a natural question is whether there exist subclasses of relations analogous to those subclasses of REG that correspond to lower levels of the FO(<) quantifier alternation hierarchy. Fortunately, a first order logical characterization of synchronous relations given by Eilenberg et al. in 1969 proves useful here. This logic is the first order theory of finite words with the prefix, equal length, and last letter predicates [43]. A close look at the proof of the characterization theorem reveals that any synchronous relation can be defined by a formula in the third level of the alternation hierarchy of this logic, which implies that the logic collapses in expressive

power to its Σ_3 fragment. This result raises the question of characterizing levels one and two of the quantifier alternation hierarchy of this logic.

Our results show that the relations definable in these lower levels are similar in combinatorial description to the regular languages definable in the corresponding levels of the FO(<) quantifier alternation hierarchy. We further prove that the membership problem for these "small fragments" of relations is decidable. These results are published in Figueira, Ramanathan, Weil 2019 [48].

1.7 Plan of the thesis

This thesis consists of two parts comprising three and two chapters respectively. Part I introduces the graph and relational data models and associated querying formalisms. We formally define Conjunctive Queries and Regular Path Queries. Following that, we look at various extensions to RPQs, including Conjunctive RPQs (CRPQs) and Extended CRPQs (ECRPQs) which add to the expressive power of the path query formalism. We also explore the graph-based abstractions of these formalisms that have been used to determine the complexity of evaluating path queries. Finally, we present our results in the form of two theorems characterizing ECRPQ evaluation, pertaining to combined complexity and parameterized complexity respectively.

Part II deals with Sync, the set of synchronous relations. This is the class of word relations that CRPQs are equipped with to produce ECRPQs. We build upon a result by Eilenberg, Elgot, and Shepherdson which states that Sync is characterized by the first order logic on words with the prefix, equal length, and last letter predicates. We show that this logic collapses to the third level in its quantifier alternation hierarchy. This raises the question of characterizing the subsets of relations definable in the lower levels of the hierarchy and their Boolean combinations. We show that these classes of relations can be effectively characterized and admit decidable membership. Moreover, we show how the combinatorial description of these classes of relations reveals a significant connection with the lower levels of the FO(<) quantifier alternation hierarchy.

We now describe the content in each chapter, along with references to the sections containing the topics we discuss.

Part I: Queries

This part consists of Chapters 2,3 and 4.

Chapter 2 introduces the key concepts associated with databases and query formalisms. We define relational databases and conjunctive queries in Sections 2.2 and 2.2.1 respectively. Following that, we define graph databases in Section 2.3 and discuss their applications. In Section 2.3.1 we introduce labeled graphs as our chosen abstraction to study graph databases. Then, we formally introduce path queries in Section 2.3.2. After defining RPQs and CRPQs in this section, we discuss some well-known extensions of CRPQs in Section 2.3.3. This leads us to the motivation behind ECRPQs, their definition as well as semantics, which is addressed in Section 2.3.4.

Following this, we discuss the ECRPQ evaluation problem in Section 2.4, touching upon its state-of-the-art with respect to CQs and CRPQs. Here we

discuss how and why queries are *abstracted*, i.e. reduced to their essential shape in the form of a graph or hypergraph-based structure. The chapter concludes with a discussion on abstracting ECRPQs, keeping in mind the goal of obtaining a complexity theorem similar to the ones associated with CQ and CRPQ evaluation.

Chapter 3 defines the structure which serves as the ECRPQ abstraction. This is called the *two-level-multi-hypergraph*, or *2L-graph* in short. We formally define 2L-graphs and ECRPQ abstraction in Section 3.1.1. We also define *measures* on 2L-graphs, which are numerical quantities associated with classes of 2L-graphs that allow us to formulate the theorem statements. These measures are defined in Sections 3.2.1 and 3.2.2. Finally, we state our results with respect to these measures and discuss how the theorems are proved in Section 3.3.

Chapter 4 serves as the technical portion of Part I, containing the statements and full proofs of all the theorems, lemmas, and propositions required to establish our results. We begin this chapter by establishing some technical preliminaries, in Section 4.1. The lemmas needed to prove the main theorems are given in Section 4.2, which is divided into the lemmas on upper bounds (in Section 4.2.2) and the lemmas on lower bounds (given in Section 4.2.3). The final proofs of the theorems are given in Section 4.3. The chapter concludes with a discussion on related work (Section 4.4), which includes a discussion on non-Boolean queries as well as potential avenues for adapting these ideas for the query containment problem.

Part II: Relations

This part deals with synchronous relations and their logical characterization. It consists of Chapters 5 and 6.

Chapter 5 covers concepts pertaining to languages, word relations, and logic, an overview of which is given in Section 5.1. This is followed by Section 5.2 which discusses the characterization of regular languages using logic and algebra. We define the syntax and semantics of the logic MSO($<$) in Section 5.2.1. In Sections 5.2.2 and 5.2.3, we define aperiodic languages and take a closer look at the structure of first order formulæ from the perspective of quantifier alternation. Some subclasses of Reg defined by small fragments of FO($<$) are discussed in Sections 5.2.4 and 5.2.5.

The logical characterization of synchronous relations by Eilenberg et al. is explained in Section 5.3. The chapter concludes with Section 5.4, which contains the statements of our contributions on this topic.

Chapter 6 is the technical portion of Part II. In Section 6.1 we define the notion of type sequences of synchronized words, an important concept that plays a crucial role in the proof of our results. Following that, in Section 6.2, we show (via a proof of Eilenberg's result) that FO[σ] is equal in expressivity to the third level in its quantifier alternation hierarchy. The task that remains then is to characterize the subsets of Sync corresponding to levels one and two, and their Boolean closures.

The results pertaining to the characterization and membership problem of $\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable relations are given in 6.3. In Section 6.4 we cover the results pertaining to the characterization and membership of $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$-definable relations, thus giving a complete picture of the quantifier alternation hierarchy of $\mathrm{FO}[\sigma]$. We conclude this chapter with a discussion on the *separation problem* for synchronous relations, which is given in 6.5.

Part I

Queries

Chapter 2

Databases and Queries

2.1 Introduction

2.1.1 General concepts

Informally, a *database* is understood to be a repository capable of storing, updating, and modifying data. Database entries or records stored within the database can be retrieved via certain commands called *queries*. These queries are conceptualized as mathematical formulæ and expressed in special languages called *query languages*. For example, Structured Query Language (SQL) is well-known for its wide range of applications in querying relational databases. Data organization is the practice of categorizing and classifying data in order to extract meaningful information from it.

As discussed in Chapter 1 (see Section 1.3), various data models are used for a variety of applications. We study query formalisms associated with two of these models: the relational model and the graph model. Relational databases store data in tables (called relations), and query them with simple declarative statements. Conjunctive Queries (CQs) constitute a well known querying formalism on relational databases.

In contrast, the graph data model has graph-based data structures in its underlying framework. Graph databases consist of nodes, representing data points, and labeled edges, which carry data representing the relationships between those data points. These find applications in real-world scenarios where the interconnections between objects are equally or even more important than the objects. For theoretical exploration, graph databases are commonly represented via edge labeled graphs. In software applications, however, these databases are modeled via formats like RDF or property graphs [9]. In the current work, we focus on *path queries*, which output tuples of nodes connected by paths whose labels satisfy some regular language or relation.

A *database management system* (DBMS) is a software that allows end-users to create, control and access databases [35]. Some of the well-known DBMSs are Microsoft Access, Microsoft SQL Server, and Oracle DB; all of these are based on the relational data model so they are called RDBMSs (relational DBMSs). Among the DBMSs that use the graph data model, Neo4j and OrientDB are widely applied.

Formal methods refer to the analysis of a digital system from a mathemati-

cal perspective, that is, studying devices, systems, or algorithms using theoretical models to describe their behavior. The formal treatment of databases and queries is useful for two reasons: (i) it helps to design efficient query evaluation algorithms, and determine their complexity in the best and worst case, and (ii) the expressive power of these queries can be quantified in precise terms, which allows for better database design.

2.1.2 Queries and languages

The term *query* generally refers to a command to modify or retrieve data from a database. In this thesis, we only focus on queries involved in data retrieval. Queries are sometimes modeled as formulæ in First Order (FO) logic, using variables and logical operators like $\exists$ and $\forall$. Moreover, relations over tuples of variables are represented using special symbols, called relation symbols.

Associated with each data model there are various types of queries. For example, CQs are written in FO logic using only relational symbols and the existential operator $\exists$. We discuss this querying mechanism in more detail in Section 2.2.1.

RPQs constitute the standard querying formalism on graph databases. As stated earlier, an RPQ evaluated on a graph database extracts all pairs of nodes such that there exists a path between them labeled by a word in a given regular language. These languages are specified as regular expressions over the labeling alphabet. A common extension to RPQs and CQs is the query formalism *Conjunctive Regular Path Queries* (CRPQs), which are conjunctions of RPQs with shared node variables and projection. CRPQs are involved in the ongoing standardization effort for Graph Query Languages (GQL) [37, 1] and G-core [7]. While RPQs and CRPQs find direct application in Neo4j and SPARQL queries, they are limited by the inability to compare labels on paths using relations. Further, they do not allow paths as output. Libkin et al. proposed extending CRPQs by allowing synchronous relations on paths [15]. As mentioned earlier, these word relations are recognized by multi-tape automata with synchronous movement of heads [43, 32]. This class of queries is called ECRPQ, which stands for *Extended CRPQ*. The expressive power of ECRPQs allows versatile application for many problems including those in the domain of Semantic Web and biological sequences. We discuss CRPQs and ECRPQs in more detail in Section 2.3.2.

2.1.3 Topic of research

In this part of the thesis we focus on the *query evaluation problem* for queries on graph databases. This problem is central to both theory and practice; the logical properties of queries can be exploited to obtain an algorithm for their evaluation. Given the increasingly large volumes of data that modern DBMSs are confronted with, optimizing query evaluation continues to be a central theme of research in the field of database theory. We only deal with the mathematical abstractions of databases and queries, as this allows us to discuss general query evaluation algorithms without being cluttered by the specifics of different query languages or DBMSs.

Given a query q and database D as input, the query evaluation problem involves the computation of the answer set $\text{ans}(q, D)$. The evaluation problem

has been extensively studied for several types of queries on different data models, including CQs [58], CRPQs and ECRPQs [15]. In this part of the thesis, we study the ECRPQ evaluation problem in depth, which is an algorithmic problem stated as follows:

ECRPQ-evaluation (eval-ECRPQ)
Input: An ECRPQ q, a graph database D, and a pair $(\bar{v}, \bar{p})$ of nodes and paths in D.
Question: Does $(\bar{v}, \bar{p})$ belong to $\text{ans}(q, D)$?

In the introductory chapter (see Section 1.4), we talked about several complexity measures for the query evaluation algorithm. Recall that the combined complexity measure considers both the query and the database as the input whereas the parameterized complexity measure considers the database to be the input and query size as a parameter. We study ECRPQ evaluation with respect to these two measures. The parameterized ECRPQ evaluation problem is stated as follows:

Parameterized ECRPQ evaluation (p-eval-ECRPQ)
Input: An ECRPQ q, a graph database D, and a pair $(\bar{v}, \bar{p})$ of nodes and paths in D.
Question: Does $(\bar{v}, \bar{p})$ belong to $\text{ans}(q, D)$?
Parameter: The size of the query, $|q|$.

The data complexity, which is the complexity measure when query size is constant, is NL-complete for CRPQs and ECRPQs [15].

2.1.4 Contributions

The complexity of the query evaluation problem has been investigated in depth for many query formalisms. For instance, CQ evaluation is known to be NP-complete [28], a result that leads to CRPQ evaluation also being NP-complete by way of a simple polynomial time reduction (which is formally proved in Corollary 1). A central theme of research is to identify the subclasses of queries whose evaluation is *tractable* – in other words in PTime – and to characterize the structural properties of such subclasses. In the case of CQs, the *tree-width* measure on their graph representation provides a solution: the evaluation problem is tractable for any subclass of CQs whose underlying graph abstractions have bounded tree-width [29, 70]. Indeed, tractability is retrieved in the case of bounded tree-width CRPQs as well.

For ECRPQs, the general evaluation problem is known to be PSpace-complete [15]. In order to simplify the problem, we define a new abstraction for ECRPQs called *two-level graphs*, or 2L-graphs. Every class $\mathcal{C}$ of 2L-graphs corresponds to a class of ECRPQs abstracted by members of $\mathcal{C}$, which is denoted by ECRPQ($\mathcal{C}$). Under some mild hypothesis, we give precise criteria for $\mathcal{C}$ such that the complexity of evaluating ECRPQ($\mathcal{C}$) is in PTime, NP or PSpace. Unsurprisingly, we find that tree-width plays a role in evaluating ECRPQs, similar to CQs and CRPQs. However, we go beyond tree-width and define additional measures on 2L-graphs that give a full picture of ECRPQ evaluation.

We also characterize the parameterized evaluation problem, specifying precisely the classes $\mathcal{C}$ where the parameterized problem for ECRPQ($\mathcal{C}$) is in FPT (fixed parameter tractable), W[1]-complete (the parameterized complexity equivalent of NP) or XNL-complete.

2.1.5 Organization

In the current chapter we introduce data models, relational databases, graph databases, and queries. Following that, we formally state the decisions problems studied in this work, chiefly the ECRPQ evaluation problem. Finally, we explain our goals in regard to ECRPQ evaluation and the methodology we employ to achieve them.

In Chapter 3 we introduce 2L-graphs and corresponding measures on them. We illustrate how this abstraction naturally arises from an analysis of the PSPACE-hardness of the general evaluation problem. We end Chapter 3 with a statement of our results, which take the form of two theorems characterizing the combined as well as parameterized complexity of ECRPQ evaluation. Full proofs of these theorems are provided in Chapter 4, which serves as the technical section of this part of the thesis.

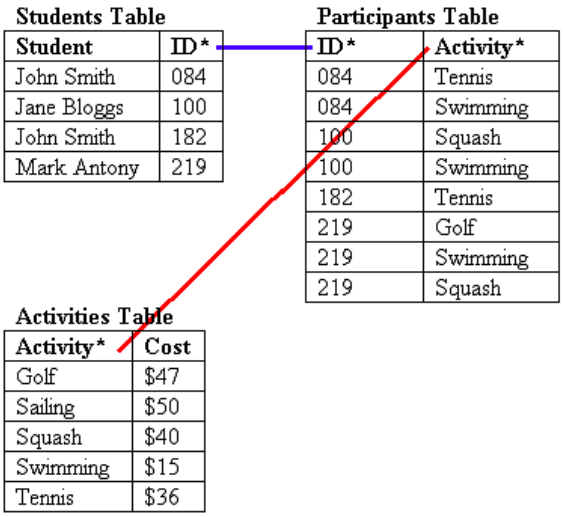
2.2 Relational databases

In 1970, while working for IBM, an English computer scientist called E. F. Codd revolutionized the field of database theory by proposing a new data model called the relational model [33]. A relational database consists of a set of tables (called relations), whose rows represent instances of some entity (like names of customers or products) and columns represent values or attributes of that entity (like ages of customers, or prices of products).

The introduction of relational databases signified a paradigm shift in data modeling; using the mathematical notion of a relation one could express different properties of an object. Structured Query Language or SQL is the paradigmatic and most well-known query language for relational databases. The relational model is best suited for applications where the structure of the data, or its schema, is fixed. Relational databases are applied ubiquitously in real-world scenarios, including but not limited to administrative and commercial software. However, this model has a limitation: databases with different underlying schema cannot be integrated, making it difficult to combine or extend them. Further, the query languages over relational databases are not expressive enough to specify the subtler, more implicit relationships between elements of the database like paths, neighborhoods, and patterns.

Example 7. The following relational database `StudentActivityList` records the participation of students in certain sports activities[1]:

[1]Image courtesy of http://www.databasedev.co.uk/table_design_tips.html



Students Table

Student	ID*
John Smith	084
Jane Bloggs	100
John Smith	182
Mark Antony	219

Participants Table

ID*	Activity*
084	Tennis
084	Swimming
100	Squash
100	Swimming
182	Tennis
219	Golf
219	Swimming
219	Squash

Activities Table

Activity*	Cost
Golf	$47
Sailing	$50
Squash	$40
Swimming	$15
Tennis	$36

Now, if we wanted to know students with names x_1, x_2 participating in the same activity, we could write a query $\texttt{SameActivity}(x_1, x_2)$:

$$\texttt{SameActivity}(x_1, x_2) = \exists y_1, y_2, z : \texttt{StudentsTable}(x_1, y_1) \wedge \texttt{StudentsTable}(x_2, y_2) \wedge \texttt{ParticipantsTable}(y_1, z) \wedge \texttt{ParticipantsTable}(y_2, z)$$

`SameActivity` is an example of a conjunctive query – it uses only existentially quantified variables y_1, y_2, z and the relations given in the database.

Codd proposed a wide range of operations on relations on databases, including permutation, projection, composition, and so on [33]. His model also specified precise conditions to check for redundancy, consistency, and data integrity. While SQL is the dominant query language paradigm in the realm of relational databases, it does deviate from Codd's proposed model by allowing duplicate rows, columns, and the `NULL` entry in any cell. However, for our purposes, we study a purely mathematical abstraction of relational databases which employs tuples and relations, with set semantics.

The definition of relational databases (given in Definition 6 below) is borrowed from the textbook of Abiteboul et al. [4]. We introduce some terminology first. Let us fix infinite, disjoint sets $\mathsf{Dom}, \mathsf{Rel}$, called the *domain* and the set of *relation names* respectively. A *signature* is a set of relation names $\mathcal{R} = \{R_1, \ldots, R_n\} \subseteq \mathsf{Rel}$ and a set of arities $\text{arity}(R_1), \ldots, \text{arity}(R_n) \in \mathbb{N}$ associated with them. For any finite set $V \subseteq \mathsf{Dom}$ and every relation R belonging to a signature $\mathcal{R}$, an **interpretation** of R over V associates to R a relation $[R] \subseteq V^{\text{arity}(R)}$.

Definition 6. A **relational database** consists of (i) a pair $\mathcal{D} = (V, \mathcal{R})$ where $V \subseteq \mathsf{Dom}$ is a finite set (called the **active domain** of $\mathcal{D}$) and $\mathcal{R}$ is a signature, and (ii) an *interpretation* $[R_i] \subseteq V^{\text{arity}(R_i)}$ for every $R_i \in \mathcal{R}$.

Example 8. The database `StudentActivityList` given in Example 7 is ex-

plicitly written as $(V, \mathcal{R})$, where:

$$\begin{aligned} V = \{&\text{`John Smith',`Jane Bloggs',`Mark Antony',} \\ &\text{`084',`100',`182',`219',} \\ &\text{`Golf',`Sailing',`Squash',`Swimming',`Tennis',} \\ &\text{`\$15',`\$36',`\$40',`\$47',`\$50'}\} \end{aligned}$$

$\mathcal{R}$ consists of the relation names

$$\{\texttt{StudentsTable,ParticipantsTable,ActivitiesTable}\}$$

interpreted as:

$$\begin{aligned} [\texttt{StudentsTable}] = \{&\text{(`John Smith',`084'),(`Jane Bloggs','100'),(`John Smith',`182'),} \\ &\text{(`Mark Antony',`219')}\} \\ [\texttt{ParticipantsTable}] = \{&\text{(`084',`Swimming'),(`084',`Tennis'),(`100',`Squash'),(`100',`Swimming')} \\ &\text{(`182',`Golf'),(`219',`Golf'),(`219',`Swimming'),(`219',`Squash')}\} \\ [\texttt{ActivitiesTable}] = \{&\text{(`Golf',`\$47'),(`Sailing',`\$50'),(`Squash',`\$40')} \\ &\text{(`Swimming',`\$15',),(`Tennis',`\$36')}\} \end{aligned}$$

2.2.1 Conjunctive Queries

Conjunctive Queries are the elements of the relational calculus that are expressible using only existentially quantified variables and conjunctions of relation atoms. In a practical setting, CQs make up a large portion of the queries posed to relational databases. Moreover, this class of queries enjoys many desirable theoretical properties such as the decidability of problems like query evaluation and containment, which is not the case for larger subclasses of the relational calculus.

In relational algebra, CQs are expressed as queries that only use the operators {`select, join, project`}. Within SQL, Conjunctive Queries are expressed using the command `SELECT FROM WHERE` such that the condition that follows only uses conjunctions of equality atoms.

Definition

Let us formalize the notion of Conjunctive Queries. Fix an infinite set of variables $\mathscr{V} = \{x, y, z \dots\}$.

Definition 7. A **Conjunctive Query** (CQ) over a signature $\{R_1, \dots, R_n\}$ is a first order formula of the form

$$q(\bar{x}) = \exists \bar{y} R_1(\bar{z}_1) \wedge \cdots \wedge R_n(\bar{z}_n)$$

where $\bar{x}$ and $\bar{y}$ are pairwise disjoint tuples of variables in $\mathscr{V}$, called **free** and **bound** variables respectively, with tuples $\bar{z}_1, \dots, \bar{z}_n$ from $\bar{x} \cup \bar{y}$ (possibly repeated). Further, $R_1, \dots, R_n$ are called the **relation atoms** of q.

In addition to $\texttt{SameActivity}(x_1, x_2)$ defined on `StudentActivityList` given in Example 7, let us look at some more examples of CQs:

Example 9. Here we have CQs listing the names of all students who sail, and pairs of students (y_1, y_2) who have the same name.

$$\texttt{StudentSailor}(x) = \exists y\, \texttt{StudentsTable}(x, y) \wedge \texttt{ParticipantsTable}(y, \text{`Sailing'})$$
$$\texttt{SameName}(y_1, y_2) = \exists x\, \texttt{StudentsTable}(x, y_1) \wedge \texttt{StudentsTable}(x, y_2)$$

Intuitively, evaluating `StudentSailor` on the database `StudentActivityList` should yield the empty set, because there is no occurrence of 'Sailing' in any entry in `ParticipantsTable`. Meanwhile, evaluating $\texttt{SameName}(y_1, y_2)$ on the same database yields {('084','182')}, with 'John Smith' being the repeated name. Now we formalize the notion of CQ evaluation.

Definition 8. Let q be a CQ with relation atoms $R_1, \ldots, R_n$ and $\mathcal{D} = (V, \mathcal{R})$ be a relational database where $\{R_1, \ldots, R_n\} \subseteq \mathcal{R}$. A **valuation** of q on $\mathcal{D}$ is a function σ from the set of variables of q to V. One can extend σ to tuples of variables in the natural way.

We say that $(\mathcal{D}, \sigma)$ **satisfies** q if and only if there exists a valuation σ of q on $\mathcal{D}$ such that for every relation atom R_i of q, $R_i(\sigma(\bar{z}_i))$ holds in $\mathcal{D}$. We say $\mathcal{D}$ **satisfies** q, written $\mathcal{D} \vDash q$, if there exists some valuation σ such that $(\mathcal{D}, \sigma)$ satisfies q.

Let $\bar{x}_q$ be the tuple of free variables of q. We define the set of *answers* of q evaluated on $\mathcal{D}$ as the set

$$\text{ans}(q, \mathcal{D}) = \{\sigma(\bar{x}_q) \in V^{|\bar{x}_q|} \colon (\mathcal{D}, \sigma) \text{ satisfies } q\}$$

Finally, a CQ is called *binary* if it has exactly two free node variables.

Conjunctive Query evaluation (stated below) is known to be NP-complete [28] in combined complexity, and W[1]-hard in parameterized complexity [91].

Conjunctive Query evaluation (eval-CQ)
Input: A CQ q, a relational database $\mathcal{D} = (V, \mathcal{R})$ and a tuple $\bar{v}$ of elements in V.
Question: Does $\bar{v}$ belong to $\text{ans}(q, \mathcal{D})$?

For simplicity, we only consider *Boolean* CQs, CRPQs, and ECRPQs. Boolean queries are those that do not contain any free variables, thus returning TRUE or FALSE when evaluated on a database. Our results on query evaluation are introduced in Chapter 3 and proved in Chapter 4. Under certain conditions, these results may be extended to non-Boolean ECRPQs with and without free path variables – we discuss this topic in Section 4.4.1.

(Boolean) CQ evaluation
Input: A Boolean CQ q and a relational database $\mathcal{D}$.
Question: Does $\mathcal{D} \vDash q$?

It was shown in [121] that CQs whose *shape* is acyclic have tractable evaluation. This shape is given by the Gaifman graph of a CQ: for any CQ q, its Gaifman abstraction is a graph over its variables which contains an edge $x \to y$ if and only if x, y appear in some atom of q. Chekuri and Rajaraman proved that evaluation is tractable for any class of CQs whose Gaifman graphs have bounded tree-width [29]. Gottlob et al. further improved this result by showing that for all $k \in \mathbb{N}$, evaluating CQs of tree-width k is LogCFL-complete [54].

2.3 Graph databases

Graph databases use graph structures to store and model data. They are used in scenarios where there is a need to name and specify semantic properties on the interrelationships between objects (i.e. paths in the graph) in addition to object properties. One of the earliest instances of this model was in the form of the *graph oriented object database model* (GOOD), introduced by Gyssens et al. [62], who observed that graphs were already an integral part of the design of semantic and object-oriented data models. Aside from having desirable theoretical properties and powerful query formalisms associated with it, the graph data model also finds various real-world applications due to having several advantages over prior data models. We outline some of these applications below.

- The graph data model improves upon prior data models like relational, network and hierarchical databases [71], whose limitations in representing data connectivity, restricted semantics, and difficulty in modeling complex relationships between database objects are addressed by graph databases [76]. Moreover, the query formalisms associated with prior data models suffer from the same restrictions concerning expressivity – graph databases come equipped with better formalisms suited to handle applications of greater complexity [92].

- Compared to relational databases, graph databases are better suited in applications pertaining to spatially embedded networks like roads and public transport [81], and other transportation networks like air routes, telecommunications, and waterworks [61].

- Graph databases offer a higher degree of functionality than object-oriented databases in many applications including CAD, image processing, and scientific data analysis software [99].

- Since the early 2000s, graph databases have addressed the need for managing huge amounts of data connected via networks, in various domains. These are called *complex networks* and have been classified and studied in great depth [89, 39, 6]. These networks are grouped into four categories: social, information, technological, and biological networks. In social networks, a set of persons or groups of people are modeled using nodes linked by various interpersonal relationships [65, 119, 107, 19]. Information networks involve representing the flow of information, most importantly in hypertext networks such as the World Wide Web (see Florescu et al. [50] for a survey) and peer-to-peer networks [88]. Technological networks are those complex networks that are characterized by geographical and/or

spatial components, such as servers and nodes on the Internet, air routes, power grids, delivery networks. These are also sometimes classified under the term *geographical information systems* (GISs) [106, 85]. Biological networks pertain to biological systems, chiefly in the areas of genome sequences, food webs, and neural networks [56, 63, 90].

- The various formalisms of *path queries* on graph databases are used in many applications associated with semistructured data, Semantic Web, and biological sequences [21, 10, 74, 73]. We discuss the applications of path queries in more detail in Sections 2.3.2 and 2.3.4.

We now define the theoretical abstraction of graph databases that we work with. Following that, we define path queries on graph databases, whose evaluation is the topic of our study.

2.3.1 Abstraction with labeled graphs

There are two theoretical models for graph databases: labeled graphs and property graphs [8]. While labeled graphs are graphs whose edges are labeled with letters from a finite alphabet, property graphs extend this formalism by allowing nodes to be labeled and edges to possess additional properties as well. The simpler abstraction for studying path queries is the labeled graph formalism. From now on, we simply consider all graph databases to be labeled graphs over some alphabet $\mathbb{A}$.

Definition 9. A **graph database** over a finite alphabet $\mathbb{A}$ is a tuple $D = (V, E, \eta)$ where (V, E) is an directed graph and $\eta\colon E \to \mathbb{A}$ is a labeling function which assigns to every edge e a label $\eta(e) \in \mathbb{A}$ (where $\mathbb{A}$ is a finite alphabet).

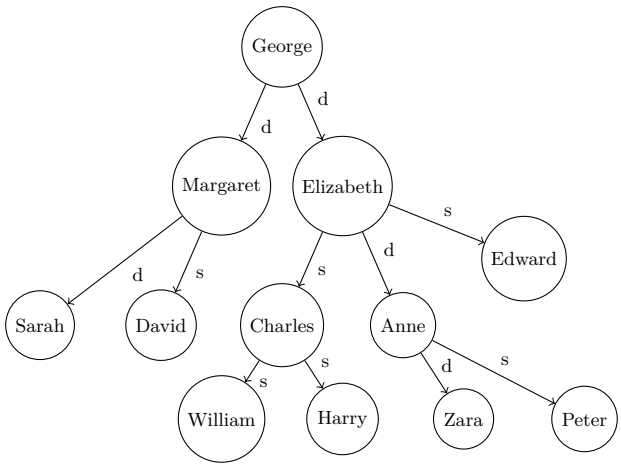
A **non-empty path** is a sequence of edges $(e_1, e_2, \ldots, e_n)$ in E such that for all $1 \leq i \leq n-1$, the edges e_i and e_{i+1} are consecutive, i.e. the target vertex of e_i is the same as the source vertex of e_{i+1}. An **empty path** consists of a single vertex $v \in V$ and contains no edges. The labeling function η is extended to paths

$$\eta\colon E^* \to \mathbb{A}^*$$
$$\eta(e_1 \cdots e_n) = \eta(e_1) \cdots \eta(e_n)$$

thus assigning to every path p a word $\eta(p)$. The label of an empty path is the empty word ε.

Let us look at an example of a graph database represented as a labeled graph. Our example database is a family tree where the nodes represent persons, and the edges labeled by s and d stand for the relationships 'son' and 'daughter' respectively.

Example 10. Below we have a family tree modeled with a labeled graph `Windsor`:



The above graph has 12 nodes and 11 edges. Therefore, it expresses 11 'basic atoms' of information, each concerning a pair of nodes and whether the edge between them is labeled by s or d.

2.3.2 Path Queries on graph databases

A crucial feature of graph database query languages is the ability to extract pairs of nodes connected according to some rule or condition. This is the essential function of a Regular Path Query.

Definition 10. An **RPQ** is a formula

$$q(x, y) = \exists \pi \, x \xrightarrow{\pi} y \wedge L(\pi)$$

where x and y are node variables, π is a path variable, and L is a language given by some regular expression.

RPQs were proposed as a tool for information extraction and knowledge representation in large semistructured databases such as the World Wide Web, which is not characterized by a well-defined schema (Buneman 1997 [21], Calvanese et al. 1999 [23]). Since their introduction, RPQs and their extensions have come to play a similar role in graph databases as CQs do in relational databases [26, 5, 2].

Example 11. Recall the graph database `Windsor` defined in Example 10, where the edge labels s and d refer to 'son' and 'daughter' respectively. Formally, we represent these relationships with the RPQs:

$$\begin{aligned}
\texttt{isSon}(x, y) &= \exists \pi \, x \xrightarrow{\pi} y \wedge L_s(\pi); L_s = s \\
\texttt{isDaughter}(x, y) &= \exists \pi \, x \xrightarrow{\pi} y \wedge L_d(\pi); L_d = d \\
\texttt{isAncestor}(x, y) &= \exists \pi \, x \xrightarrow{\pi} y \wedge L_1(\pi); L_1 = (s + d)^+
\end{aligned}$$

Informally, a graph database *satisfies* an RPQ if it contains a pair of nodes and a path between them labeled by the given regular language (as defined in the RPQ). As our topic of study is ECRPQs, which is a much more general

formalism containing RPQs, we skip the formal definition of RPQ semantics for now. Definition 12 covers the syntax and semantics of ECRPQs.

The output of an RPQ on a graph database is limited to pairs of nodes. In order to extract larger tuples of nodes with more complex interconnections, conjunctions of RPQs with shared node variables are considered. This querying formalism, called Conjunctive Regular Path Queries (CRPQs), serves as a common extension to RPQs as well as CQs.

Definition 11. A **Conjunctive Regular Path Query**, or CRPQ, is a formula

$$q(\bar{x}) = \exists \bar{y} \, \exists \bar{\pi} \, z_1 \xrightarrow{\pi_1} z_1' \wedge \cdots \wedge z_n \xrightarrow{\pi_n} z_n' \wedge L_1(\pi_1) \wedge \cdots \wedge L_n(\pi_n)$$

where the π_i's are pairwise distinct, z_i's are among the x_i's and y_i's and L_i's are regular languages. Further, the variables in $\bar{x}$ are called **free** variables and those in $\bar{y}_i$ are called **bound** variables. We denote by **CRPQ** the set of all CRPQ.

Although CQs and CRPQs operate on different data models, CRPQs contain CQs in terms of expressivity. To demonstrate this fact, consider a single relation CQ, say $q = \exists \bar{z} R(z_1, \ldots, z_n)$. Then the CRPQ (over the singleton alphabet $\{R\}$)

$$\hat{q} = \exists \bar{x} \, \exists \bar{y} \, x_1 \xrightarrow{R} y_1 \wedge \cdots \wedge x_n \xrightarrow{R} y_n$$

is "equivalent to" q, that is, evaluating q over a relational database D is equivalent to evaluating $\hat{q}$ over a graph database $\hat{D}$ via a polynomial time transformation $D \to \hat{D}$. In general, if a CQ has ℓ relations and k variables, then the corresponding CRPQ translation has $2k$ variables, over an alphabet of size ℓ consisting of the relation names of the CQ. We formally define this transformation in Chapter 4 (see Corollary 1) and show that it can be achieved in polynomial time.

RPQs and CRPQs may be evaluated under different semantics. The default mode of evaluating path queries is with *arbitrary path semantics*: the query return pairs of nodes connected by any path whose label belongs to the given regular expression. A CRPQ evaluated under *trail semantics* only output nodes connected by trails – paths with no repeated edges – whose labels satisfy the regular conditions. A further restriction is to consider only simple paths, i.e. paths without any repeating nodes; this is called simple path semantics. The choice of semantics plays a role in the complexity of query evaluation. For example, RPQ evaluation is NL-complete under arbitrary path semantics, but NP-complete under trail and simple path semantics [13, 83].

The default semantics for CRPQ evaluation is understood intuitively from the semantics of CQ evaluation – the relational atoms in a CQ are replaced by reachability atoms in a CRPQ. However, we formally define it in Definition 14, where we contextualize CRPQs as a subclass of the more general formalism of Extended CRPQs.

Example 12. The following CRPQ, when evaluated on `Windsor`, returns triples (x, y, z) such that x is some ancestor of y and z.

$$\texttt{commonAncestor}(x, y, z) = \exists \pi_1 \, \exists \pi_2 \, x \xrightarrow{\pi_1} y \wedge x \xrightarrow{\pi_2} z \wedge L(\pi_1) \wedge L(\pi_2); L = (s + d)^+ \blacksquare$$

.

There are several ways in which CRPQs can be extended to obtain more expressive querying formalisms.

2.3.3 Extending CRPQs with two-wayness and union

The disjunction of n CRPQs $\{q_1, q_2, \ldots, q_n\}$ – all operating on the same set of node variables $\{x_1, \ldots, x_m\}$ – is written as $q_1 \vee \cdots \vee q_n$. This is an instance of an extended query formalism called UCRPQ (which stands for *unions of* CRPQs). The interpretation of $q = q_1 \vee \cdots \vee q_n$ on a graph database $D = (G, E, \eta)$ is as follows: a tuple of nodes $(v_1, \ldots, v_m) \in G^m$ is in $ans(q, D)$ if and only if $(v_1, \ldots, v_m) \in ans(q_i, D)$ for some $i \in \{1, \ldots, n\}$.

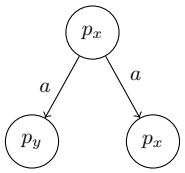
To see that UCRPQs are strictly more expressive than CRPQs, consider two CRPQs

$$q_1(x, y, z) = x \xrightarrow{a} y \wedge x \xrightarrow{b} z$$
$$q_2(x, y, z) = x \xrightarrow{b} y \wedge x \xrightarrow{a} z$$

We show by contradiction that $q_1 \vee q_2$ is not equivalent to any CRPQ. Assume there exists a CRPQ q that produces the same set of answers on every graph database D as $q_1 \vee q_2$. Then it must contain x, y, z as its free node variables, with conditions on paths connecting x to y and x to z, with no "intermediate" quantified node variables (since the conditions on paths in q_1 and q_2 force their length to be 1). Therefore it must be of the form

$$q(x, y, z) = x \xrightarrow{\pi_1} y \wedge x \xrightarrow{\pi_2} z \wedge L_1(\pi_1) \wedge L_2(\pi_2)$$

Moreover, both L_1 and L_2 must contain $\{a, b\}$ as a subset. Therefore the following graph database



satisfies q but not $q_1 \vee q_2$. It follows that $q_1 \vee q_2$ is not equivalent to any CRPQ. Thus, UCRPQs are stronger than CRPQs in expressive power. Formally, a UCRPQ is a query of the form $Q_1 \vee Q_2 \vee \cdots \vee Q_n$, where $Q_1, Q_2, \ldots, Q_n$ are CRPQs with shared free variables, all using relations of the same arity. This is a standard extension and all the major results pertaining to CRPQ decision problems also hold for the UCRPQ formalism [24].

Models of RPQs with two-way or backward traversal of paths have also been studied in depth [24, 25]. A 2RPQ over an alphabet $\mathbb{A}$ uses regular expressions over $\mathbb{A} \dot{\cup} \mathbb{A}^-$, where $\mathbb{A}^- = \{a^-\}_{a \in \mathbb{A}}$ is a copy of $\mathbb{A}$ containing 'reverse' letters. The occurrence of a letter a in a path is seen as a forward step reading a in the

graph database, whereas reading a^- amounts to traveling one step back along the edge reading a. For example, the 2RPQ

$$x \xrightarrow{(a+b^-)^*} y$$

says that y can be reached from x via a path which reads a going forwards and b going backwards. This mode of traversal in a graph database makes this formalism much more useful in real-world scenarios, especially in the context of dealing with unstructured and semistructured data [22, 21, 87]. Similar to its one-way counterpart, the 2RPQ formalism is also extended using conjunction (Conjunctive 2RPQs or C2RPQs) and further, their closure under finite union to produce UC2RPQs. Decision problems on these formalisms using two-way automata have also been explored in detail [24].

2.3.4 Word relations in path queries

The regular expressions used in a CRPQ impose conditions on path labels somewhat independently – each path label is in a given regular language or not, with no relation to the other paths. Therefore, a natural extension of CRPQs is to allow path labels to conform to a given word relation. For example, to the reachability condition

$$\text{reach}(x, y_1, y_2) = \exists \pi_1 \, \exists \pi_2 \, x \xrightarrow{\pi_1} y_1 \wedge x \xrightarrow{\pi_2} y_2$$

we may add the condition of a binary relation $\hat{R}$ on path labels

$$\text{rel}(\pi_1, \pi_2) = (\mathit{label}(\pi_1), \mathit{label}(\pi_2)) \in \hat{R}, \text{ where}$$
$$\hat{R} = (a\mathbb{A}^* \times b\mathbb{A}^*) \cup (b\mathbb{A}^* \times a\mathbb{A}^*)$$

The resulting query

$$\hat{q} = \exists \pi_1 \, \exists \pi_2 \, x \xrightarrow{\pi_1} y_1 \wedge \xrightarrow{\pi_2} y_2 \wedge \hat{R}((\mathit{label}(\pi_1), \mathit{label}(\pi_2)))$$

can be seen as a member of the class "CRPQ$+\{\hat{R}\}$" whose syntax extends that of CRPQ with a relation symbol $\hat{R}$, and whose semantics can be understood intuitively. We call $\text{reach}(x, y_1, y_2)$ the *reachability part* of $\hat{q}$, and $\text{rel}(\pi_1, \pi_2)$ its *relational part*. Evaluated over a graph database, $\hat{q}$ returns triples (x, y, z) of nodes such that there are paths (π_1, π_2) from x to y_1 and x to y_2 whose labels begin with (a, b) or (b, a). Note that this property can be expressed in a UCRPQ as well. In fact, each query in CRPQ$+\{\hat{R}\}$ is equivalent to a UCRPQ.

Example 13. Consider the CRPQ

$$\exists \pi_1 \, \exists \pi_2 \, x \xrightarrow{\pi_1} y \wedge x \xrightarrow{\pi_2} z \wedge L(\pi) \wedge L(\pi_2); L = (s + d)^*$$

over the alphabet $\{s, d\}$, to be evaluated on the graph database `Windsor` given in Example 10. Let us denote by $\mathsf{equal\text{-}length}(\pi_1, \pi_2)$ the relation which is true if and only if the paths π_1, π_2 have equal length. Then the query

$$\texttt{equal-ancestor-depth}(x, y, z) = \exists \pi_1 \, \exists \pi_2 \, x \xrightarrow{\pi_1} y \wedge x \xrightarrow{\pi_2} z \wedge L(\pi) \wedge L(\pi_2) \wedge \mathsf{equal\text{-}length}(\pi_1, \pi_2)$$

is a member of CRPQ+{equal-length}. The query `equal-ancestor-depth` collects all (x, y, z) such that x is a common ancestor of y and z of the same depth, i.e. y, z are both children of x, or grandchildren of x, or great-grandchildren of x, and so on. ■

This notion is further lifted by (i) by allowing relations of arbitrary arity, and (ii) by allowing sets of relations. For a set of relations $\mathcal{R}$, the formalism CRPQ+$\mathcal{R}$ contains queries of the form

$$\exists \pi_1 \ldots \exists \pi_n \, x_1 \xrightarrow{\pi_1} y_1 \wedge \cdots \wedge x_n \xrightarrow{\pi_n} y_n \wedge R_1(\bar{\pi}_1) \wedge \cdots \wedge R_m(\bar{\pi}_m)$$

where each $\bar{\pi}_i$ ranges over a arity(R_i)-sized subset of $\{\pi_1, \ldots, \pi_n\}$ and $R_1, \ldots, R_m \in \mathcal{R}$. Recall that taking direct products of regular languages and closing them under union gives us the class Rec of *recognizable relations*, and the relation $\hat{R}$ used in $\hat{q}$ (defined above) is in this class.

It is a folklore result that UCRPQ contains CRPQ+Rec in terms of expressivity. Given that Rec does not even contain the equality relation $\{(w, w) \colon w \in \mathbb{A}^*\}$, it is an apparently weak class. More expressive classes of word relations have been considered to extend CRPQs. As defined earlier in Chapter 1 (see Section 1.5.5), rational relations (denoted by Rat) constitute a superclass of Rec with many useful relations like subsequence, prefix and suffix. Therefore CRPQ+Rat is a powerful class of queries in terms of expressivity. However, Rat is an unwieldy class of relations in many respects. It is not closed under intersection. Many decision problems involving rational relations are undecidable, including the evaluation problem for CRPQ+Rat [15].

When extending CRPQs with relations a balance must be struck between efficiency of query evaluation and expressive power of the resulting formalism. The class of synchronous relations (denoted as Sync) serves as an ideal candidate for this purpose. We define synchronous relations in Chapter 1 (see Section 1.5.5). Considered the analogue of Reg in higher dimensions, Sync is strictly sandwiched between recognizable and rational relations

$$\textsc{Rec} \subsetneq \textsc{Sync} \subsetneq \textsc{Rat}$$

containing relations like equality and prefix but not suffix and subsequence. Moreover, it is a Boolean algebra with robust properties like closure under projection and homomorphism. It also admits a first-order logical characterization, whose study is the subject of Part II (see Chapter 5). The query formalism CRPQ+Sync is also called Extended CRPQ, or ECRPQ. The properties of this query formalism make it suitable for application in a variety of fields, including approximate sequence matching [55] and computational biology [59].

Definition 12. An **Extended Conjunctive Regular Path Query** (ECRPQ) is a formula q with free variables $\bar{x}$ of the form

$$q(\bar{x}) = \exists \bar{y} \, \exists \bar{\pi} \, q_{\text{reach}}(\bar{x}, \bar{y}, \bar{\pi}) \wedge q_{\text{rel}}(\bar{\pi})$$

where

- q_{reach}, called the **reachability subquery of** q, is a formula

  $$z_1 \xrightarrow{\pi_1} z_1' \wedge z_2 \xrightarrow{\pi_2} z_2' \wedge \cdots \wedge z_n \xrightarrow{\pi_n} z_n'$$

  where $\bar{z}, \bar{z}'$ are variables ranging over $\bar{x} \cup \bar{y}$ (not necessarily pairwise distinct), and for all $1 \leq i \leq n$, π_i appears in exactly one atom in q_{reach}.

- q_{rel}, called the **relation subquery of** q, is a formula

  $$R_1(\bar{\pi}_1) \wedge \cdots \wedge R_m(\bar{\pi}_m)$$

  where each $\bar{\pi}_i \subseteq \bar{\pi}$ is a tuple of size $\text{arity}(R_i)$ of pairwise distinct path variables. Furthermore, $R_1, \ldots, R_m$ are relation symbols associated to synchronous relations, each relation R_i given as an NFA over $(\mathbb{A} \dot{\cup} \{\perp\})^{\text{arity}(R_i)}$, for some fixed alphabet $\mathbb{A}$.

In their original definition, ECRPQs allow path variables to be free, thus permitting the output of paths along with nodes when evaluated on graph databases. Since paths can be arbitrarily long, this poses some difficulties in producing a finite output. Therefore, for simplicity, we quantify all path variables and treat paths as implicit objects. We can now define CRPQs as special cases of ECRPQs, as follows:

Definition 13 (Alternative definition for CRPQs)**.** A **Conjunctive Regular Path Query** is an ECRPQ q such that

- Every relation atom in q_{rel} has arity 1, i.e. each is a regular language.
- Each path variable appears in exactly one atom of q_{rel}.

The query `equal-ancestor-depth` given in Example 13 is a member of CRPQ + Sync, as the relation equal-length used in it is synchronous. Therefore `equal-ancestor-depth` is an ECRPQ. We now define ECRPQ semantics before looking at more examples.

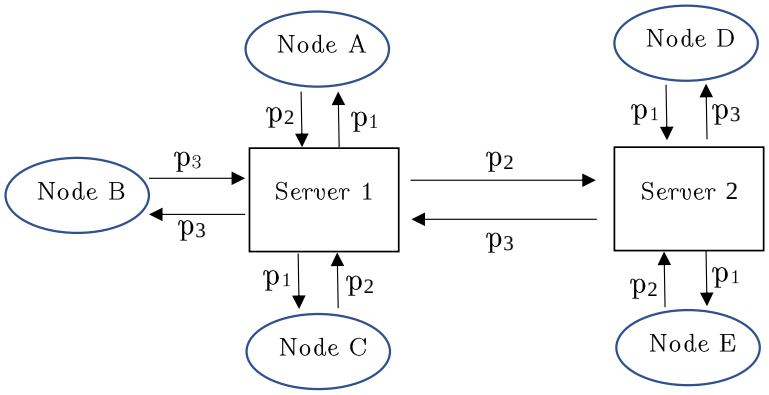
Definition 14. Let $D = (V, E, \eta)$ be a graph database and q be an ECRPQ with node variables $\bar{x} \cup \bar{y}$ and path variables $\bar{\pi}$. A **valuation** of q on D is a pair of functions $\sigma = (\sigma_{\text{node}}, \sigma_{\text{path}})$ where σ_{node} sends node variables to elements in V and σ_{path} sends variables in $\bar{\pi}$ to paths in E^*. We further extend σ_{node} to tuples of node variables in the usual way. Now, we say (D, σ) **satisfies** q, if:

- For all atoms $x \xrightarrow{\pi} y$ in q_{reach}, $\sigma_{\text{path}}(\pi)$ is a path from $\sigma_{\text{node}}(x)$ to $\sigma_{\text{node}}(y)$.
- for every relation symbol $R(\pi_1, \ldots, \pi_m)$ in q_{rel}, $(\eta(\sigma_{\text{path}}(\pi_1)), \ldots, \eta(\sigma_{\text{path}}(\pi_m))) \in R$.

If there exists some valuation σ for which (D, σ) satisfies q for some valuation σ, we say D **satisfies** q and denote it as $D \vDash q$. Moreover, σ is said to **witness** $D \vDash q$. Let $\bar{x}_q$ be the tuple of free (node) variables of q. Then the set of answers of q evaluated on D, denoted $ans(q, D)$, is defined as:

$$\{\sigma_{\text{node}}(\bar{x}_q) \in V^{|\bar{x}_q|} : (D, \sigma) \text{ satisfies } q\}$$

Example 14. Let us consider a network of 5 nodes and 2 servers communicating via processes $\{p_1, p_2, p_3\}$, modeled using the following graph database D:



John wants to know if there exists some path in this network which reads only p_1 and p_2. So he queries the database with the following CRPQ:

$$q = \exists \pi\, x \xrightarrow{\pi} y \wedge L_1(\pi); L_1 = (p_1 + p_2)^*$$

The algorithm returns all vertices (x, y) such that a path from x to y labeled by $\{p_1, p_2\}^*$ exists in D. These include the pairs

(Node A,Node A),(Node A,Node C),(Server 2,Server 2)

Emily wants to know if there exist pairs of paths π_1, π_2 such that

- π_1 goes from x to y and π_2 goes from y to x

- π_1 only reads p_1, p_2, π_2 only reads p_2, p_3
- π_1 and π_2 have the same length

She modifies John's query by adding an equal length condition to the paths, to produce an ECRPQ q' as follows:

$$q' = \exists\pi_1\, \exists\pi_2\, x \xrightarrow{\pi_1} y \wedge y \xrightarrow{\pi_2} x \wedge L_1(\pi_1) \wedge L_2(\pi_2) \wedge \mathsf{eq\text{-}length}(\pi_1, \pi_2);$$
$$L_1 = (p_1 + p_2)^*, L_2 = (p_2 + p_3)^*$$

The pair (Node A,Node E) is in $ans(q, D)$ due to the existence of the paths

$$\pi_1 = \text{Node A} \xrightarrow{p_1} \text{Server 1} \xrightarrow{p_2} \text{Server 2} \xrightarrow{p_1} \text{Node E}$$
$$\pi_2 = \text{Node E} \xrightarrow{p_2} \text{Server 2} \xrightarrow{p_3} \text{Server 1} \xrightarrow{p_2} \text{Node A} \quad \blacksquare$$

Boolean queries

It was mentioned earlier that (i) a CQ or ECRPQ with no free variables is called a *Boolean* query, and (ii) for technical brevity we shall only work with Boolean CQs and ECRPQs. However, all our results can be easily extended to non-Boolean ECRPQs (see Section 4.4.1 in Chapter 4 for a discussion on this). The Boolean ECRPQ evaluation problem is formally stated as:

(Boolean) ECRPQ-evaluation (eval-ECRPQ)
Input: An ECRPQ q and a graph database D.
Question: Does $D \models q$?

We also consider the parameterized version of the Boolean ECRPQ problem, which is stated as:

(Boolean) Parameterized ECRPQ evaluation (p-eval-ECRPQ)
Input: An ECRPQ q and a graph database D.
Question: Does $D \models q$?
Parameter: The size of the query, $|q|$.

2.4 ECRPQ evaluation

Before we discuss the evaluation problem for ECRPQs, let us look at the state-of-the-art of the general query evaluation problem for various formalisms, with respect to the different complexity measures (introduced in Section 1.4 of Chapter 1). We know that:

- eval-ECRPQ is PSpace-complete [15], whereas eval-CQ and eval-CRPQ are both NP-complete [28].
- p-eval-CQ is W[1]-complete [28] and p-eval-CRPQ is W[1]-hard.
- Data complexity of CQ evaluation is in LogSpace [28], whereas the data complexity of CRPQ and ECRPQ evaluation is NL-complete [15].

The evaluation problem for ECRPQs was tackled by Libkin et al. [15]. The strategy used to evaluate an ECRPQ q on a graph database D was to construct a product database D^n and a CRPQ q', such that the evaluation of q' on D^n coincided with the evaluation of q on D. Here n is the maximum arity of relations used in q. In other words, using elements of $\text{ans}(q', D^n)$, the set $\text{ans}(q, D)$ can be constructed. This is a PSPACE procedure in combined complexity, while keeping the data complexity unchanged from that of CRPQ evaluation (NL-complete).

In the case of both CQs and CRPQs there exist abstractions that allow us to simplify the evaluation problem. What do we mean by this? Abstractions allow us to simplify a query to some essential shape or form. Then, by restricting the structural properties of this shape, we can extract sub-classes of queries that may be limited in their expressivity but yield easy query evaluation, owing to their well-defined structural properties.

First, we will discuss these abstractions and see if they can shape our intuition in simplifying EVAL-ECRPQ. We consider the Gaifman abstraction for CQs and multi-graph abstraction for CRPQs, beginning with the formal definitions of these graph structures.

Definition 15. A **multi-hypergraph** is a structure $G = (V, H, \eta)$ where V is a finite set of vertices, H is a finite set of hyper-edges and $\eta\colon H \to \mathscr{P}(V)$. If $\text{range}(\eta) \subseteq \mathscr{P}_1(V) \cup \mathscr{P}_2(V)$, then we say G is a **multigraph**.

If further η is injective, then we say G is a **simple graph**, writing it as $G = (V, E)$, where $E = \{\eta(h)\colon h \in H\}$.

Abstracting CQs with Gaifman graphs

Consider a CQ q with a set of variables V. Let G_q be the graph whose vertices are V, with the edge set

$$\{x \to y\colon x \text{ and } y \text{ appear in some relational atom of } q\}$$

Then G_q is called the *Gaifman graph* or the *Gaifman abstraction* of q. Formally,

Definition 16. Let $q \in$ **CQ**. A graph $G = (V, E)$ is said to be the **Gaifman abstraction** of q, or simply the **abstraction** of q if and only if:

- V is the set of variables of q.
- $E = \{\{x, y\}\colon x \text{ and } y \text{ appear in some relational atom of } q\}$.

Let $\mathcal{C}$ be any class of simple graphs. We define $\text{CQ}(\mathcal{C})$ to be the set of all CQs q whose Gaifman abstraction is in $\mathcal{C}$.

Then the corresponding query evaluation problem EVAL-$\text{CQ}(\mathcal{C})$ takes as input (q, D) where q is CQ whose Gaifman graph is in $\mathcal{C}$ and a D is a relational database, and determines whether $D \vDash q$.

Why are Gaifman graphs important? It turns out that query evaluation is in polynomial time for classes of CQs whose Gaifman graphs have bounded tree-width [29, 70]. Since this result, the Gaifman abstraction as well as other graph-based abstractions for CQs have been explored in great detail. We will formally define tree-width and elaborate on these results in the next chapter.

Multi-graph abstraction for CRPQs

For any CRPQ q let G_q be the multigraph which has an edge (x, y) labeled π if and only if $x \xrightarrow{\pi} y$ is an atom in q_{reach}. Then G_q is called the *multi-graph abstraction* of q. Now, for a class $\mathcal{C}$ of multi-graphs we have its corresponding class of queries

$$\text{CRPQ}(\mathcal{C}) = \{q \in \textbf{CRPQ} \colon G_q \in \mathcal{C}\}$$

as well as the corresponding decision problem EVAL-CRPQ($\mathcal{C}$). It was shown in [15] that using the previously established result of tractability of bounded tree-width conjunctive queries we can obtain tractability of bounded tree-width CRPQs as well. The following is also proved in the present work, as a corollary to Proposition 2 in Section 4.2 of Chapter 4.

Abstracting ECRPQs

We introduce a data structure called a *two-level graph*, or *2L-graph*, which captures the underlying structure of ECRPQs. Assuming $\mathcal{C}$ is a class of 2L-graphs, ECRPQ($\mathcal{C}$) denotes the set of queries whose abstractions lie in $\mathcal{C}$, and EVAL-ECRPQ($\mathcal{C}$) denotes the corresponding decision problem. We aim to find a measure for $\mathcal{C}$ (similar to tree-width for graphs) so that we can make an assertion of the form

EVAL-ECRPQ($\mathcal{C}$) is tractable if and only if $\langle$measure$\rangle(\mathcal{C})$ is bounded

We define two-level graphs and their measures, and state our results in the next chapter (Chapter 3). We provide full proofs of our theorems in Chapter 4.

Chapter 3

A two-level abstraction for ECRPQs

At the end of the previous chapter we talked about a two-level structure for abstracting ECRPQs with respect to their evaluation. We dubbed these structures *2L-graphs*, short for two-level multi-hyper-graphs. This chapter formally introduces 2L-graphs and the measures which play a role in ECRPQ evaluation. We first state our results. Following that, we examine the underlying structure of an ECRPQ. Next, we formally define 2L-graphs and show how they abstract ECRPQs. We also define the corresponding graph measures. We end the chapter with a discussion of our results, as well as the reductions used in them. Chapter 4 contains the full proofs of these theorems, and concludes Part I of this thesis.

Recall that, for simplicity, we work only with Boolean queries (among CQs, CRPQs, and ECRPQs). These results can be extended to non-Boolean queries as well. A discussion of this along with other easy extensions of our results can be found in Section 4.4 of Chapter 4. We now state our results.

Evaluation theorems

Informally, a 2L-graph is a structure that has three components: a set of vertices V, a set of "first level" edges $E \subseteq V \times V$ connecting those vertices, and a set of "second level" hyperedges $H \subseteq 2^E$ spanning over first level edges. We say that this 2L-graph"abstracts" an ECRPQ q if there exists a mapping which associates the node variables of q to V, its path variables to E, and its relations to H, such that this mapping respects the structure of the query. The formal definition of 2L-graphs and ECRPQ abstractions can be found in Definitions 17, 18, in Section 3.1.

Recall that for every 2L-graph G, we denote by ECRPQ(G) the set of ECRPQs abstracted by it. Similarly, a class of 2L-graphs $\mathcal{C}$ the set of ECRPQ abstracted by its elements is denoted as ECRPQ($\mathcal{C}$).

Given a 2L-graph G, we associate to it the quantities $\mathrm{cc}_{\mathrm{vertex}}(G), \mathrm{cc}_{\text{h-edge}}(G) \in \mathbb{N}$, which we call *measures*. These are dependent upon the first and second level edges of G, and we cover them formally in Section 3.2.1 (see Definition

20). Moreover, we associate a notion of *treewidth* of a 2L-graph, written as $tw(G) \in \mathbb{N}$. These measures are lifted to classes of 2L-graphs in the natural way: for $\star \in \{\text{cc}_{\text{vertex}}, \text{cc}_{\text{h-edge}}, tw\}$, we define $\star(\mathcal{C})$ to be the maximum of $\{\star(G)\}_{G \in \mathcal{C}}$ whenever it exists, and ∞ otherwise. Now our results can be stated as follows:

Theorem. *[Combined complexity] Let $\mathcal{C}$ be a class of 2L-graphs. Under some mild assumption*[1] *on $\mathcal{C}$, and assuming* W[1] $\neq$ FPT*, we have:*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ or $cc_{h\text{-}edge}(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) *is* PSPACE*-complete,*

(2) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) *is in* NP *and not in polynomial time, and*

(3) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}), tw(\mathcal{C}) < \infty$ then EVAL-ECRPQ($\mathcal{C}$) *is in polynomial time.*

Theorem. *[Parameterized complexity] Let $\mathcal{C}$ be a class of 2L-graphs. Assuming* W[1] $\neq$ FPT*, we have:*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ then P-EVAL-ECRPQ($\mathcal{C}$) *is* XNL*-complete,*

(2) if $cc_{vertex}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}) = \infty$, then P-EVAL-ECRPQ($\mathcal{C}$) *is* W[1]*-complete,*

(3) if $cc_{vertex}(\mathcal{C}), tw(\mathcal{C}) < \infty$ then P-EVAL-ECRPQ($\mathcal{C}$) *is* FPT*.*

These results are formally stated towards the end of this chapter, in Section 3.3.

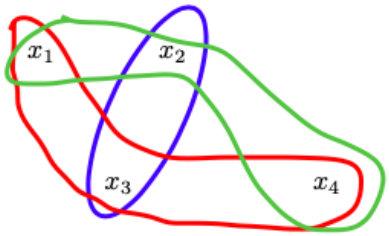
3.1 Underlying structure of ECRPQs

Recall that in an ECRPQ, every path variable appears in exactly one atom in the reachability subquery, but it may be re-used in any number of relations in the relational subquery. To understand the underlying structure of ECRPQs, we first look at the way the relational structure of a CQ is abstracted with hypergraphs. Note that this abstraction is different from the Gaifman graph abstraction for CQs.

The hypergraph-based abstraction of CQs works as follows: a CQ over the variables $x_1, \ldots, x_n$ with relations $R_1, \ldots, R_m$ is abstracted by a hypergraph whose nodes are $x_1, \ldots, x_n$, with hyperedges $h_1, \ldots, h_m$ such that each h_i corresponds to R_i and consists of the variables over which R_i ranges.

Example 15. The CQ $\exists x_1, x_2, x_3, x_4\, R_1(x_2, x_3) \wedge R_2(x_1, x_3, x_4) \wedge R_3(x_1, x_2, x_4)$ is abstracted by the hypergraph

[1] this is called cc-tameness and we cover it in Section 3.2.1 (see Definition 21)



where hyperedges are denoted by the colors of their corresponding relations.

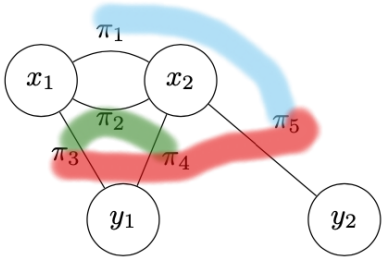
In the case of CRPQs, abstracting away the regular languages yields the underlying graph, whose nodes are node variables of the CRPQ, and edges are named by the paths between those nodes as specified in the CRPQ.

An ECRPQ contains a reachability subquery and a relational subquery. The reachability subquery can be simply seen as a trivial CRPQ, that is, a CRPQ which only uses $\mathbb{A}^*$ in its regular expressions. Therefore a multigraph is sufficient to abstract this subquery. This forms the "first level" of the 2L-graph structure, i.e. a set of nodes with level one edges connecting them. Next, the relational subquery is a conjunction of relations, each ranging over a subset of the path variables. Similar to the hypergraph abstraction of CQs, these relations are represented as hyperedges over the path variables. In other words, we have "second level" hyperedges covering first level edges. This is the idea behind 2L-graphs. We illustrate this underlying structure in Example 16 below.

Example 16. Let $q = q_{\text{reach}} \wedge q_{\text{rel}}$ be an ECRPQ where

$$\begin{aligned} q_{\text{reach}} &= x_1 \xrightarrow{\pi_1} y_1 \wedge x_1 \xrightarrow{\pi_2} y_1 \wedge y_1 \xrightarrow{\pi_3} x_1 \\ &\quad \wedge y_1 \xrightarrow{\pi_4} x_2 \wedge x_2 \xrightarrow{\pi_5} y_2 \\ q_{\text{rel}} &= R_1(\pi_1, \pi_5) \wedge R_2(\pi_2, \pi_3, \pi_4) \wedge R_3(\pi_3, \pi_4, \pi_5) \end{aligned}$$

Then q can be represented with a graph-based structure G_q:



3.1.1 Defining 2L-graphs

We now define 2L-graphs formally.

Definition 17. A **two-level multi-hypergraph**, or **2L-graph** for short, is a structure $G = (V, E, H, \eta, \nu)$ where (V, E, η) is a multigraph and (E, H, ν) is a multi-hypergraph. We consider E to be a set of "first level" edges connecting pairs of elements in V via

$$\eta \colon E \to \mathscr{P}_2(V)$$

and H a set of "second level" hyper-edges connecting (non-empty) sets of elements in E via

$$\nu \colon H \to \mathscr{P}(E) \setminus \{\emptyset\}$$

Moreover, for all $h \in H$ the **size** of h refers to the cardinality of the set $\nu(h)$. (For clarity, we always assume $E \cap H = \emptyset$).

Notation: Unless mentioned otherwise, we will denote vertices of a 2L graph as $\{v_1, v_2, \dots\}$ and first level edges as $\{\pi_1, \pi_2, \dots\}$. In the context of queries, the π_i's also denote path variables in an ECRPQ. However, our goal is to abstract ECRPQs with 2L-graphs, where ECRPQ path variables are identified with their first level edges. So this is not a total abuse of notation.

Example 17. Recall the query q of Example 16 and its graph-based representation G_q. Then G_q is a 2L-graph.

Next, we formalize the abstraction of ECRPQs by 2L-graphs.

Definition 18. Let q be an ECRPQ and G be a 2L-graph. We say $G = (V, E, H, \eta, \nu)$ **abstracts** q, or G **is the abstraction of** q if:

- V is the set of node variables of q.
- E is the set of path variables of q.
- H is the set of relation atoms of q.
- For all $\pi \in E$, $\eta(\pi) = \{z, z'\}$ if and only if $z \xrightarrow{\pi} z'$ is an atom in q_{reach}.
- For all $h \in H$, $\nu(h) = \{\pi_1, \dots, \pi_n\}$ if and only if h corresponds to an atom $R(\pi_1, \dots, \pi_n)$ in q_{rel}.

Example 18. The ECRPQ q of Examples 16 and 17 is abstracted by the 2L-graph G_q.

Proposition 1. *Every 2L-graph abstracts some ECRPQ.*

Proof. Let $G = (V, E, H, \eta, \nu)$ be a 2L-graph. For every $\pi \in E$ such that $\eta(\pi) = \{x, y\}$, let t_π be the atom $x \xrightarrow{\pi} y$. Further, for every $h \in H$ where $\nu(h) = \{\pi_1, \dots, \pi_m\}$, let $R_h = \mathbb{U}_m$ be the universal m-ary relation over $\pi_1, \dots, \pi_m$. Then

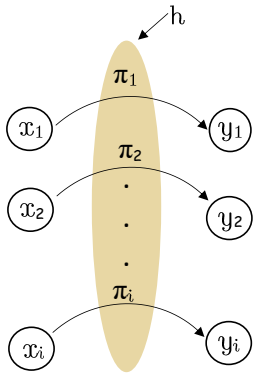
$$q = \bigwedge_{\pi \in E} t_\pi \wedge \bigwedge_{h \in H} R_h$$

is an ECRPQ whose abstraction is G. □

In the following examples, $\mathcal{C}, \mathcal{C}'$ denote classes of 2L-graphs with some carefully chosen parameters. The observations on the number of hyperedges and hyperedge size of the classes will make sense in a later section, when we analyze measures on 2L-graphs, in Section 3.2.

Example 19. Let $\mathcal{C} = \{G_i = (V_i, E_i, H_i, \eta_i, \nu_i)\}_{i \in \mathbb{N}}$ be a set of 2L-graphs such that for every $i \in \mathbb{N}$:

- $V_i = \{x_1, y_1, \ldots, x_i, y_i\}$
- $E_i = \{\pi_1, \ldots, \pi_i\}$ such that $\eta_i(\pi_j) = \{x_j, y_j\}$ for all $1 \leq j \leq i$.
- $H_i = \{h\}$ such that $\nu_i(h) = \{\pi_1, \ldots, \pi_i\}$.



Note that $\mathcal{C}$ is a class of 2L-graphs with

(i) unbounded hyperedge size, i.e., for all $n \in \mathbb{N}$ there exists some 2L-graph $G \in \mathcal{C}$ containing some hyperedge of size at least n, and

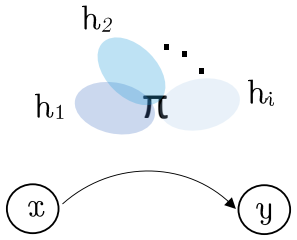
(ii) there exists a bound k on the number of hyperedges in any member of $\mathcal{C}$; in this case $k = 1$. ■

A "skeletal" ECRPQ is defined only with its relational atoms, without specifying the actual synchronous relations corresponding to the atoms. We now note that in Example 19, for any $i \in \mathbb{N}$, G_i abstracts the skeletal ECRPQ

$$\exists \pi_1 \ldots \exists \pi_i \, x_1 \xrightarrow{\pi_1} y_1 \wedge \cdots \wedge x_i \xrightarrow{\pi_i} y_i \wedge R(\pi_1, \ldots, \pi_i)$$

Example 20. Let $\mathcal{C}' = \{G'_i = (V'_i, E'_i, H'_i, \eta'_i, \nu'_i)\}_{i \in \mathbb{N}}$ be a set of 2L-graphs such that for every $i \in \mathbb{N}$:

- $V'_i = \{x, y\}$
- $E'_i = \{\pi\}$ such that $\eta_i(\pi) = \{x, y\}$
- $H'_i = \{h_1, h_2, \ldots, h_i\}$ such that $\nu_i(h_j) = \{\pi\}$ for all $1 \leq j \leq i$.



In contrast to $\mathcal{C}$, we see that $\mathcal{C}'$ has bounded hyperedge size (which is 1), as opposed to the hyperedge size of $\mathcal{C}$ being unbounded. Further, the maximum number of hyperedges in any member of $\mathcal{C}$ was 1, in case of $\mathcal{C}'$ it is unbounded. Each hyperedge is shown in a different shade of blue, covering the path variable π. ■

In Example 20, for every integer i, we note that G_i abstracts the skeletal ECRPQ

$$\exists \pi\, x \xrightarrow{\pi} y \wedge R_1(\pi) \wedge \cdots \wedge R_i(\pi)$$

We now formally define ECRPQ abstraction, and lift this notion to classes of 2L-graphs in the natural way:

Definition 19. Let $\mathcal{C}$ be a class of 2L-graphs and G be an arbitrary member of it. We define:

$$\begin{aligned}
\mathrm{ECRPQ}(G) &= \{q \in \textbf{ECRPQ} \colon G \text{ abstracts } q\} \\
\mathrm{CRPQ}(G) &= \{q \in \textbf{CRPQ} \colon G \text{ abstracts } q\} \\
\mathrm{ECRPQ}(\mathcal{C}) &= \bigcup_{G \in \mathcal{C}} \mathrm{ECRPQ}(G) \\
\mathrm{CRPQ}(\mathcal{C}) &= \bigcup_{G \in \mathcal{C}} \mathrm{CRPQ}(G)
\end{aligned}$$

Our decision problems, with respect to the combined complexity and parameterized complexity measures, are stated as follows:

ECRPQ($\mathcal{C}$) EVALUATION (EVAL-ECRPQ($\mathcal{C}$)); $\mathcal{C}$ is a class of 2L-graphs
Input: A query $q \in \mathrm{ECRPQ}(\mathcal{C})$ and a graph database D.
Question: Does $D \vDash q$?

PARAMETERIZED ECRPQ($\mathcal{C}$) EVALUATION (P-EVAL-ECRPQ($\mathcal{C}$))
Input: A query $q \in \mathrm{ECRPQ}(\mathcal{C})$ and a graph database D.
Question: Does $D \vDash q$?
Parameter: The size of the query, $|q|$.

Recall $\mathcal{C}$ and $\mathcal{C}'$, the classes of 2L-graphs of Examples 19 and 20. Let $\mathcal{Q} = \mathrm{ECRPQ}(\mathcal{C})$ and $\mathcal{Q}' = \mathrm{ECRPQ}(\mathcal{C}')$. What can be said about the complexity of EVAL-$\mathcal{Q}$ and EVAL-$\mathcal{Q}'$? Both are PSPACE-complete, as it turns out from

the analysis of the complexity of the general problem EVAL-ECRPQ. We can make much more general assertions, by introducing measures on 2L-graphs – hyperedge size being an example of one of these measures.

We state our theorems in Section 3.3. For a given class $\mathcal{C}$ of 2L-graphs, we use our 2L-graph measures to specify the conditions under which ECRPQ($\mathcal{C}$) is in PTIME, NP or PSPACE.

3.2 Measures on 2L-graphs

The relevant measure in simplifying EVAL-CQ is the tree-width of the Gaifman graph of a CQ. We now define measures on 2L-graphs to state a similar result for EVAL-ECRPQ.

3.2.1 Relational hypergraph and PSpace-hardness

We begin by showing that EVAL-ECRPQ is PSPACE-hard [15], via a reduction from the INTERSECTION EMPTINESS (IE) for DFA, which is a well-known PSPACE-complete problem [72].

INTERSECTION EMPTINESS (IE)
Input: A set of DFA $\mathcal{S}$
Question: Is $\bigcap_{\mathcal{A}\in\mathcal{S}} L(\mathcal{A}) \neq \emptyset$?

Reduction 1 (Many path variables)**.** Let $\mathcal{A}_1, \ldots, \mathcal{A}_n \subseteq \mathbb{A}^*$ be DFAs. For each $1 \leq i \leq n$, consider the synchronous relation

$$R_i = \mathbb{U}_{i-1}(\mathbb{A}) \times L(\mathcal{A}_i) \times \mathbb{U}_{n-i}(\mathbb{A})$$

where for all $k \in \mathbb{N}$, $\mathbb{U}_k(\mathbb{A})$ is the k-ary universal relation on $\mathbb{A}^*$. A DFA $\mathcal{A}'_i$ for R_i can be constructed trivially: simply ignore every component of the synchronized letters except i and run $\mathcal{A}_i$ on the i'th projection of the input. Now, let q_1 be the ECRPQ

$$\exists \bar{x}\, \exists \bar{y}\, x_1 \xrightarrow{\pi_1} y_1 \wedge \cdots \wedge x_n \xrightarrow{\pi_n} y_n \wedge R_1 \wedge \cdots \wedge R_n \wedge R^n_{\text{eq}}$$

where $R^n_{\text{eq}} = \{(w, \ldots, w) \in \mathbb{U}_n(\mathbb{A}) \colon w \in \mathbb{A}^*\}$. Now the trivial database (which contains a single node v and edges $v \xrightarrow{a} v$ for all $a \in \mathbb{A}$) satisfies q_1 if and only if $L(\mathcal{A}_i) \cap \cdots \cap L(\mathcal{A}_n) \neq \emptyset$. ■

Note that an instance of IE which has n languages requires an ECRPQ with n path variables for this reduction to go through. What if we only allowed ECRPQs with a limited number of path variables? Would the complexity of evaluating such ECRPQs be lower than the general case? Unfortunately, the answer turns out to be no.

Reduction 2 (Single path variable)**.** Consider reducing an instance $\{\mathcal{A}_i\}_{1\leq i\leq n}$ of IE to evaluating the ECRPQ

$$q_2 = \exists x\, \exists y\; x \xrightarrow{\pi} y \wedge L_1(\pi) \wedge \cdots \wedge L_n(\pi)$$

(where for all $1 \leq i \leq n$, $L_i = L(\mathcal{A}_i)$), on the trivial database $D = (\{v\}, \{v \xrightarrow{a} v \colon a \in \mathbb{A}\})$. Here too we have that $D \vDash q_2$ if and only if $L_1 \cap \cdots \cap L_n \neq \emptyset$. ■

Remark 1. Recall the 2L-graph classes $\mathcal{C}, \mathcal{C}'$ defined in Examples 19 and 20. Also recall that the classes of ECRPQ which they abstract are called $\mathcal{Q}, \mathcal{Q}'$. We observe that

- In Reduction 1, the ECRPQ q_1 constructed has a single relation of arbitrary arity (equal to the number of DFAs in the IA instance). The 2L-graphs in $\mathcal{C}$ also have hyperedges of arbitrary size. Therefore, $q_1 \in \text{ECRPQ}(\mathcal{C}) = \mathcal{Q}$.

- Reduction 2 uses an ECRPQ q_2 with arbitrarily many relations (again, equal to the number of DFAs in the IA instance). Meanwhile $\mathcal{C}'$ contains 2L-graphs of arbitrarily many hyperedges. Therefore, $q_2 \in \mathcal{Q}'$.

Therefore, from the PSPACE-completeness of INTERSECTION EMPTINESS [72], we conclude that EVAL-$\mathcal{Q}$ and EVAL-$\mathcal{Q}'$ are both PSPACE-complete. ■

Note that the conditions leading to PSPACE-hardness of $\mathcal{Q}$ and $\mathcal{Q}'$ only depend upon the relational subqueries of the ECRPQs. Therefore, the measures we introduce now will be centered around the *relational hypergraph*, which we define now.

Definition 20. Let q be an ECRPQ with

$$q_{\text{rel}} = R_1(\bar{\pi}_1) \wedge \cdots \wedge R_n(\bar{\pi}_n)$$

Then the **relational hypergraph** of q is the hyper-graph

$$G^q_{\text{rel}} = (\{\pi_1, \ldots, \pi_n\}, \{R_1, \ldots, R_n\}, \nu)$$

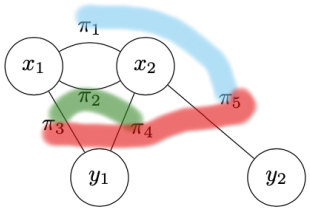
where (i) $\pi_1, \ldots, \pi_n$ are the path variables used in q_{rel}, and (ii) for all $1 \leq i \leq n$, $\nu(R_i) = \{\pi \colon \pi \text{ occurs in } \bar{\pi}_i\}$. Moreover, let C be the set of connected components of G, and for every $c \in C$ let $\mathsf{vertex}(c)$ denote the set of path variables in c and $\mathsf{hyperedge}(c)$ denote the set of hyperedges in c. Then we define

$$\text{cc}_{\text{vertex}}(G) = \max_{c \in C} |\mathsf{vertex}(c)|$$
$$\text{cc}_{\text{h-edge}}(G) = \max_{c \in C} |\mathsf{hyperedge}(c)|$$

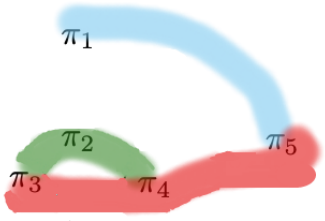
Furthermore, for any class of 2L-graphs $\mathcal{C}$, let

$$\text{cc}_{\text{vertex}}(\mathcal{C}) = \sup_{G \in \mathcal{C}} \text{cc}_{\text{vertex}}(G) \text{ (if it exists, and } \infty \text{ otherwise)}$$
$$\text{cc}_{\text{h-edge}}(\mathcal{C}) = \sup_{G \in \mathcal{C}} \text{cc}_{\text{h-edge}} \text{ (if it exists, and } \infty \text{ otherwise)}$$

Example 21. Recall the ECRPQ q in Example 16 whose 2L-abstraction G_q is



Then the relational hypergraph of q, shown here



has one connected component with five vertices and three hyperedges. Therefore $\mathrm{cc}_{\text{vertex}}(G_q) = 5, \mathrm{cc}_{\text{h-edge}}(G_q) = 3$.

Clearly, the evaluation problem is no longer PSPACE-hard, when considering classes of 2L-graphs with bounded $\mathrm{cc}_{\text{vertex}}$ and $\mathrm{cc}_{\text{h-edge}}$. However, this is not a sufficient condition for tractability. In order to evaluate ECRPQ($\mathcal{C}$) in polynomial time, we also need the additional condition of bounded treewidth. In the case of a CQ or a CRPQ, the notion of tree-width is simple enough and relies on the Gaifman graph of the query. However, in the case of ECRPQs and 2L-graphs, we find that the appropriate tree-width notion relies on both the reachability and the relational subquery.

While stating our results in the chapter introduction, we mentioned a "mild assumption" on 2L-graphs. This is called the *cc-tameness* property.

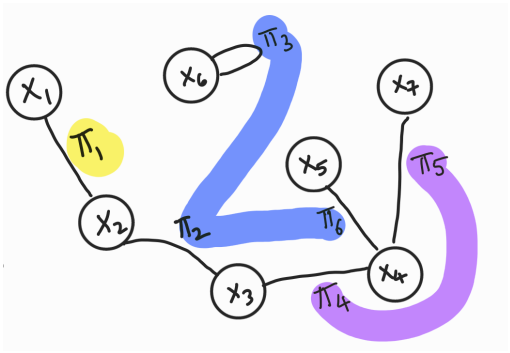
Definition 21. We say a class $\mathcal{C}$ of 2L-graphs is **cc-tame** if there exists a function $f\colon \mathbb{N} \to \mathcal{C}$ computable in polynomial time such that either $\mathrm{cc}_{\text{vertex}}(f(n)) + \mathrm{cc}_{\text{h-edge}}(f(n)) < \infty$, or for all $n \in \mathbb{N}$, $\mathrm{cc}_{\text{vertex}}(f(n)) + \mathrm{cc}_{\text{h-edge}}(f(n)) \geq n$.

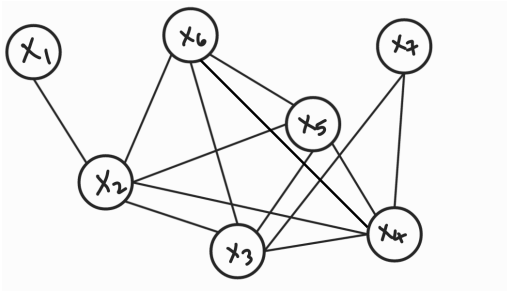
The tree-width of a 2L-abstraction G is defined on the basis of its *node graph*, which we denote by G^{node}. We define it as follows:

Definition 22. Let $\mathcal{C}$ be a class of 2L-graphs and G be some element in $\mathcal{C}$.

- Any $v_1, v_2 \in V$ are said to be **cc-close** if and only if there exist $\pi_1, \pi_2 \in E$ in the same connected component of G^{rel} such that $v_1 \in \eta(\pi_1), v_2 \in \eta(\pi_2)$.
- We denote by G^{node} the graph (V, E_{node}), where E_{node} is the set of all cc-close pairs.
- $\mathcal{C}^{\text{node}} = \{G^{\text{node}} \colon G \in \mathcal{C}\}$.

Below we have a 2L-graph G and its corresponding graph G^{node}:





3.2.2 Tree-width of an ECRPQ

We rely on the notion of tree-width, building upon the corresponding theorem for EVAL-CQ.

Definition 23. A **tree-decomposition** of a graph $G = (V, E)$ is a tree $T = (V', E')$ along with a function $\lambda\colon V' \to \mathscr{P}(V)$ such that:

(1) for every $\{u, v\} \in E$, there exists $s \in V'$ such that $\{u, v\} \subseteq \lambda(s)$.

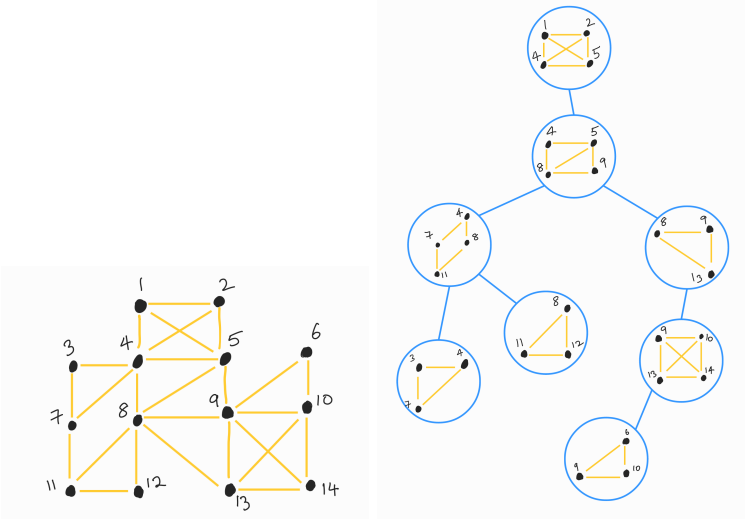
(2) For every $v \in V$, the subgraph of T induced by $\{s \in V' \colon v \in \lambda(s)\}$ is a tree.

For any $s \in V'$, we refer to $\lambda(s)$ as the *bag of* s. The width of T is the maximum size of its bags, i.e. $\max_{s \in V'} |\lambda(s)|$.

The treewidth of G, denoted as $tw(G)$, is the minimum width among all its tree-decompositions. The tree-width of a class of graphs $\mathcal{C}$ is defined as $tw(\mathcal{C}) = \sup_{G \in \mathcal{C}} tw(G)$ if it exists, and ∞ otherwise.

For any class $\mathcal{C}$ of 2L-graphs, the tree-width of $\mathcal{C}$ is defined as $tw(\mathcal{C}) = tw(\mathcal{C}^{\text{node}})$.

Example 22. Below we have a graph and one of its tree-decompositions:



In Lemma 6, we use an alternative definition of tree-decomposition which is easier to work with. This can be found in Section 4.2.3 of Chapter 4.

3.3 Theorem statements

With the definition of 2L-graphs and their measures, we are ready to state our results in full formality. Following that, we give a sketch of their proofs.

Theorem. *[Combined complexity] Let $\mathcal{C}$ be a cc-tame class of 2L-graphs. Assuming* W[1] $\neq$ FPT*:*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ or $cc_{h\text{-}edge}(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) *is* PSPACE*-complete, and*

(2) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}) < \infty$ *and* $tw(\mathcal{C}) = \infty$*, then* EVAL-ECRPQ($\mathcal{C}$) *is in* NP *and not in polynomial time, and*

(3) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}), tw(\mathcal{C}) < \infty$ *then* EVAL-ECRPQ($\mathcal{C}$) *is in polynomial time.*

Theorem. *[Parameterized complexity] Let* $\mathcal{C}$ *be a cc-tame class of 2L-graphs. Assuming* W[1] $\neq$ FPT*:*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ *then* p-EVAL-ECRPQ($\mathcal{C}$) *is* XNL*-complete, and*

(2) if $cc_{vertex}(\mathcal{C}) < \infty$ *and* $tw(\mathcal{C}) = \infty$*, then* p-EVAL-ECRPQ($\mathcal{C}$) *is* W[1]*-complete, and*

(3) if $cc_{vertex}(\mathcal{C}), tw(\mathcal{C}) < \infty$ *then* p-EVAL-ECRPQ($\mathcal{C}$) *is* FPT*.*

Proof sketch. We provide a general idea of the reductions used to prove these theorems. The full proofs of theorems and all the lemmas cited below can be found in Chapter 4.

-When $\text{cc}_{\text{vertex}}$ and $\text{cc}_{\text{h-edge}}$ are both unbounded, the proof of PSPACE-hardness is essentially contained in Reductions 1 and 2 (see Remark 1). In the proof of Lemma 5 we consider both reductions as separate cases, in order to obtain the PSPACE lower bound.

-When $\text{cc}_{\text{vertex}}$ is bounded, we give a procedure that serves as an FPT reduction from p-EVAL-ECRPQ($\mathcal{C}$) to p-EVAL-CQ($\mathcal{C}^{\text{node}}$) (see Lemma 4). Upon adding the condition of bounded $\text{cc}_{\text{h-edge}}$, the same procedure reduces an instance of EVAL-ECRPQ($\mathcal{C}$) to an instance of EVAL-CQ($\mathcal{C}^{\text{node}}$). Therefore we obtain the NP upper bound for the combined complexity.

-When $\text{cc}_{\text{vertex}}, \text{cc}_{\text{h-edge}}, \text{tw}$ are all bounded, we treat the resultant ECRPQs as CRPQs with languages over a product alphabet (with the arity of the product being bounded by $\text{cc}_{\text{h-edge}}$ and $\text{cc}_{\text{vertex}}$). Then, we apply the reduction from EVAL-CRPQ to EVAL-CQ and use the bounded tree-width property of CQ evaluation to obtain tractability.

□

In the upcoming chapter, we state and prove the lemmas to establish these theorems. The proofs involve reductions between instances of EVAL-CQ and EVAL-ECRPQ. These same reductions will also yield fixed parameter tractability when query size is considered to be fixed.

Chapter 4

Theorems and proofs

We ended Chapter 3 with two theorem statements, concerning the combined complexity and the parameterized complexity. The upcoming section establishes some technical preliminaries. Following that, we state and prove some lemmas divided according to the results associated with combined and parameterized complexity. Finally, we put all the lemmas together to prove the two theorems. We conclude with a section on future research on these topics.

4.1 Technical preliminaries

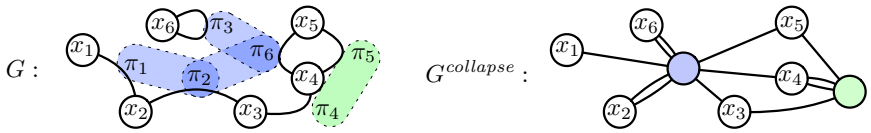
4.1.1 The collapse multigraph

We use the following construction in the proofs of Lemmas 6 and 7.

Definition 24. Let $G = (V, E, H, \eta, \nu)$ be a 2L-graph, and let C be the set of connected components of G^{rel}. We define a multigraph $G^{\text{collapse}} = (V_{\text{col}}, E_{\text{col}}, \eta_{\text{col}})$ where:

- For each $c \in C$, we introduce a new vertex v_c and define $V_C = \{v_c\}_{c \in C}$. Then $V_{\text{col}} = V \dot{\cup} V_C$.
- For every $\pi \in E$ (with $\eta(\pi) = \{v_1, v_2\}$), we introduce edges π^1, π^2 connecting $\{v_1, v_c\}, \{v_2, v_c\}$ respectively, where $c \in C$ is the connected component containing the hyperedge in which π occurs. Then
$$E_{\text{col}} = \bigcup_{\pi \in E} \{\pi^1, \pi^2\}$$
where $\eta_{\text{col}}(\pi^1) = \{v_1, v_c\}$, $\eta_{\text{col}}(\pi^2) = \{v_2, v_c\}$ such that $\eta(\pi) = \{v_1, v_2\}$ and π is in c.

Example 23. The following picture shows a 2L-graph and its collapse:



4.2 Lemmata

First, we state some folklore results which we use in our proofs. Then, we state and prove two sets of lemmas concerning upper and lower bounds respectively of the complexity of EVAL-ECRPQ($\mathcal{C}$) for various conditions on $\mathcal{C}$ (see Sections 4.2.2 and 4.2.3).

4.2.1 Folklore

We shall use the following well-established results:

Proposition 2. *([58]) Let $\mathcal{C}$ be a computably enumerable class of graphs. Then,*

(1) if tw($\mathcal{C}$) $< \infty$ then EVAL-CQ($\mathcal{C}$) *is in polynomial time.*

(2) otherwise, P-EVAL-CQ($\mathcal{C}$) *is W[1]-complete.*

Proposition 3. *(Theorem 17, [58]) Let $\mathcal{C}$ be a computably enumerable class of multigraphs such that tw($\mathcal{C}$) $= \infty$. Then,* P-EVAL-CQ$_{bin}$($\mathcal{C}$) *is W[1]-complete.*

Corollary 1. *Let $\mathcal{C}$ be a computably enumerable class of graphs. Then,*

(1) if tw($\mathcal{C}$) $< \infty$ then EVAL-CRPQ($\mathcal{C}$) *is in polynomial time.*

(2) otherwise, P-EVAL-CRPQ($\mathcal{C}$) *is W[1]-complete.*

Proof. For (1) we show a polynomial time reduction from EVAL-CRPQ($\mathcal{C}$) to EVAL-CQ($\mathcal{C}$). Let (q, D) be an instance of EVAL-ECRPQ, where

$$q = \exists \bar{x}\, \exists y_i\, \exists \bar{\pi} \bigwedge_{i=1}^{i=n} x_i \xrightarrow{\pi_i} y_i \wedge L_i(\pi_i)$$

and D is a graph database (V, E, η). We construct a CQ

$$q' = \exists \bar{z}\, \exists \bar{z}' \bigwedge_{i=1}^{i=n} R_i(z_i, z_i')$$

where for each $1 \leq i \leq n$,

$R_i = \{(v, v') \in V \times V :$ there exists a path from v to v' labeled by some word in $L_i\}$

is a relation that can be computed in polynomial time. Let $\mathcal{D}'$ be the relational database $(V, \{R_1, \ldots, R_n\})$. By definition of the query q', any valuation that witnesses $D \vDash q$ produces a valuation witnessing $\mathcal{D}' \vDash q'$, and vice-versa. Therefore, $D \vDash q$ if and only if $\mathcal{D}' \vDash q'$. This completes the proof of (1).

For (2), we give a polynomial time reduction from p-eval-$\mathrm{CQ}_{\mathrm{bin}}$ to p-eval-CRPQ which preserves tree-width. More precisely, given a binary CQ q of tree-width k and a relational database $\mathcal{D} = (V, \{R_1, \ldots, R_n\})$, we produce in polynomial time a CRPQ q' of tree-width k and a graph database D' such that $\mathcal{D} \models q$ if and only if $D' \models q'$. Let q be of the form

$$\exists \bar{x}\, \exists \bar{y}\, R_1(x_1, y_1) \wedge R_2(x_2, y_2) \ldots R_n(x_n, y_n)$$

Now, let q' be the CRPQ

$$\exists \bar{x}\, \exists \bar{y}\, \exists \bar{\pi} \bigwedge_{i=1}^{i=n} (x_i \xrightarrow{\pi_i} y_i \wedge L_i(\pi_i))$$

where for each $1 \leq i \leq n$, L_i is the regular language $\{R_i\}$ over the alphabet $\{R_1, \ldots, R_n\}$. Further, let D' be the graph database (V, E, η) where v, v' have an edge labeled R_i if and only if $R_i(v, v')$ holds in $\mathcal{D}$. Then by definition, $\mathcal{D} \models q$ if and only if $D' \models q'$. Further, q and q' have the same tree-width as they have the same underlying graph.

□

Proposition 4. *[57] Let $\mathcal{Q}$ be a computably enumerable class of CQs. Then,*

(1) if for some $k \in \mathbb{N}$ every $q \in \mathcal{Q}$ is equivalent to some CQ q' of treewidth at most k, then eval-$\mathcal{Q}$ *is in polynomial time.*

(2) otherwise, p-eval-$\mathcal{Q}$ *is $W[1]$-complete.*

4.2.2 Upper bounds

Next, we prove the upper bound results for the complexity of eval-ECRPQ and p-eval-ECRPQ.

Lemma 3. p-eval-ECRPQ *is in* XNL.

Proof. We describe a procedure which takes an ECRPQ q and generates an NL algorithm for $\{D\colon D \models q\}$. Fix a graph database D and an ECRPQ q with abstraction G. We transform q into an equivalent query $\hat{q}$ with a 2L abstraction $\hat{G}$ such that every connected component in $\hat{G}^{\mathrm{rel}}$ contains only one hyper-edge. There are two steps in this transformation: (i) for every connected component of $\hat{G}^{\mathrm{rel}}$, the hyper-edges occurring in it are coalesced into a single "large" hyper-edge by taking the union of their constituent vertex sets, and (ii) the "join" of the synchronous relations corresponding to these hyper-edges is computed to produce one synchronous relation per connected component, which is then associated to the large hyper-edge. As an example, consider the query $q = \gamma \wedge \rho$ where

$$\gamma = x \xrightarrow{\pi_1} y \wedge y \xrightarrow{\pi_4} z \wedge z \xrightarrow{\pi_2} t \wedge t \xrightarrow{\pi_3} y$$
$$\rho = U(\pi_1) \wedge S(\pi_1, \pi_2) \wedge T(\pi_2, \pi_3) \wedge L(\pi_4)$$

Now, we produce $\hat{q} = \gamma \wedge \rho'$ where $\rho' = R(\pi_1, \pi_2, \pi_3) \wedge L(\pi_4)$ and R is the "join" relation

$$\{(u_1, u_2, u_3) \in \mathbb{A}^* \times \mathbb{A}^* \times \mathbb{A}^*\colon u_1 \in U, (u_1, u_2) \in S, (u_2, u_3) \in T\}$$

Note that these joins can be computed in PSPACE.

Having computed $\hat{q}$ (along with $\hat{G}$), we describe the NL algorithm for $\{D\colon D \vDash \hat{q}\}$ (since $q, \hat{q}$ are equivalent, $\{D\colon D \vDash q\} = \{D\colon D \vDash \hat{q}\}$). Let $D = (V, E, \eta)$ be the input graph database. We guess a valuation σ such that σ_{node} sends each node variable in $\hat{q}_{\text{reach}}$ to nodes in D (this can be done in NSPACE($|\hat{q}|\cdot log(|D|)$)). For every atom $R(\pi_1, \ldots, \pi_n) \in \hat{q}_{\text{rel}}$ (with corresponding node variables x_i, y_i for all $1 \leq i \leq n$), we

- non-deterministically guess simultaneous paths p_i from $\sigma_{\text{node}}(x_i)$ to $\sigma_{\text{node}}(y_i)$ (for all i).

- verify the paths labels correspond to R, i.e. $(\eta(p_1), \ldots, \eta(p_n)) \in R$.

This suffices because, by construction, no path variable in $\hat{q}$ occurs in more than one relation. □

Lemma 4. *Let $\mathcal{C}$ be a class of 2L-graphs such that $cc_{vertex}(\mathcal{C}) < \infty$.*

(1) there is an FPT reduction from p-eval-ECRPQ($\mathcal{C}$) *to* p-eval-CQ($\mathcal{C}^{node}$)

(2) Furthermore if $cc_{h\text{-}edge}(\mathcal{C}) < \infty$, then there is a polynomial time reduction from eval-ECRPQ($\mathcal{C}$) *to* eval-CQ($\mathcal{C}^{node}$).

Proof. The following procedure will establish both (1) and (2) (we elaborate upon this in the last paragraph of the proof). First, we describe it below:

Let $G \in \mathcal{C}$ abstracting some ECRPQ q and let $D = (V, E, \eta)$ be a graph database. We produce a CQ q' in polynomial time (depending on only q) , and a relational database $\mathcal{D}'$ in polynomial space (depending only q and D), such that G^{node} is the underlying graph of q' and $D \vDash q$ if and only if $D' \vDash q'$. Furthermore, we guarantee that if $\text{cc}_{\text{h-edge}}(G) < \infty$ then this procedure is in polynomial time.

Just like in the proof of Lemma 3, we begin by producing a query $\hat{q}$ equivalent to q by merging the relations in every connected component of the latter into a single relation per connected component. We also mentioned in that proof that this is a PSPACE operation. Note, however, that if $\text{cc}_{\text{vertex}}(\mathcal{C})$ and $\text{cc}_{\text{h-edge}}(\mathcal{C})$ are both bounded, then we only need to join a constant number of join operations to produce $\hat{q}$. In this case, the operation to produce $\hat{q}$ takes polynomial time. Henceforth we will use q and $\hat{q}$ interchangeably as necessary, given that they are equivalent and have the same node and path variables.

Next we outline how q' is produced from q. For every relation symbol $R(\pi_1, \ldots, \pi_n) \in \hat{q}_{\text{rel}}$, we introduce a relation symbol

$$R'(x_1, y_1, x_2, y_2 \ldots, x_n, y_n)$$

where for all $1 \leq i \leq n$, x_i, y_i are such that $x_i \xrightarrow{\pi_i} y_i \in q_{\text{reach}}$. Define the CQ

$$q' = \exists \bar{x}\, \exists \bar{y} \bigwedge_{R \in \hat{q}_{\text{rel}}} R'$$

Note that q' is dependent only on q. Recall $D = (V, E, \eta)$. Let D' be the relational database $(V, \{R' : R \in \hat{q}\})$, where each R' is interpreted as

$$R' = \{(u_1, v_1, \ldots, u_n, v_n) \in V^{2n} : \text{ for all } i, \text{ there exists a path } u_i \to v_i \text{ in } D \text{ with label } w_i \in \mathbb{A}^* \text{ such that } (w_1, \ldots, w_n) \in R\}$$

It immediately follows that $D \models q$ if and only if $D' \models q'$. Here q' uses relations of arity at most $2 \cdot \mathrm{cc}_{\mathrm{vertex}}(G)$. Thus D' can be constructed from $\hat{q}$ in $O(|D|^{2\mathrm{cc}_{\mathrm{vertex}}(G)})$, (i.e. in polynomial time). Therefore, it takes time $f(|q|) \cdot |D|^{2 \cdot \mathrm{cc}_{\mathrm{vertex}}(G)}$ to compute D', where f is the time taken to compute $\hat{q}$ from q. Seen as a reduction from p-eval-ECRPQ($\mathcal{C}$) to p-eval-CQ($\mathcal{C}$), this reduction is FPT. Further, if $\mathrm{cc}_{\text{h-edge}}(\mathcal{C})$ is finite, f is a polynomial. Therefore the classical eval-ECRPQ($\mathcal{C}$) to eval-CQ($\mathcal{C}$) reduction is polynomial time when $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C})$ and $\mathrm{cc}_{\text{h-edge}}(\mathcal{C})$ are both bounded (for this we use the fact that G^{node} is the Gaifman graph of q', this follows easily from the merging of hyper-edges by which we produced $\hat{q}$ from q). □

4.2.3 Lower bounds

We state and prove the hardness results here, which yield the required lower bounds.

Lemma 5. *Let $\mathcal{C}$ be a cc-tame class with $cc_{vertex}(\mathcal{C}) = \infty$ or $cc_{h\text{-}edge}(\mathcal{C}) = \infty$. Then* eval-ECRPQ($\mathcal{C}$) *is* PSpace*-complete.*

Proof. We know that eval-ECRPQ is in PSpace (see [15]). Now we need to show that if $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) = \infty$ or $\mathrm{cc}_{\text{h-edge}}(\mathcal{C}) = \infty$ then eval-ECRPQ($\mathcal{C}$) is PSpace-hard. We will do this by giving a reduction from the Intersection Emptiness problem (IE) for regular languages, which we recall here:

INTERSECTION EMPTINESS (IE)
Input: A set of DFA $\mathcal{S}$
Question: Is $\bigcap_{\mathcal{A} \in \mathcal{S}} L(\mathcal{A}) \neq \emptyset$?

It is well-known that IE is PSpace-complete [72]. Now, fix an instance $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ of IE. For all i, let $L_i = L(\mathcal{A}_i)$. We construct, in polynomial time, a graph database D and ECRPQ q whose abstraction $G \in \mathcal{C}$, such that $D \models q$ if and only if $\{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n\}$ is a yes instance of IE, i.e. $L_1 \cap \cdots \cap L_n \neq \emptyset$.

Recall that $\mathcal{C}$ is cc-tame, so either $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) + \mathrm{cc}_{\text{h-edge}}(\mathcal{C}) < \infty$ or there exists a computable function $f : \mathbb{N} \to \mathcal{C}$ such that for all $k \in \mathbb{N}$, $\mathrm{cc}_{\mathrm{vertex}}(f(k)) + \mathrm{cc}_{\text{h-edge}}(f(k)) \geq n$. Here $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) + \mathrm{cc}_{\text{h-edge}}(\mathcal{C}) < \infty$ clearly does not hold (because either $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) = \infty$ or $\mathrm{cc}_{\text{h-edge}}(\mathcal{C}) = \infty$). So there exists a computable function f such that for all $k \in \mathbb{N}$, $\mathrm{cc}_{\mathrm{vertex}}(f(k)) + \mathrm{cc}_{\text{h-edge}}(f(k)) \geq n$. Let $G = f(n + (n-1)^2)$.

We argue that G^{rel} contains a connected component c which has either (i) at least n vertices, or (ii) at least one vertex incident to n hyper-edges. If (i) is not true, then every connected component of G^{rel} will have at most $(n-1)$ vertices, so $\mathrm{cc}_{\mathrm{vertex}}(G) \leq n - 1$. Further, if (ii) is not true, then every vertex in G^{rel} is

incident to at most $(n-1)$ hyper-edges, so $\mathrm{cc}_{\text{h-edge}}(G) \leq (n-1)^2$. Therefore, the largest possible value of $\mathrm{cc}_{\text{vertex}}(G) + \mathrm{cc}_{\text{h-edge}}(G)$ is $(n-1) + (n-1)^2$. However, $G = f(n + (n-1)^2)$ and by definition f satisfies

$$\mathrm{cc}_{\text{vertex}}(f(n + (n-1)^2)) + \mathrm{cc}_{\text{h-edge}}(f(n + (n-1)^2)) \geq n + (n-1)^2$$

This is a contradiction. Therefore either (i) or (ii) is true. In both cases, we construct in polynomial time an ECRPQ q whose abstraction is G, and a graph database D such that $D \vDash q$ if and only if $L_1 \cap \cdots \cap L_n \neq \emptyset$.

Case (i): Say G^{rel} has a connected component c with vertices $\{\pi_1, \ldots, \pi_m\}$, where $m \geq n$. In fact, we can assume $m = n$ (otherwise we just extend the IE instance with $(m-n)$ 'dummy' languages $\mathbb{A}^*$). Fix c. Let $q_c \in \mathrm{ECRPQ}(G)$ with the following relations:

- For every hyper-edge h in c with $\nu(h_R) = \{\pi_{i_1}, \ldots, \pi_{i_k}\}$, the relation symbol R_h corresponds to the relation

  $$R_h(\pi_{i_1}, \ldots, \pi_{i_k}) = \{(\$u\#^{i_1}\$, \ldots, \$u\#^{i_k}\$) \colon u \in \mathbb{A}^*\}$$

  Note that an NFA recognizing R_h can be constructed in $O(\max\{i_1, \ldots, i_k\})$ states. We describe this construction below.

- Additionally, for every hyper-edge h not in c of some size k, R_h is set to $\mathbb{U}_k(\mathbb{A})$, i.e. the universal relation on $\mathbb{A}^*$ of arity k.

We now construct an NFA $\mathcal{A}$ that recognizes

$$R_h(\pi_{i_1}, \ldots, \pi_{i_k}) = \{(\$u\#^{i_1}\$, \ldots, \$u\#^{i_k}\$) \colon u \in \mathbb{A}^*\}$$

in polynomial time, and establishes that R_h is synchronous. Let $n = \max\{i_1, \ldots, i_k\}$. Then $\mathcal{A}$ is an NFA over the alphabet $(\mathbb{A} \dot\cup \{\#, \$, \perp\})^k$ whose states are

$$\{q_0, q_1, \ldots, q_{n+1}, q_f\}$$

with its initial and final states being q_0 and q_f respectively. Its transition relation δ has the following transitions:

- $q_0 \xrightarrow{(\$,\ldots,\$)} q_1$
- $q_1 \xrightarrow{(a,\ldots,a)} q_1$ for every $a \in \mathbb{A}$
- $q_i \xrightarrow{(z_1,\ldots,z_k)} q_{i+1}$ for each $1 \leq i \leq n$, where for each j:
  - $z_j = \#$ if $i \leq i_j$, and
  - $z_j = \$$ if $i = i_j + 1$, and
  - $z_j = \perp$ otherwise.
- $q_{n+1} \xrightarrow{(z_1,\ldots,z_k)} q_f$, where for each j:
  - $z_j = \$$ if $n = i_j$, and
  - $z_j = \perp$ otherwise.

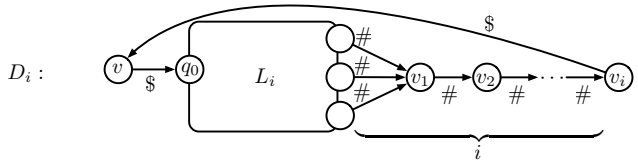
Note that $\mathcal{A}$ can be constructed in polynomial time and accepts the set of all synchronized words of the form

$$(\$u\underbrace{\#\cdots\#}_{i_1},\ldots,\$u\underbrace{\#\cdots\#}_{i_k})$$

that is, it accepts the relation R_h. This completes the construction of q_c. Next we construct the graph database D. To do this, we will define sub-databases $\{D_i\}_{1\leq i\leq n}$, whose vertices are all mutually disjoint except for one distinguished vertex v which belongs to each D_i. For each $1 \leq i \leq n$, let D_i be a graph database which contains:

- a copy of $\mathcal{A}_i$, along with additional vertices $v_1, \ldots, v_i$ and the distinguished vertex v
- a path $v_1 \to v_2 \cdots \to v_i$ reading $\#^{i-1}$
- an edge from every final state of $\mathcal{A}_i$ to v_1 reading $\#$, and an edge from v_i to v of $\mathcal{A}_i$ reading $\$$

We illustrate D_i as follows:



D is now defined as the union of all D_i, which are pairwise disjoint except for the distinguished vertex v common to all of them. All that remains to be shown is that $D \vDash q_c$ if and only if $L(\mathcal{A}_1)\cdots\cap L(\mathcal{A}_n) \neq \emptyset$.

Say $D \vDash q_c$ via a witnessing valuation σ. Recall that c contains n path variables $\{\pi_1, \ldots, \pi_n\}$. There exists $w \in \mathbb{A}^*$ such that for all $1 \leq i \leq n$, σ assigns π_i to some path p_i in D labeled $\$w\#^i\$$. The shape of D dictates that for all i, p_i must be a path in D_i witnessing $w \in L_i$. Therefore, $w \in L_1 \cap \cdots \cap L_n$. For the converse, assume there exists some $w \in L_i$ for all $1 \leq i \leq n$. Observe that the assignment which sends:

- every node variable of q_c to v
- every path variable of π_i of q_c in c to some path in D_i of the form:

$$v \xrightarrow{\$} q_0 \xrightarrow{w} q_f \xrightarrow{\#} v_1 \xrightarrow{\#} \ldots \xrightarrow{\#} v_i \xrightarrow{\$} v$$

  (there is guaranteed to be at least one such path, because $w \in L_i$)

- every path variable π_i not in c to any path in D which starts and ends in v

is a witness for $D \vDash q_c$.

Case (ii): If c contains a path variable π incident to n hyper-edges $h_1, \ldots, h_n$, then we define a synchronous relation R_i for the hyper-edge h_i as follows: if h_i

is incident to $k \geq 0$ additional vertices such that $\nu(h_i) = \{\pi, \pi_1, \ldots, \pi_k\}$, then $R_i(\pi, \pi_1, \ldots, \pi_k)$ is defined as the synchronous relation

$$\{(u, u_1, \ldots, u_k) \colon u \in L_i \text{ and } u_1, \ldots, u_k \in A^*\}$$

For every hyper-edge not in c, we define the corresponding relation to be the universal relation of its arity, just like we did in Case (i). All these relations are constructed in polynomial time. Let q be the ECRPQ whose abstraction is G containing the synchronous relations defined above. Now we simply define D to be the trivial graph database that has only one vertex v and a self-loop for every $a \in A$.

If there exists a witness for $D \vDash q$, then the path assigned to π is labeled by some word w which is in every L_i by definition of the relations in c. Conversely, if the non-emptiness of $L_1 \cap \cdots \cap L_n$ is witnessed by a word w, then by assigning to π the path from v to v labeled by w we produce a valuation witnessing $D \vDash q$. So in this case also we have $D \vDash q$ if and only if $L_1 \cap \cdots \cap L_n \neq \emptyset$. □

We now state an alternative definition of tree decomposition which will be useful in proving Lemma 6 (stated after this definition).

Definition 25 (Alternative definition of tree decomposition)**.** Let $G = (V, E)$ be a graph, $T = (V', E')$ be a tree and $\lambda \colon V' \to \mathscr{P}(V)$. For all $v \in V$, a path p in T is called (λ, v)**-faithful** if and only if for all $s \in V'$ in p, $v \in \lambda(s)$. Then, T is said to be a tree-decomposition of G if and only if

(1) for every $\{u, v\} \in E$, there exists $s \in V'$ such that $\{u, v\} \subseteq \lambda(s)$.

(2) For every $v \in V$ contained in the bags of some $s_1, s_2 \in V'$, there exists a (λ, v)-faithful path p from s_1 to s_2 in T.

The original definition of tree-decomposition (Definition 23) stated that λ must satisfy: (1) for every $\{u, v\} \in E$, there exists $s \in V'$ such that $\lambda(s) \subseteq \{u, v\}$, and (2) for every $v \in V$, the subgraph of T induced by $\{s \in V' \colon v \in \lambda(s)\}$ is a tree. To show that Definition 25 is indeed an alternative definition of tree-decomposition, we only need to show that condition (2) of Definition 25 is equivalent to condition (2) of Definition 23 (since condition (1) is identical for both definitions).

In other words, for every $v \in V$, we need to show that the subgraph of T induced by $\{s \in V' \colon v \in \lambda(s)\}$ is a tree if and only if there exists a (λ, v)-faithful path p from s_1 to s_2 in T for every $s_1, s_2 \in V'$ such that $v \in \lambda(s_1) \cap \lambda(s_2)$.

If the subgraph of T induced by $\{s \in V' \colon v \in \lambda(s)\}$ is a sub-tree, then for any s_1, s_2 whose bags contain v, a (λ, v)-faithful path from s_1 to s_2 can be found in this sub-tree itself. Conversely, assume that for all $s_1, s_2 \in V'$ whose bags contain v, there exists a (λ, v)-faithful path from s_1 to s_2. The union of all such paths is a sub-tree given by $\{s \in V' \colon v \in \lambda(s)\}$. Therefore, Definition 25 is indeed an alternative definition of tree-decomposition as given in Definition 23.

Lemma 6. *Let $\mathcal{C}$ be a class of 2L-graphs with $cc_{vertex}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}^{node}) = \infty$. Then $tw(\mathcal{C}^{collapse}) = \infty$.*

Proof. Assume $tw(\mathcal{C}^{\text{collapse}})$ is finite. We prove that for all $G \in \mathcal{C}$,

$$tw(G^{\text{node}}) \leq 2 \cdot \text{cc}_{\text{vertex}}(\mathcal{C}) \cdot tw(G^{\text{collapse}})$$

This is sufficient to show that $tw(\mathcal{C}^{\text{collapse}}) = \infty$, given that $tw(\mathcal{C}^{\text{node}}) = \infty$. Now, fix a 2L-graph $G = (V, E, H, \eta, \nu) \in \mathcal{C}$ such that $G^{\text{collapse}} = (V_{\text{col}}, E_{\text{col}}, \eta_{\text{col}})$ and a tree-decomposition $T = (V_t, E_t, \gamma)$ of width k, where

$$\gamma \colon V_t \to \mathscr{P}(V_{\text{col}})$$

is the bag function witnessing that T is a tree-decomposition of G^{collapse}. We construct a new bag function

$$\gamma' \colon V_t \to \mathscr{P}(V)$$

such that $T' = (V_t, E_t, \gamma')$ will prove to be a tree-decomposition of G^{node} of width $2 \cdot \text{cc}_{\text{vertex}}(G) \cdot k$.

[The transformation $\gamma \mapsto \gamma'$] $T = (V_t, E_t, \gamma)$ is transformed into $T' = (V_t, E_t, \gamma')$ via the transformation $\gamma \mapsto \gamma'$ which is as follows: for every $s \in V_t$, look at the bag $\gamma(s)$. Now, remove every cc vertex v_c in the bag and replace it with the set of node vertices incident to it, i.e.

$$\mathsf{node\text{-}vertices}(v_c) = \{v \in V \colon \mu(\pi') = \{v, v_c\} \text{ for some } \pi' \in E_{\text{col}}\}$$

For every connected component c, let $N_c = \mathsf{node\text{-}vertices}(v_c) \setminus \{v_c\}$. Moreover, let $N(s)$ be the union of N_c over all connected components c such that $v_c \in \gamma(s)$. Now, γ' is formally defined as

$$\gamma'(s) = \gamma(s) \cup N(s)$$

Next we show that $T' = (V_t, E_t, \gamma')$ is a tree-decomposition of $G^{\text{node}} = (V, E_{\text{node}})$. We need to check that:

(1) for every edge $\{v_1, v_2\} \in E_{\text{node}}$, there exists some $s \in V_t$ such that $\{v_1, v_2\} \subseteq \gamma(s)$.

(2) for all $v \in V$, and $s_1, s_2 \in V_t$, if $v \in \gamma'(s_1) \cap \gamma'(s_2)$ then there exists a (γ', v)-faithful path from s_1 to s_2 in (V_t, E_t) (see Definition 25 for the alternative definition of tree-width).

For (1), let $\{v_1, v_2\} \in E_{\text{node}}$. Then there exist $e_1, e_2 \in E$ in some connected component c such that $v_1 \in \eta(e_1), v_2 \in \eta(e_2)$. It directly follows that $v_1, v_2 \in \mathsf{node\text{-}vertices}(v_c)$. Let $s \in V_t$ such that $v_c \in \gamma(s)$. Then, $\mathsf{node\text{-}vertices}(v_c) \subseteq \gamma'(s)$, therefore $\{v_1, v_2\} \subseteq \gamma'(s)$.

To show (2), let $v \in V$ and $s_1, s_2 \in V_t$ such that $v \in \gamma'(s_1) \cap \gamma'(s_2)$. Without loss of generality, three cases arise:

- $v \in \gamma(s_1) \cap \gamma(s_2)$: We know that (V_t, E_t, γ) is a tree-decomposition for G^{rel}. So, there exists a (γ, v)-faithful path p from s_1 to s_2. We do not remove any vertices in V from any of the bags in the transformation $\gamma \to \gamma'$. Therefore, p is (γ', v)-faithful as well.

- $v \in \gamma(s_1) \setminus \gamma(s_2)$: In this case, v was present in the γ-bag of s_1 but not of s_2. Therefore, v was added in $\gamma'(s_2)$ because there exists some cc vertex $v_c \in \gamma(s_2)$ such that $v \in \mathsf{neighbors}(v_c)$. Since (V_t, E_t, γ) is a tree-decomposition of G^{collapse}, there exists some $s_3 \in V_t$ such that $\{v, v_c\} \in \gamma(s_3)$. Further, $v \in \gamma(s_1) \cap \gamma(s_3)$ and $v_c \in \gamma(s_2) \cap \gamma(s_3)$. So there exists a path p_1 from s_1 to s_3 that is (γ, v)-faithful, and a path p_2 from s_3 to s_2 that is (γ, v_c)-faithful. As per the transformation $\gamma \to \gamma'$, (i) p_2 is also (γ', v)-faithful because v is added to every bag containing v_c, and (ii) p_1 is (γ', v)-faithful because we never remove any vertex from V. Therefore, the conjunction $p_1 \cdot p_2$ is a path from s_1 to s_2 that is (γ', v)-faithful.

- $v \notin \gamma(s_1) \cap \gamma(s_2)$: In this case, there exist connected components $c_1, c_2 \in C$ (not necessarily distinct) such that $v_{c_1} \in \gamma(s_1), v_{c_2} \in \gamma(s_2)$ and $v \in \mathsf{neighbors}(v_{c_1}) \cap \mathsf{neighbors}(v_{c_2})$. Applying the argument made in the previous point to v_{c_1} and v_{c_2}, we know that there exist tree-vertices $s_3, s_4 \in V_t$ such that $\{v, v_{c_1}\} \in \gamma(s_1) \cap \gamma(s_3)$ and $\{v, v_{c_2}\} \in \gamma(s_2) \cap \gamma(s_4)$. So there exist paths p_1 from s_1 to s_3 which is (γ, v_{c_1})-faithful, p_2 from s_3 to s_4 which is (γ, v)-faithful, and p_3 from s_4 to s_2 which is (γ, v_{c_2})-faithful. After the transformation $\gamma \to \gamma'$, p_1 and p_2 become (γ', v)-faithful when v is added to all the γ-bags containing either v_{c_1} or v_{c_2}. Therefore, the conjunction $p_1 \cdot p_2 \cdot p_3$ is a (γ', v)-faithful path from s_1 to s_2.

We see that in all three cases, there exists a (γ', v)-faithful path from s_1 to s_2. So T' satisfies (1) and (2) and is hence a decomposition of G^{node}.

Finally, let k be the tree-width of T. In the transformation $\gamma \to \gamma'$, we took every bag of T, removed at most k cc vertices (the maximum size of the bags is k), and added a set of size at most $2 \cdot \text{cc}_{\text{vertex}}(G)$ for every cc vertex that we removed. Therefore, the size of any bag in T' is at most $2 \cdot \text{cc}_{\text{vertex}}(G) \cdot k$. This shows that $tw(G^{\text{node}}) \leq 2 \cdot \text{cc}_{\text{vertex}}(G) \cdot tw(G^{\text{collapse}})$.

□

Lemma 7. *Let $\mathcal{C}$ be a computably enumerable class of 2L-graphs. There exists an FPT reduction from* p-eval-$\text{CQ}_{bin}(\mathcal{C}^{collapse})$ *to* p-eval-$\text{ECRPQ}(\mathcal{C})$.

Proof. Recall that CQ_{bin} is the set of all CQs which use only binary relations. Given $q \in \text{CQ}_{\text{bin}}(\mathcal{C}^{\text{collapse}})$ and a relational database $\mathcal{D} = (V, \{R_1, \ldots, R_n\})$, there exists a 2L-graph $G \in \mathcal{C}$ such that G^{collapse} is the multi-graph abstraction of q. Moreover, G is computable because $\mathcal{C}$ is computably enumerable. The goal is to produce an ECRPQ $q' \in \text{ECRPQ}(G)$ and a graph database D' such that $D' \models q'$ if and only if $\mathcal{D} \models q$.

Without loss of generality, we assume that for every relation R in the database alphabet, its inverse R^{-1} is also in the database alphabet (interpreted as $\{(v, u) \colon (u, v) \in R\}$).

Recall the definition of G^{collapse} given in Definition 24. $G^{\text{collapse}} = (V_{\text{col}}, E_{\text{col}}, \eta_{\text{col}})$ has two kinds of vertices: the node vertices in V and cc vertices $\{v_c\}_{c \in C}$, where C is the set of connected components of G^{rel}. Further, every edge in E named π with $\eta(\pi) = \{v, v'\}$ is split into $\hat{\pi}$ and $\hat{\pi}'$ which connect $\{v, v_c\}$ and $\{v', v_c\}$ respectively (where $c \in C$ contains π in G^{rel}). Here G^{collapse} is the underlying

graph of q. So we partition the variables of q into two sets $\{x_1, x'_1, \ldots, x_\ell, x'_\ell\}$ and $\{y_1, \ldots, y_\ell\}$ corresponding to node vertices and cc vertices respectively. Here the $\{x_i, x'_i\}_{i \leq \ell}$ are not necessarily pairwise distinct, and neither are the $\{y_i\}_{i \leq \ell}$

The idea is that for every pair of E_{col} edges $\{v, v_c\}, \{v', v_c\}$ named $\hat{\pi}, \hat{\pi}'$ respectively, we pick an i such that the vertices v, v' are identified with the node variables x_i, x'_i and v_c is identified with y_i. The edge $\{v, v_c\}$ named $\hat{\pi}$ is identified with the relation symbol R and $\hat{\pi}'$ is identified with R'. Now we can re-write q as

$$q = \exists \bar{x} R_1(x_1, y_1) \wedge R'_1(y_1, x'_1) \wedge \cdots \wedge R_\ell(x_\ell, y_\ell) \wedge R'_\ell(y_\ell, x'_\ell)$$

where the x_i's and x'_i's are node variables corresponding to V, and y_i's are cc variables corresponding to V_C. We can rewrite any binary CQ in this form using inverses while ensuring its underlying multigraph does not change.

Let q' be some ECRPQ whose abstraction is G. Let $\mathbb{A}$ be the alphabet containing all relation symbols used in q_{rel}, along with two additional symbols 0 and 1. Now we shall precisely define q by associating synchronous relations to the relation symbols. For every atom $R(\pi_1, \ldots, \pi_r) \in q_{\text{rel}}$ and for every $1 \leq i \leq r$, there is exactly one atom $x_{j_i} \xrightarrow{\pi_i} x'_{j'_i} \in q_{\text{reach}}$. Fix the pairs $\{(j_i, j'_i)\}_{\pi_i \text{ occurs in } R}$ for all R. Then we build the synchronous relation

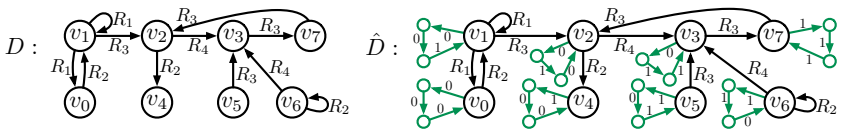
$$R = \{(R_{j_1} w R'_{j'_1}, \ldots, R_{j_r} w R'_{j'_r}) \colon w \in \{0,1\}^*\}$$

for every $R \in q_{\text{rel}}$. Intuitively, R is defined this way to ensure there exist paths

$$x_{j_i} \xrightarrow{R_{j_i}} y_{j_i} \xrightarrow{R'_{j'_i}} x'_{j'_i}$$

where the y_{j_i}'s are equal for all $1 \leq i \leq r$, ensured by the common word $w \in \{0,1\}^*$. This completes the definition of q'. We proceed to define D'.

The graph database D' is an extension of the relational database $\mathcal{D}$ over $\mathbb{A}$, constructed by adding a simple cycle on every node whose path is labeled with some word in $\{0,1\}^*$. Let $V = \{v_1, \ldots, v_n\}$ be the domain of $\mathcal{D}$. To every vertex v_i, add a cycle containing $n' - 1$ vertices, where n' is the $\lceil log(n) \rceil$-bit binary expansion of i. Further, the cycle should be labeled by this binary expansion, as a word over $\{0,1\}^*$. We provide an example below:



In this way, by adding $n \cdot (n' - 1)$ new vertices to $\mathcal{D}$, we can construct D' in polynomial time. This procedure is FPT, because (i) D' is generated in polynomial time independent of q, and (ii) the generation of q' is effective by $\mathcal{C}$ being computably enumerable and independent of $\mathcal{D}$. It remains to show that

Claim 1. *$D' \vDash q'$ if and only if $\mathcal{D} \vDash q$.*

For ($\Longleftarrow$), assume a valuation σ witnesses $\mathcal{D} \vDash q$. So for every $1 \leq i \leq \ell$ we know that $R_i(\sigma(x_i), \sigma(y_i))$ and $R'_i(\sigma(y_i), \sigma(x'_i))$ hold in $\mathcal{D}$. Recall that G is the abstraction of q', and G^{collapse} is obtained from G by splitting its edges and adding component vertices. Consider an atom $R(\pi_1, \ldots, \pi_r) \in q'_{\text{rel}}$, where for all $1 \leq i \leq r$, $x_{j_k} \xrightarrow{\pi_i} x'_{j'_k} \in q_{\text{reach}}$. This implies the existence of a subquery $R_{j_k}(x_{j_k}, y_{j_k}) \wedge R'_{j_k}(y_{j_k}, x'_{j_k})$ of q, where y_{j_k} is identified with v_{c_i}, the cc vertex (corresponding to the connected component c_i of G^{rel}) in which π_i occurs. By construction, $\hat{D}$ contains a path

$$\sigma(x_{j_k}) \xrightarrow{R_{j_k}} \sigma(y_{j_k}) \xrightarrow{w} \sigma(y_{j_k}) \xrightarrow{R'_{j'_k}} \sigma(x'_{j'_k})$$

where $w \in \{0,1\}^*$ is the word labeled by the simple cycle attached to $\sigma(y_{j_k})$. In this way we obtain a valuation witnessing $D' \vDash q'$.

For the other direction, assume a valuation σ witnesses $D' \vDash q'$. To construct a valuation witnessing $D \vDash q$, first send every node variable x to $\sigma(x)$. For a cc variable y identified with some v_c, let π be any path variable of c such that the the path $\sigma_{\text{path}}(\pi)$ is labeled by RwR' for some $R, w, R' \in \mathbb{A} \cup \{0,1\}^*$, with w being the binary encoding of some integer i. Now, send y to v_i. This witnesses $D \vDash q$.

□

Lemma 8. *For any computably enumerable class $\mathcal{C}$ of 2L-graphs such that $cc_{vertex}(\mathcal{C}) = \infty$,* p-eval-ECRPQ($\mathcal{C}$) *is XNL-hard.*

Proof. We show an FPT reduction from the parameterized IE problem, which is known to be XNL-complete under FPT reductions [118].

PARAMETERIZED INTERSECTION EMPTINESS (p-IE)
Input: A set of DFA $\mathcal{S}$
Question: Is $\bigcap_{\mathcal{A} \in \mathcal{S}} L(\mathcal{A}) \neq \emptyset$?
Parameter: $|\mathcal{S}|$

We split the reduction into two cases, depending on the size of hyper-edges in elements of $\mathcal{C}$.

(1) The size of hyper-edges in $\{G^{\text{rel}} \colon G \in \mathcal{C}\}$ is bounded

(2) For every $n \in \mathbb{N}$ there exists $G \in \mathcal{C}$ such that G^{rel} contains a hyper-edge of size $\geq n$.

In both cases we will produce an ECRPQ q with abstraction G and a graph database D such that $D \vDash q$ if and only if $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$.

Case (1) [Bounded hyper-edges]: First we look for a 2L-graph G such that G^{rel} contains a connected component with a sufficiently long path. More precisely, we need a graph $G = (V, E, H, \eta, \nu)$ such that there exist k hyper-edges $h_1, \ldots, h_k \in H$ of size at least 2, and $k-1$ edges $\pi_1, \ldots, \pi_{k-1} \in E$ such that each π_i is in h_i and h_{i+1}, and nowhere else. Formally, for every i,

$$\pi_i \in \nu(h_i) \cap \nu(h_{i+1}) \setminus \bigcup_{j \neq i, i+1} \nu(h_j)$$

We know that $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) = \infty$ and $\mathcal{C}$ is computably enumerable, so it is guaranteed that we will find a suitable $G \in \mathcal{C}$ via an effectively computable procedure.

Consider some ECRPQ q abstracted by G, without its synchronous relations specified. Let $h_{k+1}, \ldots, h_N$ be the remaining hyper-edges in H. By the choice of G, q_{rel} would have the form $\bigwedge_{1 \leq i \leq N} R_i(\bar{t}_i)$, where for all $1 \leq i \leq N$: (i) R_i is the relation symbol corresponding to h_i spanning over the set given by $\bar{t}_i$, and (ii) π is the last element of $\bar{t}_i$ and the first element of $\bar{t}_{i+1}$, and it appears nowhere else. To complete the definition of q, we have to define its synchronous relations precisely. These will be defined over a new alphabet $\mathbb{B} = \mathbb{A} \dot{\cup} \{\#, \$\}$. When $i \leq k$:

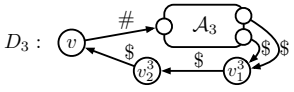
$$R_i = \{(\#w\$^i, u_1, \ldots, u_{r-2}, \#w\$^{i+1}) \colon w \in \mathbb{A}^*, u_1, \ldots, u_{r-2} \in \mathbb{B}^*\}$$

When $i > k$, R_i is taken to be the universal relation $\mathbb{U}_{\mathrm{arity}(R_i)}(\mathbb{B}^*)$.

It is easily verified that each R_i is synchronous and an automaton for it can be constructed in polynomial time. The final step is to define D, which we do by defining k sub-databases $D_1, \ldots, D_k$ and taking their disjoint union (up to one common vertex v). For every $i \leq k$, D_i is a graph database over $\mathbb{B}$ which has a copy of the transition graph of $\mathcal{A}_i$ along with additional i vertices $v, v_1^i, \ldots, v_{i-1}^i$, one edge from v to the initial state of $\mathcal{A}_i$ labeled by $\#$, and for every final state q_f of $\mathcal{A}_i$ we construct the path

$$q_f \xrightarrow{\$} v_1^i \xrightarrow{\$} v_2^i \ldots \xrightarrow{\$} v_{i-1}^i$$

reading $\$^i$. Here is an example:



D is now the disjoint union of all D_i, up to the common vertex v. We have designed q in such a way that the valuation that maps every node variable of q to v in D is a witness for $D \vDash q$ if and only if $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$. On the other hand, any valuation which maps some node variable to a vertex other than v will not be a witness for $D \vDash q$, due to the delimiting symbols $\#, \$$. Thus we have produced D and q such that D, q is a 'yes' instance of p-eval-ECRPQ($\mathcal{C}$) if and only if $\{\mathcal{A}_i\}_{i \leq k}$ is a positive instance of p-IE. The size of D is linear with respect to size of $\{\mathcal{A}_i\}_{i \leq k}$. Further, the size of q depends only on k and $\mathcal{C}$. Since $\mathcal{C}$ is computably enumerable, the size of q is bound by a computable function. Therefore, this reduction is FPT.

Case (2) [Unbounded hyper-edges]: In this case, we simply adapt the FPT reduction shown in Case (1) by finding a 2L-graph G with a hyper-edge h of size $r \geq k$. Fix G and h. Next, we intialize an ECRPQ q with abstraction G. Let $R(\pi_1, \ldots, \pi_r)$ be the relation symbol corresponding to h, and define its corresponding synchronous relation as

$$R = \{\#w\$, \#w\$\$, \ldots, \#w\$^k \colon w \in \mathbb{A}^*\} \times \mathbb{U}_{r-k}(\mathbb{B}^*) \subseteq \mathbb{U}_r(\mathbb{B}^*)$$

Every other relation R' is defined as the universal relation $\mathbb{U}_{\text{arity}(R')}(\mathbb{B}^*)$. All of these synchronous relations are produced in polynomial time, and D is defined as in Case (1).

Now, assume $D \vDash q$. Then there exists a witness tuple of paths $(\pi_1, \ldots, \pi_r)$ whose labels are in R. Moreover, the $\{1, \ldots, k\}$-projection of their labels is of the form $(\#w\$, \#w\$\$, \ldots, \#w\$^k)$ for some $w \in \mathbb{A}^*$. Since D is a union of D_i with a common vertex v (see Case (1) for its definition), $w \in L(\mathcal{A}_i)$ for each $i \in \{1, \ldots, k\}$. Therefore

$$L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k) \neq \emptyset$$

Conversely, assume $L(\mathcal{A}_1) \cap \cdots \cap L(\mathcal{A}_k)$ contains some word w. Then for each i, there exists a path $v \xrightarrow{\pi_i} v$ in D_i labeled by $\#w\$^i$. Moreover, $\pi_{k+1}, \ldots, \pi_r$ have some non-empty words as their labels, as do all the path variables of q which are not $\pi_1, \ldots, \pi_r$. Thus we obtain a valuation that witnesses $D \vDash q$. We conclude that $D \vDash q$ if and only if $L(\mathcal{A}_1) \ldots L(\mathcal{A}_k) \neq \emptyset$. □

4.3 Final proofs

Having proven the necessary prerequisites, we put all our lemmas together to give a final proof of the two ECRPQ evaluation theorems. We begin with the combined complexity:

Theorem 1. *[Combined complexity of* EVAL-ECRPQ*] Let $\mathcal{C}$ be a cc-tame class of 2L-graphs. Assuming* W[1] $\neq$ FPT*:*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ or $cc_{h\text{-}edge}(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) *is* PSPACE*-complete, or*

(2) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) *is in* NP *but not in* PTIME*, or*

(3) if $cc_{vertex}(\mathcal{C}), cc_{h\text{-}edge}(\mathcal{C}), tw(\mathcal{C}) < \infty$ then EVAL-ECRPQ($\mathcal{C}$) *is in polynomial time.*

Proof. We examine each case separately:

(1) By Lemma 5, we know that if $\text{cc}_{\text{vertex}}(\mathcal{C}) = \infty$ or $\text{cc}_{\text{h-edge}}(\mathcal{C}) = \infty$, then EVAL-ECRPQ($\mathcal{C}$) is PSPACE-hard. It was shown in [15] that EVAL-ECRPQ is in PSPACE. PSPACE-completeness follows by definition.

(2) If EVAL-ECRPQ($\mathcal{C}$) was in polynomial time, then P-EVAL-ECRPQ($\mathcal{C}$) would be FPT. However, (2) of Theorem 2 states that if $\text{cc}_{\text{vertex}}(\mathcal{C}) = \infty$ then P-EVAL-ECRPQ($\mathcal{C}$) is W[1]-complete. So EVAL-ECRPQ($\mathcal{C}$) being in PTIME would imply FPT = W[1], contradicting our assumption that FPT $\neq$ W[1]. Therefore, EVAL-ECRPQ($\mathcal{C}$) is in NP but not in PTIME.

For the upper bound, we know that EVAL-CQ is in NP. So we use the polynomial time reduction to EVAL-CQ detailed in (2) of Lemma 4 to obtain that EVAL-ECRPQ($\mathcal{C}$) is in NP.

(3) We again use the polynomial time reduction to EVAL-CQ explained in (2) of Lemma 4. Further, $tw(\mathcal{C}^{\text{node}}) = tw(\mathcal{C}) < \infty$, so EVAL-CQ($\mathcal{C}^{\text{node}}$) is in polynomial time by (1) of Proposition 2.

□

Next, we prove the parameterized ECRPQ evaluation theorem:

Theorem 2. *[Parameterized complexity of* p-eval-ECRPQ*] Let $\mathcal{C}$ be a class of 2L-graphs.*

(1) if $cc_{vertex}(\mathcal{C}) = \infty$ then p-eval-ECRPQ($\mathcal{C}$) *is* XNL*-complete, or*

(2) if $cc_{vertex}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}) = \infty$, then p-eval-ECRPQ($\mathcal{C}$) *is* W[1]*-complete, or*

(3) if $cc_{vertex}(\mathcal{C}), tw(\mathcal{C}) < \infty$ then p-eval-ECRPQ($\mathcal{C}$) *is* FPT*.*

Proof. We again go on a case-by-case basis:

(1) The upper and lower bounds follow from Lemmas 3 and 8 respectively.

(2) Assume $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) < \infty$ and $tw(\mathcal{C}) = \infty$. By Lemma 6, we know that $tw(\mathcal{C}^{\mathrm{collapse}}) = \infty$. Corollary 1 implies that p-eval-$\mathrm{CQ}_{\mathrm{bin}}(\mathcal{C}^{\mathrm{collapse}})$ is W[1]-complete. To establish the lower bound, we apply the FPT reduction of Lemma 7 to obtain that p-eval-ECRPQ($\mathcal{C}$) is W[1]-hard.

For the upper bound, we combine the upper bounds given by (1) of Lemma 4 and (2) of Proposition 2.

(3) When $tw(\mathcal{C}^{\mathrm{node}}) < \infty$, we know that p-eval-$\mathrm{CQ}(C^{\mathrm{node}})$ is FPT (by Proposition 2). Moreover, when when $\mathrm{cc}_{\mathrm{vertex}}(\mathcal{C}) < \infty$, there exists an FPT reduction from p-eval-ECRPQ($\mathcal{C}$) to p-eval-$\mathrm{CQ}(C^{\mathrm{node}})$ (see (2) of Lemma 4). Therefore p-eval-ECRPQ($\mathcal{C}$) is FPT.

□

4.4 Related work

4.4.1 The case of non-Boolean ECRPQs

In all the instances of the query evaluation problem mentioned above, we only considered Boolean queries, that is, queries that return only True or False when evaluated on a database. These queries have free node variables (in the case of CQs and CRPQs), and free node and path variables in the case of ECRPQs. For eval-CQ and eval-CRPQ, all the procedures that we considered have been extended to non-Boolean queries. We now show how instances of eval-ECRPQ can be handled when considering non-Boolean ECRPQs. We split non-Boolean ECRPQs into two groups: queries that have free node variables only (and hence output only nodes), and queries that have free path variables as well, outputting nodes and paths. We call the former "node-only non-Boolean ECRPQs".

Node-only non-Boolean ECRPQs

These are ECRPQs whose path variables are all existentially quantified, and that have a non-empty set of free node variables. Given an ECRPQ q with a set of k free node variables, and a graph database D, there are only polynomially many elements in $ans(q, D)$ (bounded by $|D|^k$). Therefore, any polynomial time reduction from EVAL-CQ to EVAL-ECRPQ and vice-versa remains in polynomial time when the non-Boolean version of these problems are considered.

Non-Boolean ECRPQs which output paths

Unlike node-only non-Boolean ECRPQs, the output of a general non-Boolean ECRPQ q evaluated on a graph database D might be infinite in size. This is because potentially infinitely many paths may satisfy a reachability condition. The number of elements in the output $ans(q, D)$ can be restricted, we can either limit the allowed paths by placing an upper bound on their length, or by asking for the shortest path which satisfies the conditions of the query. Given a regular language L which is recognized by a DFA $\mathcal{A}$, the length of the shortest word in L is bounded above by the number of states in $\mathcal{A}$. Therefore, the length of the shortest path in a graph database D which appears in $ans(q, D)$ is linear in $|q|$ and $|D|$. Thus restricting the output by only allowing the shortest paths is an effective way to generate an output of finite size from the potentially infinite set $ans(q, D)$. A precise formulation of these ideas is an avenue for future work.

4.4.2 The containment problem

In the context of query formalisms, *query containment* is an important decision problem that has been widely studied alongside query evaluation. We know that a query q evaluated on a database D yields a set of answers $ans(q, D)$. Let (q, q') be a pair of queries (q, q') belonging to an arbitrary querying formalism. We say that q *is contained in* q', denoted by $q \subseteq q'$, if for every database D (corresponding to that formalism), we have $ans(q, D) \subseteq ans(q', D)$. The containment problem for a general query formalism $\mathcal{Q}$ is stated as follows:

QUERY CONTAINMENT (CONT-$\mathcal{Q}$); where $\mathcal{Q}$ is a set of queries
Input: A pair of queries $q, q' \in \mathcal{Q}$.
Question: Is $q \subseteq q'$?

The containment problem is useful in several applications related to querying formalisms. Containment algorithms are crucial in optimizing query evaluation algorithms [80, 87], knowledge base verification [78, 79] and information integration [27, 77].

Chandra and Merlin proved that CONT-CQ is NP-complete [28]. Chekuri and Rajaraman introduced the notion of *query width* of a CQ and gave an algorithm for CONT-CQ that runs in time polynomial to n^k, where n is the size of the input and k is the query width [29].

In regard to query formalisms in graph databases, it is easy to see that CONT-RPQ is PSPACE-complete. This follows from the reduction of CONT-RPQ to the containment problem for regular languages; the containment of $x \xrightarrow{L_1} y$

in $x \xrightarrow{L_2} y$ boils down to checking whether $L_1 \subseteq L_2$. The containment problem for regular languages is known to be PSPACE-complete [67]. CONT-CRPQ is known to be significantly harder, being EXPSPACE-complete [24, 51]. This problem continues to remain EXPSPACE-hard for classes of CRPQs abstracted by multigraphs containing only two nodes; the required reduction is obtained by considering the containment of $x \xrightarrow{L} y$ in $\bigwedge_i x \xrightarrow{L_i} y$ for arbitrary regular languages $\{L, L_1, \ldots, L_i\}$.

It was shown by Libkin et al. that CONT-ECRPQ is undecidable [15]. In fact, deciding whether an ECRPQ q is contained in a CRPQ q' is undecidable. An even stronger result can be stated: let (q, q') be a pair of queries such that one of them is a CRPQ and the other is an ECRPQ that only uses unary relations in REG and the (binary) equal length relation eq, then it is undecidable to check whether $q \subseteq q'$ [52]. Therefore any attempt at finding subclasses of ECRPQs whose containment is decidable is restricted by the class of relations used in the query class. While the structural conditions to improve the complexity of CRPQ containment (see [46]) have been identified, no such result has been given for ECRPQs yet. The task of specifying restrictions on (i) the 2L-graph abstractions of these subclasses, and (ii) the sets of relations that are used in them, to obtain decidability of the containment problem, remains open.

Part II

Relations

Chapter 5

The theory of synchronous relations

In Part I, we studied the evaluation problem for Extended CRPQs based upon an analysis of their structural properties. Recall that ECRPQs extend CRPQs with the ability to specify synchronous relations on path labels. In characterizing ECRPQs with 2L-graphs, we abstracted away these relations using relation symbols. It follows that every 2L-graph abstracts a set of ECRPQs, each determined by a particular choice of synchronous relations assigned to the path variables.

Consider two ECRPQs having the same underlying 2L-graph, but different sets of relations mapped onto their common relation symbols. Though structurally identical, these two queries might describe very different properties, depending upon the synchronous relations used in them. We have already studied ECRPQs by way of their graphical structure; in this part, we focus on studying the relations used in them.

5.1 Overview

Recall that Synchronous Relations, or Sync, is the class of relations recognized by synchronous automata. We defined them formally in Section 1.5.5 in Chapter 1. Synchronous automata are finite multi-tape automata whose heads move synchronously on the tapes. Among the existing models of automata-definable word relations, Sync is seen as a natural analogue of Reg, the set of regular languages. It is also related closely to the letter-to-letter transducer model, as these transducers capture the set of equal length synchronous relations. In terms of expressive power, synchronous relations cover Rec, or recognizable relations, which are given by Cartesian products of regular languages closed under finite union. Meanwhile Sync is a strict subclass of Rat, or rational relations, which are rational subsets of $(\mathbb{A}^*)^k$, over all $k \in \mathbb{N}$. Synchronous relations also have all the robust characteristics of regular languages, being a Boolean algebra closed under projection and homomorphism. This part of the thesis is devoted to the study of synchronous relations from the point-of-view of its logical characterization.

Logic. Traditionally, regular languages have been studied from multiple perspectives; language-theoretic, algebraic, and logical characterizations have been given for REG in the literature. Büchi showed that REG is precisely the set of languages definable in MSO($<$), that is, the Monadic Second Order logic over finite words (with the predicate $x < y$ signifying that x is assigned a prior position to y in the word) [20]. This seminal result has been the foundation for much of the study of regular languages. In particular, a lot of attention has been given to studying the expressivity of *fragments* of MSO($<$); that is subsets of MSO($<$) sentences formed by restricting their structure by imposing some syntactic conditions. These fragments are important insofar as they have good closure properties and admit effective characterizations that are interesting from a combinatorial point of view. Moreover, decision problems that are computationally hard for the full logic may turn out to be tractable for these smaller fragments.

The set of first order MSO sentences, denoted as FO($<$), is one such important fragment. It defines the class of star-free or aperiodic languages, given by regular expressions without using the Kleene star. In the algebraic world, these languages are captured by aperiodic monoids [105]. Any formula in first order logic can be translated to its *prenex normal form*, a standard syntactic format containing blocks of quantifiers followed by a quantifier-free formula. This leads to the *quantifier alternation hierarchy*, whereby the set of FO formulæ is stratified into subsets called *levels* of the hierarchy. A formula is in level i in the hierarchy if its prenex normal form contains i alternating blocks of quantifiers (see Definition 29). The lower levels of the FO($<$) quantifier alternation hierarchy have been effectively characterized.

Our topic of study is the logical characterization of SYNC. Eilenberg, Elgot, and Shepherdson [43] showed that a relation is synchronous if and only if it is definable by a formula in the first order theory of words with the prefix, equal length, and last letter predicates in its signature, abbreviated as FO[σ]. We further explore this result by considering the quantifier alternation hierarchy of FO[σ]. A corollary that is surprising at first glance is that FO[σ] collapses to its $\mathbf{\Sigma}_3[\sigma]$ fragment, the third level in the hierarchy. Therefore, the only members in this hierarchy are $\mathbf{\Sigma}_1[\sigma], \mathbf{\Sigma}_2[\sigma], \mathbf{\Sigma}_3[\sigma]$ - the first, second, and third levels respectively – as well as the Boolean closures of the first two, denoted by $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ respectively. We effectively characterize the set of relations defined by each of these fragments, focusing on the *membership problem*, which asks, for a given relation R and a fragment $\mathcal{F}$, if R is definable by a formula in $\mathcal{F}$. We show that membership is decidable for all levels of the FO[σ] quantifier alternation hierarchy.

Contributions. There is an interesting link between the lower levels of the quantifier alternation hierarchies of FO[σ] and FO($<$). It is well-known that $\mathbf{\Sigma}_1(<)$-definable languages are precisely those which are upward-closed under the subword relation [115]. We generalize this relation to k-synchronized words, and call this the *synchronized subword*. Then we prove a similar characterization result for $\mathbf{\Sigma}_1[\sigma]$-definable relations: a synchronous relation is $\mathbf{\Sigma}_1[\sigma]$ if and only if its synchronized language is upward-closed under the synchronized subword relation. A similar generalization occurs in the characterization of $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable relations too. A $\mathcal{B}\mathbf{\Sigma}_1(<)$ language is totally fixed by fixing its set of

k-length subwords (for some k which depends on the language). Analogously, we show that a $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ relation is fixed by fixing a finite set of synchronized subwords.

The characterization of $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ follows even more directly from their FO($<$) counterparts. We show that a relation is in $\mathbf{\Sigma}_2[\sigma]$-definable if and only if its synchronized language is $\Sigma_2(<)$-definable; as a corollary we get the corresponding characterization theorem for $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ definable relations as well.

Organization. We begin with a brief discussion on regular languages, logic, and automata-definable relations. Then, we define FO[σ] and illustrate its quantifier alternation hierarchy in more detail. Finally, we explain our goals and state our contributions. All the technical details, lemmas, and proofs are presented in the next chapter (see Chapter 6).

5.2 Characterizing regular languages

5.2.1 Monadic Second Order logic

A standard result in automata theory is the characterization of regular languages by Monadic Second Order (MSO($<$)) formulæ, proved by Büchi in 1960 [20]. MSO($<$) formulæ are constructed using first and monadic second order variables, Boolean operators, as well as the predicates $x < y$ and $x \in X$, where x, y are first order variables and X is a second order variable.

Definition 26. Fix infinite sets $\mathcal{V}_1 = \{x, y, z \dots\}, \mathcal{V}_2 = \{X, Y, Z \dots\}$ of first and second order variables respectively. Fix a finite alphabet $\mathbb{A}$. Let MSO($<$) denote the set of all formulæ generated by the production rules

$$\mathbf{a}(x)_{a\in\mathbb{A}} | x \in X | x < y | \exists x\varphi | \exists X\varphi | \forall x\varphi | \forall X\varphi | \varphi_1 \wedge \varphi_2 | \varphi_1 \vee \varphi_2 | \neg\varphi$$

If a variable x (alternatively, X) occurs in φ in the form of a subformula $\exists x\psi$ or $\forall x\psi$ (alternatively, $\exists X\psi$ or $\forall X\psi$) then its occurrence is said to be **bound** within the scope of that quantifier.

The occurrence of x or X in φ is said to be **free** if it is not bound. Moreover, φ is said to be a **sentence** if it has no free variables. If φ contains no bound occurrences of variables, then it is said to be **quantifier-free**.

Definition 27. FO($<$) is precisely the set of MSO($<$) formulæ in which no second order variables occur.

Semantics

Here we use a presentation of MSO($<$) semantics given due to Straubing [111]. Fix a word $w = a_1 \dots a_n \in \mathbb{A}^*$. Let φ be an MSO formula with F and S being the set of its first order and second order free variable occurrences respectively. Hereby we conflate the notion of variables and their occurrences in formulæ and

refer to variables as being "free" or "bound" within the quantifiers in which they occur.

A (w, F, S)-*decorated word* is of the form

$$\tilde{w} = (a_1, F_1, S_1)(a_2, F_2, S_2)\dots(a_n, F_n, S_n) \in (\mathbb{A} \times \mathcal{P}(F) \times \mathcal{P}(S))^*$$

where

- $F_i \cap F_j = \emptyset$ whenever $i \neq j$, and
- $F = \bigcup_{i=1}^{n} F_i$

We now inductively define the conditions under which a (w, F, S)-decorated word $\tilde{w}$ *satisfies* φ. The induction is on the recursive definition of MSO formulæ. We define the semantics for each subformula φ' of φ, whose free first order and second order variables are, say, F' and S' respectively (with $F \subseteq F'$ and $S \subseteq S'$), and a (w, F', S')-decorated word $\tilde{w}'$ is mapped to $\tilde{w}$ via the homomorphism

$$(a, \hat{F}, \hat{S}) \mapsto (a, (\hat{F} \setminus F') \cup F, (\hat{S} \setminus S') \cup S)$$

extended to $(\mathbb{A} \times \mathcal{P}(F') \times \mathcal{P}(S'))^*$. Since regular languages are closed under homomorphism, defining our semantics this way preserves recognizability of regular languages. We now consider the case when φ' is a quantifier free formula. The word $\tilde{w} = (a_1, F_1, S_1)\dots(a_n, F_n, S_n)$ satisfies

- $x < y$ if there exist $i, j \in \{1, \dots, n\}$ such that $i < j$, $x \in F_i$ and $y \in F_j$.
- $\mathbf{a}(x)$ for some $a \in \mathbb{A}$ if there exists $i \in \{1, \dots, n\}$ such that $x \in F_i$ and $a_i = a$.
- $x \in X$ if there exists $i \in \{1, \dots, n\}$ such that $x \in F_i$ and $X \in S_i$.
- $\varphi_1 \wedge \varphi_2$ if it satisfies both φ_1 and φ_2.
- $\varphi_1 \vee \varphi_2$ if it satisfies either φ_1 or φ_2.
- $\neg\psi$ if it does not satisfy ψ (for any subformula ψ of φ).
- $\exists x.\varphi'$ (where φ' is a subformula of φ whose first and second order free variables are F', S' respectively) if there exists some $\tilde{w}'$ which satisfies φ' and $\tilde{w}$ is the image of $\tilde{w}'$ under the homomorphism defined by extending

  $$(a, \hat{F}, \hat{S}) \mapsto (a, \hat{F} \setminus \{x\}, f_{-x}(\hat{S}))$$

  (where $f_{-x}(\{F_1, F_2, \dots, F_\ell\}) = \{F_1 \backslash \{x\}, F_2 \backslash \{x\}, \dots, F_\ell \backslash \{x\}\}$) to $(\mathbb{A} \times \mathcal{P}(F') \times \mathcal{P}(S'))^*$.
- $\exists X.\varphi'$ (where φ' is a subformula of φ whose first and second order free variables are F', S' respectively) if there exists some $\tilde{w}'$ which satisfies φ' and $\tilde{w}$ is the image of $\tilde{w}'$ under the homomorphism defined by extending

  $$(a, \hat{F}, \hat{S}) \mapsto (a, \hat{F}, \hat{S} \setminus \{X\})$$

  to $(\mathbb{A} \times \mathcal{P}(F') \times \mathcal{P}(S'))^*$.

Definition 28. The **language defined** by an MSO(<) formula φ (whose sets of first order and second order free variables are denoted by F, S, respectively), denoted $|\varphi|$ is the set of all words in $(\mathbb{A} \times \mathcal{P}(F) \times \mathcal{P}(S))^*$ that satisfy it.

When φ is a sentence, F and S are empty. Then we define $|\varphi|$ be the subset of $\mathbb{A}^*$, given by the homomorphism $(a, \emptyset, \emptyset) \mapsto a$ extended over $(\mathbb{A} \times \mathcal{P}(\emptyset) \times \mathcal{P}(\emptyset))^*$.

Let $\mathcal{F} \subseteq \text{MSO}(<)$. Then we define by $\mathcal{F}$ the subset of languages

$$\{|\varphi| : \varphi \in \mathcal{F}\}$$

These semantics establish the correspondence between MSO(<) formulæ and regular languages over decorated words.

Proposition 5 (Straubing, 1994 [111])**.** *Let F and S be sets of first and second order variables respectively. A language L over $\mathbb{A} \times \mathcal{P}(F) \times \mathcal{P}(S)$ is regular if and only if $L = |\varphi|$ for some MSO(<) formula φ whose first order and second order free variables are subsets of F and S respectively.*

Theorem 3 (Büchi, 1960 [20])**.** *A language $L \subseteq \mathbb{A}^*$ is regular if and only if $L = |\varphi|$ for some MSO(<) sentence φ.*

For any $\mathcal{F} \subseteq \text{MSO}(<)$, we have thereby the $\mathcal{F}$-membership problem, stated as:

MEMBERSHIP-$\mathcal{F}$ (where $\mathcal{F} \subseteq \text{MSO}(<)$)
Input: A regular language L.
Question: Is L definable by a formula in $|\mathcal{F}|$?

Example 24. For all variables x, y, let $x \leq y$ be shorthand for $x < y \vee x = y$. Some FO(<) formulæ are

$$\begin{aligned} \mathit{successor}(x, y) &= x < y \wedge \forall z (z < y \implies z \leq x) \\ \mathit{first}(x) &= \forall y (x \leq y) \\ \mathit{last}(x) &= \forall y (y \leq x) \end{aligned}$$

Example 25. The language of all even length words is defined by the sentence

$$\exists X_e \exists X_o \Big(\big(\exists x \exists y (\mathit{first}(x) \wedge x \in X_o \wedge \mathit{last}(y) \wedge y \in X_e) \big) \\ \wedge \forall x_1, x_2 (\mathit{successor}(x_1, x_2) \implies (x_1 \in X_o \wedge x_2 \in X_e) \vee (x_1 \in X_e \wedge x_2 \in X_o)) \Big)$$

which encodes the set of even and odd positions in the subset variables X_e, X_o respectively, using the first order definable formulæ *first*, *last* and *successor*. ■

5.2.2 Regular languages definable in FO(<)

As stated earlier, FO(<) is the subset of MSO(<) formulæ in which no second order variables occur. It was shown by McNaughton and Papert [84] that the set of languages definable in FO(<) is precisely the set of star-free languages (see Section 1.5.2, Definition 1 in Chapter 1 for a definition of star-free regular expressions). This logical characterization of star-free languages, along with Schützenberger's algebraic characterization of them [105], shows that this is a robust subclass of regular languages with interesting properties. A short presentation of algebraic characterization of regular languages is given in Section 1.5.4 of Chapter 1.

5.2.3 Quantifier alternation of FO(<)

Using DeMorgan's laws and semantic rules, any first-order formula ψ can be re-written as an equivalent formula of the form

$$\{\exists, \forall\}^*.(\text{some quantifier-free formula})$$

This re-writing is a standard transformation and this is called the *prenex normal form* of ψ, unique up to logical equivalence. Two important features of a formula in prenex normal form are: (i) the alternation of the existential and universal blocks $\exists^*$ and $\forall^*$, and (ii) whether the normal form begins with $\exists$ or $\forall$. This gives rise to the *quantifier alternation hierarchy*, a stratification of FO formulæ into *levels* of increasing complexity. These levels are named Σ_i and Π_i; a formula is placed in them depending upon its outermost quantifier (Σ denotes $\exists$ and Π denotes $\forall$), and i is the maximum number of (possibly) alternating $\exists^*$ and $\forall^*$ blocks. Formally, we have

Definition 29. Let \$ be a predicate signature, and FO[\$] denote the set of first order formulæ over it. We define $\mathbf{\Sigma}_0[\$] = \mathbf{\Pi}_0[\$]$ to be the set of quantifier-free formulæ.

For each $i \in \mathbb{N}$, we denote by $\mathbf{\Sigma}_i[\$]$ the subset of $\mathbf{FO}[\$]$ whose elements have the prenex normal form

$$\underbrace{\exists^*\forall^* \dots}_{\text{at most } i \text{ alternating blocks}} \quad \varphi$$

for some quantifier-free formula φ. Next, for each $i \in \mathbb{N}$, we denote by $\mathbf{\Pi}_i[\$]$ the subset of $\mathbf{FO}[\$]$ whose elements have the prenex normal form

$$\underbrace{\forall^*\exists^* \dots}_{\text{at most } i \text{ alternating blocks}} \quad \varphi$$

for some quantifier-free formula φ. The set of Boolean combinations of $\mathbf{\Sigma}_i[\$]$ is denoted by $\mathcal{B}\mathbf{\Sigma}_i[\$]$, which equals $\mathcal{B}\mathbf{\Pi}_i[\$]$.

Recall that a sentence is a logical formula without any free variables. For every $i \in \mathbb{N}$, we denote by $\mathbf{\Sigma}_i(\$)$ the set of sentences in $\mathbf{\Sigma}_i[\$]$.

Since the formulæ in $\mathbf{\Sigma}_i[\$]$ can have up to i alternating blocks of quantifiers in their prenex normal form, we have the inclusions $\mathbf{\Sigma}_i[\$] \subseteq \mathcal{B}\mathbf{\Sigma}_i[\$] \subseteq \mathbf{\Sigma}_{i+1}[\$]$ and $\mathbf{\Pi}_i[\$] \subseteq \mathcal{B}\mathbf{\Pi}_i[\$] = \mathcal{B}\mathbf{\Sigma}_i[\$] \subseteq \mathbf{\Pi}_{i+1}[\$]$ for every $i \in \mathbb{N}$. This induces the quantifier alternation hierarchy

$$\mathbf{\Sigma}_1[\$] \subseteq \mathcal{B}\mathbf{\Sigma}_1[\$] \subseteq \mathbf{\Sigma}_2[\$] \subseteq \mathcal{B}\mathbf{\Sigma}_2[\$] \dots$$
$$\mathbf{\Pi}_1[\$] \subseteq \mathcal{B}\mathbf{\Sigma}_1[\$] \subseteq \mathbf{\Pi}_2[\$] \subseteq \mathcal{B}\mathbf{\Sigma}_2[\$] \dots$$

Recall that for any $\mathcal{F} \subseteq \mathrm{MSO}(<)$, we denote by $|\mathcal{F}|$ the set of regular languages definable by formulæ in $\mathcal{F}$. The problem of characterizing languages definable in the various levels of the hierarchy has been investigated in great depth for decades. In particular, the lower levels – $\mathbf{\Sigma}_1(<), \mathbf{\Sigma}_2(<), \mathbf{\Sigma}_3(<)$, and their Boolean combinations, admit effective combinatorial and algebraic characterizations [115, 109, 112, 110, 114, 11, 96].

5.2.4 Characterizing $\mathbf{\Sigma}_1(<)$ with subwords

It was shown by Thomas [115] that every $|\mathbf{\Sigma}_1(<)|$ language is a finite union of languages of the form

$$\mathbb{A}^* a_1 \mathbb{A}^* a_2 \dots a_n \mathbb{A}^*$$

for some $a_1, \dots, a_n \in \mathbb{A}$. This result can be restated using the subword relation (see Definition 5). Recall that we denote the subword relation by $\sqsubseteq$ and for any $S \subseteq \mathbb{A}^*$, $\uparrow S$ denotes the set of all words having a subword in S.

Theorem 4 ([115])**.** *For any regular language L, $L \in |\mathbf{\Sigma}_1(<)|$ if and only if $L = \uparrow L$.*

In fact, by Higman's lemma [66], the subword relation is a well-quasi order. Therefore, the characterization of $|\mathbf{\Sigma}_1(<)|$ can be restated as follows:

Theorem 5 ([115, 66])**.** *For any regular language L, $L \in |\mathbf{\Sigma}_1(<)|$ if and only if there exists a finite set $S \subseteq \mathbb{A}^*$ such that $L = \uparrow S$.*

The connection of subwords and $\mathbf{\Sigma}_1(<)$-definable languages extends to $\mathcal{B}\mathbf{\Sigma}_1(<)$-definable languages as well. These languages are called *piecewise-testable* and are given by equivalence classes of words that have the same set of bounded subwords.

Definition 30. For every $h \in \mathbb{N}$, we define an equivalence relation $\sim_h$ on $\mathbb{A}^*$ as follows: for all $u, v \in \mathbb{A}^*$, $u \sim_h v$ if and only if u and v have the same set of subwords of length at most h. Let $[u]_{\sim_h}$ denote the equivalence class of u generated by $\sim_h$. Further, we denote by $\mathscr{B}_{\sim_h}$ the set of finite unions of $[u_i]_{\sim_h}$, and let

$$\mathscr{B}_{\sim} = \bigcup_{h \in \mathbb{N}} \mathscr{B}_{\sim_h}$$

The languages in $\mathscr{B}_{\sim}$ are called **piecewise testable languages**.

The logical characterization of piecewise testable languages via $\mathcal{B}\mathbf{\Sigma}_1(<)$ sentences easily follows from their definition. The algebraic characterization of this

class of languages was given by Simon [109], using J-trivial monoids (see Section 1.5.4 in Chapter 1 for an introduction to the algebraic characterization of regular languages).

Theorem 6 (Simon 1975 [109])**.** *It is decidable to check whether a given regular language is piecewise testable.*

Moreover, Straubing and Thérien provided a more enlightening proof of Simon's theorem. They also formally proved that membership in piecewise testable languages is decidable [112]. Stern provided a more efficient algorithm for membership and showed that it is in PTime [110].

Theorem 7 (Straubing and Thérien '88[112], Stern '85[110])**.** *Checking whether a regular language is piecewise testable is decidable in polynomial time.*

5.2.5 Characterizing $\mathbf{\Sigma}_2(<)$ with polynomials

Finite unions of languages of the form

$$\mathbb{A}_1^* a_1 \mathbb{A}_2^* \dots a_n \mathbb{A}_n^*;\ \{a_1, \dots, a_n\}, \mathbb{A}_1, \dots, \mathbb{A}_n \subseteq \mathbb{A}$$

are called *polynomials*. It was first shown by Thomas [114] that the $\mathbf{\Sigma}_2(<)$-definable languages exactly corresponds to the set of polynomials, and Arfi [11] gave a decidable characterization of the membership problem for this set. The algebraic characterization was given by Pin and Weil [94]. Consolidated proofs can be found in [94] and Diekert et al. [38].

Theorem 8 (Arfi 1987 [11])**.** *It is decidable to check whether a language is a polynomial.*

The characterization of $\mathcal{B}\mathbf{\Sigma}_2(<)$-definable is more involved. Decidability of $|\mathcal{B}\mathbf{\Sigma}_2(<)|$ membership was proven fairly recently by Place and Zeitoun [96].

Theorem 9 (Place and Zeitoun 2014 [96])**.** *It is decidable to check whether a language is in* $|\mathcal{B}\mathbf{\Sigma}_2(<)|$.

5.3 Logically characterizing synchronous relations

We now describe Eilenberg et al.'s logical characterization of synchronous relations. Fix an alphabet $\mathbb{A}$.

Definition 31. We denote by $\mathbf{FO}[\sigma]$ the set of all first order formulæ over the signature $\sigma = \{\preceq, \mathsf{eql}(x,y), \{\ell_a\}_{a\in\mathbb{A}}\}$, interpreted as:

- $(w_1, w_2) \vDash x \preceq y$ if and only if w_1 is a prefix of w_2
- $(w_1, w_2) \vDash \mathsf{eql}(x,y)$ if and only if $|w_1| = |w_2|$
- $w \vDash \ell_a$ if and only if $w \in \mathbb{A}^* a$

A formula $\varphi \in \mathbf{FO}[\sigma]$ with k-free variables defines a k-ary relation which we denote by $\|\varphi\|$.

Example 26. Let $R_a = \{(ua, va) \colon u, v \in \mathbb{A}^* \text{ and } |u| = |v|\}$ for some $a \in \mathbb{A}$. Then R_a is defined by the $FO[\sigma]$ formula

$$\varphi_a(x, y) = \ell_a(x) \wedge \ell_a(y) \wedge \mathsf{eql}(x, y)$$

Theorem 10 (Eilenberg, Elgot, Shepherdson '69 [43])**.** *A relation on $\mathbb{A}^*$ is synchronous if and only if it is definable in* $\mathbf{FO}[\sigma]$.

Definition 32. The quantifier alternation hierarchy of $\mathbf{FO}[\sigma]$ consists of the classes of formulæ

$$\mathbf{\Sigma}_1[\sigma] \subseteq \mathcal{B}\mathbf{\Sigma}_1[\sigma] \subseteq \mathbf{\Sigma}_2[\sigma] \subseteq \mathcal{B}\mathbf{\Sigma}_2[\sigma] \ldots$$

along with the classes of their logical complements

$$\mathbf{\Pi}_1[\sigma] \subseteq \mathcal{B}\mathbf{\Pi}_1[\sigma] \subseteq \mathbf{\Pi}_2[\sigma] \subseteq \mathcal{B}\mathbf{\Pi}_2[\sigma] \ldots$$

We study the *membership problem* for the various fragments of $\mathbf{FO}[\sigma]$, which is stated as follows:

MEMBERSHIP-$\mathcal{F}$ (where $\mathcal{F} \subseteq \mathrm{FO}[\sigma]$).
Input: A relation $R \in$ SYNC.
Question: Is $R \in \|\mathcal{F}\|$?

5.4 Contributions

The characterization theorem of Eilenberg et al. states that: (i) every $\mathrm{FO}[\sigma]$ formula with k free variables defines a k-ary synchronous relation, and (ii) every synchronous relation is defined by some $\mathrm{FO}[\sigma]$ formula. The easier direction in its proof is (i), which relies on the atomic predicates - equal length, last letter, and prefix - all being synchronous relations, as well as some closure properties of SYNC. When proving (ii), we take a DFA recognizing a synchronized language and build a $\mathrm{FO}[\sigma]$ formula defining the corresponding relation.

In essence, we encode the behavior of the DFA (its transition relation) as a synchronized word and then describe its properties using $\mathrm{FO}[\sigma]$ formulæ. Here we come across an interesting observation: irrespective of the DFA, the formula constructed is always in $\mathbf{\Sigma}_3[\sigma]$, the third level of the quantifier alternation hierarchy. The corollary of this result is the $\mathbf{\Sigma}_3[\sigma]$ collapse. The detailed proof is given in Theorem 12, Section 6.2.

Theorem 11. *(Extension of [43]) Over any alphabet containing at least two letters,*

$$\|\mathbf{\Sigma}_3[\sigma]\| = \|\mathbf{FO}[\sigma]\| = \text{SYNC}$$

At a first glance, this is a surprising result because such a collapse doesn't occur for $\mathrm{FO}(<)$. Moreover, it provides an opportunity to characterize the sets of relations definable in each level of the quantifier alternation hierarchy of $\mathrm{FO}[\sigma]$, namely $\mathbf{\Sigma}_1[\sigma]$ and $\mathbf{\Sigma}_2[\sigma]$. In addition to that, we also address the question of characterizing relations definable in their respective Boolean closures, that is, $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$.

5.4.1 The first level

In order to characterize $\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$, we introduce the notion of *synchronized subword*. As mentioned previously, this is a generalization of the classical subword notion. We give formal definitions of both subword and synchronized subword in Section 6.3 (see Definition 5). For now, we define some notation:

- When $\bar{u}, \bar{v}$ are synchronized words (of tuples of the same arity), we write $u \sqsubseteq_s v$ to denote that $\bar{u}$ is a synchronized subword of $\bar{v}$.
- Moreover, if S and S' are sets of $\mathbb{A}^*$ words and k-ary synchronized words respectively (for some k), then $\uparrow S$ and $\uparrow_s S'$ are defined as:

$$\uparrow_s S' = \{\bar{v} \text{: there exists } \bar{u} \in S' \text{ such that } \bar{u} \sqsubseteq_s \bar{v}\}$$

- Let $h \in \mathbb{N}$. We define an equivalence relation on SW_k as follows: $\bar{w} \equiv_h \bar{w}'$ if and only if $\bar{w}$ and $\bar{w}'$ have the same set of synchronized subwords of length at most h. Let $[\bar{w}]_{\equiv_h}$ denote the equivalence class of $\bar{w}$ generated by $\equiv_h$, and let $\mathscr{B}_{\equiv_h}$ denote the set of finite unions of $[\bar{w}_i]_{\equiv_h}$, and let

$$\mathscr{B}_{\equiv} = \bigcup_{h \in \mathbb{N}} \mathscr{B}_{\equiv_h}$$

The relations definable in $\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ are characterized. In the following statements, L_R refers to the synchronized language of R.

Theorem ($\mathbf{\Sigma}_1[\sigma]$-characterization)**.** *For any relation R, $R \in \|\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R = \uparrow_s L_R$.*

Theorem ($\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-characterization)**.** *For any relation R, $R \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R \in \mathscr{B}_{\equiv}$.*

5.4.2 The second level

The characterization theorems for $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ parallel their classical counterparts much more closely.

Theorem ($\mathbf{\Sigma}_2[\sigma]$-characterization)**.** *For any relation R, $R \in \|\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R \in |\Sigma_2[<]|$.*

Theorem ($\mathcal{B}\mathbf{\Sigma}_2[\sigma]$-characterization)**.** *For any relation R, $R \in \|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ if and only if $L_R \in |\mathcal{B}\Sigma_2[<]|$.*

Finally, we have the decidability of the membership problem for all levels of the quantifier alternation hierarchy of $FO[\sigma]$, stated as follows:

Theorem (Membership)**.** *Let $\mathcal{F} \in \{\mathbf{\Sigma}_1[\sigma], \mathbf{\Sigma}_2[\sigma], \mathcal{B}\mathbf{\Sigma}_1[\sigma], \mathcal{B}\mathbf{\Sigma}_2[\sigma]\}$ be a fragment of $FO[\sigma]$. Then* MEMBERSHIP-$\mathcal{F}$ *is decidable.*

Our results are published in [48]. In the next chapter, we delve into the proofs of these theorems and other technicalities.

Chapter 6

The FO[σ] quantifier alternation hierarchy

In this chapter, we give a decidable characterization of every level of the $\mathbf{FO}[\sigma]$ quantifier alternation hierarchy. We first define the notions of types and type sequences of synchronized words, which are crucial in proving our results. In Section 6.2 we give a formal proof of Eilenberg et al.'s theorem, showing in the process that $\mathbf{FO}[\sigma]$ collapses in expressive power to the third level of its quantifier alternation hierarchy. Following that, in Section 6.3, we characterize the synchronous relations definable in the $\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ fragments. We also prove that the membership problems for these fragments are decidable. In Section 6.4 we characterize the subsets of synchronous relations definable in $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$, and subsequently we show that their corresponding membership problems are decidable. We conclude Part II with a discussion on future work on relations, in Section 6.5.

6.1 Types and type sequences

Recall that for a finite alphabet $\mathbb{A}$, and $k \in \mathbb{N}$, and a symbol $\perp \notin \mathbb{A}$, we let $\mathbb{A}_\perp = \mathbb{A} \cup \perp$ and define the $k-$*synchronized alphabet* of $\mathbb{A}$ to be $\mathbb{A}_\perp^k = \{(a_1, \ldots, a_k) \colon a_i \in \mathbb{A}_\perp\}$. With every k-tuple $(w_1, \ldots, w_k)$ of words in $\mathbb{A}^*$ we associate its *synchronized word* $w_1 \otimes \cdots \otimes w_k$, which is a word over $\mathbb{A}_\perp$.

This leads to the notion of *type* of a synchronized word, which is an equivalence relation over the indices of the tuple which carry non-empty words. The type of a synchronized word indicates which of its constituents are non-empty and equal to each other. For example, the synchronized word of a pair of non-empty words (w_1, w_2) has the type $\{(1,1),(2,2),(1,2),(2,1)\}$ if $w_1 = w_2$, and $\{(1,1),(2,2)\}$ otherwise (the presence of $(1,2)$ and $(2,1)$ in the type is a necessary and sufficient condition for $w_1 = w_2$). We formally define this notion below:

Definition 33. Let $\bar{a} = (a_1, \ldots, a_k) \in \mathbb{A}_\perp^k$. The **type** of $\bar{a}$ is the subset of $\{1, \ldots, k\}^2$ given by

$$\mathsf{type}(\bar{a}) = \{(i,j) \colon a_i = a_j \in \mathbb{A}\}$$

Furthermore, for any $\bar{w} \in (\mathbb{A}_\perp^k)^*$ we let $\mathsf{type}(\bar{w}) = \bigcap_{p \leq |\bar{w}|} \mathsf{type}(\bar{w}[p])$.

We make the following observations on type:

(1) if $\bar{u} \sqsubseteq \bar{u}'$, then $\mathsf{type}(\bar{u}) \supseteq \mathsf{type}(\bar{u}')$

(2) $|w_i| = |\bar{w}|$ if and only if $(i,i) \in \mathsf{type}(\bar{w})$.

(3) For each type T, let $T^\star = \{(i,j) \colon (i,i), (j,j) \in T\}$. Then for every $\bar{a} \in \mathbb{A}_\perp^k$ such that $T \subseteq \mathsf{type}(\bar{a}) \subseteq T^\star$, and for every pair $(i,j) \in T^\star$, we have $a_i = a_j \in \mathbb{A}$.

Definition 34. For each type T, T', we define the following sub-alphabets of $\mathbb{A}_\perp^k$:

$$\begin{aligned} \mathbb{A}_T &= \{\bar{a} \in \mathbb{A}_\perp^k \colon T \subseteq \mathsf{type}(\bar{a}) \subseteq T^\star\}) \\ \mathbb{A}_{-,T} &= \{\bar{a} \colon \mathsf{type}(\bar{a}) = T\} \\ \mathbb{A}_{T,T'} &= \{\bar{a} \in \mathbb{A}_\perp^k \colon T = T' \cap \mathsf{type}(\bar{a})\} \end{aligned}$$

Let us look at some examples of synchronized words and their types.

Example 27. Let $\mathbb{A} = \{a, b, c\}$. Here are the types of two 3-synchronized words

$$\mathsf{type}(ab \otimes ab \otimes ab) = \{(1,1),(2,2),(3,3),(1,2),(2,1),(1,3),(3,1),(2,3),(3,2)\}$$
$$\mathsf{type}(abaac \otimes abaab \otimes \varepsilon) = \{(1,1),(2,2)\}$$

We observe that $\bar{a} \in \mathbb{A}_T$ if and only if for all synchronized words $\bar{w}$ such that $\mathsf{type}(\bar{w}) = T$, we have $\mathsf{type}(\bar{w}\bar{a}) = T$. Put together with the earlier observations, this leads us to the notion of *type sequence* of a synchronized word, which is the (non-repeating) subsequence of types of its prefixes. Formally,

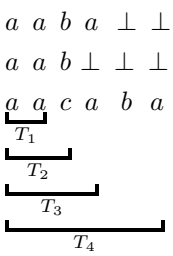
Definition 35. Let $\bar{w} = \bar{a}_1 \ldots \bar{a}_m$ be an arbitrary synchronized word. Let

$$\hat{T} = \{\mathsf{type}(\bar{a}_1 \ldots \bar{a}_i)\}_{1 \leq i \leq m}$$

be the sequence of types of successive prefixes of $\bar{w}$. The **type sequence** of $\bar{w}$ is the longest subsequence $\bar{T} = (T_1, \ldots, T_n)$ of $\hat{T}$ such that each of the T_i's are pairwise distinct.

By definition, every synchronized word $\bar{w} = w_1 \otimes \cdots \otimes w_k$ has exactly one type sequence. Then $\bar{w}$ can be factorized as $\bar{w} = \bar{w}_1 \cdots \bar{w}_n$ (where $\bar{w}_i$ is called the i-th **type factor** of $\bar{w}$) such that $\bar{w}_1$ is in $\mathbb{A}^*_{-,T_1}$ and for every $2 \leq i \leq n$, $\bar{w}_i \in \mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i}$. We say T_i is an **end type** in $\bar{T}$ if $|\bar{w}_1 \ldots \bar{w}_i| = |w_j|$ for some $1 \leq j \leq k$.

Example 28. The type sequence of a synchronized word $\bar{u} = aaba \otimes aab \otimes aacaba$ is (T_1, T_2, T_3, T_4), as displayed below.



where

$$T_1 = \{(1,1),(2,2),(3,3),(1,2),(2,1),(1,3),(3,1),(2,3),(3,2)\}$$
$$T_2 = \{(1,1),(2,2),(3,3),(1,2),(2,1)\}$$
$$T_3 = \{(1,1),(3,3)\}$$
$$T_4 = \{(3,3)\}$$

The type factors of $\bar{u}$ are

$$\bar{u}_1 = aa \otimes aa \otimes aa$$
$$\bar{u}_2 = b \otimes b \otimes c$$
$$\bar{u}_3 = a \otimes \varepsilon \otimes a$$
$$\bar{u}_4 = \varepsilon \otimes \varepsilon \otimes ba$$

6.2 FO[σ] collapses to $\mathbf{\Sigma}_3[\sigma]$

Eilenberg et al. showed that the set of relations defined by $\mathbf{FO}[\sigma]$ is exactly the set of synchronous relations [43]. We begin by re-proving this result.

Theorem 12 ([43])**.** *Over an alphabet containing at least two letters,*

$$\|\mathbf{FO}[\sigma]\| = \textsc{Sync}$$

From now on we assume that the alphabet $\mathbb{A}$ in consideration always contains at least two letters. First, we show that every $\mathbf{FO}[\sigma]$ formula defines a synchronous relation.

Lemma 9. *Let $\varphi(x_1, \ldots, x_k)$ be a formula in $\mathbf{FO}[\sigma]$. Then L_φ is a regular language over $\mathbb{A}_\perp^k$.*

Proof. We proceed by induction on the quantifier depth of $\mathbf{FO}[\sigma]$ formulæ. The atomic formulæ $\{\preceq, \mathsf{eq}, \{\ell_a\}_{a\in\mathbb{A}}\}$ all define synchronous relations (with ℓ_a defining the regular language $\mathbb{A}^*a$). Further, synchronous relations are closed under Boolean operations, so every quantifier free $\mathbf{FO}[\sigma]$ formula defines a synchronous relation. Now we consider the case where

$$\varphi = \exists x \psi(x_1, \ldots, x_k, x)$$

for some formula ψ of lower quantifier depth, such that ψ defines a $(k+1)$-ary synchronous relation given by a regular language L_ψ over $\mathbb{A}_\perp^{k+1}$. Now, $L_\varphi = \pi_{k+1}(L_\psi)$ and synchronized regular languages are closed under projection. This completes the proof. □

Proposition 6. *Let R be a synchronous relation. Then there exists some formula $\varphi \in \mathbf{FO}[\sigma]$ such that $L_\varphi = L_R$.*

Proof. Let $\mathcal{A}$ be a DFA recognizing L_R whose states are $Q = \{q_1, \dots, q_n\}$ (with starting state q_1, accepting states $F \subseteq Q$ and transition function δ). Without loss of generality we assume that $\mathbb{A}$ contains the letters 0 and 1. For every $i \in \{1, \dots, n\}$, we associate with q_i the synchronized letter $\bar{t}_i = (\underbrace{0, \dots, 0}_{i-1}, 1, \underbrace{0, \dots, 0}_{n-i})$. Moreover, for any pair of synchronized letters

$$\begin{aligned}\bar{a} &= (a_1, \dots, a_m) \in \mathbb{A}_\perp^m \\ \bar{b} &= (b_1, \dots, b_{m'}) \in \mathbb{A}_\perp^{m'}\end{aligned}$$

we denote by $\bar{a} + \bar{b}$ the synchronized letter $(a_1, \dots, a_m, b_1, \dots b_{m'}) \in \mathbb{A}_\perp^{m+m'}$.

Now, a synchronized word $\bar{w} = \bar{a}_1 \dots \bar{a}_m$ is in L_R if and only if there exists a run ρ of $\mathcal{A}$ on $\bar{w}$ of the form

$$q_{i_1} \xrightarrow{\bar{a}_1} q_{i_2} \dots q_{i_m} \xrightarrow{\bar{a}_m} q_{i_{m+1}}$$

where $q_{i_1} = q_1$ and $q_{i_{m+1}} \in F$. If this is the case, we let

$$\bar{w}_\rho = (\bar{a}_1 + \bar{t}_{i_1}) \cdot (\bar{a}_2 + \bar{t}_{i_2}) \cdots (\bar{a}_m + \bar{t}_{i_m}) \in \mathbb{A}_\perp^{k+n}$$

and call it the word *witnessing* ρ. The idea is to construct a $\mathbf{FO}[\sigma]$ formula of the form

$$\varphi(x_1, \dots, x_k) = \exists v_1 \dots \exists v_n \, \psi_{\text{run}}(x_1, \dots, x_k, v_1, \dots, v_n)$$

where ψ_{run} encodes the property:

$$\big(x_1 \otimes \cdots \otimes x_k \otimes v_1 \otimes \cdots \otimes v_k \text{ is a word witnessing an accepting run } \rho\big)$$

We denote $x_1 \otimes \cdots \otimes x_k$ and $v_1 \otimes \cdots \otimes v_n$ as $\bar{x}$ and $\bar{v}$ respectively. In order for $\bar{x} \otimes \bar{v}$ to witness an accepting run ρ, it must satisfy the following properties:

- The word $\bar{v}$ must encode a sequence of states, that is, for each $1 \leq i \leq n$, $v_i \in \{0,1\}^*$ such that $v_i[j] = 1$ if and only if q_i occurs in ρ at position j, otherwise $v_i[j] = 0$. Thus the v_i's all have the same length, say r, and for every $j \leq |r|$, the synchronized letter $v_1[j] \otimes \cdots \otimes v_n[j]$ contains 1 at exactly one component and 0 in all the others. We encode this property with the formula:

$$\begin{aligned}\psi_1(\bar{v}) = &\bigwedge_{i,j} \mathsf{eq}(v_i, v_j) \wedge \\ &\forall u_1, \dots, u_n \Big(\bigwedge_{i=1}^{i=n} (u_i \preceq v_i)) \implies \Big(\bigvee_{i=1}^{n} \ell_1(u_i) \wedge \big(\bigwedge_{j \neq i} \ell_0(u_j)\big)\Big)\end{aligned}$$

- The starting state must be q_1, that is the first letter of v_1 is 1. This is encoded by the formula

$$\psi_{\mathsf{start}} = \exists u (u \preceq v_1) \wedge \ell_1(u) \wedge \forall u' ((u' \preceq v \wedge u' \neq \varepsilon) \implies u \preceq u')$$

- If $\bar{a} + \bar{t}_j$ is the last letter of $\bar{x} \otimes \bar{v}$, then there exists a final state q_f such that $\mathcal{A}$ contains the transition $q_j \xrightarrow{\bar{a}} q_f$. For any pair of states $q, q' \in Q$, let $S(q, q') = \{\bar{a} \in \mathbb{A}_\perp^k : (q, \bar{a}, q') \in \delta\}$. For any $\bar{a} \in \mathbb{A}_\perp^k$ let $\tau(\bar{a}) = \{i \in \{1, \ldots, k\} : a_i \neq \perp\}$. Then the last letter property is encoded by the formula

$$\psi_{\mathsf{end}} = \bigwedge \ell_1(v_j) \wedge \ell_{\bar{a}}(x_1, \ldots, x_k)$$

  where the conjunction ranges over all triples $(\bar{a}, q_j, q_f)$ with $(q_j, q_f) \in Q \times F$, $\bar{a} \in S(q_j, q_f)$, and further

$$\ell_{\bar{a}}(x_1, \ldots, x_k) = \bigwedge_{i \in \tau(\bar{a})} \ell_{a_i}(x_i)$$

- The letters $\bar{a} + \bar{u}_j$ and $\bar{a}' + \bar{u}_{j'}$ occur consecutively in $\bar{x} \otimes \bar{v}$ if and only if there exists a δ-transition $q_{i_j} \xrightarrow{\bar{a}} q_{i_{j+1}}$. We denote this by the formula

$$\psi_{\mathsf{transition}} = \bigwedge \forall u, u' (u \preceq v_j \wedge u' \preceq v_{j'} \wedge \mathsf{length\text{-}successor}(u, u') \implies \left(\bigwedge_{i \in \tau(\bar{a})} \exists u'' (\ell_{a_i}(u'') \wedge \mathsf{eq}(u, u'') \wedge u'' \preceq x_i) \right)$$

  with the conjunction ranging over all triples $(j, j', \bar{a})$ where $j, j' \leq n, \bar{a} \in S(q_j, q_{j'})$, and $\mathsf{length\text{-}successor}(x, y)$ is true if and only if $|x| + 1 = |y|$, given as

$$\mathsf{length\text{-}successor}(x, y) = \forall y' ((y' \preceq y \wedge y' \neq y) \implies \exists x' (\mathsf{eq}(x', y') \wedge x' \preceq x))$$

Therefore, $\varphi(\bar{x}) = \exists \bar{v} (\psi_{\mathsf{start}} \wedge \psi_{\mathsf{end}} \wedge \psi_{\mathsf{transition}})$ is the required $\mathbf{FO}[\sigma]$ formula describing L_R. We note that $\psi_{\mathsf{start}} \in \mathbf{\Sigma}_1[\sigma]$, ψ_{end} is quantifier-free and $\psi_{\mathsf{transition}} \in \mathbf{\Pi}_2[\sigma]$. Therefore, $\varphi(\bar{x}) \in \mathbf{\Sigma}_3[\sigma]$. This leads to an important corollary. □

Corollary 2. *Let R be a synchronous relation. Then R is defined by a formula in $\mathbf{\Sigma}_3[\sigma]$.*

This shows that $\mathbf{FO}[\sigma]$ collapses to its $\mathbf{\Sigma}_3$ fragment in terms of expressive power. Therefore, we have our first characterization result:

Theorem 13. *Over any alphabet containing at least two letters,*

$$\|\mathbf{\Sigma}_3[\sigma]\| = \|\mathbf{FO}[\sigma]\| = \textsc{Sync}$$

It follows that characterizing $\mathbf{\Sigma}_1[\sigma]$ and $\mathbf{\Sigma}_2[\sigma]$-definable relations and their Boolean closures yields a complete characterization of $\mathbf{FO}[\sigma]$. Now we tackle the characterization results for these smaller fragments. We will also show the decidability of the membership problem for the classes of relations corresponding to them.

6.3 Characterizing $\|\mathbf{\Sigma}_1[\sigma]\|$ and $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$

Recall the *subword relation* on words in $\mathbb{A}^*$, which is also sometimes also called *subsequence/scattered subword* relation:

Let $u = a_1 \dots a_n, v = b_1 \dots b_m \in \mathbb{A}^*$. We say u is a **subword** of v, denoted by $u \sqsubseteq v$, if and only if there exists an increasing function $p\colon \{1,\dots,n\} \to \{1,\dots,m\}$ (called the **witness function**) such that for every $1 \leq i \leq n$, $b_{p(i)} = a_i$. For any $S \subseteq \mathbb{A}^*$, we let

$$\uparrow S = \{v \in \mathbb{A}^* \colon \text{ there exists } u \in S \text{ such that } u \sqsubseteq v\}$$

It is well known that a regular language L is $\mathbf{\Sigma}_1(<)$-definable if and only if $L = \uparrow L$ (see [97]). Higman's lemma, a corollary of Konig's lemma on well-quasi orders, establishes that $\sqsubseteq$ is a well-quasi order [66]. Therefore, for every L, there exists a finite set S such that $\uparrow S = \uparrow L$. We obtain as a corollary that a regular language is $\mathbf{\Sigma}_1(<)$-definable if and only if it is equal to $\uparrow S$ for some finite set S. Now, we define synchronized subwords and state an analogous result.

Definition 36. Let $\bar{u} = u_1 \otimes \dots \otimes u_k, \bar{v} = v_1 \otimes \dots \otimes v_k \in \mathrm{SW}_k$ for some $k \in \mathbb{N}$. We say $\bar{u}$ is a **synchronized subword** of $\bar{v}$, denoted by $\bar{u} \sqsubseteq_s \bar{v}$, if and only if $\bar{u} \sqsubseteq \bar{v}$ via a witness function p which is:

- type preserving: $\mathsf{type}(\bar{u}[1 \dots i]) = \mathsf{type}(\bar{v}[1 \dots p(i)])$, for every $1 \leq i \leq |\bar{u}|$
- end preserving: $p(|u_j|) = |v_j|$ for every $1 \leq j \leq k$.

For any set $S \subseteq \mathrm{SW}_k$ we let

$$\uparrow_s S = \{\bar{v} \in \mathbb{A}^* \colon \text{ there exists } \bar{u} \in S \text{ such that } \bar{u} \sqsubseteq_s \bar{v}\}$$

6.3.1 $\mathbf{\Sigma}_1[\sigma]$-definable relations

Theorem 14 ($\mathbf{\Sigma}_1[\sigma]$-characterization)**.** *For any relation R, $R \in \|\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R = \uparrow_s L_R$.*

The proof of this theorem relies on three technical statements:

- If φ is a $\mathbf{\Sigma}_1[\sigma]$ formula, then L_φ is $\sqsubseteq_s$-upward closed, i.e. $L_\varphi = \uparrow_s L_\varphi$ (see Lemma 12).
- For any $\bar{w} \in \mathrm{SW}_k$, the relation given by $\uparrow_s \bar{w}$ is $\mathbf{\Sigma}_1[\sigma]$-definable (see Lemma 13).
- On SW_k, $\sqsubseteq_s$ is a well-quasi order (see Proposition 7).

These statements are proven below. Now we proceed with the proof of Theorem 14:

Proof of Theorem 14. Let $R \in \|\mathbf{\Sigma}_1[\sigma]\|$. Then $L_R = \uparrow_s L_R$ by Lemma 12. Conversely, suppose $L_R = \uparrow_s L_R$. Let S be a set of $\sqsubseteq_s$-minimal elements of

L_R. Then S is finite because $\sqsubseteq_s$ is a well-quasi order (by Proposition 7). Say $S = \{\bar{w}_1, \ldots, \bar{w}_m\}$. By Lemma 13 there exist $\mathbf{\Sigma}_1[\sigma]$ formulæ $\varphi_1, \ldots, \varphi_m$ such that for every $1 \leq i \leq m$, $L_{\varphi_i} = \uparrow_s \bar{w}_i$. Let $\varphi = \bigvee_{i=1}^{i=m} \varphi_i$, then

$$L_\varphi = \bigcup_{i=1}^{i=n} L_{\varphi_i} = \bigcup_{i=1}^{i=n} \uparrow_s \bar{w}_i$$

Further, $\varphi \in \mathbf{\Sigma}_1[\sigma]$ and we have

$$L_\varphi = \bigcup_{i=1}^{i=n} \uparrow_s \bar{w}_i = \uparrow_s S = \uparrow L_R = L_R$$

□

Next, we move towards the proofs of statements used in the proof of Theorem 14, namely Proposition 7 and Lemmas 12,13.

Lemma 10. *Let $\bar{u}, \bar{u}' \in \mathrm{SW}_k$ with type sequences are $\bar{T} = (T_1, \ldots, T_n), \bar{T}' = (T'_1, \ldots, T'_m)$, and type factors $(\bar{u}_1, \ldots, \bar{u}_n), (\bar{u}'_1, \ldots, \bar{u}'_m)$, respectively. Then $\bar{u} \sqsubseteq_s \bar{u}'$ if and only if $\bar{T}$ is a subsequence of $\bar{T}'$ whose witness*

$$t\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$$

satisfies for every $1 \leq i \leq n$,

(i) $\bar{u}_i \sqsubseteq \bar{u}'_{t(i)}$, and

(ii) if T_i is an end type of $\bar{T}$, then $\bar{u}_i$ and $\bar{u}'_{t(i)}$ have the same last letter.

Proof. Assume first that $\bar{u} \sqsubseteq_s \bar{u}'$ via a witness function $p\colon \{1, \ldots, |u|\} \to \{1, \ldots, |u'|\}$. For every $1 \leq i \leq n$, let $\bar{v}_i = \bar{u}_1 \ldots \bar{u}_i$. Then $\mathsf{type}(\bar{v}_i) = T_i = \mathsf{type}(\bar{u}'[1 \ldots p(|\bar{v}_i|)]$, which follows from $\bar{u} \sqsubseteq_s \bar{u}'$. This shows that T_i is in some position in $\bar{T}'$, say $T_i = T'_{t(i)}$. Consequently, $\bar{T}$ is a subsequence of $\bar{T}'$, via the witness $t\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$. Now, for every $i \leq n$: since p is type preserving, $\bar{u}_i \sqsubseteq \bar{u}'_{t(i)}$; since p is end preserving, they have the same last letter if T_i is an end type of $\bar{T}$.

To prove the converse, let $\bar{T}$ be a subsequence of $\bar{T}'$ via a witness function t and for every $i \leq n$, suppose that the witness function for $\bar{u}_i \sqsubseteq \bar{u}'_{t(i)}$ is $p_i\colon \{1, \ldots, |\bar{u}_i|\} \to \{1, \ldots, |\bar{u}'_{t(i)}|\}$. In order to build a witness p for $\bar{u} \sqsubseteq_s \bar{u}'$, we let

$$n_i = |\bar{u}_1| + \cdots + |\bar{u}_i|$$
$$m_i = |\bar{u}'_{t(1)}| + \cdots + |\bar{u}'_{t(i)}|$$

for every $1 \leq i \leq n$. Then p is a 'concatenation' of the p_i's which sends every $j \in \{1, \ldots, |\bar{u}|\}$ to $m_i + p_i(j')$, where

$$(i, j') \in \{1, \ldots, n\} \times \{1, \ldots, n_{i+1} - n_i\}$$

is the unique pair for which $j = n_i + j'$. Now, we verify that p witnesses $\bar{u} \sqsubseteq_s \bar{u}'$.

- To prove that p is type preserving, let $T = \mathsf{type}(\bar{u}[1 \dots j])$ for some arbitrary $j \leq |\bar{u}|$. Since $\bar{T}$ is a subsequence of $\bar{T}'$, $T = T'_{t(j)}$ is the type of some prefix of $\bar{u}'$. Let (i, j') be the unique pair for which $j = n_i + j'$. Then $p(j) = m_i + j'$ and by definition of p_i's, we have

$$\mathsf{type}(\bar{u}'[1 \dots p(j)]) = \mathsf{type}(\bar{u}'[1 \dots (m_i + j')]) = T'_{t(j)} = T = \mathsf{type}(\bar{u}[1 \dots j])$$

- For every $i \in \{1, \dots, n\}$ such that T_i is an end type in $\bar{T}$, $p(|n_i|) = |m_i|$ (by definition of p_i). Therefore, p is end preserving.

□

Proposition 7. *For all $k \in \mathbb{N}$, $(\mathrm{SW}_k, \sqsubseteq_s)$ is a well-quasi-order.*

Proof. Let $W = \{\bar{w}_n\}_{n \in \mathbb{N}}$ be an infinite sequence of synchronized words in SW_k. Since the number of type sequences are finite, there exists an infinite subsequence of W, say W_1, in which all the synchronized words have the same type sequence. Let $\bar{T} = (T_1, \dots, T_m)$ be the type sequence of the elements of W_1. For every $\bar{w}_i \in W$, let us denote its type factors as $\bar{w}_i^1, \dots, \bar{w}_i^m$. Moreover, we say that a pair of words $\bar{w}_i, \bar{w}_j \in W_1$ are *end-similar* if and only if for every end type T_r in $\bar{T}$, $\bar{w}_i^r$ and $\bar{w}_j^r$ have the same last letter. In other words, if we write $\bar{w}_i, \bar{w}_j$ as

$$\begin{aligned} \bar{w}_i &= w_{1,i} \otimes \cdots \otimes w_{k,i} \\ \bar{w}_j &= w_{1,j} \otimes \cdots \otimes w_{k,j} \end{aligned}$$

then they are end-similar if and only if for every $p \in \{1, \dots, k\}$, $w_{p,i}$ and $w_{p,j}$ have the same last letter. Note that end-similarity is an equivalence relation with a bounded number of classes (at most $\mathbb{A}^m$). Therefore, there exists an infinite subsequence of W_1, say W_2, whose elements are all end-similar (in addition to having the same type, since they come from W_1).

Next we consider the set of $\{\bar{w}_i^1 \colon \bar{w}_i \in W_2\}$. Since $\sqsubseteq$ is a well-quasi order, there exists an infinite subsequence W_3 of W_2 which is $\sqsubseteq$-increasing in its first type factor. To be precise, for each $\bar{w}_i, \bar{w}_j \in W_3$, if $i < j$ then $\bar{w}_i^1 \sqsubseteq \bar{w}_i^1$. Similarly, there exists an infinite subsequence of W_3, call it W_4, which is $\sqsubseteq$-increasing in the second type factor. Iterating this reasoning, we find a chain of infinite subsequences

$$W_{m+2} \subseteq W_{m+1} \cdots \subseteq W_1 \subseteq W$$

such that for all pairs of elements $\bar{w}_i, \bar{w}_j \in W_{m+2}$ (with $i < j$), we have, (i) $\bar{w}_i^p \sqsubseteq \bar{w}_j^p$ for all $p \in \{1, \dots, m\}$, and (ii) $\bar{w}_i, \bar{w}_j$ are end-similar. By Lemma 10, we have $\bar{w}_i \sqsubseteq_s \bar{w}_j$ whenever $\bar{w}_i, \bar{w}_j \in W_{m+2}$ and $i < j$. In particular, there exists a pair $(\bar{u}, \bar{v})$ of synchronized words in W such that $\bar{u} \sqsubseteq_s \bar{v}$. This completes the proof. □

Lemma 11. *Let $\bar{w} = w_1 \otimes \cdots \otimes w_k$ and $\bar{w}' = w'_1 \otimes \cdots \otimes w'_k$ such that $\bar{w} \sqsubseteq_s \bar{w}'$. For all $u \in \mathbb{A}^*$, there exists $u' \in \mathbb{A}^*$ such that $\bar{w} \otimes u \sqsubseteq_s \bar{w}' \otimes u'$.*

Proof. Let $p\colon \{1,\dots,|\bar{w}|\} \to \{1,\dots,|\bar{w}'|\}$ be the function witnessing $\bar{w} \sqsubseteq_s \bar{w}'$. Let $u \in \mathbb{A}^*$. To produce $u' \in \mathbb{A}^*$ such that $\bar{w}\otimes u \sqsubseteq_s \bar{w}'\otimes u'$, we do the following:

Fix $\ell \in \{1,\dots,k\}$ such that $|u \sqcap w_\ell| = \max_{i\le k} |u \sqcap w_i|$, and let $s_1 = |u \sqcap w_\ell|$. Let $s_2 = \min(|\bar{w}, |u|)$, and let us denote by $a_1 \dots a_m$ the word $u[s_1+1 \dots s_2]$. Then

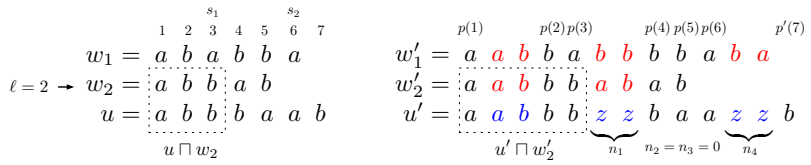
$$u = (u \sqcap w_\ell)(a_1 \dots a_m)(u[s_2+1 \dots |u|])$$

where $u[s_2+1\dots|u|] = \varepsilon$ if $|u| \le |\bar{w}|$.

For every $1 \le i \le m$, let $n_i = p(s_1+i) - p(s_1) - 1$, and let $n_{m+1} = |\bar{w}'| - p(s_2)$. Finally, let z be an arbitrary letter in $\mathbb{A}$. We let

$$u' = w'_\ell[1\dots p(s_1)]z^{n_1}a_1 \cdots z^{n_m}a_m z^{n_{m+1}} u[s_2+1\dots|u|]$$

We provide an example of the construction of u' below:



We claim that $\bar{w}\otimes u \sqsubseteq_s \bar{w}'\otimes u'$ via the witness function

$$p'\colon \{1,\dots,\max(|\bar{w}|,|u|)\} \to \{1,\dots,\max(|\bar{w}'|,|u'|)\}$$

defined as

$$p'(j) = \begin{cases} p(j) \text{ if } j \le |\bar{w}| \\ |\bar{w}'| + j \text{ otherwise} \end{cases}$$

By construction, p' is increasing. Further, for every $1 \le i \le \max(|\bar{w}|,|u|)$, $(\bar{w}\otimes u)[i] = (\bar{w}'\otimes u')[p'(i)]$. Therefore p' is a witness for $\bar{w}\otimes u \sqsubseteq \bar{w}'\otimes u'$. Since $p'(|w_j|) = p(|w_j|) = |w'_j|$ for every $1 \le j \le k$, to establish that p' is end preserving we only need to show that $p'(|u|) = |u'|$. Further, we need to show that p' is type preserving, i.e. for every i, $\mathsf{type}(\bar{w}\otimes u[1\dots i]) = \mathsf{type}(\bar{w}'\otimes u'[1\dots p'(i)])$. We denote $\bar{w}\otimes u$ as $\bar{w}_u$ and $\bar{w}'\otimes u'$ as $\bar{w}'_{u'}$ for convenience. Let $i \in \{1,\dots,|\bar{w}_u|\}$. The following cases arise:

- If $1 \le i \le s_1$, then $p'(i) = p(i)$, and the u-component and w_ℓ-component of each letter of $\bar{w}_u[1\dots i]$ coincide, as do the u'-component and w'_ℓ-component of each letter of $\bar{w}'_{u'}[1\dots p(i)]$. Therefore $\mathsf{type}(\bar{w}'_u)[1\dots i]$ is the symmetric transitive closure of $\mathsf{type}(\bar{w}[1\dots i])\cup\{(\ell,k+1)\}$, and $\mathsf{type}(\bar{w}'_{u'})[1\dots p(i)]$ is the symmetric transitive closure of $\mathsf{type}(\bar{w}'[1\dots p(i)])\cup\{(\ell,k+1)\}$. We know that $\mathsf{type}(\bar{w}[1\dots i]) = \mathsf{type}(\bar{w}'[1\dots p(i)])$, therefore

  $$\mathsf{type}(\bar{w}_u[1\dots i]) = \mathsf{type}(\bar{w}'_{u'}[1\dots p'(i)])$$

  Further,

  $$u[i] = w_\ell[i] = w'_\ell[p'(i)] = u'[p'(i)]$$

  If $i = |u|$ then $s_1 = i$ and by definition, $u' = w'_\ell[1\dots p(i)]$. It follows that $|u'| = p(i) = p'(i)$.

- If $s_1 < i \leq s_2$, then too we have $p'(i) = p(i)$. Note that in this case,

$$\mathsf{type}(\bar{w}_u[1 \dots i]) = \mathsf{type}(\bar{w}[1 \dots i]) \cup \{(k, k+1)\}$$

because there exists at least one position not greater than i where the u-component of $\bar{w}_u$ differs from some other component. This also implies that

$$\mathsf{type}(\bar{w}'_{u'}[1 \dots p'(i)]) = \mathsf{type}(\bar{w}'[1 \dots p(i)]) \cup \{(k, k+1)\} = \mathsf{type}(\bar{w}_u[1 \dots i])$$

By definition of the n_j's, we have $u[i] = u'[p(i)]$, in particular when $i = |u|$ then $p(|u|) = p(i) = p(|u'|)$.

- If $s_2 < i \leq |u|$, then $p'(i) - |\bar{w}'| = i - |\bar{w}'| = |u[s_2 + 1 \dots i]|$, in particular when $i = |u|$, $p'(i) = p'(|u|) = u[s_2 + 1 \dots |u|] = |u'|$. Further, $\mathsf{type}(\bar{w}_u[1 \dots i]) = \{(k+1, k+1)\} = \mathsf{type}(\bar{w}'_{u'}[1 \dots p'(i)])$.

□

Observation 1. Let $\bar{T} = (T_1, \dots, T_n)$ be the type sequence of some synchronized word in SW_k. Then $K(\bar{T}) \subseteq \mathrm{SW}_k$, the set of all synchronized words in SW_k whose type-sequence is $\bar{T}$, can be written as

$$K(\bar{T}) = \mathbb{A}_{-,T_1} \mathbb{A}^*_{T_1} \mathbb{A}_{T_1,T_2} \mathbb{A}^*_{T_2} \dots \mathbb{A}_{T_{n-1},T_n} \mathbb{A}^*_{T_n}$$

Lemma 12. *Let φ be a formula in $\mathbf{\Sigma}_1[\sigma]$. Then L_φ is $\sqsubseteq_s$-upward closed.*

Proof. First we observe that for any pair of synchronized words $\bar{w} = w'_1 \dots w'_k, \bar{w}' = w'_1 \otimes \dots \otimes w'_k$, such that $\bar{w} \sqsubseteq_s \bar{w}'$ via a witness function p, we must have, for every $i, j \in \{1, \dots, k\}$:

(i) $w_i \preceq w_j$ if and only if $w'_i \preceq w'_j$: Let $T = \mathsf{type}(\bar{w}[1 \dots |w_i|])$. By definition of p, $T = \mathsf{type}(\bar{w}[1 \dots |w'_i|])$. Therefore, $(i, j) \in \mathsf{type}(\bar{w}[1 \dots |w'_i|])$ and therefore $w'_i \preceq w'_j$. The converse holds by the same reasoning.

(ii) $|w_i| = |w_j|$ if and only if $|w'_i| = |w'_j|$: this follows from the fact that p is end preserving.

(iii) If $|w_i| > 0$ then $|w'_i| > 0$ and w_i, w'_i have the same last letter: this follows from the fact that p is type preserving and end preserving.

Now we prove the lemma by induction on the number of quantified variables of φ. In the case where φ is quantifier-free, the statements (i),(ii),(iii) given above show that L_φ is $\sqsubseteq_s$-upward closed.

If φ is not quantifier-free then it is a positive Boolean combination (using only $\vee$ or $\wedge$) of formulæ of the form

$$\exists x \, \psi(y_1, \dots, y_k, x)$$

for some $\psi \in \mathbf{\Sigma}_1[\sigma]$ such that L_ψ is $\sqsubseteq_s$-upward closed (by inductive hypothesis). Let $\bar{w}, \bar{w}' \in \mathrm{SW}_k$ such that $\bar{w} \sqsubseteq_s \bar{w}'$ and $\bar{w} \vDash \varphi$. Then there exists $u \in \mathbb{A}^*$ such that $\bar{w} \otimes u \vDash \psi$. By Lemma 11 there also exists $u' \in \mathbb{A}^*$ such that $\bar{w} \otimes u \sqsubseteq_s \bar{w}' \otimes u'$. Since L_ψ is $\sqsubseteq_s$-upward closed, $\bar{w}' \otimes u' \vDash \psi$. Therefore, $\bar{w}' \vDash \exists x \, \psi$. Now, if L_{φ_1} and L_{φ_2} are both $\sqsubseteq_s$-upward closed, then $L_{\varphi_1 \vee \varphi_2}$ and $L_{\varphi_1 \wedge \varphi_2}$ are $\sqsubseteq_s$-upward closed. Therefore, L_φ is$\sqsubseteq_s$-upward closed. This completes the proof. □

Corollary 3. *Let $\varphi \in \mathbf{\Sigma}_1[\sigma]$ be a formula with k free variables. Then there exists a finite set $\mathcal{S} \subset \mathrm{SW}_k$ such that $L_\varphi = \uparrow_s \mathcal{S}$.*

Proof. Let $\mathcal{S}$ be the set of all $\sqsubseteq_s$-minimal elements of L_φ. By Proposition 7, $\sqsubseteq_s$ is a well-quasi order. Therefore, $\mathcal{S}$ must be finite. Next we show that $L_\varphi = \uparrow_s \mathcal{S}$.

By the minimality of $\mathcal{S}$, every element $\bar{w} \in L_\varphi$ has a synchronized subword in $\mathcal{S}$. Therefore, $L_\varphi \subseteq \uparrow_s \mathcal{S}$. Finally, $\mathcal{S} \subseteq L_\varphi$ and $\mathbf{\Sigma}_1[\sigma]$-definable relations are $\sqsubseteq_s$-upward closed, by Lemma 12. Therefore, $\uparrow_s \mathcal{S} \subseteq L_\varphi$. This completes the proof. □

Lemma 13. *Let $\bar{w} \in \mathrm{SW}_k$. Then the relation given by $\uparrow_s \bar{w}$ is $\mathbf{\Sigma}_1[\sigma]$-definable.*

Proof. Let $\bar{w} = w_1 \otimes \cdots \otimes w_k$. We define a formula φ over free variables $z_1, \ldots, z_k$ whose synchronized language is $\uparrow_s \bar{w}$. In order to do this, let us introduce a variable $x_{i,j}$ for each $i \in \{1, \ldots, k\}$ and $j \in \{1, \ldots, |w_i|\}$. Then $X = \{x_{i,j}\}$ is the set of bound (in this case existentially quantified) variables of φ. Formally we define $\varphi(z_1, \ldots, z_k)$ as $\exists X.\psi(z_1, \ldots, z_k, X)$, where ψ is the conjunction of the following formulæ for every $1 \leq i \leq k$:

(i) $z_i = x_{i,|w_i|}$;

(ii) for every $1 \leq j < |w_i|$: $x_{i,j} \preceq x_{i,j+1}$;

(iii) for every $1 \leq j \leq |w_i|$: $\ell_{w_i[j]}(x_{i,j})$;

(iv) for every $1 \leq i' \leq k$ and $j \leq \min\{|w_i|, |w_{i'}|\}$: $\mathsf{eq}(x_{i,j}, x_{i',j})$;

(v) for every $1 \leq i' \leq k$ and $j \leq \min\{|w_i|, |w_{i'}|\}$ such that $w_i[1 \ldots j] = w_{i'}[1 \ldots j]$: $x_{i,j} = x_{i',j}$.

Let $\bar{w}' = w'_1 \otimes \cdots \otimes w'_k$. Assume $\bar{w} \sqsubseteq_s \bar{w}'$ via a witness function p, we show that $\bar{w}' \in L_\varphi$. For each $1 \leq i \leq k$ and $1 \leq j \leq |w_i|$, let $x_{i,j}$ be assigned the word $v_{i,j} = w'_i[1 \ldots p(j)]$. We know that p witnesses $\bar{w} \sqsubseteq_s \bar{w}'$. We see that $v_{i,j}$'s satisfy (ii)-(iv) because p is type-preserving, and (v) because p is end-preserving. Therefore, $(w'_1, \ldots, w'_k)$ satisfies $\varphi(z_1, \ldots, z_k)$, and we have $\bar{w}' \in L_\varphi$.

To show the converse, let $\bar{w}' \in L_\varphi$. Then $(w'_1, \ldots, w'_k)$ satisfies φ via a witnessing assignment α. We define a function $p\colon \{1, \ldots, |w|\} \to \{1, \ldots, |w'|\}$ as follows: for every $1 \leq j \leq |\bar{w}|$, there exists some $1 \leq i \leq k$ such that $j \leq |w_i|$; define $p(j)$ to be $|\alpha(x_{i,j})|$. We observe that p is well-defined due to the subformulæ defined in (iv) above. Further, the subformulæ in (ii) show that p is increasing, and the subformulæ in (iii) show that for every j, $\bar{w}[j] = \bar{w}'[p(j)]$, so p witnesses $\bar{w} \sqsubseteq \bar{w}'$. Finally, the subformulæ defined in (i) and (v) show that p is end preserving and type preserving respectively. Thus p witnesses $\bar{w} \sqsubseteq_s \bar{w}'$. □

6.3.2 $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable relations

Recall that for any $h \in \mathbb{N}$, the equivalence relations $\sim_h$ and $\equiv_h$ are defined as follows (on words in $\mathbb{A}^*$ and synchronized words respectively):

- for all $u, v \in \mathbb{A}^*$, $u \sim_h v$ if and only if u and v have the same set of subwords of length at most h. Let $[u]_{\sim_h}$ denote the equivalence class of u

generated by $\sim_h$. Furthermore, let $\mathscr{B}_{\sim_h}$ be the set of unions of $\sim_h$-classes. Finally we let

$$\mathscr{B}_{\sim} = \bigcup_{h\in\mathbb{N}} \mathscr{B}_{\sim_h}$$

- for all k-synchronized words $\bar{u}, \bar{v}$, $\bar{u} \equiv_h \bar{v}$ if and only if $\bar{u}$ and $\bar{v}$ have the same set of synchronized subwords of length at most h. Let $[\bar{w}]_{\equiv_h}$ denote the equivalence class of $\bar{w}$ generated by $\equiv_h$, let $\mathscr{B}_{\equiv_h}$ denote the set of finite unions of $[\bar{w}_i]_{\equiv_h}$, and let

$$\mathscr{B}_{\equiv} = \bigcup_{h\in\mathbb{N}} \mathscr{B}_{\equiv_h}$$

Theorem 15. *For any relation R, $R \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if $L_R \in \mathscr{B}_{\equiv}$.*

Proof. Let $\varphi \in \mathcal{B}\mathbf{\Sigma}_1[\sigma]$ be a formula defining R, and let $L = L_R$. We use the standard disjunctive normal form for $\mathcal{B}\Sigma_1$ formulæ and assume φ is logically equivalent to $\bigvee_{i=1}^{i=n} \psi_i \wedge \psi_i'$ where the ψ_i's are in $\mathbf{\Sigma}_1[\sigma]$ and the ψ_i''s are in $\mathbf{\Pi}_1[\sigma]$. By Corollary 3, then, for every $1 \leq i \leq n$ there exist finite sets $\mathcal{S}_i, \mathcal{S}_i' \subset \mathrm{SW}_k$ such that $L_{\psi_i} = \uparrow_s \mathcal{S}_i$ and $L_{\psi_i'} = \mathrm{SW}_k \setminus \uparrow_s \mathcal{S}_i'$. We define

$$\mathcal{S} = \bigcup_{i=1}^{i=n} \mathcal{S}_i \cup \mathcal{S}_i'$$
$$h = \max_{w\in\mathcal{S}} |w| \text{ (exists because } \mathcal{S} \text{ is finite)}$$

Now consider synchronized words $\bar{w}, \bar{w}' \in \mathrm{SW}_k$ such that $\bar{w} \equiv_h \bar{w}'$. Then, for every $1 \leq i \leq n$ and $\bar{u} \in \mathcal{S}_i$, $\bar{u} \sqsubseteq_s \bar{w}$ if and only if $\bar{u} \sqsubseteq_s \bar{w}'$ because by definition $|\bar{u}| \leq h$ and $\bar{w} \equiv_h \bar{w}'$. Similarly, $\bar{w}$ and $\bar{w}'$ have the same set of synchronized subwords in $\mathcal{S}_i'$ as well, for every $1 \leq i \leq n$. Therefore, $\bar{w} \in L_\varphi$ if and only if $\bar{w}' \in L_\varphi$. Thus φ is a union of $\equiv_h$-classes, and hence $L_\varphi \in \mathscr{B}_{\equiv_h}$.

Conversely, let $h \in \mathbb{N}$ such that $L = L_R \in \mathscr{B}_{\equiv_h}$. Then L is the finite union $[\bar{w}_1]_h, \ldots, [\bar{w}_n]_h$ for some $\bar{w}_1, \ldots, \bar{w}_n \in \mathrm{SW}_k$. To produce a $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ formula φ which characterizes L, we let $W_h \subset \mathrm{SW}_k$ be the (finite) set of all synchronized words of length at most h. Further, for every $1 \leq i \leq n$, let $\mathcal{S}_i \subseteq W_h$ be the set of all subwords of $\bar{w}_i$ of length at most h, and let $\mathcal{S}_i' = W_h \setminus \mathcal{S}_i$; note that both of these sets are finite. By Theorem 14, there exist $\mathbf{\Sigma}_1[\sigma]$ formulæ $\psi_1, \ldots, \psi_n, \psi_1', \ldots, \psi_n'$ such that for every $1 \leq i \leq n$

$$\begin{aligned} L_{\psi_i} &= \uparrow_s \mathcal{S}_i \\ L_{\psi_i'} &= \uparrow_s \mathcal{S}_i' \\ [\bar{w}_i]_h &= \uparrow_s \mathcal{S}_i \setminus \uparrow_s \mathcal{S}_i' = L_{\psi_i} \setminus L_{\psi_i'} \end{aligned}$$

Therefore,

$$L = \bigcup_{i=1}^{i=n} [\bar{w}_i]_h = L_{\psi_i} \setminus L_{\psi_i'}$$

and the $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ formula

$$\bigvee_{i=1}^{i=n} \psi_i \wedge \neg\psi_i'$$

characterizes L. □

6.3.3 Deciding membership in $\|\mathbf{\Sigma}_1[\sigma]\|$ and $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$

This section deals with the $\mathbf{\Sigma}_1[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ membership problems.

Theorem 16. MEMBERSHIP-$\mathbf{\Sigma}_1[\sigma]$ *is decidable.*

Proof. By Theorem 14, a synchronous relation R is $\mathbf{\Sigma}_1[\sigma]$-definable if and only if $L_R = \uparrow_s L_R$. To check whether a given relation R is $\mathbf{\Sigma}_1[\sigma]$-definable, we simply compute $\uparrow_s L_R$ and check whether $L_R = \uparrow_s L_R$. In Proposition 8 below, we show that for any synchronized regular language L, the language $\uparrow_s L$ is regular and computable. Assuming this proposition holds, we obtain the decidability of MEMBERSHIP-$\mathbf{\Sigma}_1[\sigma]$. □

Now we state and prove Proposition 8. We begin with some definitions.

Definition 37. For $\mathcal{S} \subseteq \mathrm{SW}_k$, we denote by $\uparrow^\ell \mathcal{S}$ the set

$\{\bar{w} \in \mathrm{SW}_k \colon \exists \bar{u} \in \mathcal{S}$ such that $\bar{u} \sqsubseteq \bar{w}$ and $\bar{u}, \bar{w}$ have the same last letter$\}$

Definition 38. Let $\mathcal{A} = (Q, \delta, q_0, F)$ be a DFA accepting a regular language L. For all $p, r \in Q$, let $\mathcal{A}(p, r)$ denote the DFA $(Q, \delta, p, \{r\})$, and let $\mathsf{Lang}(\mathcal{A}(p, r))$ be the language accepted by $\mathcal{A}(p, r)$.

Moreover, let $\bar{T} = (T_1, \ldots, T_n)$ be some type. A sequence $\bar{q} = (q_1, \ldots, q_n) \in Q^n$ is called $\bar{T}$-**compatible** in $\mathcal{A}$ if and only if $q_n \in F$ and q_1 is reachable from q_0 by reading a word in $\mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^*$, q_2 is reachable from q_1 by reading a word in $\mathbb{A}_{T_1,T_2}\mathbb{A}_{T_2}^*$, and so on. For any $\bar{T}$-compatible sequence $\bar{q}$, let

$$L(\bar{T}, \bar{q}, 1) = \mathbb{A}_{-,T_1}\mathbb{A}_{T_1}^* \cap \mathsf{Lang}(\mathcal{A}(q_0, q_1))$$

and for every $1 < i \leq n$ let

$$L(\bar{T}, \bar{q}, i) = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^* \cap \mathsf{Lang}(\mathcal{A}(q_{i-1}, q_i))$$

Observation 2. Let $\mathcal{A}$ be a DFA accepting $L \subseteq \mathrm{SW}_k$, and let $\bar{w} \in \mathrm{SW}_k$. Then $\bar{w} \in L$ if and only if there exists a $\bar{T}$-compatible sequence $\bar{q} \in Q^n$ such that

$$\bar{w} \in L(\bar{T}, \bar{q}, 1) \cdots L(\bar{T}, \bar{q}, n)$$

where $\bar{T} = \mathsf{type}(\bar{w})$. Further, this factorization of $\bar{w}$ produces its type factors, i.e. for every $1 \leq i \leq n$, the i-th type factor of $\bar{w}$ is in $L(\bar{T}, \bar{q}, i)$.

Observation 3 (Derived from Observation 2)**.** Let $\mathcal{A}$ be a DFA accepting $L \subseteq \mathrm{SW}_k$. Then

$$L = \bigcup L(\bar{T}, \bar{q}, 1) \cdots L(\bar{T}, \bar{q}, n)$$

where the union runs over all type sequences $\bar{T}$ and all $\bar{T}$-compatible state sequences $\bar{q}$. Further, this union is finite and computable.

Proposition 8. *Let $L \subseteq \mathrm{SW}_k$ be a regular language. Then its $\sqsubseteq_s$-upward closure $\uparrow_s L$ is regular and computable.*

Proof. Let $\mathcal{A} = (Q, \delta, q_0, F)$ be some DFA recognizing L. For any type $\bar{T}$ and $\bar{T}$-compatible state sequence $\bar{q} \in Q^n$, we define the language

$$\hat{L}(\bar{T}, \bar{q}, i) = \begin{cases} \uparrow^{\ell} L(\bar{T}, \bar{q}, i) \cap \mathbb{A}^*_{T_i} \text{ if } T_i \text{ is an end type} \\ \uparrow L(\bar{T}, \bar{q}, i) \cap \mathbb{A}^*_{T_i} \text{ otherwise} \end{cases}$$

Now we show that

$$\uparrow_s L = \bigcup \hat{L}(\bar{T}, \bar{q}, 1) \cdots \hat{L}(\bar{T}, \bar{q}, n) \tag{6.1}$$

where the union ranges over all types $\bar{T}$ and all $\bar{T}$-compatible state sequences $\bar{q}$. Showing Equation 6.1 holds is sufficient to prove the proposition, because (i) $\uparrow^{\ell} K$ and $\uparrow K$ are computable for any regular language K, which makes the operation $L(\bar{T}, \bar{q}, i) \mapsto \hat{L}(\bar{T}, \bar{q}, i)$ computable, and (ii) there are only finitely many types and type-compatible state sequences, so computing all of the $\hat{L}(\bar{T}, \bar{q}, i)$ would yield a DFA for $\uparrow_s L$.

The arguments made to show Equation 6.1 closely follow those in the proof of Lemma 10. Let $\bar{w} \in \uparrow_s L$, that is, there exists $\bar{u} \in L$ such that $\bar{u} \sqsubseteq_s \bar{w}$. Let $\bar{T} = \mathsf{type\text{-}seq}(\bar{u}) = (T_1, \ldots, T_n)$ and let $\bar{q}$ be the $\bar{T}$-compatible state sequence obtained upon $\mathcal{A}$ reading $\bar{u}$. By Lemma 11, $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{w})$ with a witness function $t\colon \{1, \ldots, n\} \to \{1, \ldots, |\mathsf{type\text{-}seq}(\bar{w})|\}$ such that for each $1 \leq i \leq n$, we have: $\bar{u}_i \sqsubseteq \bar{w}_{t(i)}$ (where $\bar{u}_i$ is the i-th type factor of $\bar{u}$ and $\bar{w}_{t(i)}$ is the $t(i)$-th type factor of $\bar{w}$), and they have the same last letter if T_i is an end type. Therefore, $\bar{u}_i \sqsubseteq \bar{w}_{t(i-1)+1} \ldots \bar{w}_{t(i)}$ (where again, the last letter condition holds if T_i is an end type). Now, $\bar{u}_i \in L(\bar{T}, \bar{q}, i)$, and therefore $\bar{w}_{t(i-1)+1} \ldots \bar{w}_{t(i)} \in \hat{L}(\bar{T}, \bar{q}, i)$. This implies $\bar{w} \in \hat{L}(\bar{T}, \bar{q}, 1) \cdots \hat{L}(\bar{T}, \bar{q}, n)$.

Conversely, let $\bar{w} \in \hat{L}(\bar{T}, \bar{q}, 1) \cdots \hat{L}(\bar{T}, \bar{q}, n)$, where $\bar{T}$ is a type sequence and $\bar{q}$ is a $\bar{T}$-compatible state sequence. In other words, there exists a factorization of $\bar{w}$ as $\bar{w} = \bar{w}_1 \cdots \bar{w}_n$ where for each $1 \leq i \leq n$, $\bar{w}_i \in \hat{L}(\bar{T}, \bar{q}, i)$. So for every $1 \leq i \leq n$, there exists $\bar{u}_i \in L(\bar{T}, \bar{q}, i)$ such that $\bar{u}_i \sqsubseteq \bar{w}_i$ with witness function p_i. Further, for all end types T_i, we also have $p_i(|\bar{u}_i|) = |\bar{w}_i|$. Then the word $\bar{u} = \bar{u}_1 \cdots \bar{u}_n$ is in L by construction of the $L(\bar{T}, \bar{q}, i)$'s. To produce a witness function p for $\bar{u} \sqsubseteq_s \bar{w}$, we 'concatenate' the p_i's as we did in the proof of Lemma 10; for every $1 \leq i \leq n$:

$$s_i = |\bar{u}_1| + \cdots + |\bar{u}_i|$$
$$r_i = |\bar{w}_{t(1)}| + \cdots + |\bar{w}_{t(i)}|$$

Now, we define the witness function $p\colon j \mapsto r_i + p_i(j')$, where $(i, j') \in \{1, \ldots, n\} \times \{1, \ldots, s_{i+1} - s_i\}$ is the unique pair for which $j = s_i + j'$. The argument for p witnessing $\bar{u} \sqsubseteq_s \bar{w}$ is identical to that given in the proof of Lemma 10, where we simply replace $\bar{u}'$ by $\bar{w}$. $\square$

Theorem 17. MEMBERSHIP-$\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ *is decidable.*

We first give an overview of the proof of Theorem 17. We then state and prove the necessary technical lemmas and return to a formal proof of Theorem 17 at the end of this section.

Proof sketch. Let R be a synchronous relation given as a regular language $L_R \subseteq \mathrm{SW}_k$. Recall that for any type sequence $\bar{T}$, $K(\bar{T})$ denotes the set of synchronized words in SW_k whose type sequence is $\bar{T}$. Let $L = L_R$ and for every possible type sequence $\bar{T}$ on words in SW_k, let $L_{\bar{T}} = L \cap K(\bar{T})$. The proof of this theorem relies on three lemmas:

- In Lemma 14, we show that for any type sequence $\bar{T}$, the set $K(\bar{T})$ is $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable. As a corollary, we obtain that $L \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if for each type sequence $\bar{T}$, $L_{\bar{T}} \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ (see Corollary 4).
- Lemmas 15 and 16 show that it is decidable whether $L_{\bar{T}} \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$, for each type sequence $\bar{T}$.

Below we prove Lemmas 14, 15, and 16. □

Lemma 14. *Let $\bar{T}$ be a type sequence. Then $K(\bar{T})$ is $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable.*

Proof. For any type $\bar{T}$, let $S_{\bar{T}}$ be the set of $\sqsubseteq_s$-minimal elements of $K(\bar{T})$. We established in Proposition 7 that $\sqsubseteq_s$ is a well-quasi order, therefore $S_{\bar{T}}$ is finite. Now, $K(\bar{T}) \subseteq \uparrow_s S_{\bar{T}}$. Moreover, if $\bar{u} \in \uparrow_s S_{\bar{T}}$, then $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ by Lemma 11. We observe that

$$K(\bar{T}) = \uparrow_s S_{\bar{T}} \setminus \bigcup \{\uparrow_s S_{\bar{T}'} : \bar{T} \sqsubseteq \bar{T}', \bar{T} \neq \bar{T}'\}$$

for every $\bar{T}'$ such that $\bar{T} \sqsubseteq \bar{T}'$, let $\psi_{\bar{T}'}$ be the $\mathbf{\Sigma}_1[\sigma]$ formula characterizing $\uparrow_s S_{\bar{T}'}$. Then $K(\bar{T})$ is characterized by the $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ formula

$$\psi_{\bar{T}} \wedge \neg \Big(\bigvee_{\bar{T} \sqsubseteq \bar{T}', \bar{T}' \neq \bar{T}} \psi_{\bar{T}'} \Big)$$

□

Corollary 4 (of Lemma 14)**.** *For any regular $L \subseteq \mathrm{SW}_k$, $L \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if for each type sequence $\bar{T}$, $L \cap K(\bar{T}) \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$.*

We now fix a type sequence $\bar{T} = (T_1, \ldots, T_n)$ and characterize the $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ languages within $K(\bar{T})$.

Definition 39. For each $1 \leq i \leq n$, let $\mathcal{F}_i$ be $\mathcal{B}\mathbf{\Sigma}_1(<, \mathbf{max})(\mathbb{A}^*_{T_i})$ if T_i is an end type in $\bar{T}$, and $\mathcal{B}\mathbf{\Sigma}_1(<)(\mathbb{A}^*_{T_i})$ otherwise. Further, let

$$\mathcal{G}_1 = \{\mathbb{A}_{-,T_1}\mathbb{A}^*_{T_1} \cap H : H \in |\mathcal{F}_1|\}$$

and for every $2 \leq i \leq n$, let

$$\mathcal{G}_i = \{A_{T_{i-1},T_i}\mathbb{A}^*_{T_i} \cap H : H \in |\mathcal{F}_i|\}$$

Lemma 15. *Let $\mathcal{A}_{\bar{T}} = (Q, \delta, q_0, F)$ be a DFA defining a regular language $L \subseteq K(\bar{T})$. Then, $L \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if, for each $\bar{T}$-compatible state sequence $\bar{q} \in Q^n$ and $1 \leq i \leq n$, $L(\bar{T}, \bar{q}, i) \in \mathcal{G}_i$.*

Proof. Since $\bar{T}$ is fixed, we shorten $L(\bar{T}, \bar{q}, i)$ to $L(\bar{q}, i)$. First assume that for every $1 \leq i \leq n$, $L(\bar{q}, i) \in \mathcal{G}_i$. In other words, there exists a $\mathcal{B}\mathbf{\Sigma}_1(<)$-definable language $H(\bar{q}, i) \subseteq \mathbb{A}^*_{T_i}$ such that $L(\bar{q}, i) = \mathbb{A}^*_{T_{i-1},T_i} \cap H(\bar{q}, i)$ (for the case $i = 1$, $H(\bar{q}, 1)$ is $\mathcal{B}\mathbf{\Sigma}_1(<, \mathbf{max})$-definable and $L(\bar{q}, 1) = \mathbb{A}^*_{-,T_1} \cap H(\bar{q}, 1)$). So for every i, $H(\bar{q}, i)$ is a union of $n_{\bar{q},i}$ sets of the form

$$H(\bar{q}, i, j) = \uparrow S(\bar{q}, i, j) \backslash \uparrow S'(\bar{q}, i, j) \text{ (with} \uparrow^\ell \text{ substituted for} \uparrow \text{ if } T_i \text{ is an end type in } \bar{T})$$

with j ranging over $\{1, \ldots, n_{\bar{q},i}\}$, where $S(\bar{q}, i, j), S'(\bar{q}, i, j)$ are finite sets. Consequently, $L(\bar{q}, 1)$ is the union of the

$$L(\bar{q}, 1, j) = \mathbb{A}_{-,T_1}\mathbb{A}^*_{T_1} \cap H(\bar{q}, 1, j)$$

over all $1 \leq j \leq n_{\bar{q},1}$. Further, for every $2 \leq i \leq n$, $L(\bar{q}, 1)$ is a union of

$$L(\bar{q}, 1, j) = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i} \cap H(\bar{q}, i, j)$$

over all $1 \leq j \leq n_{\bar{q},i}$. For all $\bar{j} = (j_1, \ldots, j_n)$ where for every $1 \leq i \leq n$, $j_i \in \{1, \ldots, n_{\bar{q},i}\}$, we define

$$L(\bar{q}, \bar{j}) = L(\bar{q}, 1, j_1) \cdots L(\bar{q}, n, j_n)$$

Then we can write L as the finite union

$$L = \bigcup L(\bar{q}, \bar{j})$$

Define the sets

$$\mathcal{S}(\bar{q}, \bar{j}) = \{\bar{w} \in K(\bar{T}) \colon \text{ for every } i \in \{1, \ldots, n\}, \mathsf{type\text{-}factor}_i(\bar{w}) \in S(\bar{q}, i, j_i)\}$$
$$\mathcal{S}'(\bar{q}, \bar{j}) = \{\bar{w} \in K(\bar{T}) \colon \text{ for every } i \in \{1, \ldots, n\}, \mathsf{type\text{-}factor}_i(\bar{w}) \in S'(\bar{q}, i, j_i)\}$$

Now for all state sequences $\bar{q}$ and sequences $\bar{j}$, we have

$$L(\bar{q}, \bar{j}) = K(\bar{T}) \cap \left(\uparrow \mathcal{S}(\bar{q}, \bar{j}) \setminus \uparrow \mathcal{S}'(\bar{q}, \bar{j}) \right)$$

which is $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable. Thus their finite union L is also $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ definable.

Conversely, let $L(\bar{q}, i) \notin \mathcal{G}_i$ for some $\bar{T}$-compatible state sequence $\bar{q}$ and $1 \leq i \leq n$. To show that L is not $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable, we use Theorem 15 and show that L is not a union of $\equiv_h$ classes for any $h \in \mathbb{N}$. Fix h. We only need to show the existence of words $\bar{w}, \bar{w}' \in \mathrm{SW}_k$ such that $\bar{w} \in L$, $\bar{w}' \notin L$ and $\bar{w} \equiv_h \bar{w}'$. We write $\bar{u} \sim^i_h \bar{v}$ if and only if $\bar{u} \sim_h \bar{v}$ and either the first letters of $\bar{u}$ and $\bar{v}$ are both in $\mathbb{A}_{T_{i-1}}$ or neither is.

Suppose, $L(\bar{q}, i)$ is the finite union of some $\sim^i_h$ classes $[\bar{u}_1], \ldots, [\bar{u}_m]$. Then, for each $i \leq m$, $\bar{u}_i \in \mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i}$, and by definition of $\sim^i_h$, $[\bar{u}_i] = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i} \cap [\bar{u}_i]_{\sim_h}$. Therefore,

$$L(\bar{q}, i) = \mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i} \cap M$$

where $M = \bigcup_j [\bar{u}_j]_{\sim_h} \in \mathcal{B}\mathbf{\Sigma}_1(<)(\mathbb{A}_{T_i})$. This shows that $L(\bar{q}, i) \in \mathcal{G}_i$, contradicting our assumption.

This establishes that for every i, $L(\bar{q}, i)$ is not a union of $\sim_h^i$ classes. So there exist words $\bar{u}_i, \bar{u}_i' \in \mathbb{A}_{T_{i-1},T_i}\mathbb{A}_{T_i}^*$ such that $\bar{u}_i \sim_h \bar{u}_i'$ (and if T_i is an end type they have the same last letter), such that exactly one of them, say $\bar{u}_i$, is in $L(\bar{q}, i)$. Then there exist distinct states $q_i, q_i' \in Q$ such that $\bar{u}_i \in \mathsf{Lang}(\mathcal{A}_{\bar{T}}(q_{i-1}, q_i))$ and $\bar{u}_i' \in \mathsf{Lang}(\mathcal{A}_{\bar{T}}(q_{i-1}, q_i'))$. Without loss of generality, we assume that $\mathcal{A}_{\bar{T}}$ is minimal for L.

So there exists a word $\bar{y}$ and states p, p' (where $p \in F$ and $p' \notin F$) such that $y \in \mathsf{Lang}(\mathcal{A}_{\bar{T}}(q_i, p)) \cap \mathsf{Lang}(\mathcal{A}_{\bar{T}}(q_i', p'))$. Let $\bar{x}$ be any word in $L(\bar{q}, 1) \cdots L(\bar{q}, i-1)$, and let us define $\bar{w} = \bar{x}\bar{u}_i\bar{y}, \bar{w}' = \bar{x}\bar{u}_i'\bar{y}$. Then $\bar{w} \in L$ and $\bar{w} \notin L$. Since $L \subseteq K(\bar{T})$, $\bar{w}$ has type sequence $\bar{T}$ and hence $\bar{y}$ must be in $\mathbb{A}_{T_i}^*\mathbb{A}_{T_i,T_{i+1}}\mathbb{A}_{T_{i+1}}^* \cdots \mathbb{A}_{T_n}^*$ ($\mathbb{A}_{T_n}^*$ if $i = n$).

Therefore, $\bar{w}'$ also has type sequence $\bar{T}$, i.e. $\bar{w}$ and $\bar{w}'$ have the same type sequence, and the same type factors with the exception of the i-th one. Furthermore, $\mathsf{type\text{-}factor}_i(\bar{w}) = \bar{u}_i\bar{u}'$ and $\mathsf{type\text{-}factor}_i(\bar{w}') = \bar{u}_i'\bar{u}'$, where $\bar{u}'$ is the longest prefix of $\bar{u}$ in $\mathbb{A}_{T_i}^*$. We know that $\bar{u}_i \sim_h \bar{u}_i'$, therefore $\bar{u}_i\bar{u}' \sim_h \bar{u}_i'\bar{u}'$. Then $\bar{w} \equiv_h \bar{w}'$ by Lemma 10. □

Remark 2. The final step in proving that MEMBERSHIP-$\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ is decidable is to show that checking whether $L(\bar{q}, i) \in \mathcal{G}_i$ is decidable, which amounts to checking membership in the class $\mathcal{W}_B$ (defined below in Lemma 16). For this we rely on the decidability of the $\mathcal{B}\mathbf{\Sigma}_1(<)$-separation problem [36].

Lemma 16. *Let A be an alphabet, $B \subseteq A$ and*

$$\mathcal{W}_B = \{BA^* \cap L \colon L \in |\mathcal{B}\mathbf{\Sigma}_1(<)|\}$$

Given any regular language $K \subseteq BA^*$, it is decidable whether $K \in \mathcal{W}_B$.

Proof. We reduce the problem of deciding whether $K \in \mathcal{W}_B$ to an instance of the $\mathcal{B}\mathbf{\Sigma}_1(<)$-separation problem, stated as follows:

$\mathcal{B}\mathbf{\Sigma}_1(<)$-SEPARATION
Input: Regular languages (L_1, L_2) over A such that $L_1 \cap L_2 = \emptyset$.
Question: Does there exist a language $L \in |\mathcal{B}\mathbf{\Sigma}_1(<)|$ such that $L_1 \subseteq L$ and $L \cap L_2 = \emptyset$?

$\mathcal{B}\mathbf{\Sigma}_1(<)$-SEPARATION is known to be decidable [36]. Now, given $K \subseteq BA^*$, we show that $K \in \mathcal{W}_B$ if and only if $(K, K^c \cap BA^*)$ has a separator in $\mathcal{B}\mathbf{\Sigma}_1(<)(A)$. This serves as an effectively computable procedure to check whether a given language $K \in \mathcal{W}_B$. First let us assume that $K \in \mathcal{W}_B$, that is, $K = BA^* \cap L$ for some $L \in |\mathcal{B}\mathbf{\Sigma}_1(<)|$. Then $K \subseteq L$ and further,

$$(K^c \cap BA^*) \cap L = K^c \cap (BA^* \cap L) = K^c \cap K = \emptyset$$

Therefore, L is a $\mathcal{B}\mathbf{\Sigma}_1(<)$-definable separator for $(K, K^c \cap BA^*)$. Conversely, assume that $(K, K^c \cap BA^*)$ has a separator $L \in |\mathcal{B}\mathbf{\Sigma}_1(<)|$. Then $K \subseteq L$ and hence $K = K \cap B^* \subseteq L \cap BA^*$. Moreover, $(K^c \cap BA^*) \cap L = \emptyset = K^c \cap (BA^* \cap L)$. Therefore $BA^* \cap L \subseteq K$ and hence $K = BA^* \cap L$, implying $K \in \mathcal{W}_B$. This completes the proof. □

We explore the separation problem for synchronous relations in Section 6.5. We now prove Theorem 17.

Proof of Theorem 17. Let the input R be a k-ary synchronous relation whose synchronized language L_R is recognized by some DFA $\mathcal{A}$. By Corollary 4, R is $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable if and only if $L_R \cap K(\bar{T})$ corresponds to a $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ relation for every type sequence $\bar{T}$. Since there are only finitely many type sequences of k-synchronized words, it is sufficient to check whether $L_R \cap K(\bar{T})$ corresponds to a $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$-definable relation for every type sequence $\bar{T}$. Now we fix a type sequence $\bar{T}$ and let $L_{\bar{T}} = L_R \cap K(\bar{T})$. Moreover, let $\mathcal{A}_{\bar{T}}$ be a DFA recognizing $L_{\bar{T}}$.

Now, $L_{\bar{T}} \subseteq K(\bar{T})$, and by Lemma 15, $L \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$ if and only if, for each $\bar{T}$-compatible state sequence $\bar{q} \in Q^n$ (of states in $\mathcal{A}_{\bar{T}}$) and $1 \leq i \leq n$, $L_{\bar{T}}(\bar{T}, \bar{q}, i) \in \mathcal{G}_i$. Therefore, to check whether $L_{\bar{T}} \in \|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$, we go through every $\bar{T}$-compatible state sequence $\bar{q}$ and check whether $L_{\bar{T}}(\bar{T}, \bar{q}, i) \in \mathcal{G}_i$. There are only finitely many $\bar{T}$-compatible state sequences, and checking the membership of a language in $\mathcal{G}_i$ is decidable (see Remark 2 and Lemma 16). Therefore, it is decidable to check whether $L_{\bar{T}}$ is in $\|\mathcal{B}\mathbf{\Sigma}_1[\sigma]\|$. This completes the proof. □

6.4 Characterizing $\|\mathbf{\Sigma}_2[\sigma]\|$ and $\|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$

The classes $\|\mathbf{\Sigma}_2[\sigma]\|$ and $\|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ are closely related to their classical counterparts, as shown by the following statements:

Theorem 18. *Let R be any relation. Then $R \in \|\mathbf{\Sigma}_2[\sigma]\|$ if and only if $L_R \in |\mathbf{\Sigma}_2(<)|$.*

The proof of Theorem 18 will be given later in this section. As a direct corollary, we obtain a characterization of $\|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ relations as well. Furthermore, the membership problems of $\mathbf{\Sigma}_2(<)$ and $\mathcal{B}\mathbf{\Sigma}_2(<)$ are known to be decidable [97]. Therefore we obtain the following results at no additional cost:

Theorem 19. *Let R be any relation, then $R \in \|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$ if and only if $L_R \in |\mathcal{B}\mathbf{\Sigma}_2(<)|$.*

Theorem 20. MEMBERSHIP-$\mathbf{\Sigma}_2[\sigma]$ *and* MEMBERSHIP-$\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ *are decidable.*

We now explain the proof of Theorem 18. Let A be an alphabet, then a *polynomial* over A is a language of the form

$$A_0^* a_1 A_1^* \dots A_{n-1}^* a_n A_n^*$$

We know that $|\mathbf{\Sigma}_2(<)|$ is exactly the set of finite unions of polynomials [114, 11]. Therefore, we consider polynomials over $\mathbb{A}_\perp^k$ as a starting point for characterizing $\|\mathbf{\Sigma}_2[\sigma]\|$. However, not every polynomial over $\mathbb{A}_\perp^k$ is a valid synchronized language. Consider for example the case of $\mathbb{A} = \{a, b\}$ and $k = 2$ and the subsets $\bar{A}_1 = \{(\perp, a)\}$ and $\bar{A}_2 = \{(b, \perp)\}$, then the polynomial

$$\bar{A}_1^*(a, a)\bar{A}_2^*$$

contains words such as $(\perp, a)(\perp, a)(a, a)(b, \perp)$ which are not synchronized. Therefore, we introduce the notion of a $\perp$-*consistent polynomial*, that is a polynomial over $\mathbb{A}_\perp^k$ which is also a synchronized language.

Definition 40. For any synchronized letter $\bar{a} = a_1 \otimes \cdots \otimes a_k \in \mathbb{A}_\perp^k$, we denote by $\tau(\bar{a})$ the set $\{i \in \{1,\ldots,k\} \colon a_i = \perp\}$. A non-empty subset $\bar{A} \subseteq \mathbb{A}_\perp^k$ is said to be **$\perp$-consistent** if and only if for every $\bar{a}_1, \bar{a}_2 \in \bar{A}$, $\tau(\bar{a}_1) = \tau(\bar{a}_2)$. In that case, we define $\tau(\bar{A})$ to be $\tau(\bar{a})$ for any $\bar{a} \in \bar{A}$. We say that the monomial

$$\bar{A}_0^* \bar{a}_1 \bar{A}_1^* \ldots \bar{a}_n \bar{A}_n^*$$

is **$\perp$-consistent** if and only if all the non-empty sets among $\bar{A}_0, \bar{A}_1, \ldots, \bar{A}_n$ are $\perp$-consistent and $\tau(\bar{A}_0) \subseteq \tau(\bar{A}_1) \subseteq \cdots \subseteq \tau(\bar{A}_n)$ (where the term $\bar{A}_i$ is skipped if $\bar{A}_i = \emptyset$). We define by $\bar{\mathcal{P}}$ the set of all $\perp$-consistent polynomials, that is, the set of all finite unions of $\perp$-consistent monomials.

Observation 4. Let L be an $\mathbb{A}_\perp^k$-monomial. Then $L \subseteq \mathrm{SW}_k$ if and only if $L \in \bar{\mathcal{P}}$. Furthermore, if L is $\perp$-consistent, then for every$S \subseteq \{1,\ldots,k\}$, $\pi_S(L)$ is a $\perp$-consistent polynomial as well.

Lemma 17. *Let $R \in \|\mathbf{\Pi}_1[\sigma]\|$. Then $L_R \in \bar{\mathcal{P}}$.*

Proof. Every relation in $\|\mathbf{\Pi}_1[\sigma]\|$ is the complement of a relation in $\|\mathbf{\Sigma}_1[\sigma]\|$. By Corollary 3, then, $L_R = \mathrm{SW}_k \setminus \uparrow_s \mathcal{S}$ for some finite set $\mathcal{S} \subset \mathrm{SW}_k$. Since polynomials are closed under intersection (due to $\mathbf{\Sigma_2}(<)$-definability), we only need to show that $\mathrm{SW}_k \setminus \uparrow_s \bar{w} \in \bar{\mathcal{P}}$ for any synchronized word $\bar{w}$. Fix $\bar{w} \in \mathrm{SW}_k$ and let $\bar{T} = (T_1, \ldots, T_n)$ be its type sequence. By Lemma 10, a synchronized word $\bar{u}$ is *not* a synchronized subword of $\bar{w}$ if and only if one of the following holds (note that the conditions are not exclusive):

(i) $\bar{T} \not\sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$

(ii) $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ via a witness t but for some i, $\bar{w}_i \not\sqsubseteq \bar{u}_{t(i)}$

(iii) $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ via a witness t, but for some i such that T_i is an end type for $\bar{T}$, $\bar{w}_i$ and $\bar{u}_{t(i)}$ do not have the same last letter

Let C_1, C_2, C_3 be the sets of synchronized words that satisfy conditions $(i), (ii), (iii)$ respectively. Then

$$\mathrm{SW}_k \setminus (\uparrow_s \bar{w}) = C_1 \cup C_2 \cup C_3$$

It is sufficient to show that each C_i is a $\perp$-consistent polynomial to prove that $\mathrm{SW}_k \setminus \uparrow_s \bar{w} \in \bar{\mathcal{P}}$.

Firstly, C_1 is equal $\bigcup K(\bar{T}')$, where the union runs over all type sequences $\bar{T}'$ such that $\bar{T}' \not\sqsubseteq \bar{T}$. (Recall that $K(\bar{T}')$ is the set of synchronized words whose type-sequence is $\bar{T}'$). By Observation 1, $K(\bar{T})$ is a $\perp$-consistent polynomial for every type-sequence $\bar{T}$. Therefore $C_1 \in \bar{\mathcal{P}}$.

Next, the set C_2 is equal to $\bigcup_{i=1}^{i=n} C_{2,i}$, where for every $1 \le i \le n$, $C_{2,i}$ is the set of words $\bar{u}$ such that $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ via a witness function t such that $\bar{w}_i \not\sqsubseteq \bar{u}_{t(i)}$. Formally,

$$C_{2,i} = \mathbb{A}_{-,T_1} \mathbb{A}_{T_1}^* \mathbb{A}_{T_1,T_2} \mathbb{A}_{T_2}^* \ldots \mathbb{A}_{T_{i-2},T_{i-1}} \mathbb{A}_{T_{i-1}}^* L_i \mathbb{A}_{T_i,T_{i+1}} \mathbb{A}_{T_{i+1}}^* \ldots \mathbb{A}_{T_{n-1},T_n} \mathbb{A}_{T_n}^*$$

where L_i is the set of elements of $\mathbb{A}_{T_{i-1},T_i}\mathbb{A}^*_{T_i}$ that do not have $\bar{w}_i$ as their subword. Then L_i is $\mathbf{\Sigma}_1(<)$-definable over the alphabet $\mathbb{A}_{T_{i-1},T_i} \cup \mathbb{A}_{T_i}$, and hence $\mathbf{\Sigma}_2(<)$-definable. Therefore, $L_i \in \bar{\mathcal{P}}$ and hence $C_{2,i} \in \mathcal{P}$. It follows that $C_2 = \bigcup C_{2,i} \in \bar{\mathcal{P}}$.

Lastly, note that C_3 is equal to $\bigcup_{i=1}^{i=n} C_{3,i}$, where $C_{3,i}$ is the set of words $\bar{u}$ such that $\bar{T} \sqsubseteq \mathsf{type\text{-}seq}(\bar{u})$ via a witness function t such that $\bar{w}_i$ and $\bar{u}_{t(i)}$ do not have the same last letter. For each i we can write this as

$$C_{3,i} = \mathbb{A}_{-,T_1}\mathbb{A}^*_{T_1}\mathbb{A}_{T_1,T_2}\mathbb{A}^*_{T_2}\dots\mathbb{A}_{T_{i-2},T_{i-1}}\mathbb{A}^*_{T_{i-1}}L'_i\mathbb{A}_{T_i,T_{i+1}}\mathbb{A}^*_{T_{i+1}}\dots\mathbb{A}_{T_{n-1},T_n}\mathbb{A}^*_{T_n}$$

where $L'_i = (\mathbb{A}_{T_i} \setminus \mathbb{A}_{T_{i-1}})\mathbb{A}^*_{T_i} \cap \mathbb{A}^*_{T_i}B_i$ with B_i being the set of letters of $\mathbb{A}_{T_i}$ different from the last letter of $\bar{w}_i$. Again we have $L_i \in \bar{\mathcal{P}}$. This implies $C_{3,i} \in \bar{\mathcal{P}}$ for every i. Therefore, $C_3 \in \bar{\mathcal{P}}$. □

Proof of Theorem 18. Let R be a synchronous relation such that $R \in \|\mathbf{\Sigma}_2[\sigma]\|$. Then $L_R = L_\varphi$ for some $\mathbf{\Sigma}_2[\sigma]$ formula φ, of the form $\exists\bar{x}\psi(\bar{x})$ where $\psi \in \mathbf{\Pi}_1[\sigma]$. By Lemma 17, $L_\psi \in \bar{\mathcal{P}}$. By Observation 4, $\bar{\mathcal{P}}$ is closed under projection, so $L_\varphi = \pi_{\bar{x}}(L_\psi)$ is a $\perp$-consistent polynomial as well. Therefore, $L_\varphi \in |\mathbf{\Sigma}_2(<)(\mathbb{A}^k_\perp)|$.

Conversely, let R be a synchronous relation such that $L_R \in |\mathbf{\Sigma}_2(<)|$. Then L_R is a polynomial over $\mathbb{A}^k_\perp$. Since L_R is also the synchronized language of R, it must be a $\perp$-consistent polynomial. Therefore, $L_R \in \bar{\mathcal{P}}$. By Lemma 18, $R \in \|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$. □

Lemma 18. *Let R be a relation such that L_R is a $\perp$-consistent polynomial. Then $R \in \|\mathbf{\Sigma}_2[\sigma]\|$.*

Proof. Due to the closure of $\|\mathbf{\Sigma}_2[\sigma]\|$ under union, the proof reduces to the case where L_R is a $\perp$-consistent monomial $\bar{A}^*_0\bar{a}_1\dots\bar{A}^*_1\dots\bar{a}_n\bar{A}^*_n$. We construct a $\mathbf{\Sigma}_2[\sigma]$ formula $\varphi(Z)$ with free variables $Z = \{z_1,\dots,z_k\}$ defining R.

Due to $\perp$ -consistency, $\bar{a}_1\bar{a}_2\dots\bar{a}_n$ is a synchronized word. Hence there exist words $w_1,\dots,w_k \in \mathbb{A}^*$ such that $\bar{w} = w_1 \otimes \dots \otimes w_k = \bar{a}_1\dots\bar{a}_n$. Let us define additional sets of variables $X = \{x_{i,j} \colon 1 \le i \le k, 1 \le j \le |w_i|\}$ and $Y = \{y_1,\dots,y_k\}$. First we define a formula $\psi_1(X,Z)$ as a conjunction of the following formulæ over $1 \le i \le k$:

- for every $1 \le j < |w_i|$: $x_{i,j} \prec x_{i,j+1}$
- for every $1 \le j \le |w_i|$: $\ell_{w_i[j]}(x_{i,j})$
- $x_{i,|w_i|} \prec z_i$
- for every $1 \le i' \le k$ and $j \le \min(|w_i|,|w_{i'}|)$: $\mathsf{eq}(x_{i,j},x_{i',j})$

Note that $\exists X\psi_1(X,Z)$ is satisfied by a word $\bar{u} \in \mathrm{SW}_k$ if and only if $\bar{w}$ is a subword of $\bar{x}$ with the witness function given by $p(j) = |x_{h,j}|$ for any $1 \le h \le k$. The k variables in Y represent the k components of a prefix of $\bar{u}$, which is expressed by the formula $\psi_2(X,Y)$ given as a disjunction over all $H \subseteq \{1,\dots,k\}$ (H represents components of $\bar{u}$ which are shorter than the prefix $y_1 \otimes \dots \otimes y_k$), of the formulæ

$$\bigwedge_{h\in H}(y_h \preceq z_h \wedge \mathsf{eq}(y_h,z_h)) \wedge \bigwedge_{h,i\notin H}(y_h \prec z_h \wedge \mathsf{eq}(y_h,y_i)) \wedge \bigwedge_{h\in H, i\notin H} \exists r(r \preceq y_i \wedge \mathsf{eq}(r,y_h))$$

Recall that for a synchronized letter $\bar{a} \in \mathbb{A}_\perp^k$, $\tau(\bar{a}) = \{h\colon \pi_h(\bar{a}) = \perp\}$. For all $\bar{a} \in \mathbb{A}_\perp^k$ and $\bar{A} \subseteq \mathbb{A}_\perp^k$, we define

$$\psi_{\bar{a}}(Y) = \bigwedge_{h\notin\tau(\bar{a})} \ell_{\pi_h(\bar{a})}(y_h)$$
$$\psi_{\bar{A}}(Y) = \bigvee_{\bar{a}\in\bar{A}} \psi_{\bar{a}}(Y)$$

If $\bar{y}$ is a prefix of u and $\bar{w}$ is a subword of $\bar{u}$ with a witness function p, if for some $1 \leq j \leq n$ we have $|\bar{y}| = p(j)$, then $\bar{y}$ satisfies $\psi_{\bar{a}}$. Finally we verify that if $p(j) < |\bar{y}| < p(j+1)$ for some $1 \leq j \leq n-1$, then $\bar{y}$ satisfies $\psi_{\bar{A}_j}$. We ensure this by the subformulæ $\psi_3(X,Y) = \chi_0 \wedge \chi_1 \cdots \wedge \chi_n$, where

$$\chi_0(X,Y) = \left(\bigwedge_{h\notin\tau(\bar{A}_0)} y_h \prec x_{h,1}\right) \implies \psi_{\bar{A}_0}(Y)$$
$$\chi_n(X,Y) = \left(\bigwedge_{h\notin\tau(\bar{A}_n)} x_{h,n} \prec y_h\right) \implies \psi_{\bar{A}_n}(Y)$$

and for every $0 < j < n$,

$$\chi_j(X,Y) = \left(\bigwedge_{h\notin\tau(\bar{A}_j)} x_{h,j} \prec y_h \wedge y_h \prec x_{h,j+1}\right) \implies \psi_{\bar{A}_j}(Y)$$

Finally, we define R with the formula

$$\varphi(Z) = \exists X \psi_1(X,Z) \wedge \forall Y(\psi_2(X,Y) \wedge \psi_3(X,Y))$$

Now, $(w_1,\ldots,w_k)$ satisfies φ if and only if (i) $w_1 \otimes \cdots \otimes w_k$ contains $\bar{a}_1 \ldots \bar{a}_n$ as a subword, say at positions $i_1,\ldots,i_n$, and (ii) the synchronized letters in $w_1 \otimes \cdots \otimes w_k$ between positions i_p and i_{p+1} are in $\bar{A}_p$. The positions $i_1,\ldots,i_n$ are fixed by X, (i) is verified by ψ_1, and (ii) is verified by $\forall Y(\psi_2 \wedge \psi_3)$. Therefore, φ defines R.

□

6.5 Beyond membership: the separation problem for relations

We consider the separation problem for subclasses of relations. Consider a class of relations $\mathcal{R} \subseteq$ Sync. Then the $\mathcal{R}$-separation problem is:

Separation-$\mathcal{R}$

Input: A pair of synchronous relations (R_1, R_2) over $\mathbb{A}$ such that $R_1 \cap R_2 = \emptyset$.

Question: Does there exist a relation $R \in \mathcal{R}$ such that $R_1 \subseteq R$ and $R \cap R_2 = \emptyset$?

We consider the separation problem for every level of the FO[σ] quantifier alternation hierarchy. We show that it is decidable for $\mathbf{\Sigma}_1[\sigma]$, $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$, while it is open for $\mathcal{B}\mathbf{\Sigma}_1[\sigma]$. We begin with the case of $\mathbf{\Sigma}_1[\sigma]$, with a lemma:

Lemma 19. *Let (R_1, R_2) be a pair of synchronous relations such that $R_1 \cap R_2 = \emptyset$. Then they are separated by a relation in $\|\mathbf{\Sigma}_1[\sigma]\|$ if and only if $R_2 \cap \uparrow_s R_1 = \emptyset$.*

Proof. Assume first that (R_1, R_2) have a separator $R \in \|\mathbf{\Sigma}_1[\sigma]\|$. In other words, $R_1 \subseteq R$ and $R \cap R_2 = \emptyset$. This implies that

$$\uparrow_s R_1 \subseteq \uparrow_s R$$

Since $\uparrow_s R = R$ (by Theorem 14) and $R \cap R_2 = \emptyset$, it follows that $R_2 \cap \uparrow_s R_1 = \emptyset$.

Conversely, assume $R_2 \cap \uparrow_s R_1 = \emptyset$. Then $\uparrow_s R_1$ separates (R_1, R_2). Moreover, $\uparrow_s R_1 \in \|\mathbf{\Sigma}_1[\sigma]\|$ by Theorem 14. □

Corollary 5. SEPARATION-$\mathbf{\Sigma}_1[\sigma]$ *is decidable.*

Proof. Checking that an input pair (R_1, R_2) has a separator in $\|\mathbf{\Sigma}_1[\sigma]\|$ is equivalent to checking whether $\uparrow_s R_1 \cap R_2 = \emptyset$ (by Lemma 19). By Proposition 8, $\uparrow_s R_1$ is computable from an NFA for L_{R_1}. This completes the proof. □

Theorem 21. SEPARATION-$\mathbf{\Sigma}_2[\sigma]$ *and* SEPARATION-$\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ *are decidable.*

Proof. The characterization theorems for $\mathbf{\Sigma}_2[\sigma]$ and $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$ definable relations (Theorems 18 and 19) state that a relation is definable in $\mathbf{\Sigma}_2[\sigma]$ (resp. $\mathcal{B}\mathbf{\Sigma}_2[\sigma]$) if and only if its synchronized language is in $|\mathbf{\Sigma}_2(<)|$ (resp. $\mathcal{B}\mathbf{\Sigma}_2(<)$).

It follows that input pair of relations (R_1, R_2) has a $\|\mathbf{\Sigma}_2[\sigma]\|$ (resp. $\|\mathcal{B}\mathbf{\Sigma}_2[\sigma]\|$) separator if and only if (L_{R_1}, L_{R_2}) have a separator in $|\mathbf{\Sigma}_2(<)|$ (resp. $|\mathcal{B}\mathbf{\Sigma}_2(<)|$).

Therefore, each instance of SEPARATION-$\mathbf{\Sigma}_2(<)$ reduces to an instance of SEPARATION-$\mathbf{\Sigma}_2(<)$. Similarly, SEPARATION-$\mathcal{B}\mathbf{\Sigma}_2(<)$ reduces to SEPARATION-$\mathcal{B}\mathbf{\Sigma}_2(<)$. It is known that SEPARATION-$\mathbf{\Sigma}_2(<)$ and SEPARATION-$\mathcal{B}\mathbf{\Sigma}_2(<)$ are decidable [96, 98]. This completes the proof. □

The decidability of SEPARATION-$\mathcal{B}\mathbf{\Sigma}_1[\sigma]$ remains open.

Bibliography

[1] GQL Standards website, https://www.gqlstandards.org, 2020.

[2] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the web: from relations to semistructured data and XML*. Morgan Kaufmann, 2000.

[3] Serge Abiteboul and Richard Hull. IFO: A formal semantic database model. *ACM Transactions on Database Systems*, 12(4):525–565, 1987.

[4] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[5] Serge Abiteboul and Victor Vianu. Regular path queries with constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[6] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Review of Modern Physics*, 74:47–97, 2002.

[7] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutiérrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *Proceedings of the International Conference on Management of Data*, pages 1421–1432. ACM, 2018.

[8] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys*, 50(5):1–40, 2017.

[9] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. Mapping RDF databases to property graph databases. *IEEE Access*, 8:86091–86110, 2020.

[10] Kemafor Anyanwu and Amit Sheth. ρ-queries: enabling querying for semantic associations on the Semantic Web. In *Proceedings of the 12th International Conference on World Wide Web*, pages 690–699, 2003.

[11] Mustapha Arfi. Polynomial operations on rational languages. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, pages 198–206. Springer, 1987.

[12] Charles W Bachman. The programmer as navigator. In *Readings in Artificial Intelligence and Databases*, pages 52–59. Elsevier, 1989.

[13] Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 261–272, 2013.

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

[15] Pablo Barceló, Leonid Libkin, Anthony W Lin, and Peter T Wood. Expressive languages for path queries over graph-structured data. *ACM Transactions on Database Systems*, 37(4):1–46, 2012.

[16] Jean Berstel and Luc Boasson. *Transductions and context-free languages*. Springer-Verlag, 1979.

[17] Achim Blumensath and Erich Grädel. Automatic structures. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, pages 51–62. IEEE, 2000.

[18] Angela Bonifati, Wim Martens, and Thomas Timm. Navigating the maze of Wikidata query logs. In *The World Wide Web Conference*, pages 127–138, 2019.

[19] Ulrik Brandes, Linton C. Freeman, and Dorothea Wagner. Social networks. In *Handbook on Graph Drawing and Visualization*, pages 805–839. Chapman and Hall/CRC, 2013.

[20] J. Büchi. Weak second order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6:66–92, 1960.

[21] Peter Buneman. Semistructured data. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121. ACM New York, 1997.

[22] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 505–516, 1996.

[23] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[24] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y Vardi. Containment of conjunctive regular path queries with inverse. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning*, pages 176–185. Morgan Kaufmann Publishing Inc., 2000.

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

[26] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y Vardi. Reasoning on regular path queries. *ACM SIGMOD Record*, 32(4):83–92, 2003.

[27] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 2–13. Morgan Kaufmann Publishers Inc., 1998.

[28] Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th annual ACM Symposium on Theory of Computing*, pages 77–90, 1977.

[29] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000.

[30] Peter Pin-Shan Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[31] Yijia Chen, Jörg Flum, and Martin Grohe. Bounded nondeterminism and alternation in parameterized complexity theory. In *Proceedings of the 18th Annual IEEE Conference on Computational Complexity*, pages 13–29. IEEE Computer Society, 2003.

[32] Christian Choffrut. Relations over words and logic: A chronology. *Bulletin of the European Association for Theoretical Computer Science EATCS*, 89, 2006.

[33] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[34] Edgar F. Codd. Derivability, redundancy and consistency of relations stored in large data banks. *ACM SIGMOD Record*, 38(1):17–36, 2009.

[35] Thomas M Connolly and Carolyn E Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[36] Wojciech Czerwiński, Wim Martens, and Tomáš Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 150–161. Springer, 2013.

[37] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoc, Mingxi Wu, and Fred Zemke. Graph pattern matching in GQL and SQL/PGQ. *CoRR*, abs/2112.06217, 2021.

[38] Volker Diekert, Paul Gastin, and Manfred Kufleitner. A survey on small fragments of first-order logic over finite words. *International Journal of Foundations of Computer Science*, 19(03):513–548, 2008.

[39] Sergei N Dorogovtsev and José Fernando Mendes. *Evolution of networks: From biological nets to the Internet and WWW*. Oxford University Press, 2003.

[40] Rodney G Downey and Michael R. Fellows. *Parameterized complexity*. Springer, 2012.

[41] Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*, volume 4. Springer, 2013.

[42] Samuel Eilenberg. *Automata, languages, and machines*. Academic Press, 1974.

[43] Samuel Eilenberg, Calvin C. Elgot, and J. C. Shepherdson. Sets recognized by n-tape automata. *Journal of Algebra*, 13:447–464, 1969.

[44] Samuel Eilenberg and Marcel-Paul Schützenberger. On pseudovarieties. *Advances in Mathematics*, 19(3), 1976.

[45] Calvin C. Elgot and Jorge E. Mezei. On relations defined by generalized finite automata. *IBM Journal of Research and Development*, 9(1):47–68, 1965.

[46] Diego Figueira. Containment of UC2RPQ: the hard and easy cases. In *In Proceedings of the 23rd International Conference on Database Theory*, volume 155 of *LIPIcs*, pages 9:1–9:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[47] Diego Figueira and Varun Ramanathan. When is the evaluation of Extended CRPQ tractable? September 2021. Working paper/preprint, https://hal.archives-ouvertes.fr/hal-03353483.

[48] Diego Figueira, Varun Ramanathan, and Pascal Weil. The quantifier alternation hierarchy of synchronous relations. In *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 138, pages 29–1. LIPIcs, 2019.

[49] Patrick C Fischer and Arnold L Rosenberg. Multitape one-way nonwriting automata. *Journal of Computer and System Sciences*, 2(1):88–101, 1968.

[50] Daniela Florescu, Alon Levy, and Alberto Mendelzon. Database techniques for the World Wide Web: A survey. *ACM Sigmod Record*, 27(3):59–74, 1998.

[51] Daniela Florescu, Alon Levy, and Dan Suciu. Query containment for conjunctive queries with regular expressions. In *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 139–148, 1998.

[52] Dominik D Freydenberger and Nicole Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *Journal of Computer and System Sciences*, 79(6):892–909, 2013.

[53] Abraham Ginzburg. *Algebraic theory of automata*. ACM monograph series. Academic Press, 1968.

[54] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

[55] Gösta Grahne and Alex Thomo. Query answering and containment for regular path queries under distortions. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems*, pages 98–115. Springer, 2004.

[56] Mark Graves, Ellen R Bergeman, and Charles B Lawrence. A graph-theoretic data model for genome mapping databases. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, volume 5, pages 32–41. IEEE, 1995.

[57] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1–24, 2007.

[58] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *Proceedings on 33rd Annual ACM Symposium on Theory of Computing*, pages 657–666. ACM, 2001.

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

[60] Claudio Gutierrez, Carlos Hurtado, Alberto Mendelzon, and Jorge Perez. Foundations of Semantic Web databases. *Journal of Computer System Sciences*, 77:520–541, 2011.

[61] Ralf Hartmut Güting. GraphDB: Modeling and querying graphs in databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 297–308. Morgan Kaufmann Publishers Inc., 1994.

[62] M. Gyssens, J. Paredaens, J. van den Bussche, and D. van Gucht. A graph-oriented object database model. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):572–586, 1994.

[63] Joachim Hammer and Markus Schneider. The GenAlg project: developing a new integrating data model, language, and tool for managing and querying genomic information. *ACM SIGMOD Record*, 33(2):45–50, 2004.

[64] Michael Hammer and Dennis McLeod. The Semantic Data Model: a modelling mechanism for data base applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 26–36, 1978.

[65] Robert A Hanneman and Mark Riddle. *Introduction to social network methods.* publisher: University of California, Riverside (url: http://faculty.ucr.edu/˜hanneman/), 2005.

[66] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of The London Mathematical Society*, pages 326–336, 1952.

[67] Harry B Hunt (III), Daniel J Rosenkrantz, and Thomas G Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *Journal of Computer and System Sciences*, 12(2):222–268, 1976.

[68] Masami Ito. *Algebraic theory of automata and languages*. World Scientific, 2004.

[69] Stephen C Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 34:3–41, 1956.

[70] Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *Journal of Computer and System Sciences*, 61(2):302–332, 2000.

[71] Gabriel M. Kuper and Moshe Y. Vardi. A new approach to database logic. In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 86–96. ACM, 1984.

[72] Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. In *Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 346–354. Springer, 1992.

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

[74] Jens Lehmann, Jörg Schüppel, and Sören Auer. Discovering unknown connections - the DBpedia relationship finder. In *Proceedings of the 1st Conference on Social Semantic Web*. Gesellschaft für Informatik e.V., 2007.

[75] Ulf Leser. A query language for biological networks. In *Proceedings of the 4th European Conference on Computational Biology*, page 39, 2005.

[76] M. Levene and A. Poulovassilis. The hypernode model and its associated query language. In *Proceedings of the 5th Jerusalem Conference on Information Technology*, pages 520–530, 1990.

[77] Alon Y. Levy, Anand Rajaraman, and Joann Ordille. Query-answering algorithms for information agents. *Proceedings of the National Conference on Artificial Intelligence*, 1:40–47, 2003.

[78] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases: a unifying logical view. In *Proceedings of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems*, pages 7–17, 1997.

[79] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[80] Alon Y. Levy and Yehoshua Sagiv. Semantic query optimization in Datalog programs. In *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 163–173. ACM, 1995.

[81] M Mainguenaud. Modelling of the Geographical Information System network component. *International Journal of Geographical Information Systems*, 9(6):575–593, 1995.

[82] Stanislav Malyshev, Markus Krötzsch, Larry González, Julius Gonsior, and Adrian Bielefeldt. Getting the most out of Wikidata: semantic technology usage in Wikipedia's knowledge graph. In *Proceedings of the International Semantic Web Conference*, pages 376–394. Springer, 2018.

[83] Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. In *37th International Symposium on Theoretical Aspects of Computer Science*, volume 154 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[84] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata (M.I.T. Research Monograph No. 65)*. The MIT Press, 1971.

[85] Claudia Bauzer Medeiros and Fatima Pires. Databases for GIS. *ACM Sigmod Record*, 23(1):107–115, 1994.

[86] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

[87] Tova Milo and Dan Suciu. Index structures for path expressions. In *International Conference on Database Theory*, pages 277–295. Springer, 1999.

[88] Wolfgang Nejdl, Wolf Siberski, and Michael Sintek. Design issues and challenges for RDF and schema-based peer-to-peer systems. *ACM SIGMOD Record*, 32(3):41–46, 2003.

[89] M.E.J. Newman. The structure and function of complex networks. *Computer Physics Communications*, 147:40–45, 2003.

[90] Frank Olken. Tutorial on graph data management for biology. In *Proceedings of the IEEE Computer Society Bioinformatics Conference*, volume 3, 2003.

[91] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 12–19. ACM, 1997.

[92] J. Paredaens, P. Peelman, and L. Tanca. G-Log: a graph-based query language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):436–453, 1995.

[93] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. In *Proceedings of the International Semantic Web Conference*, pages 30–43. Springer, 2006.

[94] J-E Pin and Pascal Weil. Polynomial closure and unambiguous product. *Theory of Computing Systems*, 30(4):383–422, 1997.

[95] Jean-Éric Pin. Mathematical foundations of automata theory. *Lecture Notes, LIAFA, Université Paris 7*, 2020.

[96] Thomas Place and Marc Zeitoun. Going higher in the first-order quantifier alternation hierarchy on words. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*, pages 342–353. Springer, 2014.

[97] Thomas Place and Marc Zeitoun. The tale of the quantifier alternation hierarchy of first-order logic over words. *ACM SIGLOG News*, 2(3):4–17, 2015.

[98] Thomas Place and Marc Zeitoun. Separation for dot-depth two. In *Proceedings of the 32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–12. IEEE, 2017.

[99] Alexandra Poulovassilis and Mark Levene. A nested-graph model for the representation and manipulation of complex objects. *ACM Transactions on Information Systems*, 12(1):35–68, 1994.

[100] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

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

[102] Mauro San Martín and Claudio Gutierrez. Representing, querying and transforming social networks with RDF/SPARQL. In *Proceedings of the European Semantic Web Conference*, pages 293–307. Springer, 2009.

[103] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[104] Marcel-Paul Schützenberger. Une théorie algèbrique du codage. *Séminaire Dubreil. Algèbre et théorie des nombres*, 9:1–24, 1955.

[105] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[106] Shashi Shekhar, Mark Coyle, Brajesh Goyal, Duen-Ren Liu, and Shyamsundar Sarkar. Data models in geographic information systems. *Communications of the ACM*, 40(4):103–111, 1997.

[107] Amit Sheth, Boanerges Aleman-Meza, I Budak Arpinar, Clemens Bertram, Yashodhan Warke, Cartic Ramakrishanan, Chris Halaschek, Kemafar Anyanwu, David Avant, F. Sena Arpinar, and Krys Kochut. Semantic association identification and knowledge discovery for national security applications. *Journal of Database Management (JDM)*, 16(1):33–53, 2005.

[108] Avi Silberschatz, Henry F Korth, and S Sudarshan. Data models. *ACM Computing Surveys*, 28(1):105–08, 1996.

[109] Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222. Springer, 1975.

[110] Jacques Stern. Complexity of some problems from the theory of automata. *Information and Control*, 66(3):163–176, 1985.

[111] Howard Straubing. *Finite Automata, Formal Logic, and Circuit Complexity*. Birkhauser Verlag, 1994.

[112] Howard Straubing and Denis Thérien. Partially ordered finite monoids and a theorem of I. Simon. *Journal of Algebra*, 119(2):393–399, 1988.

[113] Robert W Taylor and Randall L Frank. Codasyl data-base management systems. *ACM Computing Surveys*, 8(1):67–103, 1976.

[114] Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

[115] Wolfgang Thomas. Automata on infinite objects. In *Formal Models and Semantics*, pages 133–191. Elsevier, 1990.

[116] DC Tsichritzis and Frederick H. Lochovsky. Hierarchical data-base management: A survey. *ACM Computing Surveys*, 8(1):105–123, 1976.

[117] Moshe Y Vardi. The complexity of relational query languages. In *Proceedings of the 14th Annual ACM Aymposium on Theory of Computing*, pages 137–146, 1982.

[118] Michael Wehar. *On the complexity of intersection non-emptiness problems*. PhD thesis, University at Buffalo, 2016.

[119] Barry Wellman, Janet Salaff, Dimitrina Dimitrova, Laura Garton, Milena Gulia, and Caroline Haythornthwaite. Computer networks as social networks: Collaborative work, telework, and virtual community. *Annual Review of Sociology*, 22(1):213–238, 1996.

[120] M. Yannakakis. Perspectives on database theory. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 224–246. IEEE Computer Society, 1995.

[121] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of the 7th International Conference on Very Large Data Bases*, volume 7, pages 82–94, 1981.