



HAL
open science

Path Logics for Querying Graphs: Combining Expressiveness and Efficiency

Diego Figueira, Leonid Libkin

► **To cite this version:**

Diego Figueira, Leonid Libkin. Path Logics for Querying Graphs: Combining Expressiveness and Efficiency. Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), Jul 2015, Kyoto, Japan. 10.1109/LICS.2015.39 . hal-01778476

HAL Id: hal-01778476

<https://hal.science/hal-01778476>

Submitted on 25 Apr 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Path Logics for Querying Graphs: Combining Expressiveness and Efficiency

Diego Figueira
CNRS, LaBRI
Bordeaux, France
diego.figueira@labri.fr

Leonid Libkin
School of Informatics
University of Edinburgh
Edinburgh, UK
libkin@inf.ed.ac.uk

Abstract—We study logics expressing properties of paths in graphs that are tailored to querying graph databases: a data model for new applications such as social networks, the Semantic Web, biological data, crime detection, and others. The basic construct of such logics, a regular path query, checks for paths whose labels belong to a regular language. These logics fail to capture two commonly needed features: counting properties, and the ability to compare paths. It is known that regular path-comparison relations (e.g., prefix or equality) can be added without significant complexity overhead; however, adding common relations often demanded by applications (e.g., subword, subsequence, suffix) results in either undecidability or astronomical complexity.

We propose, as a way around this problem, to use automata with counting functionalities, namely Parikh automata. They express many counting properties directly, and they approximate many relations of interest. We prove that with Parikh automata defining both languages and relations used in queries, we retain the low complexity of the standard path logics for graphs. In particular, this gives us efficient approximations to queries with prohibitively high complexity. We extend the best known decidability results by showing that even more expressive classes of relations are possible in query languages (sometimes with restriction on the shape of formulae). We also show that Parikh automata admit two convenient representations by analogs of regular expressions, making them usable in real-life querying.

I. INTRODUCTION

Our main object of interest is logics for querying edge-labeled graphs. Of course such logics exist in abundance, and they come in many flavors, depending on the intended applications. Many such applications arise in verification, and logics are designed to specify temporal properties observed along paths in graphs. We are interested in logics that arise in the study of semistructured data, i.e., data that does not conform to the standard relational model [1]. These classes of logics have many commonalities [34], [37], but they are still sufficiently distinct due to the nature of applications and specific computational requirements posed by them.

The main need for the development and study of the logics we are looking at comes from *graph databases* [39]. They have emerged as a new trend in data management technology fairly recently, and have attracted much attention due to numerous applications in many active areas where the underlying data is naturally modeled as a graph; these include social networks, Semantic Web and RDF, biological networks, crime detection, etc. Multiple prototypes and commercial systems exist, includ-

ing NEO4J, INFINITEGRAPH, ALLEGROGRAPH, ORIENTDB; see also [3], [5], [43] for recent surveys of the area.

Among the proposals for logical languages for querying graph data, the ones that use *regular path queries*, or *RPQs* have become most commonly accepted (although other interesting proposals exist as well, for instance those based on adapting spatial calculi for semistructured data [15], [20]). RPQs, proposed initially in [18], check the existence of a path whose label belongs to a regular language. Formally, such a query is of the form

$$x \xrightarrow{L} y$$

where L is a regular language. When applied to a graph G , it returns pairs of nodes x and y such that there is an L -labeled path between them. Such queries play the role of atomic formulae in other formalisms. Most commonly, they are closed under conjunction and existential quantification, resulting in *conjunctive regular path queries* (CRPQs) [14], [24] by analogy with the usual conjunctive queries, i.e., the \exists, \wedge -fragment of first-order predicate logic. One commonly writes them as rules: for instance,

$$q(x) \text{ :- } x \xrightarrow{a^+} y, x \xrightarrow{b^+} y$$

looks for nodes x from which both an a -labeled path and a b -labeled path to the same node y exist.

CRPQs are well understood, and have many nice properties: for instance, they can be efficiently evaluated, matching the combined complexity of relational conjunctive queries [17]. However, for many applications their power is insufficient [5], [22], [43]. The main shortcomings of CRPQs stem from the fact that

- they cannot compare paths; and
- they are limited to regular properties of paths.

For instance, already over a decade ago, the RDF community proposed a query language in which paths can be compared for specific semantic associations [4]. Applications in biology and crime detection often require checking the similarity of paths, e.g., based on their edit distance, or relations such as subword or subsequence. We refer to [3], [5], [22], [43] for multiple concrete examples of properties of paths that need to be checkable by logical query languages.

Thus, finding ways to increase the power of graph query languages has been a major theme in graph data research over

the past several years. When it comes to comparing paths, the key idea is to name paths and quantify over them [6], [27]. Once we have the ability to refer to paths, we can use them in formulas by testing relationships between them. To give a concrete example, consider a formula

$$q(x, y) :- x \xrightarrow{\pi_1:a^+} z, z \xrightarrow{\pi_2:b^+} y, |\pi_1| = |\pi_2|. \quad (1)$$

It says that there is an a -labeled path π_1 from x to some node z , from which there is a b -labeled path π_2 to y such that the lengths of π_1 and π_2 are the same. In other words, x and y are connected by a path labeled $a^n b^n$ for some $n \geq 1$. The above formula comes from a class of *extended CRPQs* [6], which continue to enjoy many nice computational properties. Their data complexity is in NLOGSPACE (the best possible, as they express NLOGSPACE-complete reachability), and their combined complexity is PSPACE-complete, matching that of first-order logic (and thus of traditional relational databases).

Relations on paths used in extended CRPQs are *regular* (or *synchronized*) relations. The best way to visualize such a relation (say, binary) is to think of an automaton over a pair of words with two synchronously moving heads, first looking at the initial letters of the words, then at their second letters, and so on; when one word ends, only the remaining head continues. Examples of such relations are equal length, equality, prefix, or having edit distance bounded by a fixed constant [40].

However, many real-life relations of interest are *not* regular. Relations such as *subword*, *subsequence*, *suffix* which are very often required to express semantic associations between paths in RDF, or in biological and crime detection applications are not regular but *rational* [16], [40]: they are given by automata whose heads move asynchronously.

So it seems natural to add them to RPQ-based logical formalisms. This carries a huge price though: the query evaluation problem becomes undecidable with adding subword or suffix, and of astronomical complexity when subsequence is added [7]. To lower the complexity, severe restrictions are required: only *one* relation among paths can be used in the query, and the query itself must be restricted syntactically [7], [8]; this effectively rules out such query languages.

Problem: adding relations needed in real-life querying tasks to existing logical formalisms leads to prohibitively high complexity. But at the same time one cannot give up and dismiss such queries since they do often arise in practice.

Our approach: we look for suitable approximate solutions. To make it work, we need a *sufficiently expressive* and at the same time *efficient* logical framework in which finding approximate answers to queries involving complex nonregular properties is feasible.

To understand what such a framework might be, we appeal to an example. Consider a graph database used in a crime detection application, and assume that a query requires the comparison of two paths. One of them, π , is known to indicate criminal activity, the other, π' , is suspect, and we want to see if π' occurs as a subsequence of π . This is computationally

too expensive [7], but instead we can attempt to check whether $\#_a(\pi') \leq \#_a(\pi)$ for every label a , where $\#_a(\pi)$ is the number of occurrences of a in π . This does not give the exact answer to the query of course, but it selects paths that need to be further analyzed – and, as we shall see, this can be answered with low complexity. If we are interested in a subword relation, we can select a portion of path π so that the numbers of occurrences of all symbols in that portion will be the same as in π' .

This approach could be used with many properties that need to be tested by graph database queries, and not only counting occurrences of letters. For instance, suppose that a word w_i indicates a position of interest on a path π_i , and we want to check if letters that follow each occurrence of w_1 in π_1 , and each occurrence of w_2 in π_2 , form the same word. This similarly can be approximated by counting the numbers of occurrences of each label a after w_i in π_i , and comparing these numbers.

Thus, the general idea is to replace precise properties of words by their counting approximations. This approach has been used successfully in verification [19], [21], [26], [28], [31], and it seems natural to extend it to querying paths in graph databases.

Note that we go beyond simple Parikh images (i.e., counting letter occurrences). To capture properties of words formed by specific positions, we may need to express languages like

$$L_{ba=ca} = \left\{ w \mid \begin{array}{l} \text{the number of } as \text{ that follow a } b \\ \text{the number of } as \text{ that follow a } c \end{array} \right\}$$

What makes it possible for us to provide efficient query evaluation mechanism is the existence of an automata model for these properties: *Parikh automata* [30]. The intuition behind them is as follows. Assume that we have a fixed number k of *counters*. Each automaton transition, in addition to the usual NFA transition, carries a k -vector of integers; in a run, when such transition applies, these numbers are added to the counters. At the end, one checks a linear arithmetic constraint to determine acceptance. To accept the language $L_{ba=ca}$, we have counters r_1 and r_2 ; then r_1 is incremented in the state reached after reading ba , and r_2 in the state reached after reading ca . Acceptance is given by $r_1 = r_2$.

We demonstrate that these automata are very suitable for asking complex graph queries, due to three reasons:

- They can express counting approximations for many relevant properties that are otherwise too expensive to use directly in queries;
- they lead to good complexity of query evaluation; and
- there are natural ways of expressing languages given by such automata, making them usable in real-life queries.

Contributions We prove some results about Parikh automata that are necessary for understanding how to use them in graph querying, and demonstrate a few query languages based on Parikh automata that are both expressive and have good query evaluation complexity; these languages go significantly beyond previously known decidable languages in terms of their expressiveness. In more detail, we do the following.

- We pinpoint the exact complexity of some decidable problems related to Parikh automata, as these are needed to establish the complexity of query evaluation.
- We show that paths definable by Parikh automata can be added to CRPQs without an increase in complexity.
- We develop a theory of Parikh-definable relations, rather than languages, coming up with natural analogs of regular and rational relations over words. We do this by using a notion of synchronizations recently proposed in [23] as it lets one define classes of relations without relying on modes of acceptance, as is usually done in formal language theory. We also provide two natural analogs of regularity, allowing us to incorporate more relations into queries, and show that some problematic relations like subword and subsequence can naturally be approximated by regular Parikh relations.
- We look at extending CRPQs with Parikh-definable languages, and with relations on paths definable by two types of regular Parikh relations. Our main result is that for these extensions, data complexity is tractable, and combined complexity is in PSPACE, matching, for instance, first-order logic over relations. The resulting languages subsume and significantly extend all those for which acceptable complexity bounds have previously been known.
- We show that arbitrary Parikh-definable relations on words can also be used in queries, under appropriate syntactic restrictions.
- Finally, we look at usable representations of languages defined by Parikh automata. In practical languages, automata are usually represented by regular expressions; thus, to be able to use Parikh automata in real-life query languages, we need an analog of regular expressions for them. We present in fact two such analogs, one based on the idea of marking positions in a regular expression, and the other on choosing regular expressions that apply to prefixes of a word.

Organization Basic notations are given in Section II. In Section III we give necessary information about Parikh automata. Section IV studies evaluation of CRPQs with Parikh automata. In Section V we investigate relations based on Parikh automata, and in Section VI we study query evaluation with such relations. In Section VII we provide two types of regular expressions for Parikh automata. Due to space limitations, complete proofs are in the appendix.

II. PRELIMINARIES

Basic notations Let $\mathbb{N}_+ = \{1, 2, \dots\}$ and $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. We write \underline{k} to denote the set $\{1, \dots, k\}$. We use letters \mathbb{A}, \mathbb{B} to denote finite alphabets. The set of finite words over \mathbb{A} is denoted by \mathbb{A}^* . For a word $w = a_1 \dots a_n \in \mathbb{A}^*$, the letter a_i in the i th position is denoted by $w[i]$, the subword $a_i \dots a_j$ by $w[i, j]$, and the length n of the word by $|w|$. Given $w \in \mathbb{A}^*$ and a letter $a \in \mathbb{A}$, we define $\#_a(w)$ as the number of occurrences of a in w .

For two words $u = a_1 \dots a_n \in \mathbb{A}^*$ and $v = b_1 \dots b_n \in \mathbb{B}^*$, we write $u \otimes v$ for the word $(a_1, b_1) \dots (a_n, b_n) \in (\mathbb{A} \times \mathbb{B})^*$. This definition can be extended to words of different length: in that case, the shorter word (say, u) is padded with $|v| - |u|$ letters \perp that do not occur in \mathbb{A}, \mathbb{B} to a word u' (of the same length as v) and then we take $u \otimes v$ to be $u' \otimes v$. This definition straightforwardly extends to tuples of words, i.e., to $w_1 \otimes w_2 \otimes \dots \otimes w_k$.

Parikh images, semilinear sets, and Presburger arithmetic For a finite, ordered, alphabet $\mathbb{A} = \{a_1, \dots, a_k\}$, the *Parikh image* of a word $w \in \mathbb{A}^*$, written $\Pi(w)$, is the vector of \mathbb{N}^k whose i th component contains $\#_{a_i}(w)$, the number of occurrences of a_i in w . The Parikh image of a language L is $\Pi(L) = \{\Pi(w) \mid w \in L\}$.

A *linear set* is a subset of \mathbb{N}^k that can be described as

$$\{\bar{v}_0 + \alpha_1 \bar{v}_1 + \dots + \alpha_n \bar{v}_n \mid \alpha_1, \dots, \alpha_n \in \mathbb{N}\} \quad (2)$$

for some $n \in \mathbb{N}$ and $\bar{v}_0, \dots, \bar{v}_n \in \mathbb{N}^k$. A *semilinear set* is a finite union of linear sets. Henceforward we assume that linear sets are represented by the *offset* \bar{v}_0 and the *generators* $\bar{v}_1, \dots, \bar{v}_n$, where numbers are represented in binary; and that semilinear sets are represented as a collection of linear set representations. *Presburger arithmetic* refers to first-order logic over $\langle \mathbb{N}, \leq, 0, 1, +, f_2 \rangle$, where $f_2 : \mathbb{N} \rightarrow \mathbb{N}$ is the doubling function: $f_2(x) = 2x$. Of course f_2 is definable with $+$, but we include it here to generate Presburger formulae of smaller size (see below). It is well-known that semilinear sets correspond precisely to Presburger arithmetic (or its existential fragment) [25] and to Parikh images of context free (or regular) languages [38]. Moreover, using the doubling function to represent each $n \in \mathbb{N}$ as a term of Presburger arithmetic of size $O(\log n)$, we have the following.

Lemma II.1. *For any given semilinear set, an equivalent existential Presburger formula can be produced in linear time.*

Graph databases, RPQs and CRPQs The standard abstraction of *graph databases* [3], [5], [39] is finite \mathbb{A} -labeled graphs $G = \langle V, E \rangle$, where V is a finite set of nodes, or vertices, and $E \subseteq V \times \mathbb{A} \times V$ is a set of labeled edges. A *path* p from v_0 to v_m in G is a sequence of edges $(v_0, a_1, v_1), (v_1, a_2, v_2), \dots, (v_{m-1}, a_m, v_m)$ from E , for some $m \geq 0$. The *label* of p , denoted by $\lambda(p)$, is the word $a_1 \dots a_m \in \mathbb{A}^*$.

The main building blocks for graph queries are *regular path queries*, or *RPQs* [18]; they are expressions of the form $x \xrightarrow{L} y$, where L is a regular language over \mathbb{A} . Given an \mathbb{A} -labeled graph $G = \langle V, E \rangle$, the answer to an RPQ above is the set of pairs of nodes (v, v') such that there is a path p from v to v' with $\lambda(p) \in L$. *Conjunctive RPQs*, or *CRPQs* [13], [17] are the closure of RPQs under conjunction and existential quantification. We present them in a rule-based notation; formally, a CRPQ is an expression of the form

$$\varphi(\bar{z}) \quad :- \quad x_1 \xrightarrow{L_1} y_1, \dots, x_m \xrightarrow{L_m} y_m, \quad (3)$$

where x_i, y_i s are variables, and \bar{z} is a tuple of variables among those (as usual, variables that appear in the body of the rule but

not in the head are assumed to be existentially quantified; that is, we can view CRPQ (3) as $\varphi(\bar{z}) = \exists \bar{z}' \bigwedge_{i=1}^m (x_i \xrightarrow{L_i} y_i)$, where \bar{z}' refers to the tuple of variables used in the body but not in the head).

The semantics naturally extends the semantics of RPQs: $\varphi(\bar{a})$ is true in G iff there is a tuple \bar{b} of nodes so that \bar{a}, \bar{b} witness every RPQ in the body: if u_i, v_i from \bar{a}, \bar{b} interpret x_i and y_i , then there is a path p_i between u_i and v_i whose label $\lambda(p_i)$ is in L_i .

Relations on words *Regular relations* on words are simply defined by the acceptance of the word $w_1 \otimes \dots \otimes w_k$ by an NFA. That is, to determine if a tuple $(w_1, \dots, w_k) \in (\mathbb{A}^*)^k$ is in the relation, one runs an NFA over the word $w_1 \otimes \dots \otimes w_k$ over the alphabet $(\mathbb{A} \cup \{\perp\})^k$. Such an automaton first looks at the first symbols of all words, then at their second symbols, etc.; when a word ends, it is padded with the extra symbol \perp . We denote the class of such relations REG.

Examples of regular relations are prefix, equality, equal length, and bounded edit distance. There are, however, many other important relations that are non-regular. These are *suffix*: $w \preceq_{\text{suffix}} u$ iff $u = v \cdot w$ for some v ; *subword*: $w \preceq_{\text{subw}} u$ iff $u = v \cdot w \cdot v'$ for some v, v' , and *subsequence*: $w \preceq_{\text{subseq}} u$ if some letters can be deleted from u in a way that results in w . These three relations are not regular but *rational*: they can be defined by asynchronous automata with k heads (for k -ary relations), or alternatively by regular expressions over $(\mathbb{A} \cup \{\varepsilon\})^k$, see [40].

III. PARIKH AUTOMATA

For presentational purposes, we give a slight syntactic variation of the usual definition of Parikh automata from [29], [30]. This easier to grasp definition follows the intuition explained in [30] and is in the spirit of models such as [2]. It is readily seen to be equivalent to the original definition, which we present as well, and which in fact is often better suited for proofs.

Definition III.1. A Parikh Automaton, or PA, of dimension k , is a tuple $\mathcal{PA} = (Q, \mathbb{A}, \delta, q_0, F, \mathbf{C})$, where Q is a finite set of states, \mathbb{A} is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, \mathbf{C} is a semilinear set in \mathbb{N}^k (called a constraint set), and the transition relation δ is a finite subset of $Q \times \mathbb{A} \times \mathbb{N}^k \times Q$.

A run ρ of \mathcal{PA} on a word $w = a_1 \dots a_m \in \mathbb{A}^*$ is a map from $\{0, \dots, m\}$ to $Q \times \mathbb{N}^k$ such that

- $\rho(0) = (q_0, \bar{0})$;
- if $\rho(i) = (q, \bar{r})$ and $\rho(i+1) = (q', \bar{r}')$ then there exists a transition $(q, a_{i+1}, \bar{v}, q') \in \delta$ such that $\bar{r}' = \bar{r} + \bar{v}$.

A run is *accepting* if $\rho(m) \in F \times \mathbf{C}$. A word is *accepted* if it has an accepting run. The set of accepted words is denoted by $\mathcal{L}(\mathcal{PA})$.

In other words, the run starts in q_0 and has all zeros in k counters \bar{r} . If it is in state q and reads a , and there is a transition (q, a, \bar{v}, q') , it can enter q' and add \bar{v} to the counters, componentwise. For acceptance, the automaton must be in an accepting state, and have the tuple of counters in the set \mathbf{C} .

Note that one can easily model the condition $q \in F$ by an extra counter, only accepting by the configuration of counters. We also call the set of all tuples from \mathbb{N}^k that appear in transitions of \mathcal{PA} its *auxiliary set*.

Languages accepted by PAs are closed under union, intersection, homomorphisms, inverse homomorphisms and concatenation (though not complementation and iteration). A deterministic version of PAs is known to be strictly weaker than arbitrary PAs [30]. PAs are also known to be equivalent to reversal-bounded multicounter machines [28] in the sense that they describe the same class of languages [30], [11].

Original definition We now present the definition of k -dimensional Parikh automata from [30], [29]. Such automata are pairs $\mathcal{PA} = (\mathcal{A}', \mathbf{C})$ where $\mathcal{A}' = (Q, \mathbb{A} \times \mathbb{D}, \delta, q_0, F)$ is an NFA over $\mathbb{A} \times \mathbb{D}$, and \mathbb{D} is a finite subset of \mathbb{N}^k , which we call *auxiliary set*. A word $w = a_1 \dots a_n$ is accepted by $(\mathcal{A}', \mathbf{C})$ if there is a word $v = \bar{v}_1 \dots \bar{v}_n$ over \mathbb{D} such that $w \otimes v$ is accepted by \mathcal{A}' and $\sum_{i \leq n} \bar{v}_i \in \mathbf{C}$.

Note that we simply incorporated the tuples of numbers over the finite set $\mathbb{D} \subset \mathbb{N}^k$ into transitions, and the summation into the definition of a run. The translations between the two versions preserve all complexity bounds. When we use this definition, we shall write $\oplus v$ for $\sum_{i \leq n} \bar{v}_i$ when $v = \bar{v}_1 \dots \bar{v}_n$ (with $\oplus v = \bar{0}$ when v is the empty word).

An alternative characterization. We now provide two closely related views of Parikh automata that will help us better visualize their power and later define analogs of regular expressions for them. Let $\mathcal{A} = (Q, \mathbb{A}, \delta, q_0, F)$ with $\delta \subseteq Q \times \mathbb{A} \times Q$ be a usual NFA. We can view a run ρ of \mathcal{A} on a word w as a $|w| + 1$ -tuple of states, i.e., a map from $\{0, 1, \dots, |w|\}$ to Q .

Proposition III.1. For a language $L \subseteq \mathbb{A}^*$ the following are equivalent:

- 1) L is accepted by a Parikh automaton;
- 2) there is an NFA \mathcal{A} over \mathbb{A} with the set of states Q and a semilinear set $\mathbf{C} \subseteq \mathbb{N}^{|Q|}$ such that

$$L = \{w \mid \exists \text{ accepting run } \rho \text{ of } \mathcal{A} \text{ on } w \text{ with } \Pi(\rho) \in \mathbf{C}\};$$

- 3) there is an alphabet \mathbb{B} , an NFA \mathcal{A}' over $\mathbb{A} \times \mathbb{B}$ and a semilinear set $\mathbf{C} \subseteq \mathbb{N}^{|\mathbb{A}| \cdot |\mathbb{B}|}$ such that

$$L = \{w \mid \exists v \in \mathbb{B}^* : w \otimes v \in \mathcal{L}(\mathcal{A}') \text{ and } \Pi(w \otimes v) \in \mathbf{C}\}.$$

A. Complexity of Parikh automata

It is known that language nonemptiness (is $\mathcal{L}(\mathcal{PA}) \neq \emptyset$?) and membership (is $w \in \mathcal{L}(\mathcal{PA})$?) are decidable for Parikh automata [30], [29]. Here we pinpoint their exact complexity, as it will be needed later.

Proposition III.2. Both the nonemptiness and membership problem for Parikh automata are NP-complete. They can be solved in polynomial time if the dimension is fixed and all numbers are represented in unary.

Proof sketch. We explain the proof for nonemptiness. Consider $\mathcal{PA} = (\mathcal{A}, \mathbf{C})$ of dimension k so that $\mathcal{L}(\mathcal{A}) \subseteq (\mathbb{A} \times \mathbb{D})^*$

with $\mathbb{D} = \{\bar{d}_1, \dots, \bar{d}_n\}$. We first build an existential Presburger formula $\varphi_{\mathbf{C}}(y_1, \dots, y_k)$ describing \mathbf{C} in linear time, by Lemma II.1. We then produce an existential Presburger formula $\varphi_{\mathcal{A}}^{\Pi}(x_1, \dots, x_k)$ for $\Pi(\{v \mid u \otimes v \in \mathcal{L}(\mathcal{A})\}) \subseteq \mathbb{N}^n$ in linear time. This is possible by a combination of [42] and a small fix to their (slightly underconstructed) formula given in [26]. From $\varphi_{\mathcal{A}}^{\Pi}$ one can obtain, in polynomial time, a formula $\varphi_{\mathcal{A}}$ describing $\{\oplus v \mid u \otimes v \in \mathcal{L}(\mathcal{A})\} \subseteq \mathbb{N}^k$. Finally, we check for satisfiability of $\varphi_{\mathcal{A}} \wedge \varphi_{\mathbf{C}}$ in NP [41]. For NP-hardness, one can reduce from the Subset Sum Problem: given integers n_1, \dots, n_k , whether there is a non-empty subset of them whose sum is 0.

For the unary representation of $(\mathcal{A}, \mathbf{C})$, one can construct in polynomial time an automaton \mathcal{B} over an alphabet of size k such that $\Pi(\mathcal{L}(\mathcal{B})) = \mathbf{C}$, and an automaton \mathcal{A}' from \mathcal{A} so that $\Pi(\mathcal{L}(\mathcal{A}'))$ is $\{\oplus v \mid w \otimes v \in \mathcal{L}(\mathcal{A})\}$. To test emptiness, we just test if $\Pi(\mathcal{L}(\mathcal{A}')) \cap \Pi(\mathcal{L}(\mathcal{B})) = \emptyset$, using the fact that the problem of whether the Parikh images of two given regular languages are disjoint is in PTIME, when the alphabet is fixed [32], [36]. \square

Another problem whose precise complexity we need is the problem of nonemptiness of intersection of languages. For regular languages, it is a classical PSPACE-complete problem [33].

The *language intersection problem* for PAs is defined as follows: its input is a set $\mathcal{PA}_1, \dots, \mathcal{PA}_m$ of Parikh automata, and the question is whether

$$\bigcap_{i=1}^m \mathcal{L}(\mathcal{PA}_i) \neq \emptyset.$$

Solving this will be essential for understanding combined complexity of several classes of graph queries. We now show that going from NFAs to PAs, we keep the upper bound on the complexity of the problem.

Proposition III.3. *The language intersection problem for Parikh automata is PSPACE-complete.*

Proof sketch. Let $\mathcal{PA}_i = (\mathcal{A}_i, \mathbf{C}_i)$ for $i \in \underline{m}$, so that each \mathcal{A}_i has an alphabet $\mathbb{A} \times \mathbb{D}_i$ with $\mathbb{D}_i \subseteq \mathbb{N}^{n_i}$, under the definition of [30]. We define a NFA \mathcal{A} over $\underline{\mathbf{S}}^N \subseteq \mathbb{N}^N$ for $N = \sum_i n_i$, and $S = \max\{\|v\|_{\infty} \mid v \in \bigcup_i \mathbb{D}_i\}$, so that $\oplus \mathcal{L}(\mathcal{A}) = \{\oplus v \mid v \in \mathcal{L}(\mathcal{A})\}$ has nonempty intersection with $\prod_{i=1}^m \mathbf{C}_i$ iff the instance has the positive answer. This automaton \mathcal{A} has the alphabet \mathbb{D} of size $n = |\mathbb{D}_1| + \dots + |\mathbb{D}_m|$ that consists of all vectors

$$\underbrace{(0, \dots, 0, \bar{x}_i, 0, \dots, 0)}_{\sum_{j < i} n_j} \quad \underbrace{(0, \dots, 0)}_{\sum_{j > i} n_j}$$

for $i \in \underline{m}$ and $\bar{x}_i \in \mathbb{D}_i$. The automaton \mathcal{A} simulates the execution of $\mathcal{A}_1, \dots, \mathcal{A}_m$ in parallel by iterating the following:

- it guesses an alphabet letter $(a_i, \bar{x}_i) \in \mathbb{A} \times \mathbb{D}_i$ for each \mathcal{A}_i so that $a_1 = \dots = a_m$,
- it stores the states q_1, \dots, q_m to which transitions lead for each automaton, as well as the vectors $\bar{x}_1, \dots, \bar{x}_m$, and
- it reads $\bar{x}_1, \dots, \bar{x}_m$ one at a time using m transitions.

The idea is that, by reading one \bar{x}_i at a time, we are summing (instead of multiplying) the auxiliary sets of $\mathcal{A}_1, \dots, \mathcal{A}_m$.

By [36, Theorem 7.3.1], the Parikh image $\Pi(\mathcal{L}(\mathcal{A})) \subseteq \mathbb{N}^n$ can be described by a finite union of linear sets of the form

$$\{\bar{v}_0 + \sum_{i=1}^{n'} \alpha_i \bar{v}_i \mid \alpha_1, \dots, \alpha_{n'} \in \mathbb{N}\} \quad (4)$$

so that $n' \leq n$, and $\max_i \|\bar{v}_i\|_{\infty}$ is bounded by $(n|Q|)^{O(n)}$, where Q is the statespace of \mathcal{A} . From this representation one can easily obtain a semilinear set for $\oplus \mathcal{L}(\mathcal{A}) \subseteq \mathbb{N}^n$ whose every linear set is of the form (4) so that $n' \leq n$ and $\max_i \|\bar{v}_i\|_{\infty}$ is bounded by $(n|Q|)^{O(n)}$; hence these numbers are at most exponential in $\sum_i |\mathcal{A}_i|$.

Therefore, due to Lemma II.1 and the use of binary representation of numbers, $\oplus \mathcal{L}(\mathcal{A})$ can be given as a disjunction $\bigvee_i \varphi_i$ of existential Presburger formulas, each of which has size linear in $n + N$ and logarithmic in $|Q|$.

The existential Presburger formula $\varphi_{\mathbf{C}}$ for $\prod_{i=1}^m \mathbf{C}_i$ can be built in linear time due to Lemma II.1. Hence, we have that $\varphi = \bigvee_i \varphi_i \wedge \varphi_{\mathbf{C}}$ represents $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}$, and that each $\varphi_i \wedge \varphi_{\mathbf{C}}$ is of size polynomial in $\{(\mathcal{A}_i, \mathbf{C}_i)\}_{i \in \underline{m}}$. By [41, Theorem 6 (A)], if an existential Presburger formula such as $\theta = \varphi_i \wedge \varphi_{\mathbf{C}}$ is satisfiable, there is a satisfying assignment so that each variable is assigned a number that is at most exponential in $|\theta|$. Thus, φ is satisfiable iff there exists a word $v \in \mathbb{D}^*$ so that $\oplus v \in \oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}$ and $\|\oplus v\|_{\infty}$ is at most exponential in $\{(\mathcal{A}_i, \mathbf{C}_i)\}_{i \in \underline{m}}$. By building \mathcal{A} on-the-fly we can guess and verify the existence of such v using only polynomial space. By Savitch's theorem, we can then test, in PSPACE, whether φ is satisfiable, which is equivalent to \mathcal{A} being nonempty, which in turn is equivalent to $\bigcap_i \mathcal{L}(\mathcal{A}_i, \mathbf{C}_i) \neq \emptyset$. Thus, the statement follows. \square

Our final result is on the intersection of the language of a fixed PA with a regular language. This is needed for determining data complexity of several query languages.

Proposition III.4. *For a fixed Parikh automaton \mathcal{PA} , the problem of checking, for an input NFA \mathcal{A} , whether $\mathcal{L}(\mathcal{PA}) \cap \mathcal{L}(\mathcal{A}) \neq \emptyset$ can be solved in NLOGSPACE.*

Proof sketch. Let $(\mathcal{A}_1, \mathbf{C}_1)$ be a fixed PA and \mathcal{A} an input NFA. Let \mathcal{A}' be an NFA such that $\mathcal{L}(\mathcal{A}') = \{v \mid \exists u \in \mathcal{L}(\mathcal{A}) : u \otimes v \in \mathcal{L}(\mathcal{A}_1)\}$. Notice that \mathcal{A}' is linear in \mathcal{A} . It suffices to check whether $\oplus \mathcal{L}(\mathcal{A}') \cap \mathbf{C}_1 \neq \emptyset$. By the previous proof, $\oplus \mathcal{L}(\mathcal{A}') \cap \mathbf{C}_1$ is given by an existential Presburger formula $\varphi = \bigvee_i \varphi_i \wedge \varphi_{\mathbf{C}_1}$ so that each $\varphi_i \wedge \varphi_{\mathbf{C}_1}$ is logarithmic in the number of states of \mathcal{A}' (and thus of \mathcal{A}).

By [41], a satisfiable disjunct $\varphi_i \wedge \varphi_{\mathbf{C}_1}$ has a satisfying assignment with numbers exponential in φ_i (since $\varphi_{\mathbf{C}_1}$ is fixed). Thus, if $\oplus v \in \oplus \mathcal{L}(\mathcal{A}') \cap \mathbf{C}_1$, there is one where all numbers in $\oplus v$ are at most polynomial in \mathcal{A} . This can be checked in NLOGSPACE guessing one position of v at a time and keeping a counter of logarithmic number of bits, ensuring that the sum of the guessed word is of size at most polynomial in \mathcal{A} . \square

IV. QUERY EVALUATION WITH PARIKH AUTOMATA

Recall that CRPQs (3) are defined as formulae of the form $\varphi(\bar{z}) := x_1 \xrightarrow{L_1} y_1, \dots, x_m \xrightarrow{L_m} y_m$, where each $x_i \xrightarrow{L_i} y_i$ is an RPQ, and variables used in the body but not in the head are existentially quantified. We assume that regular languages L_i s are represented by NFAs.

If the L_i s are given by Parikh automata, we refer instead to RPQ(PA) and CRPQ(PA) queries. For instance, $L_{a^n b^n} = \{a^n b^n \mid n > 0\}$ is a PA-definable language, and hence $x \xrightarrow{L_{a^n b^n}} y$ is an RPQ(PA) query selecting pairs of nodes that have a path between with the label from $L_{a^n b^n}$. Using previous formalisms, one needed relations on paths to do this, see (1). CRPQs combine several RPQs: for instance,

$$\varphi(x) := x \xrightarrow{L_{a^n b^n}} y, x \xrightarrow{L_{a^n b^n}} z, y \xrightarrow{a^+} z$$

finds nodes x from which we can reach, by a path of the form $a^n b^n$, two nodes that are connected by an a -labeled path.

Given a class \mathcal{Q} of queries over graphs, recall that *data complexity and combined complexity* of \mathcal{Q} refer to the following problems:

- for combined complexity, the input is a query $\varphi(\bar{x})$ from \mathcal{Q} , a graph G , and a tuple \bar{u} of nodes such that $|\bar{u}| = |\bar{x}|$; the question is whether $\varphi(\bar{u})$ is true in G ;
- for data complexity, the query $\varphi(\bar{x})$ is fixed, i.e., only G and \bar{u} are the parameters.

For ordinary RPQs and CRPQs, data complexity is in NLOGSPACE. In fact it is already NLOGSPACE-complete for the very simple RPQ $x \xrightarrow{\mathbb{A}^*} y$ as this is simply the NLOGSPACE-complete reachability problem. For RPQs based on NFAs, combined complexity is in PTIME, and for CRPQs it is NP-complete. Again the latter is the best possible, since combined complexity of relational conjunctive queries is NP-complete.

In general, when it comes to good complexity of query evaluation, the following is usually assumed. First, data complexity must always be tractable. Solving intractable problems on large volumes of data is simply out of the question. Combined complexity can be single exponential, reflecting the fact that in practical query evaluation algorithms, the size of the query is usually in the exponent. In fact relational database languages are based on first-order logic, for which data complexity is very low (AC^0) and combined complexity is PSPACE-complete; when one goes to the \exists, \wedge -fragment (conjunctive queries), it drops to NP.

We now show that adding Parikh conditions to RPQs costs us very little: the bounds for data complexity, and for combined complexity of CRPQs do not change at all. Combined complexity for RPQs changes, but it still is bounded by the good complexity of evaluating CRPQs. More precisely, we have the following.

Theorem IV.1. *The following complexity bounds hold for RPQs and CRPQs based on Parikh automata:*

- (I) *Data complexity of both RPQ(PA) and CRPQ(PA) is in NLOGSPACE.*

- (II) *Combined complexity of both RPQ(PA) and CRPQ(PA) is NP-complete.*

Proof idea. For RPQs, as usual, we take the product of the automaton with the graph viewed as an NFA, where the endnodes of the RPQ serve as the initial and the final state. The bounds then follow from Proposition III.2 for combined and from Proposition III.4 for data complexity. Hardness follows from the hardness of PA nonemptiness: we simply take a fixed graph in which every path from \mathbb{A}^* is realized. For CRPQs, an extra guessing step for witnesses of existential quantifiers is involved, which does not affect the complexity (since its size depends only on the query). \square

V. SYNCHRONIZED PARIKH RELATIONS

Query (1) from the introduction used a binary *relation* on paths. In fact, much of the motivation for this work stems from the need to use relations, in particular, relations that lead to computationally infeasible queries. Our goal now is twofold:

- first, we study relations rather than languages defined by PAs, and
- second, we see how to use these relations in queries.

We now concentrate on the first item; the second one is studied in the next section.

How can we define relations given by PAs? For NFAs, one typically operates not with one, but with three relational analogs of regular languages. In terms of increasing power, these are recognizable, regular, and rational relations [16], [40]. They correspond to three different ways of defining regularity of languages: by recognizability by a finite monoid, by acceptance by an NFA, and by regular (or rational) expressions. However, already in the binary case, these result in different classes of relations.

So what can we do with if the notion of NFA is replaced by Parikh automata? We do not yet have commonly accepted regular expressions for Parikh automata that can be lifted to relations, and the algebraic theory of PAs is in its infancy [12]. Instead, we use a recently proposed approach to defining relations over words that does not depend on the underlying properties of automata but rather on how these relations are *synchronized* [23].

To explain this idea, consider two words, $w_1 = abacb$ and $w_2 = bcaac$ in \mathbb{A}^* . A possible synchronization is:

$$\begin{array}{cccccccc} 1 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & 2 \\ a & b & b & a & c & c & a & b & a & c \end{array}$$

It is a shuffle of these two words, where each position is marked with 1 or with 2, indicating which word they come from. Formally, this is a word in $(\underline{2} \times \mathbb{A})^*$ (recall that \underline{k} stands for $\{1, \dots, k\}$). If we read the gray-shaded positions marked with 1, we see the word w_1 ; if we read positions marked with 2, we see w_2 .

Now we can run an automaton (e.g., an NFA) over such a synchronization to determine if the pair (w_1, w_2) is in the relation, and it is the shape of the word of 1s and 2s that determines what types of relations we get. If such words come

from the language 1^*2^* , we get recognizable relations. If they come from $(12)^*(1^*|2^*)$ (i.e., alternate between 1s and 2s until one of the words is finished), we get regular relations. And if there is no restriction on the word over $\{1, 2\}$, we get rational relations [23].

Thus, to define relations based on Parikh automata, we shall use the same notion of synchronization, but will run Parikh automata over them; words over positions 1 and 2, rather than ad hoc notions, will determine types of relations we get. Now we present this idea formally.

A. Synchronizations and regularity for Parikh relations

A *synchronization* of a tuple (w_1, \dots, w_k) of words in \mathbb{A}^* is a word over $\underline{k} \times \mathbb{A}$ so that the projection on \mathbb{A} of positions labeled i is exactly w_i , for $i = 1, \dots, k$. Every word w in $(\underline{k} \times \mathbb{A})^*$ is a synchronization of a uniquely determined k -tuple (w_1, \dots, w_k) , where w_i is the sequence of \mathbb{A} -letters corresponding to the symbol i in the first position of $\underline{k} \times \mathbb{A}$. We denote such tuple (w_1, w_2, \dots, w_k) by $\llbracket w \rrbracket_k$ and extend it to languages $S \subseteq (\underline{k} \times \mathbb{A})^*$ by $\llbracket S \rrbracket_k = \{\llbracket w \rrbracket_k \mid w \in S\}$.

For $L \subseteq \underline{k}^*$, we say that a word $u \otimes v \in (\underline{k} \times \mathbb{A})^*$ is *L-controlled* if $u \in L$; a language is *L-controlled* if all its words are. We now look at relations given by Parikh automata over *L-controlled* synchronizations for a regular language $L \subseteq \underline{k}^*$:

$$\text{REL}_{\text{PA}}(L) = \left\{ \llbracket S \rrbracket_k \mid \begin{array}{l} S \text{ is defined by a Parikh automaton} \\ \text{over } \underline{k} \times \mathbb{A} \text{ and is } L\text{-controlled} \end{array} \right\}.$$

For a set \mathcal{L} of regular languages we define $\text{REL}_{\text{PA}}(\mathcal{L}) = \bigcup \{\text{REL}_{\text{PA}}(L) \mid L \in \mathcal{L}\}$.

Let $\text{All} = \{\underline{k}^* \mid k \in \mathbb{N}_+\}$. Then $\text{REL}_{\text{PA}}(\text{All})$ contains all the relations that can be defined in our framework; these are analogs of rational relations, which are similarly defined when the automata are NFAs rather than PAs [23]. These include all regular relations on words as well as subsequence, subword, suffix, concatenation (e.g., $\{(u, v, w) \mid w = uv\}$) and even relations that can add counting constraints, e.g., $\{(u, v, w) \mid w = uv \text{ and } |u| = 2|v|\}$, using the power of PAs.

For NFA-based relations, the notion of regularity (in the binary case) is given by the synchronizing language $(12)^*(1^*|2^*)$, and regular length-preserving k -ary relations are given by $(12 \dots k)^*$ [23]. To extend this for relations based on Parikh automata, we go beyond the standard description of regular relations: those, for instance, include prefix but not suffix. We present a way of synchronizing relations that lets us use the suffix relation, among others, in graph queries, in a way that avoids undecidability issues of [7].

Let $\pi = (p_1, \dots, p_k)$ be a permutation of \underline{k} , and let $w_{\pi, i}$ stand for the word whose letters are p_i, \dots, p_k appearing in the increasing order (for instance, $w_{\pi, 1} = 12 \dots k$). Define

$$\vec{L}_{\pi} = w_{\pi, 1}^* \cdot w_{\pi, 2}^* \cdot \dots \cdot w_{\pi, k-1}^* \cdot w_{\pi, k}^*$$

and

$$\overleftarrow{L}_{\pi} = w_{\pi, k}^* \cdot w_{\pi, k-1}^* \cdot \dots \cdot w_{\pi, 2}^* \cdot w_{\pi, 1}^*$$

Let

$$\vec{L}_{\underline{k}} = \bigcup_{\pi} \vec{L}_{\pi} \quad \text{and} \quad \overleftarrow{L}_{\underline{k}} = \bigcup_{\pi} \overleftarrow{L}_{\pi}$$

where π ranges over permutations of \underline{k} . For example, $\vec{L}_{\underline{2}} = (12)^*(1^*|2^*)$ and $\overleftarrow{L}_{\underline{2}} = (1^*|2^*)(12)^*$, for $k = 3$ we have

$$\begin{aligned} \vec{L}_{\underline{3}} &= (123)^*((12)^*(1^*|2^*) \mid (13)^*(1^*|3^*) \mid (23)^*(2^*|3^*)) \\ \overleftarrow{L}_{\underline{3}} &= ((1^*|2^*)(12)^* \mid (1^*|3^*)(13)^* \mid (2^*|3^*)(23)^*)(123)^*. \end{aligned}$$

We define

$$\overrightarrow{\text{REG}}_{\text{PA}}^k = \text{REL}_{\text{PA}}(\vec{L}_{\underline{k}}) \quad \text{and} \quad \overleftarrow{\text{REG}}_{\text{PA}}^k = \text{REL}_{\text{PA}}(\overleftarrow{L}_{\underline{k}})$$

and finally

$$\overrightarrow{\text{REG}}_{\text{PA}} = \bigcup_{k>0} \overrightarrow{\text{REG}}_{\text{PA}}^k \quad \text{and} \quad \overleftarrow{\text{REG}}_{\text{PA}} = \bigcup_{k>0} \overleftarrow{\text{REG}}_{\text{PA}}^k.$$

Relations in these classes express many properties of interest and, in a way to be described shortly, approximate every rational relation. We first list some examples of relations in these classes.

- Every relation in REG is also in $\overrightarrow{\text{REG}}_{\text{PA}}$.
- Every *length-preserving* regular relation (i.e., a regular relation R such that $(w_1, \dots, w_k) \in R$ implies $|w_1| = \dots = |w_k|$) given by an NFA (or even a PA) is in both $\overrightarrow{\text{REG}}_{\text{PA}}^k$ and $\overleftarrow{\text{REG}}_{\text{PA}}^k$. In fact, this is true even if we impose the criterion that $-d \leq |w_i| - |w_j| \leq d$ for all $i, j \leq k$, for some fixed $d \in \mathbb{N}$. For instance, the equal-length and fixed-edit-distance relations are in both $\overrightarrow{\text{REG}}_{\text{PA}}^2$ and $\overleftarrow{\text{REG}}_{\text{PA}}^2$.
- Testing for Parikh image of a k -ary relation belonging to a semilinear set is regular. That is, for a semilinear set \mathbf{C} of dimension $(|\mathbb{A}| + 1)^k$, the relation $\{(w_1, \dots, w_k) \mid \Pi(w_1 \otimes \dots \otimes w_k) \in \mathbf{C}\}$ is in both $\overrightarrow{\text{REG}}_{\text{PA}}^k$ and $\overleftarrow{\text{REG}}_{\text{PA}}^k$.
- The prefix relation is in $\overrightarrow{\text{REG}}_{\text{PA}}^2$ but not in $\overleftarrow{\text{REG}}_{\text{PA}}^2$.
- The suffix relation \preceq_{suffix} is in $\overleftarrow{\text{REG}}_{\text{PA}}^2$ but not in $\overrightarrow{\text{REG}}_{\text{PA}}^2$.
- The subword relation \preceq_{subw} and the subsequence relation \preceq_{subseq} belong to neither $\overrightarrow{\text{REG}}_{\text{PA}}^2$ nor $\overleftarrow{\text{REG}}_{\text{PA}}^2$.

To show that they approximate arbitrary rational relations, we define $\text{shuffle}_k(w) \subseteq \mathbb{A}^*$ as the set of shufflings of $w \in \mathbb{A}^*$ preserving the number of subwords of length $\leq k$, that is, the set of all words $w' \in \mathbb{A}^*$ such that $\#_v(w) = \#_v(w')$ for each $v \in \mathbb{A}^*$ with $|v| \leq k$. Here $\#_v(w)$ counts the number of occurrences of v as a subword of w . For example, $\text{shuffle}_1(w)$ is the set of all words w' with $\Pi(w) = \Pi(w')$, and in general $w' \in \text{shuffle}_k(w)$ implies $\Pi(w) = \Pi(w')$.

For a relation $R \subseteq (\mathbb{A}^*)^m$, we define

$$\text{shuffle}_k(R) = \left\{ (w'_1, \dots, w'_m) \mid \begin{array}{l} (w_1, \dots, w_m) \in R \text{ and} \\ w'_i \in \text{shuffle}_k(w_i), i \in \underline{m} \end{array} \right\}$$

That is, we approximate a relation by only insisting on preserving the numbers of occurrences of words of fixed length, for instance replacing subsequence by checking whether $\Pi(w) \leq \Pi(w')$ in the simplest case of $k = 1$.

It turns out that every such approximation falls into both classes of analogs of regular relations.

Proposition V.1. *If R is an arbitrary m -ary rational relation, and $k \geq 1$, then $\text{shuffle}_k(R)$ is in both $\overrightarrow{\text{REG}}_{\text{PA}}^m$ and $\overleftarrow{\text{REG}}_{\text{PA}}^m$.*

There is a simple relationship between classes $\overrightarrow{\text{REG}}_{\text{PA}}$ and $\overleftarrow{\text{REG}}_{\text{PA}}$. For a tuple $\bar{w} = (w_1, \dots, w_k)$, let $\bar{w}^r = (w_1^r, \dots, w_k^r)$, where w^r is the reverse of w , and let $R^r = \{\bar{w}^r \mid \bar{w} \in R\}$. Then it is immediate from the definitions that $R \in \overrightarrow{\text{REG}}_{\text{PA}}$ iff $R^r \in \overleftarrow{\text{REG}}_{\text{PA}}$. Since $(R^r)^r = R$, this implies that $R \in \overleftarrow{\text{REG}}_{\text{PA}}$ iff $R^r \in \overrightarrow{\text{REG}}_{\text{PA}}$.

We need one closure property of these analogs of regular relations (that in particular gives us other examples of relations in these classes).

Proposition V.2. *For every $k > 0$, classes $\overrightarrow{\text{REG}}_{\text{PA}}^k$ and $\overleftarrow{\text{REG}}_{\text{PA}}^k$ are closed under intersection. In fact, for any two PA-definable $\overleftarrow{L}_{\mathbf{k}}$ - or $\overrightarrow{L}_{\mathbf{k}}$ -controlled languages $S_1, S_2 \subseteq (\mathbf{k} \times \mathbb{A})^*$ we have*

$$[S_1 \cap S_2]_k = [S_1]_k \cap [S_2]_k.$$

We show in fact a slightly more general closure result that works for every class $\text{REL}_{\text{PA}}(L)$ where L is *Parikh-injective*, i.e., the function $\Pi : L \rightarrow \mathbb{N}^k$ is injective. Languages $\overrightarrow{L}_{\mathbf{k}}$ and $\overleftarrow{L}_{\mathbf{k}}$ are easily seen to be such.

VI. PARIKH RELATIONS IN QUERIES

We now look at queries in which relations on paths can be used, i.e., queries similar to (1). If one uses NFAs to define such relations, then those synchronized by languages from $\overrightarrow{L}_{\mathbf{k}}$ (i.e., relations from the class REG) can be added to CRPQs with a small complexity cost [6]. Our motivation comes from the inability to extend good complexity results to relations that are often needed in applications, e.g., subword and subsequence [7], [8]. But such relations admit good approximations given by Parikh automata.

Indeed, as already discussed in the introduction, an approximation of the subsequence relation is

$$R_{\text{subseq}} = \{(u, v) \mid \forall a \in \mathbb{A} : \#_a(u) \leq \#_a(v)\}.$$

This relation is easily seen to be $\text{shuffle}_1(\preceq_{\text{subseq}})$, and thus it is in $\overrightarrow{\text{REG}}_{\text{PA}}^2$ (and in $\overleftarrow{\text{REG}}_{\text{PA}}^2$ as well), i.e., an analog of regular relations given by Parikh automata. For the subword relation, one can get an even more precise approximation than doing shuffling. Consider

$$R_{\text{subword}} = \{(u, v) \mid \exists i, j \forall a \in \mathbb{A} : \#_a(u) = \#_a(v[i, j])\}.$$

This again is a relation from $\text{REL}_{\text{PA}}(\overrightarrow{L}_{\mathbf{2}})$: one synchronously runs a PA over $u \otimes v$, guesses the positions i and j in v and increments counters only between those positions.

Thus, important rational relations that cannot be directly used in queries due to the complexity considerations, can be approximated by regular relations based on Parikh automata. We will show that those can be evaluated with good data and combined complexity.

A. CRPQs with relations

Let \mathcal{R} be a class of relations on words. We assume that \mathcal{R} has relations of every arity $k \geq 1$ (in particular, languages as relations of arity 1); this could be a class of regular, or rational relations, or a class of the form $\text{REL}_{\text{PA}}(\mathcal{L})$, e.g., $\overrightarrow{\text{REG}}_{\text{PA}}$ or $\overleftarrow{\text{REG}}_{\text{PA}}$. The class $\text{CRPQ}_{\text{rel}}(\mathcal{R})$ of queries extends CRPQs with relations from \mathcal{R} . Formally, such a query is of the form

$$\varphi(\bar{z}) :- x_1 \xrightarrow{\pi_1:L_1} y_1, \dots, x_l \xrightarrow{\pi_l:L_l} y_l, R_1(\bar{\chi}_1), \dots, R_m(\bar{\chi}_m),$$

where

- π_1, \dots, π_l are the *path variables* of φ ; they are pairwise distinct;
- each L_i is a language (unary relation) from \mathcal{R} ;
- \bar{z} is a tuple of variables among x_i, y_i s;
- each $\bar{\chi}_j$ is a tuple of variables among the π_i s of the same arity as R_j .

An example is the query (1) from the introduction that uses a single ‘equal-length’ regular relation.

The semantics is a natural extension of the semantics of CRPQs. Given a graph G and a tuple \bar{a} of its nodes of the same arity as \bar{z} , the formula $\varphi(\bar{a})$ is true if there exists a tuple \bar{b} of nodes interpreting the variables used in the body but not in the head so that:

- for each $i \leq l$, there exists a path p_i between the interpretations u_i, v_i of x_i, y_i such that its label $\lambda(p_i)$ is in L_i ;
- for each $\bar{\chi}_j = (\pi_{j_1}, \dots, \pi_{j_s})$, the tuple of labels of the paths $(\lambda(p_{j_1}), \dots, \lambda(p_{j_s}))$ is in R_j .

The set of all tuples \bar{a} such that $\varphi(\bar{a})$ holds in G is denoted by $\varphi(G)$.

The class of CRPQs extended with regular relations, called ECRPQs in [6], is $\text{CRPQ}_{\text{rel}}(\text{REG})$ in this notation. The extension with all rational relations was considered in both [6] and [7], and, due to its undecidability, less expressive languages $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{suff}})$, $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{subw}})$, and $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{subseq}})$ have been studied, i.e., those with arbitrary regular relations and just one particular rational relation. Those turned out to have horrendous complexity as well, so [8] restricted them even further, replacing regular relations with regular languages and keeping just one rational relation (we indicate this by omitting REG in the list of relations, e.g., $\text{CRPQ}_{\text{rel}}(\preceq_{\text{subseq}})$), but of course paths can still be checked for membership in a regular language).

The table below provides a summary of known complexity results from [6], [7], [8]. When a particular complexity class is mentioned, the problem is complete for it. Non-PR stands for non-primitive-recursive.

| language | data complexity | combined complexity |
|--|-----------------|---------------------|
| $\text{CRPQ}_{\text{rel}}(\text{REG})$ | NLOGSPACE | PSPACE |
| $\text{CRPQ}_{\text{rel}}(\text{RAT})$ | undecidable | undecidable |
| $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{suff}})$ | undecidable | undecidable |
| $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{subw}})$ | undecidable | undecidable |
| $\text{CRPQ}_{\text{rel}}(\text{REG} + \preceq_{\text{subseq}})$ | nonelementary | non-PR |
| $\text{CRPQ}_{\text{rel}}(\preceq_{\text{suff}})$ | NLOGSPACE | PSPACE |
| $\text{CRPQ}_{\text{rel}}(\preceq_{\text{subw}})$ | PSPACE | PSPACE |
| $\text{CRPQ}_{\text{rel}}(\preceq_{\text{subseq}})$ | PSPACE | NEXPTIME |

Thus, from the complexity point of view, only two languages, $\text{CRPQ}_{\text{rel}}(\text{REG})$ and $\text{CRPQ}_{\text{rel}}(\preceq_{\text{suff}})$, can be viewed as potentially usable. However, they both put severe restrictions on the use of relations in queries: the former completely rules out rational but non-regular relations, and the latter only allows one particular relation, at the expense of excluding all other relations, even regular ones.

We shall now see how to generalize and extend these positive results.

B. Regularity and query evaluation

To achieve our goal and significantly extend the only positive complexity results, we deal with languages $\text{CRPQ}_{\text{rel}}(\overrightarrow{\text{REG}}_{\text{PA}})$ and $\text{CRPQ}_{\text{rel}}(\overleftarrow{\text{REG}}_{\text{PA}})$. That is, we

- extend regular languages to languages defined by PAs; and
- use relations from $\overrightarrow{\text{REG}}_{\text{PA}}$ and $\overleftarrow{\text{REG}}_{\text{PA}}$ in queries.

These languages are significant extensions of previously known decidable languages of path-based queries on graphs (we shall give some examples of their power shortly). As our main result we show that despite adding much to the expressiveness, these extensions do not cost us computationally.

Theorem VI.1. *For queries in $\text{CRPQ}_{\text{rel}}(\overrightarrow{\text{REG}}_{\text{PA}})$ and $\text{CRPQ}_{\text{rel}}(\overleftarrow{\text{REG}}_{\text{PA}})$, data complexity is in NLOGSPACE, and combined complexity is in PSPACE.*

Of course we also have NLOGSPACE- and PSPACE-completeness, as hardness was already known for small fragments of these languages [6], [8], so this result is indeed the best possible.

We now give examples of the power of these languages, before providing a sketch of the proof of the theorem.

Queries in $\text{CRPQ}_{\text{rel}}(\overrightarrow{\text{REG}}_{\text{PA}})$. This language is a natural extension of $\text{CRPQ}_{\text{rel}}(\text{REG})$ to Parikh automata: instead of relations given by synchronized NFAs, we now can use relations given by synchronized Parikh automata. In particular, in this language we can use counting approximations R_{subseq} of \preceq_{subseq} and R_{subword} of \preceq_{subw} , even though the relations themselves cannot be added to $\text{CRPQ}_{\text{rel}}(\text{REG})$ due to complexity considerations. In terms of the results of such approximations, we have the following observation, due to the monotonicity of CRPQs.

Lemma VI.2. *Given two queries*

$$\varphi(\bar{z}) := x_1 \xrightarrow{\pi_1:L_1} y_1, \dots, x_l \xrightarrow{\pi_l:L_l} y_l, R_1(\bar{\chi}_1), \dots, R_m(\bar{\chi}_m),$$

$$\varphi'(\bar{z}) := x_1 \xrightarrow{\pi_1:L_1} y_1, \dots, x_l \xrightarrow{\pi_l:L_l} y_l, R'_1(\bar{\chi}_1), \dots, R'_m(\bar{\chi}_m),$$

such that $R_j \subseteq R'_j$ for all $j = 1, \dots, m$, we have $\varphi(G) \subseteq \varphi'(G)$ for every graph G .

Hence, replacing relations with their counting approximations results in overapproximation of query results, which was exactly our intention. We now give a couple of examples motivated by a common application of graph databases, namely social networks and in particular crime detection [22]. Consider a simple version of a query that attempts to detect criminal activity:

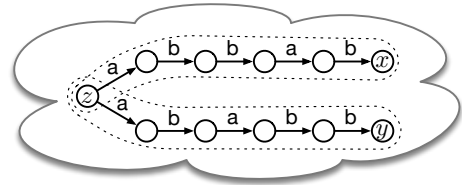
$$\varphi(x, y) := x \xrightarrow{\pi:L} z, x \xrightarrow{\pi':L'} y, \pi \preceq_{\text{subseq}} \pi'$$

Here L describes known criminal activity in a network of people, and L' gives paths to be analyzed. We want to find ys that are connected to x in a way resembling known illicit activity involving x . Due to the nature of those activities in the network, not all of them may be known yet, hence we are looking for subsequences of known activities as an indication that the connection of x and y is suspect and needs to be investigated further. This query φ is too complex to evaluate, due to the presence of \preceq_{subseq} , but its replacement

$$\varphi'(x, y) := x \xrightarrow{\pi:L} z, x \xrightarrow{\pi':L'} y, R_{\text{subseq}}(\pi, \pi')$$

from the class $\text{CRPQ}_{\text{rel}}(\overrightarrow{\text{REG}}_{\text{PA}}^2)$ will be much more efficient, and will overapproximate the answer to φ , i.e., no objects of interest will be lost.

As another example, consider comparing paths for similarity; these queries are common in both social networks [22] and Semantic Web applications [4]. Some measures, e.g., fixed edit distance, are regular, but some are not. Consider the following one: two words u and v are k -similar if $u \in \text{shuffle}_k(v)$, i.e., each word w of length $\leq k$ occurs the same number of times in u and in v . The picture below has an example of two 2-similar paths starting at the same node z and realizing the same local patterns up to length 2 over the alphabet $\{a, b\}$:



Note that k -similarity is not a regular relation, nor can it be described by Parikh images if $k \geq 2$, but this is a relation in $\overrightarrow{\text{REG}}_{\text{PA}}$. Hence, we can look, for instance, for nodes z from which k -similar paths to two distinct nodes exist by a $\text{CRPQ}_{\text{rel}}(\overrightarrow{\text{REG}}_{\text{PA}})$ query $\varphi(z) := z \xrightarrow{\pi} x, z \xrightarrow{\pi'} y, \pi \sim_k \pi', \pi \neq \pi'$, where \sim_k is the k -similarity relation. It achieves the same complexity as $\text{CRPQ}_{\text{rel}}(\text{REG})$ queries, even though it uses relations that go beyond what has so far been possible to use with reasonable complexity.

Queries in $\text{CRPQ}_{\text{rel}}(\overline{\text{REG}}_{\text{PA}})$. No decidable language that allows the use of \preceq_{suff} and other relations has been known yet: decidability was only shown for $\text{CRPQ}_{\text{rel}}(\preceq_{\text{suff}})$, i.e., extension of CRPQs with suffix and nothing else. The language $\text{CRPQ}_{\text{rel}}(\overline{\text{REG}}_{\text{PA}})$ can use suffix with other regular relations, definable by NFAs and PAs, as long as they are synchronized by $\overline{L}_{\mathbf{k}}$. For instance, we can use suffix, equal-length, bounded edit-distance, and properties of Parikh images in a query, all at the same time, without incurring any complexity penalty.

Proof sketch of Theorem VI.1. Suppose we have a formula of $\text{CRPQ}_{\text{rel}}(\overline{\text{REG}}_{\text{PA}})$ (the case for $\overline{\text{REG}}_{\text{PA}}$ being analogous) of the form

$$\varphi(\bar{z}) :- x_1 \xrightarrow{\pi_1} y_1, \dots, x_l \xrightarrow{\pi_l} y_l, R_1(\bar{\chi}_1), \dots, R_m(\bar{\chi}_m),$$

where without loss of generality we assume that languages witnessing all the π_i s are \mathbb{A}^* (since any regular condition $\lambda(\pi_i) \in L_i$ can be checked in the second part of the query by $L_i(\pi_i)$).

We first show the combined complexity case. Let $G = \langle V, E \rangle$ be a labeled graph over the alphabet \mathbb{A} . Let \bar{v} be a tuple of vertices of the arity $|\bar{z}|$. We want to check if $\bar{v} \in \varphi(G)$. We first guess a valuation $\nu : \{x_i, y_i\}_{i \leq l} \rightarrow V$ so that $\nu(\bar{z}[i]) = \bar{v}[i]$ for every $i \leq l$.

Let k_j be the arity of the relation R_j , and let $t = k_1 + \dots + k_m$. Next, we produce the following relations over $(\mathbb{A}^*)^t$:

- the relation R_{\times} containing all tuples $(\bar{w}_1, \dots, \bar{w}_m)$ so that $\bar{w}_i \in R_i$ for all $i \in \mathbf{m}$,
- for every π that appears both in position i of R_j and i' of $R_{j'}$ a relation $R_{i,j,i',j'}^{\times}$ of tuples

$$(w_{1,1}, \dots, w_{1,k_1}, \dots, w_{m,1}, \dots, w_{m,k_m}) \quad (5)$$

so that $w_{j,i} = w_{j',i'}$,

- for every RPQ $x \xrightarrow{\pi} y$ used in φ so that π appears in position i of R_j a relation $S_{i,j}^{G,\nu}$ of tuples like (5) so that $w_{j,i}$ is accepted in the automaton represented by G having $\nu(x)$, $\nu(y)$ as initial and final states respectively.

Note that the intersection of all these relations is nonempty iff $\bar{v} \in \varphi(G)$.

It is easy to see that all the aforementioned relations are $\overline{L}_{\mathbf{k}}$ -controlled since each R_j is $\overline{L}_{\mathbf{k}_j}$ -controlled. Further, it is easy to see that for every one of these relations one can build a PA synchronizing it in polynomial time, given PAs synchronizing the R_j 's of φ . By Proposition V.2, the intersection of all these relations is empty iff the intersection of all the PA synchronizing them is. We can test the emptiness of the latter intersection in PSPACE due to Proposition III.3. Even though we performed an initial polynomial guessing, by Savitch's Theorem the procedure remains PSPACE.

Let us now analyze the case where the query φ is fixed. Note that the relations $S_{i,j}^{G,\nu}$ s can be synchronized using NFAs instead of PAs. (In other words, we do not need to make use of the semilinear constraint set.) Let then \mathcal{A}_2 be the NFA synchronizing $\bigcap S_{i,j}^{G,\nu}$. Since the number of relations $S_{i,j}^{G,\nu}$ s

is bounded by a function on φ , we have that \mathcal{A}_2 can be built so that it is polynomial in the NFA synchronizing the $S_{i,j}^{G,\nu}$ s, which in turn are polynomial in G and \bar{v} . Furthermore, its transitions can be produced on-the-fly using logarithmic space. Let $(\mathcal{A}_1, \mathbf{C}_1)$ be the PA synchronizing the remaining relations $R_{\times} \cap \bigcap R_{i,j,i',j'}^{\times}$. Since these depend only on the query, this is a fixed PA. Using Proposition III.4, it follows that we can check whether $\mathcal{L}(\mathcal{A}_2) \cap \mathcal{L}(\mathcal{A}_1, \mathbf{C}_1)$ is empty in NLOGSPACE, and thus check whether $(\bar{v}, \bar{\rho}) \in \varphi(G)$. \square

C. Syntactic restrictions

We now look at query answering in the language that imposes no restrictions on the type of relations that can be used, i.e., it uses arbitrary relations from $\text{REL}_{\text{PA}}(\text{All})$. Recall that these are relations in which no restrictions whatsoever are imposed on allowed synchronizations. These include suffix, subword, subsequence, and many properties definable by Parikh automata, e.g., pairs (u, v) so that $v = wu$ (u is a suffix of v), and $\#_a(u) = \#_a(w)$ for every $a \in \mathbb{A}$.

Of course we know that in general query evaluation in $\text{CRPQ}_{\text{rel}}(\text{REL}_{\text{PA}}(\text{All}))$ is undecidable; for that, we do not even need Parikh automata [7]. To recover decidability, or, better yet, achieve good complexity, it is important to impose syntactic restrictions on the language. We now do it in the way that generalizes all previously known decidable restrictions for graph queries with relations on paths.

Consider again a query

$$\varphi(\bar{z}) :- x_1 \xrightarrow{\pi_1:L_1} y_1, \dots, x_m \xrightarrow{\pi_m:L_m} y_m, R_1(\bar{\chi}_1), \dots, R_l(\bar{\chi}_l),$$

Let $R(\bar{\chi})$ be an atom used in this query, where $\bar{\chi} = (\pi_{i_1}, \dots, \pi_{i_n})$. Define $G_{R(\bar{\chi})}$ as an undirected graph whose vertices are $\pi_{i_1}, \dots, \pi_{i_n}$, and edges are $\{\pi_{i_1}, \pi_{i_2}\}, \{\pi_{i_2}, \pi_{i_3}\}, \dots, \{\pi_{i_{n-1}}, \pi_{i_n}\}$. If $n = 1$, there are no edges. We let G_{φ} be the undirected multi-graph consisting in the union of all the $G_{R_j(\bar{\chi}_j)}$ s.

We say that φ :

- is *strongly-acyclic* if G_{φ} is acyclic (note that if there is more than one edge between two vertices, G_{φ} is *not* acyclic; the “strongly-” prefix is to distinguish it from other notions of acyclicity [6]);
- has *join-size* n if the size of each connected component of G_{φ} is at most n ;
- has *dimension* n if every relation R_i is given by a PA of dimension at most n .

Theorem VI.3. (I) For strongly acyclic queries in $\text{CRPQ}(\text{REL}_{\text{PA}}(\text{All}))$, data complexity is in NLOGSPACE and combined complexity is in PSPACE;

(II) if the query is in addition of a fixed join-size n , the combined complexity is in NP;

(III) if the query is furthermore of fixed dimension m , and all numbers in the definitions of PAs are given in unary, the combined complexity is in PTIME.

The proof follows the construction of the proof of Theorem VI.1, using Proposition III.2 for the last two items, and a

lemma below on the complexity of join evaluations. If we have relations $R_1 \subseteq (\mathbb{A}^*)^{k_1}$ and $R_2 \subseteq (\mathbb{A}^*)^{k_2}$, and numbers $i_1 \leq k_1$ and $i_2 \leq k_2$, the join $R_1 \bowtie_{i_1, i_2} R_2 \subseteq (\mathbb{A}^*)^{k_1+k_2}$ consists of the tuples (\bar{u}_1, \bar{u}_2) such that $\bar{u}_1 \in R_1, \bar{u}_2 \in R_2$, and $\bar{u}_1[i_1] = \bar{u}_2[i_2]$.

Lemma VI.4. *Given two Parikh automata \mathcal{PA}_1 and \mathcal{PA}_2 with constraint sets \mathbf{C}_1 and \mathbf{C}_2 that synchronize $R_1 \subseteq (\mathbb{A}^*)^{k_1}$ and $R_2 \subseteq (\mathbb{A}^*)^{k_2}$, respectively, one can build in polynomial time a Parikh automaton \mathcal{PA} synchronizing $R_1 \bowtie_{i_1, i_2} R_2$ such that its constraint set is $\mathbf{C}_1 \times \mathbf{C}_2$ and the size of the auxiliary set of \mathcal{PA} is linear in those of \mathcal{PA}_1 and \mathcal{PA}_2 .*

These result go beyond what is known even for regular relations; for instance, part (I) of Theorem VI.3 extends a result about acyclic queries for recognizable relations and one given binary rational relation from [7].

VII. REGULAR EXPRESSIONS FOR PARIKH AUTOMATA

So far we defined queries in languages $\text{CRPQ}_{\text{rel}}(\mathcal{R})$ by presenting languages and relations with automata, but it is highly unlikely that a user of a graph query language will be doing the same. In practice such a user is likely to use a declarative formalism, probably some type of regular expressions. For NFAs, we know very well what regular expressions are; in fact they found their way into UNIX commands (`grep` etc.) and programming languages such as PERL, PYTHON, AWK.

If we want to make Parikh automata and relations based on them usable, we need to provide an easy way to specify languages definable by them. We do this now, by presenting two candidate descriptions of languages given by Parikh automata. One of them is based on the idea of marking positions in a regular expression, and it captures the power of PAs. The other works well for some cases when marked expressions become too complicated; it is based on choosing expressions matching initial segments of words.

A. Expressions with marking

To explain the idea of these expressions, consider first the language $L_{a^n b^n}$. It can also be defined as the language given by $a^* b^*$ with the additional constraint that $\#_a(w) = \#_b(w)$. We can thus use an expression $(a^* b^*, a = b)$ to define it: the first component is a usual regular expression, and the second is a constraint on the numbers of occurrences. For instance, $(a^* b^* c^*, a + 2b > c)$ defines words $w \in \mathcal{L}(a^* b^* c^*)$ with $\#_a(w) + 2 \cdot \#_b(w) > \#_c(w)$.

Such expressions are sufficient to define languages given by NFAs together with a semilinear constraint on Parikh images, but as we saw, Parikh automata go further than that. Consider, for instance, the language $L_0 = \{ba^n ca^n \mid n \geq 0\}$. Simply counting *as* does not help us any more. What we can do instead is use a technique similar to the one that defines unambiguous regular languages [10]: namely, we *mark* positions in the regular expression. We use the expression $ba_1^* ca_2^*$, with a_1 and a_2 referring to different occurrences of the same symbol a , but now we can refer to their individual counts. To define L_0 , we shall now use $(ba_1^* ca_2^*, a_1 = a_2)$: it still defines words from

$ba^* ca^*$ but now with a semilinear constraint that the number of a_1 s (i.e., *as* after the b) is the same as the number of a_2 s (i.e., *as* after the c).

A marked expression therefore is an expression in which each symbol is marked with a natural number (we can view unmarked symbols b, c above as marked with 0); that is, an expression over the alphabet $\mathbb{A} \times \mathbb{D}$, where \mathbb{D} is a finite subset of \mathbb{N} . For a word w from $(\mathbb{A} \times \mathbb{D})^*$, its \mathbb{A} -projection $\pi_{\mathbb{A}}(w)$ simply drops the second component of each letter.

Now a *marked Parikh expression* over \mathbb{A} is a pair (e, \mathbf{C}) , where e is a regular expression over $\mathbb{A} \times \mathbb{D}$ for a finite $\mathbb{D} \subset \mathbb{N}$, and \mathbf{C} is a semilinear set in $\mathbb{N}^{|\mathbb{A}| \cdot |\mathbb{D}|}$. It defines the language

$$\mathcal{L}(e, \mathbf{C}) = \{\pi_{\mathbb{A}}(w) \mid w \in \mathcal{L}(e) \text{ and } \Pi(w) \in \mathbf{C}\}.$$

From Proposition III.1, part 3, we obtain:

Corollary VII.1. *The classes of languages defined by Parikh automata and by marked Parikh expressions are the same.*

While in the examples above these expressions are very simple and intuitive, sometimes they can grow to be a bit hard to parse, even for simple languages. Coming back to the language $L_{ba=ca}$ (the number of *as* that occurring after a b and after a c is the same), the expression to describe it is

$$\left((ba_1|\varepsilon) \left((\mathbb{A} - \{b, c\})^* (b(\mathbb{A} - \{a\})|c(\mathbb{A} - \{a\})|\varepsilon) \right)^* (ca_2|\varepsilon) \right)^*$$

together with the constraint $a_1 = a_2$. (The sublanguage in the middle, under the inner Kleene star, denotes the set of words in which neither ba nor ca occurs.) We now propose alternative expressions that, while slightly weaker in terms of their power, work better for languages like $L_{ba=ca}$.

B. Expressions with choice

For languages like $L_{ba=ca}$, it is better to use expressions with choice. We define such a language with the expression $\{(\mathbb{A}^* ba, \binom{0}{1}), (\mathbb{A}^* ca, \binom{1}{0})\}$, together with constraint $x_1 = x_2$. This works as follows. The use of pairs of numbers next to expressions indicates the use of two counters, say x_1 and x_2 . Suppose we are given a word $w = a_1 \cdots a_n$. When a prefix of it matches the first expression, $\mathbb{A}^* ba$, we add 0 and 1 to the counters, and when it matches the second expression, $\mathbb{A}^* ca$, we add 1 and 0 to the counters. So having $x_1 = x_2$ after reading the whole word is equivalent to having equal numbers of *as* following a b and a c .

Now we define these expressions formally. A *Parikh expression with choice* is an expression of the form (E, \mathbf{C}) , where $E = \{(e_1, \bar{v}_1), \dots, (e_m, \bar{v}_m)\}$, each e_i is a regular expression over \mathbb{A} , each \bar{v}_i is in \mathbb{N}^k , and \mathbf{C} is a semilinear set in \mathbb{N}^k .

To define the language (over \mathbb{A}) given by E , let e_0 define the complement of all the e_j s (i.e., its language is the complement of $\bigcup_i \mathcal{L}(e_i)$) and let $v_0 = \bar{0}$. Then a word $w = a_1 \cdots a_n$ is in $\mathcal{L}(E, \mathbf{C})$ iff for every position $i \leq n$, there is $j_i \in [0, m]$ such that:

- $a_1 \cdots a_i \in \mathcal{L}(e_{j_i})$, and
- $\sum_{i \leq n} \bar{v}_{j_i} \in \mathbf{C}$.

Proposition VII.2. *Languages given by Parikh expressions with choice are definable by Parikh automata.*

PAs resulting from translating Parikh expressions with choice have an additional property: whenever (q, a, \bar{v}, q') and (q, a, \bar{u}, q'') are transitions, then $q' = q''$. But despite being slightly weaker, this class of expressions covers many examples of interest, and appears to be intuitive enough to be used in query languages. Expressions with marking also are quite intuitive for many languages, and they do capture PAs.

VIII. CONCLUSIONS

We showed that with the use of Parikh automata we avoided the curse of rational relations in graph database queries, at the expense of approximations, of course. Such approximate solutions can be found effectively, matching bounds for best graph queries and traditional query languages for relational databases, in terms of combined complexity. The formalism of PAs, in addition, directly expresses many properties of interest in graph querying, and PAs come with at least two alternative ways of specifying languages that can be used in querying, similar to the way usual regular expressions are used.

Several questions remain. Among technical questions, there is one regarding the use of path variables in queries, as was done in [6] for $\text{CRPQ}_{\text{rel}}(\text{REG})$. One needs to find effective PA representations of sets of paths returned by a query, and understand their complexity. We also would like to show that expressions with choice are provably weaker than expressions with marking (the notion of determinism that is known to be weaker for PAs is actually more restrictive than the automata such expressions translate into).

Regarding directions for future work, we would like to look at obtaining underapproximations, as opposed to overapproximations, of query answers (say, for queries involving subword, subsequence, and similar relations). Is there a model sharing good computational properties with PAs that can be used for that purpose? Another has to do with extending the formalism beyond navigational properties, i.e., to querying the topology of the graph (say, connections in a social network) and data stored in it (e.g., data about people in the social network). Extensions of RPQs that handle data have been proposed [35]; they treat paths as data words and use automata formalisms for those [9]. One possible approach is then to enhance such automata with features of PAs.

Acknowledgment We would like to thank Anthony Widjaja Lin and Eryk Kopczyński for helpful discussions. Diego Figueira is partially supported by ANR StochMC, Project Blanc ANR-13-BS02-0011-01.

REFERENCES

- [1] S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufman, 1999.
- [2] R. Alur, L. D'Antoni, J. Deshmukh, M. Raghothaman, and Y. Yuan. Regular functions and cost register automata. In *LICS'13*.
- [3] R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Comput. Surv.* 40(1): (2008).
- [4] K. Anyanwu, A. P. Sheth. ρ -Queries: enabling querying for semantic associations on the semantic web. In *WWW'03*.

- [5] P. Barceló. Querying graph databases. In *PODS'13*, 175-188.
- [6] P. Barceló, L. Libkin, A. W. Lin, P. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst.* 37(4): 31 (2012).
- [7] P. Barceló, D. Figueira, L. Libkin. Graph logics with rational relations. *Logical Methods in Computer Science* 9(3) (2013).
- [8] P. Barceló, P. Muñoz. Graph logics with rational relations: the role of word combinatorics. In *CSL-LICS 2014*.
- [9] M. Bojańczyk, C. David, A. Muscholl, T. Schwentick, L. Segoufin. Two-variable logic on data words. *ACM TOCL* 12(4): 27 (2011).
- [10] A. Brüggemann-Klein, D. Wood. One-unambiguous regular languages. *Information and Computation* 140 (1998), 229-253.
- [11] M. Cadilhac, A. Finkel, and P. McKenzie. Affine Parikh automata. *RAIRO Theoretical Informatics and Applications*, 46(4): 511-545, 2012.
- [12] M. Cadilhac, A. Krebs, P. McKenzie. The algebraic theory of Parikh automata. In *CAI 2013*, pages 60-73.
- [13] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. View-based query processing and constraint satisfaction. In *LICS'00*, pages 361-371.
- [14] D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443-465, 2002.
- [15] L. Cardelli, P. Gardner, G. Ghelli. A spatial logic for querying graphs. In *ICALP'02*, pages 597-610.
- [16] C. Choffrut. Relations over words and logic: a chronology. *Bulletin of the EATCS* 89 (2006), 159-163.
- [17] M. P. Consens, A. O. Mendelzon. GraphLog: a visual formalism for real life recursion. In *PODS'90*, pages 404-416.
- [18] I. Cruz, A. Mendelzon, P. Wood. A graphical query language supporting recursion. In *SIGMOD'87*, pages 323-330.
- [19] Z. Dang, O. H. Ibarra, T. Bultan, R. Kemmerer, J. Su. Binary reachability analysis of discrete pushdown timed automata. In *CAV'00*, pages 69-84.
- [20] A. Dawar, P. Gardner, G. Ghelli. Expressiveness and complexity of graph logic. *Inf. & Comput.* 205 (2007), 263-310.
- [21] J. Esparza, P. Ganty. Complexity of pattern-based verification for multithreaded programs. In *POPL 2011*, pages 499-510.
- [22] W. Fan. Graph pattern matching revised for social network analysis. In *ICDT 2012*, pages 8-21.
- [23] D. Figueira and L. Libkin. Synchronizing relations on words. In *STACS 2014*, pages 93-104.
- [24] D. Florescu, A. Levy, D. Suciu. Query containment for conjunctive queries with regular expressions. In *PODS'98*.
- [25] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2): 285-296, 1966.
- [26] M. Hague, A. W. Lin. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In *CAV 2012*, pages 260-276.
- [27] J. Hellings, B. Kuijpers, J. Van den Busche, X. Zhang. Walk logic as a framework for path query languages on graph databases. In *ICDT 2013*, pages 117-128.
- [28] O. H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25(1): 116-133, 1978.
- [29] W. Karianto. Parikh automata with pushdown stack. Master's thesis, RWTH Aachen, 2004.
- [30] F. Klaedtke and H. Rueß. Monadic second-order logics with cardinalities. In *ICALP 2003*, pages 681-696.
- [31] B. König, J. Esparza. Verification of graph transformation systems with context-free specifications. *ICGT 2010*, pages 107-122.
- [32] E. Kopczyński and A. W. Lin. Parikh images of grammars: Complexity and applications. In *LICS 2010*, pages 80-89.
- [33] D. Kozen. Lower bounds for natural proof systems. In *FOCS 1977*, pages 254-266.
- [34] L. Libkin. Logics for unranked trees: an overview. *LMCS* 2(3): (2006).
- [35] L. Libkin, D. Vrgoč. Regular path queries on graphs with data. In *ICDT'12*, pages 74-85.
- [36] A. W. Lin. *Model Checking Infinite-State Systems: Generic and Specific Approaches*. PhD thesis, U. Edinburgh, 2010.
- [37] M. Marx. Conditional XPath. *ACM TODS* 30(4): 929-959 (2005).
- [38] R. Parikh. On context-free languages. *J. ACM*, 13(4): 570-581, 1966.
- [39] I. Robinson, J. Webber, E. Eifrem. *Graph Databases*. O'Reilly, 2013.
- [40] J. Sakarovitch. *Elements of Automata Theory*. CUP, 2009.
- [41] B. Scarpellini. Complexity of subcases of Presburger arithmetic. *Trans. AMS*, 284(1): 203-218, 1984.
- [42] K. N. Verma, H. Seidl, and T. Schwentick. On the complexity of equational Horn clauses. In *CADE 2005*, pages 337-352.
- [43] P. Wood. Query languages for graph databases. *Sigmod Record*, 41(1):50-60, 2012.

APPENDIX

NOTATION

For all our proofs we adopt the following presentation of Parikh automata given by Karianto [29], which, as stated in Proposition III.1, is equivalent to the definition given before. A *Parikh Finite Automaton* (PA) of dimension $k \geq 1$ over an alphabet \mathbb{A} is a system $(\mathcal{A}, \mathbf{C})$ where $\mathcal{A} = (Q, \mathbb{A} \times \mathbb{D}, \delta, q_0, F)$ is a finite automaton over $\mathbb{A} \times \mathbb{D}$, for some finite, nonempty set $\mathbb{D} \subseteq \mathbb{N}^k$ (called the *auxiliary set*), and $\mathbf{C} \subseteq \mathbb{N}^k$ is a semi-linear set (called the *constraint set*). The language recognized by $(\mathcal{A}, \mathbf{C})$ is defined as

$$\mathcal{L}(\mathcal{A}, \mathbf{C}) \stackrel{\text{def}}{=} \{w \mid w \otimes v \in \mathcal{L}(\mathcal{A}), \oplus v \in \mathbf{C}\}.$$

PROOF OF LEMMA II.1

Proof: Note that for $q \in \mathbb{N}$, we have that $2^q \cdot x$ can be simply represented as

$$f_2^q(x) = \underbrace{f_2(\cdots f_2(x))}_q$$

and thus for every $m \in \mathbb{N}$ we have that $m \cdot x$ can be represented with a term

$$t_m(x) = f_2^{q_1}(x) + \cdots + f_2^{q_r}(x)$$

of Presburger arithmetic of size $O(\log(m))$, where $m = \sum_i 2^{q_i}$. Further, this term can be built in linear time in the binary representation of m . From this observation, it follows that for any linear set represented in binary

$$\{\bar{v}_0 + \alpha_1 \bar{v}_1 + \cdots + \alpha_n \bar{v}_n \mid \alpha_1, \dots, \alpha_n \in \mathbb{N}\} \subseteq \mathbb{N}^k$$

one can build an equivalent Presburger formula $\varphi(x_1, \dots, x_k)$ in linear time:

$$\exists \alpha_1, \dots, \alpha_n \cdot \bigwedge_{i \in \mathbf{k}} (x_i = t_{v_0[i]}(1) + \sum_{j \in \mathbf{n}} t_{v_j[i]}(\alpha_j)).$$

Hence, by a disjunction of such formulae, one can also compute, in linear time, Presburger arithmetic representations of semilinear sets. \blacksquare

PROOF OF PROPOSITION III.1

Proof: 1) \Rightarrow 2) Consider a PA of dimension k given as a pair $(\mathcal{A}, \mathbf{C})$ with $\mathcal{A} = (Q, \mathbb{A} \times \mathbb{D}, \delta, q_0, F)$, where $\mathbb{D}, \mathbf{C} \subseteq \mathbb{N}^k$. We show how to build an automaton \mathcal{A}' over \mathbb{A} with statespace Q' and a constraint set \mathbf{C}' verifying 2). We define the set of states as $Q' = Q \times (\{\bar{0}\} \cup \mathbb{D})$, with an initial state $(q_0, \bar{0})$ and set of final states $F \times (\{\bar{0}\} \cup \mathbb{D})$. The idea is that we internalize the second components of the letters read by \mathcal{A} into the states of \mathcal{A}' , so that whenever \mathcal{A} reads (a, \bar{x}) reaching a state q , we have that \mathcal{A}' reads a reaching a state (q, \bar{x}) . Thus, the set of transitions of \mathcal{A}' consists of all $((q, \bar{x}), a, (q', \bar{x}'))$ so that $(q, (a, \bar{x}'), q') \in \delta$. Let us fix an order $Q' = \{p_1 < \cdots < p_m\}$ for the $m = |Q'|$ states of \mathcal{A}' . We write $\bar{p}_i \in \mathbb{D}$ to denote the vector contained in the second component of the state p_i . We define $\mathbf{C}' \subseteq \mathbb{N}^m$ as

$$\mathbf{C}' = \{\bar{v} \mid \bar{v} \cdot (\bar{p}_1, \dots, \bar{p}_m)^t \in \mathbf{C}\},$$

where $(\bar{p}_1, \dots, \bar{p}_m)^t$ is the transpose of $(\bar{p}_1, \dots, \bar{p}_m)$. The fact that \mathbf{C}' is semilinear can be readily seen when working with Presburger arithmetic formulae. That is, if we have a formula $\varphi(x_1, \dots, x_k)$ describing \mathbf{C} we have that the formula

$$\varphi'(y_1, \dots, y_m) = \exists x_1, \dots, x_k \cdot \bigwedge_{i \in \mathbf{k}} (x_i = \sum_{j \in \mathbf{m}} \bar{p}_j[i] \cdot y_j) \wedge \varphi(x_1, \dots, x_k)$$

describes \mathbf{C}' .

Now we show $w \in \mathcal{L}(\mathcal{A}, \mathbf{C})$ iff there is an accepting run ρ of \mathcal{A}' on w so that $\Pi(\rho) \in \mathbf{C}'$. From left to right, if $w \in \mathcal{L}(\mathcal{A}, \mathbf{C})$ then there is a word $v \in \mathbb{D}^*$ so that $w \otimes v \in \mathcal{L}(\mathcal{A})$ with an accepting run ρ ,

so that $\oplus v \in \mathbf{C}$. By construction of \mathcal{A}' , we have that $\rho \otimes v$ is an accepting run of \mathcal{A}' on w and that $\Pi(\rho \otimes v) \cdot (\bar{p}_1, \dots, \bar{p}_m)^t = \oplus v \in \mathbf{C}$. Hence, $\Pi(\rho \otimes v) \in \mathbf{C}'$ for the accepting run $\rho \otimes v$ of \mathcal{A}' on w . For the right-to-left direction, suppose we have an accepting run $\rho \otimes v$ of \mathcal{A}' on w so that $\Pi(\rho \otimes v) \in \mathbf{C}'$. This means that $\Pi(\rho \otimes v) \cdot (\bar{p}_1, \dots, \bar{p}_m)^t \in \mathbf{C}$. Since $\Pi(\rho \otimes v) \cdot (\bar{p}_1, \dots, \bar{p}_m)^t = \oplus v$, and ρ is an accepting run of \mathcal{A} on $w \otimes v$ by construction of \mathcal{A}' , it follows that $w \otimes v \in \mathcal{L}(\mathcal{A})$ and $\oplus v \in \mathbf{C}$.

2) \Rightarrow 3) Given a NFA \mathcal{A} with statespace Q and a semilinear set $\mathbf{C} \subseteq \mathbb{N}^k$, where $k = |Q|$. Consider the alphabet $\mathbb{B} = Q$ and the NFA \mathcal{A}' so that $\mathcal{L}(\mathcal{A}') = \{w \otimes \rho \mid \rho \text{ is an accepting run of } \mathcal{A} \text{ on } w\}$. Let us assume the lexicographic order on $\mathbb{A} \times \mathbb{B}$. We build \mathbf{C}' by replacing every offset or generator $\bar{v} \in \mathbb{N}^k$ of \mathbf{C} with $\underbrace{(\bar{v}, \dots, \bar{v})}_{|\mathbb{A}|} \in \mathbb{N}^{|\mathbb{A}| \cdot k}$. It then follows that there is a word $w \in \mathcal{L}(\mathcal{A})$ with an accepting run ρ so that $\Pi(\rho) \in \mathbf{C}$ if, and only if, $w \otimes \rho \in \mathcal{L}(\mathcal{A})$ and $\Pi(w \otimes \rho) \in \mathbf{C}'$.

3) \Rightarrow 1) Suppose we have a NFA \mathcal{A} over $\mathbb{A} \times \mathbb{B}$ and a semilinear set $\mathbf{C} \subseteq \mathbb{N}^{|\mathbb{A}| \cdot |\mathbb{B}|}$. Let $\mathbb{D} = \{v_1, \dots, \bar{v}_{|\mathbb{A}| \cdot |\mathbb{B}|}\}$, where $\bar{v}_i \in \mathbb{N}^{|\mathbb{A}| \cdot |\mathbb{B}|}$ is the vector containing a 1 in its i -th component, and zeros in every other component. Let a_i be the i -th letter in the ordering of $\mathbb{A} \times \mathbb{B}$. Given a word $w \in (\mathbb{A} \times \mathbb{B})^*$ let $w^\dagger \in \mathbb{D}^*$ be the result of replacing every a_i in w with \bar{v}_i , for every i . Note that $\Pi(w^\dagger) = \oplus(w^\dagger) = \Pi(w)$.

Consider a NFA \mathcal{A}' over $\mathbb{A} \times \mathbb{D}$ so that $\mathcal{L}(\mathcal{A}') = \{w \otimes v \mid \exists u. w \otimes u \in \mathcal{L}(\mathcal{A}), v = (w \otimes u)^\dagger\}$. We show that a word w is in the language of $(\mathcal{A}', \mathbf{C})$ iff there is some $u \in \mathbb{B}^*$ so that $w \otimes u \in \mathcal{L}(\mathcal{A})$ and $\Pi(w \otimes u) \in \mathbf{C}$.

From left to right, if w is in the language of the PA $(\mathcal{A}', \mathbf{C})$ then there is some $v \in \mathbb{D}^*$ so that $w \otimes v \in \mathcal{L}(\mathcal{A}')$ and $\oplus v \in \mathbf{C}$. Let $u \in \mathbb{B}^*$ so that $(w \otimes u)^\dagger = v$; we then have that $w \otimes u \in \mathcal{L}(\mathcal{A})$. Further, since $\oplus v = \Pi(w \otimes u)$, we have $\Pi(w \otimes u) \in \mathbf{C}$. From right to left, assume there is some $u \in \mathbb{B}^*$ so that $w \otimes u \in \mathcal{L}(\mathcal{A})$ and $\Pi(w \otimes u) \in \mathbf{C}$. Since $\Pi(w \otimes u) = \oplus(w \otimes u)^\dagger \in \mathbf{C}$ and since $w \otimes (w \otimes u)^\dagger \in \mathcal{L}(\mathcal{A}')$, it follows that w is in $\mathcal{L}(\mathcal{A}', \mathbf{C})$. ■

PROOF OF PROPOSTION III.2

Proof: We focus first on the emptiness problem. Let $(\mathcal{A}, \mathbf{C})$ be a PA of dimension k so that $\mathcal{L}(\mathcal{A}) \subseteq \mathbb{A} \times \mathbb{D}$ with $\mathbb{D} = \{\bar{d}_1, \dots, \bar{d}_n\}$. We first build an existential Presburger formula $\varphi_{\mathbf{C}}(y_1, \dots, y_k)$ describing \mathbf{C} in polynomial time (Lemma Theorem II.1). We then produce an existential Presburger formula $\varphi_{\mathcal{A}}^\Pi(x_1, \dots, x_k)$ for $\Pi(\{v \mid u \otimes v \in \mathcal{L}(\mathcal{A})\}) \subseteq \mathbb{N}^n$ in linear time [42, Theorem 4]. From $\varphi_{\mathcal{A}}^\Pi$ one can obtain, in polynomial time, a formula $\varphi_{\mathcal{A}}$ describing $\{\oplus v \mid u \otimes v \in \mathcal{L}(\mathcal{A})\} \subseteq \mathbb{N}^k$, by defining

$$\varphi_{\mathcal{A}}(y_1, \dots, y_k) = \exists x_1, \dots, x_n. \varphi_{\mathcal{A}}^\Pi \wedge (y_1, \dots, y_k) = x_1 \cdot \bar{d}_1 + \dots + x_n \cdot \bar{d}_n.$$

Finally, we check for satisfiability of $\varphi_{\mathcal{A}} \wedge \varphi_{\mathbf{C}}$ in NP [41, Corollary 1].

Now let $(\mathcal{A}, \mathbf{C})$ be a PA so that \mathbf{C} and the working alphabet are represented in unary. Note that from \mathbf{C} one can produce in polynomial time an automaton \mathcal{B} over an alphabet of size k , say $\mathbb{B} = \{b_1, \dots, b_k\}$, so that the Parikh image of $\mathcal{L}(\mathcal{B})$ is \mathbf{C} . Similarly, one can produce in polynomial time an automaton \mathcal{A}' from \mathcal{A} so that the Parikh image of $\mathcal{L}(\mathcal{A}')$ is $\{\oplus v \mid w \otimes v \in \mathcal{L}(\mathcal{A})\}$.¹ Note that both polynomial translations make use of the fact that numbers are represented in unary. In [32] (also in [36]) it is shown that the problem of whether the Parikh image of two given regular languages is disjoint is in PTIME, assuming the alphabet is fixed. Thus, we can check whether

$$\Pi(\mathcal{L}(\mathcal{A}')) \cap \Pi(\mathcal{L}(\mathcal{B})) = \emptyset$$

or, equivalently, whether

$$\{\oplus v \mid w \otimes v \in \mathcal{L}(\mathcal{A})\} \cap \mathbf{C} = \emptyset$$

(and thus whether $(\mathcal{A}, \mathbf{C})$ is empty) in polynomial time.

Let us now consider the emptiness problem. Given a word $w \in \mathbb{A}^*$ and a PA $(\mathcal{A}, \mathbf{C})$ we produce \mathcal{A}' so that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A}) \cap \{w \otimes v \mid v \in \mathbb{D}^*\}$ in polynomial time, and we then check $(\mathcal{A}', \mathbf{C})$ for emptiness as before.

Finally, for NP-hardness, one can easily reduce from the Subset Sum Problem: the problem of, given k integers n_1, \dots, n_k , whether there is a non-empty subset of them whose sum is 0. For any such instance

¹Indeed, it suffices to introduce a simple modification on the transition relation of \mathcal{A} , so that for each letter $(a, (\beta_1, \dots, \beta_k))$ read by \mathcal{A} , we have that \mathcal{A}' reads β_1 times b_1 , followed by β_2 times b_2 , etc.

we can produce a PA $(\mathcal{A}, \mathbf{C})$ of dimension 2 so that $\underbrace{a \cdots a}_k \in \mathcal{L}(\mathcal{A}, \mathbf{C})$ iff the instance has a positive answer. For this, we take $\mathbf{C} = \{(n, n) \mid n \in \mathbb{N}\}$ and we build \mathcal{A} in the following way. We use $k + 1$ states q_0, \dots, q_k so that the initial and final states are q_0 and q_k respectively. We use an auxiliary set $\mathbb{D} = \{(n_i, 0) \mid n_i > 0\} \cup \{(0, -n_i) \mid n_i < 0\} \cup \{(0, 0)\}$ and a singleton alphabet $\mathbb{A} = \{a\}$ (we assume, without any loss of generality, that no n_i is zero). For every $i \in \underline{k}$ we have a transition $(q_{i-1}, (a, (0, 0)), q_i)$; a transition $(q_{i-1}, (a, (n_i, 0)), q_i)$ if $n_i > 0$; and a transition $(q_{i-1}, (a, (0, -n_i)), q_i)$ if $n_i < 0$. Note that the automaton basically chooses either to use n_i (by reading the auxiliary element $(n_i, 0)$ or $(0, -n_i)$ depending on its signature) or not to use it (by reading $(0, 0)$). If the PA is non-empty, then its language is $\underbrace{\{a \cdots a\}_k}$, and there must be a non-empty subset of $\{n_1, \dots, n_k\}$ so that the positive numbers sum up to n and the negative numbers sum up to $-n$, for some $n \in \mathbb{N}$. Hence, $\underbrace{a \cdots a}_k \in \mathcal{L}(\mathcal{A}, \mathbf{C})$ iff the instance has a positive answer, and thus the emptiness and membership problems are NP-hard. \blacksquare

PROOF OF PROPOSITION III.3

Proof: PSPACE-hardness follows from the fact that the Intersection Problem for NFA is already PSPACE-complete [33, Lemma 3.2.3]; here we show that it belongs to PSPACE. Let $(\mathcal{A}_1, \mathbf{C}_1), \dots, (\mathcal{A}_m, \mathbf{C}_m)$ be PA given as instance of the problem, with $\mathcal{A}_i = (Q_i, \mathbb{A} \times \mathbb{D}_i, \delta_i, q_0^i, F_i)$ and $\mathbb{D}_i, \mathbf{C}_i \subseteq \mathbb{N}^{n_i}$. Let us define $\mathbf{C} = \mathbf{C}_1 \times \dots \times \mathbf{C}_m$; note that an existential Presburger formula $\varphi_{\mathbf{C}}$ for \mathbf{C} can be built from $\mathbf{C}_1, \dots, \mathbf{C}_m$ in linear time (due to Lemma II.1).

We define a NFA \mathcal{A} so that $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C} \neq \emptyset$ iff the instance has a positive answer. Let R be $\bigcup_{1 \leq i \leq m} (\mathbb{D}_i \times \dots \times \mathbb{D}_m) \cup \{\bar{0}\}$, where $\bar{0}$ denotes the 0-tuple (i.e., the tuple of dimension 0). Let $N = \sum_{i \in \underline{m}} n_i$ and $n = \sum_{i \in \underline{m}} |\mathbb{D}_i|$. We define the alphabet of \mathcal{A} as

$$\mathbb{D} \stackrel{\text{def}}{=} \bigcup_{i \in \underline{m}} \{0^{l_i}\} \times \mathbb{D}_i \times \{0^{r_i}\} \subseteq \mathbb{N}^N$$

where $l_i = \sum_{1 \leq j < i} n_j$, $r_i = \sum_{i < j \leq m} n_j$ and 0^k stands for the all 0's k -tuple.² Note that $|\mathbb{D}| = n$. We define the statespace of \mathcal{A} as $Q \stackrel{\text{def}}{=} Q_1 \times \dots \times Q_m \times R$. The idea is that \mathcal{A} simulates the execution of $\mathcal{A}_1, \dots, \mathcal{A}_m$ in parallel by iterating the following mechanism:

- it guesses an alphabet letter $(a_i, \bar{x}_i) \in \mathbb{A} \times \mathbb{D}_i$ for each \mathcal{A}_i so that $a_1 = \dots = a_m$,
- it stores the states q_1, \dots, q_m to which transitions lead for each automaton, as well as the vectors $\bar{x}_1, \dots, \bar{x}_m$, and
- it reads $\bar{x}_1, \dots, \bar{x}_m$ one at a time using m transitions.

The sole purpose of this way of simulating $\mathcal{A}_1, \dots, \mathcal{A}_m$ is to keep the alphabet of \mathcal{A} linear. The idea is that, by reading one \bar{x}_i at a time, we are summing (instead of multiplying) the auxiliary sets of $\mathcal{A}_1, \dots, \mathcal{A}_m$.

Formally, we define the initial state as $(q_0^1, \dots, q_0^m, \bar{0})$, the final states as $\{(q_1, \dots, q_m, \bar{0}) \mid q_i \in F_i, i \in \underline{m}\}$, and the transitions as

- $((\bar{q}, \bar{0}), (\bar{x}_1, 0^{r_1}), (\bar{q}', (\bar{x}_2, \dots, \bar{x}_m)))$ for every $\bar{q} = (q_1, \dots, q_m)$, $\bar{q}' = (q'_1, \dots, q'_m)$ and $a \in \mathbb{A}$ so that $(q_i, (a, \bar{x}_i), q'_i) \in \delta_i$ for every $i \in \underline{m}$
- $((\bar{q}, (\bar{x}_i, \bar{x}_{i+1}, \dots, \bar{x}_m)), (0^{l_i}, \bar{x}_i, 0^{r_i}), (\bar{q}, \bar{x}_{i+1}, \dots, \bar{x}_m))$ where $\bar{x}_i \in \mathbb{D}_i$ and $i \leq m$ —note that $\bar{x}_{i+1}, \dots, \bar{x}_m$ is $\bar{0}$ if $i = m$.

Observe that $|\mathcal{A}|$ is exponential in $\sum_i |\mathcal{A}_i|$, but the size of its alphabet \mathbb{D} is only linear in $\sum_i |\mathcal{A}_i|$. We observe that $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C} \neq \emptyset$ if, and only if, $\bigcap_i \mathcal{L}(\mathcal{A}_i, \mathbf{C}_i) \neq \emptyset$. Indeed, if $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C} \neq \emptyset$ there is a word $\bar{x}_1 \cdots \bar{x}_s \in \mathcal{L}(\mathcal{A})$ which means that $s = \ell(m+1)$ for some $\ell \in \mathbb{N}_+$, and there is a word $a_1 \cdots a_\ell \in \mathbb{A}^*$ so that for every $i \in \underline{m}$, we have that $(a_1 \cdots a_\ell) \otimes (\bar{x}'_1 \bar{x}'_{i+m+1} \bar{x}'_{i+2(m+1)} \cdots \bar{x}'_{s-(m+1)+i}) \in \mathcal{L}(\mathcal{A}_i)$, where each \bar{x}'_j is the projection of \bar{x}_j onto the components $\{\sum_{i' < i} k_{i'} + 1, \dots, \sum_{i' \leq i} k_{i'}\}$. Further, since $\bar{x}_1 + \dots + \bar{x}_s \in \mathbf{C}$, we have that $\bar{x}'_1 + \dots + \bar{x}'_s \in \mathbf{C}_i$ and since only the indices equivalent to i modulo $m+1$ contain vectors different from $\bar{0}$, it follows that $\bar{x}'_i + \bar{x}'_{i+m+1} + \bar{x}'_{i+2(m+1)} + \dots + \bar{x}'_{s-(m+1)+i} \in \mathbf{C}_i$ as well. Thus, $a_1 \cdots a_\ell \in \mathcal{L}(\mathcal{A}_i, \mathbf{C}_i)$ for every i . On the other hand, if $\bigcap_i \mathcal{L}(\mathcal{A}_i, \mathbf{C}_i) \neq \emptyset$ there is a word $a_1 \cdots a_s \in \mathbb{A}^*$ and, for every $i \in \underline{m}$, a word $\bar{x}_1^i \cdots \bar{x}_s^i \in \mathbb{D}_i^*$ so that $\bar{x}_1^i + \dots + \bar{x}_s^i \in \mathbf{C}_i$ and $(a_1 \cdots a_s) \otimes (\bar{x}_1^i \cdots \bar{x}_s^i) \in \mathcal{L}(\mathcal{A}_i)$. This means that

²For the cases $i = 1$ and $i = m$ we obtain $\mathbb{D}_1 \times \{0^{r_1}\}$ and $\{0^{l_m}\} \times \mathbb{D}_m$ respectively.

$\bar{y}_1^1 \cdots \bar{y}_1^{m+1} \cdots \bar{y}_s^1 \cdots \bar{y}_s^{m+1} \in \mathcal{L}(\mathcal{A})$ and $\bar{y}_1^1 + \cdots + \bar{y}_1^{m+1} + \cdots + \bar{y}_s^1 + \cdots + \bar{y}_s^{m+1} \in \mathbf{C}$, where \bar{y}_j^i (with $i \leq m$) is the vector containing \bar{x}_j^i in the coordinates $\{\sum_{i' < i} k_{i'} + 1, \dots, \sum_{i' \leq i} k_{i'}\}$ and 0 everywhere else, and $\bar{y}_j^{m+1} = \bar{0}$.

Let us fix an arbitrary linear order on $\mathbb{D} = \{\bar{d}_1 < \cdots < \bar{d}_n\}$. By [36, Theorem 7.3.1], the Parikh image $\Pi(\mathcal{L}(\mathcal{A})) \subseteq \mathbb{N}^n$ of \mathcal{A} can be described by a finite union of linear sets of the form

$$\{\bar{v}_0 + \sum_{i \in \mathbf{n}'} \alpha_i \bar{v}_i \mid \alpha_1, \dots, \alpha_{n'} \in \mathbb{N}\} \quad (6)$$

so that $n' \leq n$, and $\max_i \|\bar{v}_i\|_\infty$ is $O(|Q|^{O(n)} \cdot n^{O(n)})$. From this representation one can obtain a semilinear set corresponding to $\oplus \mathcal{L}(\mathcal{A}) \subseteq \mathbb{N}^N$ whose every linear set is of the form (6) so that $n' \leq n$ and $\max_i \|\bar{v}_i\|_\infty$ is $n \cdot O(|Q|^{O(n)} \cdot n^{O(n)})$ —and therefore exponential in $\sum_i |\mathcal{A}_i|$. To do this, we replace every $\bar{v}_i = (\beta_1, \dots, \beta_n) \in \mathbb{N}^n$ with $\beta_1 \cdot \bar{d}_1 + \cdots + \beta_n \cdot \bar{d}_n \in \mathbb{N}^N$, where $\beta_j \cdot \bar{d}_j = (\beta_j \cdot \bar{d}_j[1], \dots, \beta_j \cdot \bar{d}_j[N]) \in \mathbb{N}^N$.

Therefore, due to Lemma II.1, we obtain:

Observation A.1. $\oplus \mathcal{L}(\mathcal{A})$ can be represented as a disjunction $\bigvee_i \varphi_i$ of existential Presburger formulas, each of which has size linear in $n + N$ and logarithmic in $|Q|$.

Hence, $\varphi = \bigvee_i \varphi_i \wedge \varphi_{\mathbf{C}}$ represents $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}$ and each $\varphi_i \wedge \varphi_{\mathbf{C}}$ is of size polynomial in $\{(\mathcal{A}_i, \mathbf{C}_i)\}_{i \in \mathbf{m}}$. By [41, Theorem 6 (A)],³ if an existential Presburger formula such as $\theta = \varphi_i \wedge \varphi_{\mathbf{C}}$ is satisfiable, there is a satisfying assignment so that each variable is assigned a number that is exponential in $|\theta|$. Thus, φ is satisfiable iff there exists a word $v \in \mathbb{D}^*$ so that $\oplus v \in \oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}$ and $\|\oplus v\|_\infty$ is at most exponential in $\{(\mathcal{A}_i, \mathbf{C}_i)\}_{i \in \mathbf{m}}$. By building \mathcal{A} on-the-fly we can guess and verify the existence of such v using only polynomial space. By Savitch's theorem ($\text{PSPACE} = \text{NPSPACE}$), we can then test, in PSPACE , whether φ is satisfiable, which is equivalent to \mathcal{A} being nonempty, which in turn is equivalent to $\bigcap_i \mathcal{L}(\mathcal{A}_i, \mathbf{C}_i) \neq \emptyset$. Thus, the statement follows. ■

PROOF OF PROPOSITION III.4

We show the following: For any fixed PA $(\mathcal{A}_1, \mathbf{C}_1)$, given a NFA \mathcal{A}_2 , the problem of whether $\mathcal{L}(\mathcal{A}_1, \mathbf{C}_1) \cap \mathcal{L}(\mathcal{A}_2) \neq \emptyset$ is in NLOGSPACE .

Proof: Let \mathcal{A} be the projection of the second component of \mathcal{A}_1 intersected with \mathcal{A}_2 , that is, $\mathcal{L}(\mathcal{A}) = \{v \mid \exists u \in \mathcal{L}(\mathcal{A}_2) \text{ s.t. } u \otimes v \in \mathcal{L}(\mathcal{A}_1)\}$. Notice that \mathcal{A} is linear in \mathcal{A}_2 . It suffices to check whether $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}_1 \neq \emptyset$. By the previous Observation A.1 in the proof of Proposition III.3, we have that $\oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}_1$ can be represented by an existential Presburger formula $\varphi = \bigvee_i \varphi_i \wedge \varphi_{\mathbf{C}_1}$ so that each $\varphi_i \wedge \varphi_{\mathbf{C}_1}$ is logarithmic in the number of states of \mathcal{A} , and thus also logarithmic in \mathcal{A}_2 .

By [41, Theorem 6 (A)], each $\theta = \varphi_i \wedge \varphi_{\mathbf{C}_1}$ is satisfiable iff there is a satisfying assignment so that each variable is assigned a number that is exponential in θ , and hence polynomial in \mathcal{A}_2 . Thus, φ is satisfiable iff there exists a word $v \in \mathbb{D}^*$ so that $\oplus v \in \oplus \mathcal{L}(\mathcal{A}) \cap \mathbf{C}_1$ and $\|\oplus v\|_\infty$ is at most polynomial in \mathcal{A}_2 . This can be checked in NLOGSPACE guessing one position of v at a time and keeping a counter of logarithmic number of bits, ensuring that the sum of the guessed word is of size at most polynomial in \mathcal{A}_2 . ■

PROOF OF PROPOSITION V.1

Proof: Let \mathcal{A} be a NFA synchronizing $R \subseteq (\mathbb{A}^*)^n$. Let w_1, \dots, w_K be all the words of \mathbb{A}^+ of length at most k . Note that the set

$$\mathbf{C} = \{(x_1^1, \dots, x_1^K, \dots, x_n^1, \dots, x_n^K) \in \mathbb{N}^{n \cdot K} \mid \exists \bar{u} \in R \text{ s.t. } \forall i \in \mathbf{n}, j \in \mathbf{K} \text{ we have } x_i^j = \#_{w_j}(\bar{u}[i])\}$$

is semilinear. Indeed, it suffices to extend \mathcal{A} to work over the alphabet $\mathbf{n} \times \mathbb{A}^{\leq k}$ in order to keep track of the last k letters seen for each of the components $i \in \mathbf{n}$, and then take its Parikh image.

Consider then the $\vec{L}_{\mathbf{n}}$ -controlled PA $(\mathcal{A}', \mathbf{C})$, where \mathcal{A}' operates over the alphabet $(\mathbf{n} \times \mathbb{A}) \times \{0, 1\}^{nK}$ (with $\{0, 1\}^{nK}$ as the auxiliary set), so that whenever it reads $(j, a, (u_1^1, \dots, u_1^K, \dots, u_n^1, \dots, u_n^K))$ we have that $u_j^i = 1$ iff $j = j'$ and the word read so far in the j' -th component has w_i as a suffix, for every $i \in \mathbf{K}$.

³It is worth mentioning that Scarpellini shows this for the version of Presburger arithmetic we use (that is, with the doubling function).

Thus, for every $w \in \mathcal{L}(\mathcal{A}', \mathbf{C})$ there is v so that $w \otimes v \in \mathcal{L}(\mathcal{A}')$ and $\oplus v \in \mathbf{C}$. By definition of \mathcal{A}' , assuming $\llbracket w \rrbracket_n = (u_1, \dots, u_n)$ we obtain

$$\oplus v = (\#_{w_1}(u_1), \dots, \#_{w_K}(u_1), \dots, \#_{w_1}(u_n), \dots, \#_{w_K}(u_n)) \in \mathbf{C}.$$

This means that there is $(v_1, \dots, v_n) \in R$ so that $\#_{w_j}(v_i) = \#_{w_j}(u_i)$ for all $i \in \underline{n}$, $j \in \underline{K}$. This shows that $\text{shuffle}_k(R) \in \overrightarrow{\text{REG}}_{\text{PA}}^n$. It is easy to see that a $\overleftarrow{L}_{\underline{n}}$ -controlled automaton synchronizing the same relation as \mathcal{A}' can also be built. Thus, $\text{shuffle}_k(R)$ is in $\overrightarrow{\text{REG}}_{\text{PA}}^n \cap \overleftarrow{\text{REG}}_{\text{PA}}^n$. ■

PROOF OF PROPOSITION V.2

A language $L \subseteq \underline{\mathbf{k}}^*$ is *Parikh-injective* if the function $\Pi : L \rightarrow \mathbb{N}^k$ is injective.

Lemma A.2. *If $L \subseteq \underline{\mathbf{k}}^*$ is Parikh-injective, then for any two PA L -controlled languages $S_1, S_2 \subseteq (\underline{\mathbf{k}} \times \mathbb{A})^*$ we have*

$$\llbracket S_1 \cap S_2 \rrbracket_k = \llbracket S_1 \rrbracket_k \cap \llbracket S_2 \rrbracket_k.$$

Proof: This is a trivial generalization of the case for regular relations and $k = 2$ contained in [23] that we repeat here for the sake of clarity. Let $S_1, S_2 \subseteq (\underline{\mathbf{k}} \times \mathbb{A})^*$ be two L -controlled PA languages. It is immediate that the language $S_\cap = S_1 \cap S_2$ is L -controlled and $\llbracket S_\cap \rrbracket_k \subseteq \llbracket S_1 \rrbracket_k \cap \llbracket S_2 \rrbracket_k$. We show that $\llbracket S_1 \rrbracket_k \cap \llbracket S_2 \rrbracket_k \subseteq \llbracket S_\cap \rrbracket_k$. Suppose that $\bar{u} \in \llbracket S_1 \rrbracket_k \cap \llbracket S_2 \rrbracket_k$. Then, there must be $w_1 \in S_1$ and $w_2 \in S_2$ so that $\llbracket w_1 \rrbracket_k = \llbracket w_2 \rrbracket_k = \bar{u}$. Note that the projection onto the first component of w_1 must be equal to the projection onto the first component of w_2 since L is Parikh-injective. Then, we must have that $w_1 = w_2$ and thus $\bar{u} \in \llbracket S_\cap \rrbracket_k$. ■

Corollary A.3. *If $L \subseteq \underline{\mathbf{k}}^*$ is Parikh-injective, then $\text{REL}_{\text{PA}}(L)$ is effectively closed under intersection.*

The following is a simple observation from the definition of $\overrightarrow{L}_{\underline{\mathbf{k}}}$ and $\overleftarrow{L}_{\underline{\mathbf{k}}}$.

Lemma A.4. *For every $k \in \mathbb{N}_+$, $\overrightarrow{L}_{\underline{\mathbf{k}}}$ and $\overleftarrow{L}_{\underline{\mathbf{k}}}$ are Parikh-injective.*

Thus, by Lemma A.2, we obtain Proposition V.2 as a corollary.

PROOF OF THEOREM VI.1

In the proofs that follow, we will assume that, for \mathcal{S} a class of relations, formulae of $\text{CRPQ}_{\text{rel}}(\mathcal{S})$ are of the form

$$\varphi(\bar{z}) :- x_1 \xrightarrow{\pi_1} y_1, \dots, x_l \xrightarrow{\pi_l} y_l, R_1(\bar{\chi}_1), \dots, R_m(\bar{\chi}_m), \quad (\dagger)$$

where

- without loss of generality we assume that languages witnessing all the π_i s are \mathbb{A}^* (since any regular condition $\lambda(\pi_i) \in L_i$ can be checked in the second part of the query by $L_i(\pi_i)$),
- $\{\pi_i \mid i \in \underline{l}\}$ are the *path variables* of φ ,
- $\{x_i, y_i \mid i \in \underline{m}\}$ are the *node variables* of φ ,
- the π_i 's are pairwise distinct,
- every R_j is in \mathcal{S} and $\bar{\chi}_j$ is a k_j -tuple of variables from $\{\pi_i\}_i$ of the same dimension k_j as R_j ,
- the elements of \bar{z} range over the node variables of φ .

Proof: The lower bounds for this problem come from the evaluation problem for $\overrightarrow{E\text{CRPQ}}$ [6], which is a subclass of instances of this problem. Henceforward we assume we work with $\overrightarrow{\text{REG}}_{\text{PA}}$, since the proof for $\overleftarrow{\text{REG}}_{\text{PA}}$ is completely analogous.

Let us first show the combined complexity case, where both the query and graph are part of the input. Let φ be as in (\dagger) , and let $G = \langle V, E \rangle$ be a labelled graph over the alphabet \mathbb{A} . Let \bar{v} be a vector of vertices that is given. We want to check if $\bar{v} \in \varphi(G)$. We first guess a valuation $\nu : \{x_i, y_i\}_{i \in \underline{l}} \rightarrow V$ so that $\nu(\bar{z}[i]) = \bar{v}[i]$ for every $i \in \underline{l}$. Given two vertices x, y of G , let us write $G[x, y]$ to denote the language recognized by the NFA whose transition relation is the labeled graph G , and initial and final states are, respectively, x and y . Next, we produce the following relations

- the relation

$$R_{\times} \stackrel{\text{def}}{=} \{(w_{1,1}, \dots, w_{1,k_1}, \dots, w_{m,1}, \dots, w_{m,k_m}) \mid (w_{j,1}, \dots, w_{j,k_j}) \in R_j, j \in \underline{m}\}$$

which is simply the crossproduct $R_1 \times \dots \times R_m$;

- for every π that appears both in position i of R_j and i' of $R_{j'}$, a relation

$$R_{i,j,i',j'}^{\boxtimes} \stackrel{\text{def}}{=} \{(w_{1,1}, \dots, w_{1,k_1}, \dots, w_{m,1}, \dots, w_{m,k_m}) \mid w_{j,i} = w_{j',i'}\}$$

which assures that the i -th component of R_j is joined with the i' -th component of $R_{j'}$;

- for every RPQ $x \xrightarrow{\pi} y$ so that π appears in position i of R_j , a relation

$$S_{i,j}^{G,\nu} \stackrel{\text{def}}{=} \{(w_{1,1}, \dots, w_{1,k_1}, \dots, w_{m,1}, \dots, w_{m,k_m}) \mid w_{j,i} \in G[\nu(x), \nu(y)]\}$$

which ensure that the path π is compatible with G, ν , that is, that π is indeed a path of G from the node associated to x to the node associated to y in the guessing ν .

We first show that the intersection of all these relations is nonempty iff \bar{v} is a solution of $\varphi(G)$. From right-to-left, if $\bar{v} \in \varphi(G)$, through the guessed valuation ν , there are paths w_1, \dots, w_l of G corresponding to the paths π_1, \dots, π_l so that for every $R_j(\bar{\chi}_j)$ with $\chi_j = (\pi_{i_1}, \dots, \pi_{i_{k_j}})$ we have a tuple $\bar{w}_j = (w_{i_1}, \dots, w_{i_{k_j}}) \in R_j$. This means that $(\bar{w}_1, \dots, \bar{w}_m) \in R_{\times}$ and further that it verifies the equalities of $R_{i,j,i',j'}^{\boxtimes}$. Further, since the words w_i according to $x_1 \xrightarrow{\pi_1} y_1$, they also verify the restrictions imposed by $S_{i,j}^{G,\nu}$. From left-to-right, if there is a tuple $(\bar{w}_1, \dots, \bar{w}_m) \in R_{\times} \cap \bigcap R_{i,j,i',j'}^{\boxtimes} \cap \bigcap S_{i,j}^{G,\nu}$ this means that the valuation ν correctly guessed nodes that are compatible with the paths in $\bar{w}_1, \dots, \bar{w}_m$ contained in G . We are now left with the task to give a PSPACE procedure to test the above relations for emptiness.

It is easy to see that each of the aforementioned relations are \overrightarrow{L}_t -controlled for $t = \sum_{j \in \underline{m}} k_j$ since each R_j is \overrightarrow{L}_{k_j} -controlled. Further, it is easy to see that for every one of these relations one can build a PA synchronizing it in polynomial time, given PA synchronizing the R_j 's of φ . For example, for R_{\times} it suffices to take the crossproduct of their constraint sets and to compose the NFA for each relation: first the automaton on $\{1, \dots, k_1\} \times \mathbb{A}$ synchronizing R_1 , then the automaton on $\{k_1 + 1, \dots, k_1 + k_2\} \times \mathbb{A}$ synchronizing R_2 (note that we shift the alphabets), etc. The resulting automaton is then \overrightarrow{L}_t -controlled. For $R_{i,j,i',j'}^{\boxtimes}$ and $S_{i,j}^{G,\nu}$ the construction is straightforward.

Now that we have \overrightarrow{L}_t -controlled PA $\mathcal{PA}_1, \dots, \mathcal{PA}_s$ synchronizing each of the relations above, we have that, the intersection of all these PA synchronizes the intersection of all the relations above, that is

$$\begin{aligned} \llbracket \mathcal{PA}_1 \cap \dots \cap \mathcal{PA}_s \rrbracket_t &= \llbracket \mathcal{PA}_1 \rrbracket_t \cap \dots \cap \llbracket \mathcal{PA}_s \rrbracket_t \\ &= R_{\times} \cap \bigcap R_{i,j,i',j'}^{\boxtimes} \cap \bigcap S_{i,j}^{G,\nu}. \end{aligned}$$

This is a consequence of Proposition V.2. This means that $R_{\times} \cap \bigcap R_{i,j,i',j'}^{\boxtimes} \cap \bigcap S_{i,j}^{G,\nu} \neq \emptyset$ if and only if $\mathcal{L}(\mathcal{PA}_1 \cap \dots \cap \mathcal{PA}_s) \neq \emptyset$. We can then test the emptiness of the latter intersection in PSPACE thanks to Proposition III.3. Note that, even though we performed an initial polynomial guessing, by Savitch's Theorem the procedure remains PSPACE.

Let us now analyse the case where the query φ is fixed. Note that the relations $S_{i,j}^{G,\nu}$'s can be synchronized using NFA's instead of PA's. (In other words, we don't need to make use of the semilinear constraint set.) Let then \mathcal{A}_2 be the NFA synchronizing $\bigcap S_{i,j}^{G,\nu}$. Since the number of relations $S_{i,j}^{G,\nu}$'s is bounded by a function on φ , we have that \mathcal{A}_2 can be built so that it is polynomial in the NFA synchronizing the $S_{i,j}^{G,\nu}$'s, which in turn are polynomial in G, \bar{v} . Further, its transitions can be produced on-the-fly using logarithmic space. Let $(\mathcal{A}_1, \mathbf{C}_1)$ be the (fixed) PA synchronizing the remaining (fixed) relations $R_{\times} \cap \bigcap R_{i,j,i',j'}^{\boxtimes}$. Using Proposition III.4, it follows that we can check whether $\mathcal{L}(\mathcal{A}_2) \cap \mathcal{L}(\mathcal{A}_1, \mathbf{C}_1)$ is empty in NLOGSPACE, and thus whether $\bar{v} \in \varphi(G)$. ■

PROOF OF THEOREM VI.3

Proof: (I) We first show that the combined complexity for strongly acyclic queries of CRPQ(REL_{PA}(All)) is in NLOGSPACE. Let $G = \langle V, E \rangle$ be a labelled graph over the alphabet \mathbb{A} , and let \bar{v} be a vector of vertices of G . Assume φ is a strongly-acyclic query of the form (\dagger). We want to check if $\bar{v} \in \varphi(G)$. Without any loss of

generality, let us assume that G_φ is connected. (If it's not connected, one can divide the input vector and query into connected queries, and test each one for membership.) Let $(\mathcal{A}_1, \mathbf{C}_1), \dots, (\mathcal{A}_m, \mathbf{C}_m)$ be PA synchronizing R_1, \dots, R_m , where $\mathbf{C}_i \subseteq \mathbb{N}^{k_i}$ for every i . We now proceed as in the proof of Theorem VI.1, where we reduce the problem to the intersection of several relations of arity $k_1 + \dots + k_m$, assuming $R_i \subseteq (\mathbb{A}^*)^{k_i}$ for $i \in \underline{m}$. We consider the same relations defined in that proof: $R_\times, R_{i,j,i',j'}^{\boxtimes}, S_{i,j}^{G,\nu}$'s. Remember that the intersection of these relations is non-empty iff \bar{v} is a solution of $\varphi(G)$. The key observation here is that we can build, in polynomial time, m PA synchronizing the relations S_1, \dots, S_m so that

$$R_\times \cap \bigcap_{i,j} S_{i,j}^{G,\nu} = S_1 \times \dots \times S_m.$$

This can be done because, on the one hand the relation R_\times is just $R_\times = R_1 \times \dots \times R_m$, and on the other hand the relations $S_{i,j}^{G,\nu}$'s impose at most one further constraint for each of the coordinates of R_\times (here we're using the fact that in the query (\dagger) all the π_i 's are distinct).

We still have to produce the intersection with $R_{i,j,i',j'}^{\boxtimes}$. Remember that $R_{i,j,i',j'}^{\boxtimes}$ basically joins component i of the tuple from relation R_j with component i' of the tuple from relation $R_{j'}$. Due to the strong-acyclicity of φ , this join graph looks like a tree, which allows us to use the repeated application of Lemma VI.4 obtaining a PSPACE procedure. More formally, strong-acyclicity entails the existence of a permutation $p = \ell_1, \dots, \ell_m$ of $1, \dots, m$ and indices $i_1, j_1, \dots, i_{m-1}, j_{m-1}$ so that

$$S^{\boxtimes} \stackrel{\text{def}}{=} ((S_{\ell_1} \boxtimes_{i_1, j_1} S_{\ell_2}) \dots) \boxtimes_{i_{m-1}, j_{m-1}} S_{\ell_m} = \text{perm}_p(S_1 \times \dots \times S_m \cap \bigcap_{i,j,i',j'} R_{i,j,i',j'}^{\boxtimes})$$

where $\text{perm}_p(R)$ of a relation $R \subseteq \mathbb{A}^{k_1 + \dots + k_m}$ is defined as

$$\{(w_{\ell_1,1}, \dots, w_{\ell_1, k_{\ell_1}}, \dots, w_{\ell_m,1}, \dots, w_{\ell_m, k_{\ell_m}}) \mid (w_{1,1}, \dots, w_{1, k_1}, \dots, w_{m,1}, \dots, w_{m, k_m}) \in R\}.$$

Thus through the $(m-1)$ times) application of Lemma VI.4 we obtain a PA $(\mathcal{A}, \mathbf{C})$ synchronizing S^{\boxtimes} , so that:

- \mathcal{A} has an exponential number of states,
- \mathcal{A} has a linear alphabet size, and
- $\mathbf{C} = \mathbf{C}_{\ell_1} \times \dots \times \mathbf{C}_{\ell_m}$.

Finally, given the bounds above, by building \mathcal{A} on-the-fly we obtain a PSPACE procedure as described in the proof of Proposition III.3.

If the query φ is fixed, we proceed in a similar way as in the proof of Theorem VI.1. We observe that we can build the transition relations of S_1, \dots, S_m in NLOGSPACE. It is easy to see that the automaton build by Lemma VI.4 can be built on-the-fly using logarithmic space. Thus, by a constant number of applications (as m is fixed) of the construction we obtain that a PA synchronizing S^{\boxtimes} can be built on-the-fly and test for emptiness in NLOGSPACE.

(II) We proceed as before, where for each connected component we have that there is a constant number of automata involved and thus we can produce a PA for the intersection in polynomial time and check for non-emptiness in NP due to Proposition III.2.

(III) We proceed as before, using the second part of Proposition III.2 this time. ■

PROOF OF LEMMA VI.4

We show the following. Given two PA $(\mathcal{A}_1, \mathbf{C}_1), (\mathcal{A}_2, \mathbf{C}_2)$ synchronizing $R_1 \subseteq (\mathbb{A}^*)^{k_1}, R_2 \subseteq (\mathbb{A}^*)^{k_2}$, and given $i_1 \in \underline{k}_1, i_2 \in \underline{k}_2$, one can build in polynomial time a PA $(\mathcal{A}, \mathbf{C})$ synchronizing

$$R_1 \boxtimes_{i_1, i_2} R_2 \stackrel{\text{def}}{=} \{\bar{u}_1 \bar{u}_2 \mid \bar{u}_1 \in R_1, \bar{u}_2 \in R_2, \bar{u}_1[i_1] = \bar{u}_2[i_2]\} \subseteq (\mathbb{A}^*)^{k_1 + k_2}$$

so that $\mathbf{C} = \mathbf{C}_1 \times \mathbf{C}_2$ and so that the size of the auxiliary set of \mathcal{A} is linear in those of $\mathcal{A}_1, \mathcal{A}_2$.

Proof: Let $\mathcal{A}_1 = (Q_1, \mathbb{A} \times \mathbb{D}_1, \delta_1, q_0^1, F_1), \mathcal{A}_2 = (Q_2, \mathbb{A} \times \mathbb{D}_2, \delta_2, q_0^2, F_2)$. We define $\mathbb{D} = \mathbb{D}_1 \times \{0^{k_2}\} \cup \{0^{k_1}\} \times \mathbb{D}_2, Q = Q_1 \times Q_2 \times (\mathbf{2} \times \mathbb{A} \cup \{\varepsilon\})$. The idea is that a state $(q_1, q_2, (\ell, a))$ represents that we are in state q_1 of \mathcal{A}_1 , in state q_2 of \mathcal{A}_2 and that we need to move forward \mathcal{A}_i from q_i until it reads a letter of the form (i_ℓ, a) ; more formally, from such a state we can either

- if $\ell = 1$, advance through a transition $(q_1, ((j, a'), \bar{v}), q_1') \in \delta_1$ by reading a letter $((j, a'), \bar{v} 0^{k_2})$ to the state

- $(q'_1, q_2, (\ell, a))$ if $j \neq i_1$, or
- (q'_1, q_2, ε) if $(j, a') = (i_1, a)$; or
- if $\ell = 2$, advance through a transition $(q_2, ((j, a'), \bar{v}), q'_2) \in \delta_2$ by reading a letter $((j, a'), 0^{k_1} \bar{v})$ to the state
 - $(q_1, q'_2, (j, a))$ if $j \neq k_1 + i_2$, or
 - (q_1, q'_2, ε) if $(j, a') = (k_1 + i_2, a)$.

A state of the form (q_1, q_2, ε) means that component i_1 from \mathcal{A}_1 and i_2 from \mathcal{A}_2 are synchronized and we can either

- advance through a transition $(q_1, ((j, a), \bar{v}), q'_1) \in \delta_1$ by reading a letter $((j, a), \bar{v} 0^{k_2})$ to the state
 - (q'_1, q_2, ε) if $j \neq i_1$ or
 - $(q'_1, q_2, (2, a))$ if $j = i_1$, or
- advance through a transition $(q_2, ((j, a), \bar{v}), q'_2) \in \delta_2$ by reading a letter $((k_1 + j, a), 0^{k_1} \bar{v})$ to the state
 - (q_1, q'_2, ε) if $j \neq i_2$ or
 - $(q_1, q'_2, (1, a))$ if $j = i_2$.

Finally, the initial state is $(q_0^1, q_0^2, \varepsilon)$ and the final states $\{(q_1, q_2, \varepsilon) \mid q_1 \in F_1, q_2 \in F_2\}$. Note that the size of \mathcal{A} is polynomial in $\mathcal{A}_1, \mathcal{A}_2$, and that \mathbb{D} is linear in $\mathbb{D}_1, \mathbb{D}_2$. Further, we have $[\mathcal{L}(\mathcal{A}, \mathbf{C})]_{k_1+k_2} = R_1 \bowtie_{i_1, i_2} R_2$. ■

PROOF OF PROPOSITION VII.2

Proof: Consider an expression with choice (E, \mathbf{C}) where $E = \{(e_1, \bar{v}_1), \dots, (e_m, \bar{v}_m)\}$. Let \mathcal{A}_i be a DFA for $\mathcal{L}(e_i)$, for $i \in \underline{m}$. We assume that $\mathcal{A}_i = (Q_i, \mathbb{A}, q_0^i, F_i, \delta_i)$ where δ_i is a function from $Q_i \times \mathbb{A}$ to Q_i . Let $\mathcal{A}_0 = (Q_0, \mathbb{A}, q_0^0, \delta_0, F_0)$ be a DFA whose language is $\mathbb{A}^* - \bigcup \{\mathcal{L}(e_i) \mid i \in \underline{m}\}$ (it corresponds to the expression e_0 used in the definition of acceptance by a Parikh expression with choice).

This Parikh expression with choice is captured by a PA $(Q, \mathbb{A}, \bar{q}_0, \delta, F, \mathbf{C})$, where:

- the statespace Q is the subset of $Q_0 \times Q_1 \times \dots \times Q_m$ that consists of tuples (s_0, s_1, \dots, s_m) so that $s_i \in F_i$ for at least one $i \in \underline{m}$ (note that in any simultaneous run of the \mathcal{A}_i s over any word, at any point in the run at least one state will be final, by the definition of \mathcal{A}_0 ; hence, this requirement simply eliminates unreachable states);
- $\bar{q}_0 = (q_0^0, q_0^1, \dots, q_0^m)$;
- $F = Q$, i.e., every state is final and acceptance is by the constraint set;
- a transition $(\bar{s}, a, \bar{v}, \bar{s}')$, where $\bar{s} = (s_0, s_1, \dots, s_m)$ and $\bar{s}' = (s'_0, s'_1, \dots, s'_m)$ is in δ iff
 - $s'_i = \delta_i(s_i, a)$ for all $i \in \underline{m}$, and
 - there is $j \in \underline{m}$ such that $s'_j \in F_j$ and $\bar{v} = \bar{v}_j$ or $s'_0 \in F_0$ and $\bar{v} = \bar{0}$.

That is, the Parikh automaton simulates all the automata $\mathcal{A}_0, \mathcal{A}_1, \dots, \mathcal{A}_m$ in parallel; if one of them, say \mathcal{A}_i , upon reading letter a enters a final state (which means the prefix is in $\mathcal{L}(e_i)$), the corresponding vector \bar{v}_i can be added to the counters (or all zeros, in the case of \mathcal{A}_0). It is routine to verify that this automaton captures (E, \mathbf{C}) . Note also that it satisfies the condition mentioned in the paper: if $(\bar{s}, a, \bar{v}', \bar{s}')$ and $(\bar{s}, a, \bar{v}'', \bar{s}'')$ are both in δ , then $\bar{s}' = \bar{s}''$ (although \bar{v}' need not equal \bar{v}'' , so this is a weaker requirement than the determinism of PAs). ■