Data Path Queries over Embedded Graph Databases

Diego Figueira, Artur Jeż, Anthony W Lin

Data Path Queries over Embedded Graph Databases

Diego Figueira
diego.figueira@cnrs.fr
Univ. Bordeaux, CNRS, Bordeaux INP,
LaBRI, UMR 5800
Talence, France

Artur Jeż
artur.jez@uwr.edu.pl
University of Wrocław
Wrocław, Poland

Anthony W. Lin
anthony.lin@cs.uni-kl.de
Automated Reasoning Group, TU
Kaiserslautern and MPI-SWS
Kaiserslautern, Germany

ABSTRACT

This paper initiates the study of data-path query languages (in particular, regular data path queries (RDPQ) and conjunctive RDPQ (CRDPQ)) in the classic setting of embedded finite model theory, wherein each graph is "embedded" into a background infinite structure (with a decidable FO theory or fragments thereof). Our goal is to address the current lack of support for typed attribute data (e.g. integer arithmetics) in existing data-path query languages, which are crucial in practice. We propose an extension of register automata by allowing powerful constraints over the theory and the database as guards, and having two types of registers: registers that can store values from the active domain, and read-only registers that can store arbitrary values. We prove NL data complexity for (C)RDPQ over the Presburger arithmetic, the real-closed field, the existential theory of automatic structures and word equations with regular constraints. All these results strictly extend the known NL data complexity of RDPQ with only equality comparisons, and provides an answer to a recent open problem posed by Libkin et al. Among others, we introduce one crucial proof technique for obtaining NL data complexity for data path queries over embedded graph databases called "Restricted Register Collapse (RRC)", inspired by the notion of Restricted Quantifier Collapse (RQC) in embedded finite model theory.

CCS CONCEPTS

• **Theory of computation → Database query languages (principles)**; **Complexity theory and logic**; **Finite Model Theory**; **Logic and databases**; *Regular languages.*

KEYWORDS

Data graphs, complexity, embedded finite models, regular path queries

1 INTRODUCTION

The past decade witnessed a lot of practical and theoretical work in querying graph databases (e.g. [2, 3, 17, 49]), due to their rapidly growing number of applications, e.g., in social networks, semantic web, natural science, and supply chain management. On the practical side, graph databases have enjoyed a wide adoption in industry with many commercial database systems developed for them by various companies like Neo4j, Oracle, IBM, DataStax, TigerGraph, JanusGraph, and Stardog, among many others. Although such systems adopt the so-called "property graph data model", they differ in many important aspects (e.g. querying facilities), which has led to the ongoing standardization efforts of SQL/PGQ and GQL involving leading researchers and engineers in academia and industry [17].

On the theoretical side, an important research avenue has been to develop "well-behaved" graph query languages incorporating *topological* aspects of the graph, especially the existence of *paths* in a graph satisfying certain patterns and constraints. Such queries are called *path queries*, and represent an essential feature (called GPML, which stands for *Graph Pattern Matching Languages*) in the current standardization effort [17] of GQL and SQL/PGQ. Historically, Mendelzon and Wood [42] introduced in the 1990s the Regular Path Query (RPQ) language, which has been the foundation of most path query languages, e.g., Conjunctive Regular Path Queries (CRPQ) [16, 18, 25]. One of their main attractive features is that their query evaluation problem has an NL (Nondeterministic Logspace) *data complexity* (i.e. *for any fixed query*), potentially allowing traversals of large graphs in practice (e.g. see [4, 48]).

Combining data and topology. RPQ and CRPQ are unfortunately not suitable for "data querying" in graph databases, which is bread and butter for relational database query languages. More precisely, in a typical graph database each node is usually associated with some data like names and ages of the users. In relational algebra (assuming a relational encoding of graph databases), one could easily express the query that outputs two node ids for users with the same ages. In the past decade, the importance of combining data and topology in graph query languages has been brought to attention by multiple works, e.g. [6, 30, 40, 41]. There are many natural examples of such queries, e.g., find two people in the database with the same age. In particular, walk logic (WL) [30] and regular expressions with memory (REM) [41] emerged as two initial solutions for overcoming this limitation. While RPQ using REM and register automata as path constraint has NL data complexity [40, 41], WL has an non-elementary data complexity [6]. The crucial observation in [40, 41] is that paths in a graph database can be construed as a *data word*, which enables one to apply automata over data words (e.g. register automata [34], REM [41], data automata [14], etc.) as path constraints in a path-query. Other automata models over data words (e.g. data automata) increase the data complexity of RPQ from NL to NP-hard [40].

The need for reasoning about concrete data. Thus far, the solutions [6, 30, 40, 41] to combine data and topology aspects in graph queries ignore completely the *structure of the data*. For example, practical database query languages typically allow concrete data (e.g. numbers, words) and a wide variety of data operations/comparisons (e.g. substrings, inequality, difference). Consider, for instance, the "Bacon graph" (see Figure 1) and the query "*return all actors with a finite Bacon number and whose birthyear differs Bacon's birthyear by at least 30 years*," whose only answer here is Erdős. Alternatively, consider the query "*return all users who were in at least four different locations in the last month, but stayed within*

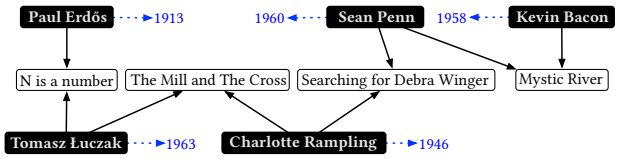

Figure 1: Part of the "Bacon Graph", where each solid (resp. dotted) edge represents the "acts in" (resp. "is born in") relation. Each actor is a black box, and each movie is a white box. Birthyears are also nodes.

Table 1: Summary of our data complexity results

$\mathbb{L}$	Data Complexity of RDPQ($\mathbb{L}, \sigma$)
FO($\mathbb{R}_{\times,+}, \sigma$)	NL
FO($\mathbb{Z}_{LA}, \sigma$)	NL
EFO$^+$($\mathbf{T}_{\text{Aut}}, \sigma$)	NP-hard
EFO($\mathbf{T}_{\text{Aut}}, \sigma$)	NL (under LogH)
$\mathbb{L}_{\text{PM}}$	NL
$\mathbb{L}^+_{\text{WE}}$	NL

a 10km radius from some point in a map," whose numbers and how they differ on a monthly basis might be insightful especially during the time of COVID-19. Finally, for a phylogenetic database, one might be interested in a path containing only nodes whose center (with respect to some edit distance measure) is not that far away from any DNA sequence along the path [15, 28]. None of the above features are expressible in the existing path-query languages. For example, while path query languages like RDPQ [40] may for instance express the existence of a path from Erdős to Bacon, sharing *the same* birth years, they only support active-domain values in their registers, as well as equality comparisons, which restrict them from expressing the three aforementioned examples.

Incorporating concrete attribute data and their operations in a first-order query language over relational databases has been intensively studied by database theorists between 1990 and early 2000s in the context of constraint databases and embedded finite model theory (e.g. see [27, 38] and [39, Chapter 13]). In particular, one considers a relational database embedded into the universe of an infinite structure with a decidable FO theory (e.g. real-closed field $\mathbb{R}_{\times,+}$ or integer linear arithmetic $\mathbb{Z}_{LA}$ [27, 38, 39]), and allows first-order queries to make use of operations in the theory. For example, over a graph embedded into $\mathbb{R}_{\times,+}$, one could ask if all edges in a graph lie on a line. The first-order query can be found in [39, Chapter 13]: $\exists u\, \exists v\, \forall x \in adom\, \forall y \in adom\, (E(x,y) \rightarrow y = u.x + v)$. Notice two kinds of quantifiers are in use: $\exists u$ means "there is a real number (not necessarily in the active domain)", while $\forall x \in adom$ means "for all elements in the active domain". The set of such first-order formulas over the theory $\mathbf{T}$ and relational database schema σ is denoted by FO($\mathbf{T}, \sigma$), and its restrictions with only active-domain quantifiers by FO$_{\text{act}}$($\mathbf{T}, \sigma$). Despite this, the issue of querying data-paths over embedded graph databases remains hitherto unaddressed and was only left as an open problem by Libkin et al. [40].

Contributions. This paper initiates the study of data-path queries in the setting of *embedded graph databases*, i.e., finite graphs embedded in some infinite structure with a decidable first-order theory $\mathbf{T}$ or fragments thereof. Because of the different graph models in the literature we assume a general relational database schema σ and define a graph by means of *first-order views* over σ and $\mathbf{T}$. This is closer to the data model adopted by SQL/PGQ (e.g. see [17] and https://pgql-lang.org/), and can be used to encode implicitly, as well as explicitly, represented property graphs. We study the problem of query evaluation over extensions of RDPQ and CRDPQ, allowing the use of FO($\mathbf{T}, \sigma$)-formulas in the queries, which capture a range of meaningful queries involving typed attribute values like integers, reals, and words (e.g. the three aforementioned examples).

Since register automata [40] use untyped attribute data which can only be compared via equality and disequality, we first define *extended register automata (ERA)* that allow attribute data in $\mathbf{T}$ which can be manipulated via operations in FO($\mathbf{T}, \sigma$). By analogy to the setting of embedded finite model theory, we allow two kinds of registers: active-domain registers and unrestricted registers. Each transition is associated with a guard $\varphi(curr, \mathcal{R}, \mathcal{R}', \mathcal{P}, \mathcal{P}') \in$ FO($\mathbf{T}, \sigma$), where $curr$ is the value in $\mathbf{T}$ representing the current node (called *node ID*), $\mathcal{R}$ (resp. $\mathcal{P}$) represents the current values of the active-domain (resp. unrestricted) registers, and $\mathcal{R}'$ (resp. $\mathcal{P}'$) represents the next values of the active-domain (resp. unrestricted) registers. The register automata model of [40] can be construed as an instance of ERA with no unrestricted registers, where $\sigma = \{E\}$, $\mathbf{T} = \langle \mathbb{N}; =, \{c_i\}_{i \in \mathbb{N}} \rangle$ with each constant c_i interpreted as the number i, and the guard uses only Boolean combinations of FO($\mathbf{T}$)-atoms.

Without further restrictions, the model easily captures the universal Minsky's counter machine already with two unrestricted registers and the theory $\langle \mathbb{N}; succ \rangle$ of natural numbers with successor [45]. Hence, we impose the so-called *bounded-rewrite assumptions* to the unrestricted registers, i.e., for some constant $b \in \mathbb{N}$, each unrestricted register can be rewritten at most b times. The resulting query languages RDPQ($\mathbf{T}, \sigma$) and CRDPQ($\mathbf{T}, \sigma$) turn out to be sufficiently general to express many interesting examples (e.g. our three aforementioned examples). Our result on the data complexity of query evaluation is summarized in Table 1. Note that under the bounded-rewrite assumption we can assume that $b = 0$ at the expense of increasing the number of unrestricted registers, see Proposition 2.4.

First, we obtain NL data complexity for RDPQ(FO, $\mathbb{R}_{\times,+}, \sigma$) and RDPQ(FO, $\mathbb{Z}_{LA}, \sigma$). With these, we can express the first two out of our three examples. To prove this, we introduce the so-called *Restricted Register Collapse (RRC)* — akin to the notion of *Restricted Quantifier Collapse (RQC)* for FO over embedded finite models [39, Chapter 13] — which states that each query in RDPQ(FO, $\mathbf{T}, \sigma$) is equivalent to a query in RDPQ(FO, $\mathbf{T}, \sigma$) without unrestricted registers and so that no guards contain unrestricted quantifiers. We show that this holds for $\mathbb{R}_{\times,+}$ and $\mathbb{Z}_{LA}$.

The case of words is more involved. To permit reasoning about properties such as edit distance, one needs a sufficiently expressive word theory $\mathbf{T}$, e.g., the theory $\mathbf{T}_{\text{Aut}}$ of automatic relations [11, 13], or theory $\mathbf{T}_{\text{WE}}$ of word equations [19]. Unfortunately, FO($\mathbf{T}_{\text{Aut}}, \sigma$) and FO($\mathbf{T}_{\text{WE}}, \sigma$) have prohibitive data complexity [11] (hard for the

polynomial hierarchy PH and undecidable, respectively). We provide restrictions that still allow us to reason about interesting properties like edit distance, while admitting NL data complexity. First we show that the existential fragment $\mathrm{EFO}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$ admits RQC (i.e. $\mathrm{EFO}(\mathbf{T}_{\mathrm{Aut}}, \sigma) \subseteq \mathrm{FO}_{\mathrm{act}}(\mathbf{T}, \sigma)$), implying an NC^1 data complexity. In contrast, we show that $\mathrm{RDPQ}(\mathrm{EFO}, \mathbf{T}_{\mathrm{Aut}}, \sigma)$ has NP-hard data complexity and does not admit RRC, unless P = NP. Fortunately, NL data complexity can be recovered for $\mathrm{RDPQ}(\mathrm{EFO}, \mathbf{T}_{\mathrm{Aut}}, \sigma)$ assuming the *LogH hypothesis* that the words in the database are of length logarithmic in the size of the database, which is reasonable for most graph database applications, with extremely large graphs and exponentially smaller data sizes (e.g. names, DNA sequences, etc.). Secondly, we provide two fragments of $\mathrm{FO}(\mathbf{T}_{\mathrm{WE}}, \sigma)$ (pattern-matching fragment $\mathbb{L}_{\mathrm{PM}}$ and positive quantifier-free fragment $\mathbb{L}^{+}_{\mathrm{WE}}$) that permit NL data complexity for $\mathrm{RDPQ}(\mathbf{T}_{\mathrm{WE}}, \sigma)$, even without assuming LogH. The proof is an intricate application of word equation solving.

Organization. We introduce our data model and query language in §2. We tackle data complexity of the theories over numbers (reals, integers) in §3, and over words in §4 (automatic structures) and §5 (word equations). We conclude in §6 with generalizations (e.g. conjunctive queries and property graphs), and future work.

Complexity Theory Preliminaries. We assume standard complexity classes esp. L (Logspace), NL (Nondeterministic Logspace), P, NP, PH, PSPACE. In this paper, we will also mention (the uniform versions of) circuit complexity classes like $\mathrm{AC}^0, \mathrm{TC}^0, \mathrm{NC}^1$. It is well-known that $\mathrm{AC}^0 \subsetneq \mathrm{TC}^0 \subseteq \mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{NL} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$. All reductions mentioned in this paper can be implemented in Logspace. See [37, 39, 47] for more.

2 DATA MODEL AND QUERIES

Embedded Finite Models. Fix a relational vocabulary σ containing finitely many relation symbols R_i with arity r_i (i.e. $\sigma = \{R_1, \ldots, R_k\}$) and an infinite structure $\mathbf{T}$ with domain $\mathcal{D}$. A *finite σ-model embedded into* $\mathbf{T}$ [39, Ch. 13] is a σ-structure $\mathbf{S} := \langle U; \{R_i^{\mathbf{S}}\}_{i=1}^{k} \rangle$ such that $U \subseteq \mathcal{D}$ is finite and $R_i^{\mathbf{S}} \subseteq U^{r_i}$ for every i. The set U is called the *active domain* of $\mathbf{S}$ and hence denoted by $adom(\mathbf{S})$. In the sequel, we shall also refer to such $\mathbf{S}$ as $(\mathbf{T}, \sigma)$*-structure*.

By $\mathrm{FO}(\mathbf{T}, \sigma)$, we denote first-order logic that may use $(\mathbf{T} \cup \sigma)$-atomic formulas, active-domain quantification ($\exists x \in adom, \forall x \in adom$), as well as unrestricted quantification ($\exists x, \forall x$). We denote by $\mathrm{FO}_{\mathrm{act}}(\mathbf{T}, \sigma)$ the restriction of $\mathrm{FO}(\mathbf{T}, \sigma)$ that prohibits unrestricted quantifications. For a formula $\varphi(\bar{x})$ and $\bar{a} \in \mathcal{D}^{|\bar{x}|}$, we define the notion of satisfaction $\mathbf{S} \models_{\mathbf{T}} \varphi(\bar{a})$ by interpreting each atomic formula from σ (resp. $\mathbf{T}$) with respect to $\mathbf{S}$ (resp. $\mathbf{T}$), each active-domain quantifier $\exists x \in adom\ \psi$ (resp. $\forall x \in adom\ \psi$) as "there exists an element b in U such that ψ" (resp. "for all elements b in U, ψ"), and each unrestricted quantifier $\exists x\ \psi$ (resp. $\forall x\ \psi$) as "there exists an element b in $\mathcal{D}$ such that ψ" (resp. "for all elements b in $\mathcal{D}$, ψ"). See the example $\exists u \exists v \forall x \in adom \forall y \in adom\ (E(x, y) \rightarrow y = u.x + v)$ in § 1 and more examples in [27, 39].

We fix now some extra notation that we will use in the sequel. We write $\models$ instead of $\models_{\mathbf{T}}$, when $\mathbf{T}$ is understood. We also write $[\![\varphi]\!]_{(\mathbf{T},\mathbf{S})}$ to denote the set of all $\bar{a} \in \mathcal{D}^{|\bar{x}|}$ such that $\mathbf{S} \models_{\mathbf{T}} \varphi(\bar{a})$. Whenever understood, we also use the same notation for a relation symbol R and the relation $R^{\mathbf{S}}$ that instantiates R in $\mathbf{S}$. Finally, for the purpose of measuring complexity of query evaluation problems, we assume a size function $|.| : \mathcal{D} \rightarrow \mathbb{N}$ that defines the size of the representation of a datum; this will be fixed with respect to each theory under consideration.

Table 2: Embedding of Bacon Graph into $\mathbb{Z}_{LA}$. Each year has itself as node ID.

ID	node
0	Kevin Bacon
1	Paul Erdős
2	Tomasz Łuczak
3	Sean Penn
4	Charlotte Rampling
-1	N is a number
-2	The Mill and The Cross
-3	Searching for Debra Winger
-4	Mystic River

Embedded Graph Databases. Fix a finite alphabet Σ of symbols. We define our notion of data graphs following [40]. Our results can be easily adapted to a notion of property graphs [17]; see § 6. We assume in the following a theory $\mathbf{T}$ and a finite $(\mathbf{T}, \sigma)$-structure $\mathbf{S}$. Our notion of embedded graphs will be defined with respect to $\mathbf{S}$ via views defined through formulas over a fragment $\mathbb{L} \subseteq \mathrm{FO}(\mathbf{T}, \sigma)$.

Fix our view vocabulary $\nu = \{V\} \cup \{E_a\}_{a \in \Sigma}$, where V is unary (denoting the vertex view) and each E_a is binary (denoting the view of a-labeled edges). A *view definition* for a view W with arity r is an $\mathbb{L}$-formula $\varphi_W(\bar{x})$ with $r = |\bar{x}|$. Each view W will then be interpreted as $[\![\varphi_W]\!] \subseteq U^r$ containing all $\bar{a} \subseteq U^r$ such that $\mathbf{S} \models \varphi(\bar{a})$. As before, whenever understood, we will confuse each view W and their interpretation $[\![\varphi_W]\!]$. We are now ready to define our notion of (data)-graph $\mathbf{G}$ embedded into $\mathbf{T}$ (or $(\mathbf{T}, \sigma)$*-graph* or just *graph* for short): it is a pair $(\Theta, \mathbf{S})$, where Θ is a set of view definitions and $\mathbf{S}$ a finite $(\mathbf{T}, \sigma)$-structure.

In the sequel, instead of writing $(v, w) \in E_a$, we shall often write $v \rightarrow_a w$. A $\mathbf{G}$*-path* from v to w, where $v, w \in V$, is simply an alternating sequence $v = v_0 a_1 \cdots a_n v_n = w$ of vertices and edge labels of $\mathbf{G}$ (i.e. each $v_i \in V$ and each $a_j \in \Sigma$) such that $v_i \rightarrow_{a_{i+1}} v_{i+1}$ for each $i \in [0, n-1]$.

Example 2.1. Consider our Bacon graph G from Figure 1. We may assume that the database schema used to represent this is $\sigma = \{E_a, E_b, Act, Mov, Y\}$, where E_a (resp. E_b) is a binary relation representing the "$\underline{a}$cts in" (resp. "was $\underline{b}$orn in") relation, Act (resp. Mov) is a unary relation representing all the actors (resp. movies), and Y is a unary relation containing all the years in the database. Let $\mathbf{T} = \mathbb{Z}_{LA}$. To embed G into $\mathbf{T}$, we will associate each node with a unique ID, as in Table 2. For example, this embedding has tuples $E_b(1, 1913)$, $E_a(0, -4)$, etc. We may define the views as follows:

$$\begin{aligned} V(x) &:= Act(x) \\ E(x_1, x_2) &:= \exists y\ (Mov(y) \wedge E_a(x_1, y) \wedge E_a(x_2, y)) \end{aligned}$$

Note that two actors are connected by an E-edge iff they act in a common movie.

Example 2.2. Let $\mathbf{T} = \mathbb{R}_{\times,+}$ and let $\sigma = \langle T \rangle$, where T is binary. Each pair (t, n) in T indicates a time stamp $t \in \mathbb{Q}$ (with day as unit, where fractions are allowed since data are inserted as they become available) and the total number $n \in \mathbb{N}$ of active COVID cases at time stamp t. Graph views will contain only the vertex view V and a single edge view E with the following view definitions:

$$\begin{aligned} V(x) &:= \exists y \in adom\ (T(x,y)) \\ E(x_1,x_2) &:= x_1 < x_2 \wedge \exists y_1, y_2\ (T(x_1,y_1) \wedge T(x_2,y_2)) \wedge \\ &\quad \neg\exists x\ (x_1 < x < x_2 \wedge \exists y\, T(x,y)) \end{aligned}$$

In other words, we obtain a graphical representation of the number of active COVID cases whose vertices represent time stamps in the DB and edges represent the time progressions. One might be interested in a long enough stretch of time stamps in T whose number of active COVID cases exceeds some sufficiently fast growing function $p(t)$. We will mention cases which can be answered with NL data complexity.

Data Path Queries. Fix a fragment $\mathbb{L} \subseteq \mathrm{FO}(\mathbf{T}, \sigma)$. For an alphabet Σ, let $\Sigma_\# = \Sigma \cup \{\#\}$, where $\# \notin \Sigma$ is a fixed "dummy symbol". In order to define our query language $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$ (or $\mathrm{RDPQ}(\mathbb{L}, \sigma)$ or just $\mathrm{RDPQ}(\mathbb{L})$ in short), we define our notion of *extended register automata (ERA)* admitting active-domain and unrestricted registers, which can take any value in the domain $\mathcal{D}$ of $\mathbf{T}$. More precisely, a $(\sigma, \mathbf{T})$-*register automaton* is a tuple $\mathcal{A} = (\mathcal{R}, \mathcal{P}, Q, Q_0, F, \Delta)$, where $\mathcal{R} = \{r_1, \ldots, r_l\}$ is a set of active-domain registers, $\mathcal{P} = \{p_1, \ldots, p_h\}$ is a set of unrestricted registers, Q is a finite set of states, $Q_0 \subseteq Q$ is the set of initial states, $F \subseteq Q$ is the set of final states, and Δ is a finite set of transitions[1] each of the form $(q, (a, \varphi(curr, \mathcal{P}, \mathcal{P}', \mathcal{R}, \mathcal{R}')), q')$, where $q, q' \in Q$, $a \in \Sigma_\#$, and $\varphi \in \mathbb{L}$ with $\mathcal{R}' := \{r'_1, \ldots, r'_l\}$ and $\mathcal{P}' := \{p'_1, \ldots, p'_h\}$. Intuitively, *curr* refers to current node ID, $\mathcal{P}$ (resp. $\mathcal{R}$) the current values of the unrestricted (resp. active-domain) registers, and $\mathcal{P}'$ (resp. $\mathcal{R}'$) the next values of the unrestricted (resp. active-domain) registers. More formally, given a $(\mathbf{T}, \sigma)$-graph $\mathbf{G}$, a path in it

$$\pi := v_0 \to_{a_1} \cdots \to_{a_n} v_n,$$

we write that $\mathbf{G}, \pi \models \mathcal{A}$ if there is a sequence of transitions

$$q_0 \to_{t_0} \cdots \to_{t_n} q_{n+1},$$

with $q_0 \in Q_0$, $q_{n+1} \in F$, and register values

$$\mathbf{v} = (\mu_0, \nu_0), \ldots, (\mu_n, \nu_n) \in U^{\mathcal{R}} \times \mathcal{D}^{\mathcal{P}} \qquad (*)$$

such that for every $0 < i \leq n$: (1) $t_0 = (q_0, (\#, \varphi_0), q_1)$ and $t_i = (q_i, (a_i, \varphi_i), q_{i+1})$; and (2) $\mathbf{G} \models \varphi_i(v_i, \nu_{i-1}, \nu_i, \mu_{i-1}, \mu_i)$. We write $[\![\mathcal{A}]\!]_{\mathbf{G}}$ to be the set of all pairs $(u, v) \in V^2$ of vertices in $\mathbf{G}$ such that there is a path π from u to v such that $\mathbf{G}, \pi \models \mathcal{A}$.

An RDPQ query is a formula F of the form $x \to_{\mathcal{A}} y$, such that $[\![F]\!]_{\mathbf{G}} = [\![\mathcal{A}]\!]_{\mathbf{G}}$. We are concerned with the *query evaluation problem*: given a graph $\mathbf{G}$, a pair of vertices $(s, t) \in V^2$, and an RDPQ formula $F := x \to_{\mathcal{A}} y$, whether $(s, t) \in [\![F]\!]_{\mathbf{G}}$.

Unsurprisingly, this problem is undecidable for a fixed $\mathcal{A}$, even for the decidable theory $\langle \mathbb{N}, succ \rangle$ of natural numbers with successors, since two unrestricted registers can be used to simulate universal Minsky's 2-counter machines (e.g. see [45]). For this reason, we impose the so-called *bounded-rewrite* restriction for the unrestricted registers. In the sequence of register values in $(*)$, a register p_i *undergoes* b *rewrites* if the number of value changes of p_i in the sequence is b. For example, if the values of p in an $\mathcal{A}$-run are 7, 7, 7, 3, 3, 10, 1, then p undergoes 3 rewrites. In the sequel, we will restrict ourselves to runs of $\mathcal{A}$ with at most b rewrites for each unrestricted register, for some constant $b \in \mathbb{N}$. We will refer to such $\mathcal{A}$ as (h, b)-rewrite register automaton, where h is the number of unrestricted registers of $\mathcal{A}$. We will tacitly assume that each $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$-query uses an (h, b)-rewrite register automaton because of the following:

[1] After gaining familiarity with ERA, the reader is referred to Section 6 for a generalization of ERA that *directly* traverse the relational database S.

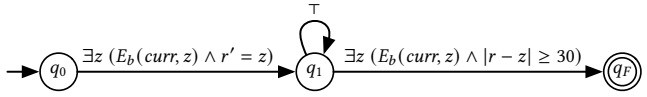

Figure 2: Automaton $\mathcal{A}$ for the query in Example 2.5.

Proposition 2.3. *Let $\mathbb{L} = \mathrm{FO}(\mathbf{T}, \sigma)$ or $\mathbb{L} = \mathrm{EFO}(\mathbf{T}, \sigma)$, and assume that query evaluation for $\mathbb{L}$ is decidable. Then, query evaluation for $\mathrm{RDPQ}(\mathbb{L}, \sigma)$ with (h, b)-rewrite register automata is decidable.*

This can be easily proven: it suffices to consider paths in the graph of length at most $|U|^l \times |Q| \times (b+1)^h$ (remember, $l = |\mathcal{R}|$) and hence we turn each such path π into a formula in $\mathbb{L}$ to check if there is a sequence of $\mathcal{A}$-configurations conforming to π.

In this paper we are concerned mostly about the issue of *data complexity*: What is the complexity of the evaluation problem for a *fixed* $\mathrm{RDPQ}(\mathbf{T}, \sigma)$-query? We count the view definitions in $\mathbf{G}$ also as part of the query, which is consistent with PGQL (see https://pgql-lang.org/). Following [40], NL data complexity is of particular interest, which is achieved for the basic RDPQ (i.e. with the base theory $\langle \mathbb{N}; =, \{c_i\}_{i \in \mathbb{N}} \rangle$, and $\sigma = \langle E_1, \ldots, E_n \rangle$). When considering data complexity, the following proposition shows that we may assume $(h, 0)$-rewrite register automata, i.e., each unrestricted register behaves like a *parameter*, a notion that appears in parametric one-counter automata [29] and parametric timed automata [1].

Proposition 2.4. $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$ *with (h, b)-rewrite register automata is (effectively) equi-expressive with* $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$ *with $(hb, 0)$-rewrite register automata.*

In the sequel, we assume that each unrestricted register is a *parameter* and we simplify notation accordingly (e.g. the guard in each transition is of the form $\varphi(curr, \mathcal{P}, \mathcal{R}, \mathcal{R}')$).

Example 2.5. Following Example 2.1, we are interested in finding the set of pairs (x, y) of actors whose age difference is at least 30. This can be expressed in $\mathrm{RDPQ}(\mathbb{Z}_{LA}, \sigma)$ as $x \to_{\mathcal{A}} y$, where $\mathcal{A}$ has three states q_0, q_1, q_F (q_0 initial and q_F final) and one active-domain register r. The automaton $\mathcal{A}$ is depicted in Figure 2. Intuitively, $\mathcal{A}$ saves the birthyear of the actor x, follows the edges in E, and nondeterministically guesses an actor y whose birthyear differs by at least 30 from the birthyear of x. It can be seen that $(0, 1)$ is an answer to this query (recall that 0 and 1 refer to Erdős and Bacon).

Example 2.6. Following Example 2.2, suppose we are interested in finding a period of at least 28 days where the total number of active COVID cases grows exponentially. It is a well-known open problem (e.g. see [27, Chapter 5]) whether $\mathbb{R}_{\times,+}$ with the exponential function e^x is decidable. To circumvent this, we could

use the standard trick in numerical methods to approximate $\lambda e^{\chi x}$ by the nth Taylor polynomial $p_n(x)$ for sufficiently large n, i.e., $\sum_{i=0}^{n} \lambda(\chi x)^i/i!$. (In practice, one uses small enough n, e.g., $n = 4$ is used in [5].) Let $g(\lambda, \chi, x) := p_n(x)$, with λ and χ treated also as variables. An example query could be to find any window between two time stamps $(t, t') \in E$ spanning across at least 28 days (i.e. $t' - t \geq 28$) such that there exist $\lambda \geq 1, \chi > 0.4$ such that the total number of active COVID cases at each time step s in this interval is within $B := 50$ from $g(\lambda, \chi, s) + c$ for some non-negative constant c. This can be expressed in $\mathrm{RDPQ}(\mathbb{R}_{\times,+}, \sigma)$ as $x \to_{\mathcal{A}} y$, where $\mathcal{A}$ has three states q_0, q_1, q_2 (q_0 initial and q_2 final), one active-domain register r and three parameters λ, χ, c. The transitions are $(q_0, \psi_0(curr, \mathcal{P}, r, r'), q_1)$, $(q_1, \psi_1(curr, \mathcal{P}, r, r'), q_1)$, $(q_1, \psi_2(curr, \mathcal{P}, r, r'), q_2)$, where

$$\psi_0 := r = curr \wedge \lambda \geq 1 \wedge \chi > 0.4 \wedge c \geq 0 \wedge \theta$$
$$\psi_1 := r' = r \wedge \theta, \quad \psi_2 := \psi_1 \wedge curr - r \geq 28$$
$$\theta := \exists x \in adom(T(curr, x) \wedge |g(\lambda, \chi, curr) + c - x| \leq B).$$

3 REALS AND LINEAR ARITHMETICS

This section deals with real-closed field $\mathbb{R}_{\times,+} = \langle \mathbb{R}; +, \times, \leq, 1, 0 \rangle$ and integer linear arithmetic $\mathbb{Z}_{LA} = \langle Z; +, \leq, 1, 0, \{\equiv_k\} \rangle$, where $\equiv_k$ is the congruence modulo k relation. Both admit (effective) quantifier elimination (QE) [24, 31]: each formula $\varphi(\bar{x})$ can be effectively converted into an equivalent quantifier-free formula in the theory. Notice that each side of an (in)equation can be any general term in the theory: a multivariate polynomial with integer coefficients in the case of $\mathbb{R}_{\times,+}$, and a linear multivariate polynomial with integer coefficients in the case of $\mathbb{Z}_{LA}$. In the case of $\mathbb{R}_{\times,+}$ each element in the database is a rational number, i.e., given as a/b with two integers a, b. The size of a number is the number of bits required to represent it. Our main result is:

Theorem 3.1. *The data complexity of* $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{R}_{\times,+}, \sigma)$ *and* $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{Z}_{LA}, \sigma)$ *is NL-complete.*

The remainder of §3 presents a sketch of proof of Theorem 3.1.

Restricted Quantifier Collapse. We recall the notion of Restricted Quantifier Collapse (RQC) only for theories $\mathbf{T}$ that admit QE; see [39, Definition 13.5] for a general definition, which is not crucial for the paper, and [39, Chapter 13.4] for positive and negative examples as well as some properties of RQC. We also give a definition that applies to fragments of FO. The logic $\mathbb{L} \subseteq \mathrm{FO}(\mathbf{T}, \sigma)$ *admits RQC* if $\mathbb{L} \subseteq \mathrm{FO}_{\mathrm{act}}(\mathbf{T}, \sigma)$, i.e., for every formula $\varphi(\bar{x}) \in \mathbb{L}$, there exists $\psi(\bar{x}) \in \mathrm{FO}_{\mathrm{act}}(\mathbf{T}, \sigma)$ (i.e. without unrestricted quantifiers) such that $[\![\psi]\!]_{\mathbf{S}} = [\![\varphi]\!]_{\mathbf{S}}$ for each finite $(\mathbf{T}, \sigma)$-structure $\mathbf{S}$. We write "effectively admits RQC" if the translation from φ to ψ is effective.

Proposition 3.2 ([9, 26]). *Both* $\mathrm{FO}(\mathbb{R}_{\times,+}, \sigma)$ *and* $\mathrm{FO}(\mathbb{Z}_{LA}, \sigma)$ *effectively admit RQC, and have data complexity* TC^0.

The proofs of RQC can also be found in the textbook [39, Theorem 13.19] and in the textbook [27, Proposition 5.32], from which data complexity immediately follows, e.g. see [9, 10] for $\mathrm{FO}(\mathbb{R}_{\times,+}, \sigma)$. The same analysis as in [9, 10], combined with that modulo comparisons are in TC^0 (e.g. see [47]), also gives TC^0 data complexity for $\mathrm{FO}(\mathbb{Z}_{LA}, \sigma)$. RQC provides us sometimes with a tool for coming up with an NL algorithm for RDPQ, as we shall see the case for $\mathbb{R}_{\times,+}$ and $\mathbb{Z}_{LA}$, but this is not always the case (see next section).

Restricted Register Collapse. We define an analogue of RQC for $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$. Let $\mathrm{RDPQ}_{\mathrm{act}}(\mathbb{L}, \mathbf{T}, \sigma)$ refer to queries $x \to_{\mathcal{A}} y$ in $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$ such that $\mathcal{A}$ has no parameters and $\mathbb{L}$ admits RQC. We say that $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma)$ admits *Restricted Register Collapse (RRC)* if $\mathrm{RDPQ}(\mathbb{L}, \mathbf{T}, \sigma) \subseteq \mathrm{RDPQ}_{\mathrm{act}}(\mathrm{FO}_{\mathrm{act}}, \mathbf{T}, \sigma)$. The following lemma is crucial in obtaining NL data complexity of $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{R}_{\times,+}, \sigma)$ and $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{Z}_{LA}, \sigma)$.

Lemma 3.3. *Both* $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{R}_{\times,+}, \sigma)$ *and* $\mathrm{RDPQ}(\mathrm{FO}, \mathbb{Z}_{LA}, \sigma)$ *admit RRC effectively.*

The proof of the lemma is quite technical, and borrows some ideas from the proofs of RQC for $\mathrm{FO}(\mathbb{R}_{\times,+}, \sigma)$ and $\mathrm{FO}(\mathbb{Z}_{LA}, \sigma)$. Essentially, it is shown that one can restrict the parameter valuations to a bounded number of values while preserving equivalence. Although the exact values might depend on the database $\mathbf{S}$, they can be uniformly FO-defined through the values of active domains, i.e., in $\mathrm{FO}_{\mathrm{act}}(\mathbb{R}_{\times,+}, \sigma)$ or $\mathrm{FO}_{\mathrm{act}}(\mathbb{Z}_{LA}, \sigma)$. This allows us to replace each parameter by active-domain registers.

We now proceed the proof of RRC for $\mathbf{T} := \mathbb{R}_{\times,+}$; the proof for $\mathbb{Z}_{LA}$ is similar (see appendix). Fix a vocabulary σ and a $(\mathbf{T}, \sigma)$-register automaton

$$\mathcal{A} = (\mathcal{R}, \mathcal{P}, Q, Q_0, F, \Delta)$$

with $\mathcal{P} = \{p_1, \ldots, p_k\}$. By RQC of $\mathrm{FO}(\mathbf{T}, \sigma)$, it follows that each formula $\varphi(curr, \mathcal{P}, \mathcal{R}, \mathcal{R}')$ in Δ can be assumed to have only active-domain quantifiers. We shall construct a $(\sigma, \mathbf{T})$-register automaton

$$\mathcal{A}' = (\mathcal{R}', \mathcal{P} \setminus \{p_k\}, Q', Q_0', F', \Delta'),$$

such that $[\![\mathcal{A}]\!]_{\mathbf{G}} = [\![\mathcal{A}']\!]_{\mathbf{G}}$ for all $\mathbf{T}$-embedded graph $\mathbf{G}$ over σ. By recursively eliminating the parameters, we will obtain an equivalent $(\sigma, \mathbf{T})$-register automaton with no parameters. Let $Z = \mathcal{P} \cup \mathcal{R} \cup \mathcal{R}' \cup \{curr\}$ and $Z_k := Z \setminus \{p_k\}$.

We may assume (e.g. see proof of [39, Theorem 13.19]) that each $\varphi(curr, \mathcal{P}, \mathcal{R}, \mathcal{R}')$ is of the form

$$Q_1 y_1 \in adom \ldots Q_m y_m \in adom\, \alpha(\bar{y}, p_k, Z_k)$$

where each $Q_1, \ldots, Q_m$ denotes an $\forall$ or $\exists$ quantifier and α is a Boolean combination of formulas of

(1) atomic σ-formulas $R(\bar{u})$ with $\bar{u} \subseteq \bar{y}$ (e.g. $E(p_k, r)$ is replaced by $\exists z, z' \in adom(E(z, z') \wedge z = p_k \wedge z' = r)$); and
(2) expressions of the form $t(\bar{y}, Z_k, p_k) \sim 0$, where t is polynomial with integer coefficients whose variables are among $\bar{y}$, Z_k and p_k, and $\sim \in \{=, >\}$.

We will further assume that each term t that appears in some formula (of a transition) in $\mathcal{A}$ appears in *each* formula in $\mathcal{A}$, e.g., if t does not appear in a formula ψ, we may simply rewrite it with $\psi \wedge (t = 0 \vee t \neq 0)$. By the same token, we may for simplicity assume that each formula φ in $\mathcal{A}$ uses the same number m of active-domain variables $y_1, \ldots, y_m$. Letting $Y_k = \bar{y} \cup Z_k$, we may enumerate all polynomial terms t involving p_k occurring in $\mathcal{A}$ as

$$t_1(Y_k, p_k), \ldots, t_m(Y_k, p_k),$$

where t_i has degree d_i in p_k. We introduce two new active-domain registers $r^1(z)$ and $r^2(z)$ for each $z \in Y_k$. Let $r^i(Y_k)$ denote the set containing each $r^i(z)$ with $z \in Y_k$. The crux is to replace p_k by

either one of the terms in the set S defined as

$$\left\{ \frac{\text{root}_i^s(r^1(Y_k)) + \text{root}_j^t(r^2(Y_k))}{2} : \begin{array}{c} i,j \in [1,m], \\ s \in [1,d_i],\ t \in [1,d_j] \end{array} \right\}$$
$$\cup \{\text{root}_i^s(r^1(Y_k)) + c : c \in \{-1,1\} \wedge i \in [1,m], s \in [1,d_i]\}$$

where $\text{root}_i^s(Y_k)$ represents the sth root of $t_i(Y_k, p_k)$ (treated as a univariate polynomial in p_k), if exists, otherwise 0. It is easy to see that there is a FO($\mathbb{R}_{\times,+}$) formula $\text{Root}_i^s(Y_k, x)$ which is true iff x equals $\text{root}_i^s(Y_k)$. By quantifier elimination, we may assume $\text{Root}_i^s(Y_k, x)$ is quantifier-free. At this point, we state a result from embedded finite model theory, which we will use later:

Proposition 3.4 (See Lemma 13.20 of [39]). *Let* $\mathbf{S}$ *be an finite structure such that* $adom(\mathbf{S}) \neq \emptyset$ *embedded into* $\mathbf{T} = \mathbb{R}_{\times,+}$. *For each valuation* $\mu : Z_k \to \mathbb{R}$, *it is the case that* $\mathbf{S}, \mu \models_{\mathbf{T}} \exists p_k \varphi$ *iff* $\mathbf{S}, \mu \models_{\mathbf{T}} \bigvee_{t \in S} \varphi[t/p_k]$, *where* $\varphi[t/p_k]$ *is* φ *but with* p_k *replaced by* t.

Note that, in this proposition, we interpret an active-domain register as an active-domain variable.

We now complete the construction of the new register automaton that replaces p_k by active domain registers. Define $Q' := Q \times S$, $Q'_0 = Q_0 \times S$, and $F'_0 := F_0 \times S$. For each transition $(q, \varphi(Z_k, p_k), q')$ and each $t \in S$, we add to Δ' the transition

$$((q,t), \varphi'(Z_k, r^1(Y_k), r^2(Y_k), r^1(Y'_k), r^2(Y'_k)), (q', t))$$

where φ' is constructed as follows depending on t:

- $t = \frac{\text{root}_i^s(r^1(Y_k)) + \text{root}_j^t(r^2(Y_k))}{2}$. Let $\varphi' := \bigwedge_{\alpha=1}^2 r^\alpha(Y_k) = r^\alpha(Y'_k) \wedge$
$$\exists z, z_1, z_2 \left(\begin{array}{cc} \varphi(Z_k, z) \wedge & Root_i^s(r^1(Y_k), z_1) \wedge \\ Root_j^t(r^2(Y_k), z_2) & \wedge\ 2z = z_1 + z_2 \end{array} \right)$$
- $t = \text{root}_i^s(r^1(Y_k)) + c$. Let $\varphi' := \bigwedge_{\alpha=1}^2 r^\alpha(Y_k) = r^\alpha(Y'_k) \wedge$
$$\exists z, z_1 (\varphi(Z_k, z) \wedge Root_i^s(r^1(Y_k), z_1) \wedge z = z_1 + c)$$

By RQC of FO($\mathbb{R}_{\times,+}, \sigma$), we may assume each φ' has only active-domain quantifiers.

We claim now that $[\![\mathcal{A}]\!]_{\mathbf{G}} = [\![\mathcal{A}']\!]_{\mathbf{G}}$ for all $(\mathbf{T}, \sigma)$-graph $\mathbf{G}$. The direction $[\![\mathcal{A}]\!]_{\mathbf{G}} \supseteq [\![\mathcal{A}']\!]_{\mathbf{G}}$ is immediate since $\mathcal{A}'$ restricts the range of values that are permitted to p_k in $\mathcal{A}$ (i.e. to values in S, where each $r^h(Y_k)$ can be instantiated by any active-domain values). To prove the other direction $[\![\mathcal{A}]\!]_{\mathbf{G}} \subseteq [\![\mathcal{A}']\!]_{\mathbf{G}}$, assume an accepting computation of $\mathcal{A}$

$$\pi := (q_0, \nu_0) \to_{(a_1,\varphi_1)} \cdots \to_{(a_n,\varphi_n)} (q_n, \nu_n)$$

for a valuation $\mu : \mathcal{P} \to \mathbb{R}$, i.e., $(\mathbb{R}_{\times,+}, \mathbf{G}), \mu \models \exists p_k \Phi(Z_k, p_k)$, where $\Phi(Z_k, p_k)$ is:

$$\exists \bar{s}, \bar{R} \in adom \bigwedge_{i=0}^{n-1} (\varphi_i(s_i, R_i, R_{i+1}, \mathcal{P}) \wedge E_{a_i}(s_i, s_{i+1})).$$

By assumption, Φ shares the same polynomial terms with any one of φ_i (up to renaming *curr*, R and R' with respectively s_i, R_i, and R_{i+1}). Using Proposition 3.4, there exist $i, j \in [1,m]$, $i \in [1,d_i]$, $j \in [1,d_j]$, and two tuples $\bar{a}, \bar{b}$ over $adom(\mathbf{G})$, each of length $|Y_k|$, such that $(\mathbb{R}_{\times,+}, \mathbf{G})$ with valuation μ satisfies

$$\Phi(Z_k, (\text{root}_i^s(\bar{a}) + \text{root}_j^t(\bar{b}))/2) \vee \bigvee_{c \in \{-1,1\}} \Phi(Z_k, \text{root}_i^s(\bar{a}) + c)$$

Hence, $\mathcal{A}'$ may simply instantiate $r^1(Y_k)$ (resp. $r^2(Y_k)$) with $\bar{a}$ (resp. $\bar{b}$) and keep them constant throughout the computation, as is done in the definition of $\mathcal{A}'$. Extend each ν_i with this instantiation yielding ν'_i. Let $\mu' := \mu_{|_{Z_k}}$ be the restriction of μ to Z_k. The following path is then an accepting path of $\mathcal{A}'$ with valuation μ':

$$\pi' := (q_0, \nu'_0) \to_{(a_1,\delta'_0)} \cdots \to_{(a_n,\delta'_n)} (q_n, \nu_n).$$

This completes the proof of Theorem 3.3.

NL algorithm. Theorem 3.1 is a direct consequence of the following lemma, Proposition 3.2, and Lemma 3.3.

Lemma 3.5. *We assume a sublogic* $\mathbb{L}$ *of* FO($\mathbf{T}, \sigma$) *with NL data complexity. Then,* RDPQ($\mathbb{L}, \mathbf{T}, \sigma$) *without parameters is NL-complete. It remains NL if we allow parameters that can be instantiated by data values whose sizes are logarithmically bounded.*

Proof idea. One nondeterministically simulates a run of $\mathcal{A}$ in a query $x \to_{\mathcal{A}} y$ over RDPQ($\mathbb{L}, \mathbf{T}, \sigma$), while using the NL algorithm for $\mathbb{L}$ as a subprocedure. References to active-domain values are saved in the working tape, hence keeping the algorithm running in NL. Full proof is in the appendix. □

4 THEORY OF AUTOMATIC RELATIONS

We consider the theory of automatic relations on finite words. Let Γ be a finite alphabet, and $\Gamma_\perp$ be $\Gamma \dot\cup \{\perp\}$. Given $u_1, \ldots, u_k \in \Gamma^*$, the *convolution* $u_1 \otimes \cdots \otimes u_k$ of these words is defined as the word $w \in (\Gamma_\perp^k)^*$ such that the projection onto the i-th component of w yields $u_i \cdot \perp^{|w|-|u_i|}$, for every $1 \leq i \leq k$. An *automatic relation* of arity k is a relation $R \subseteq (\Gamma^*)^k$ such that

$$R = \{(u_1, \ldots, u_k) : u_1 \otimes \cdots \otimes u_k \in L\} \tag{1}$$

for some regular language L over $(\Gamma_\perp)^k$. Let *Rel* be the set of all automatic relations over some fixed alphabet Γ. $\mathbf{T}_{\text{Aut}}$ is then the theory $\langle \Gamma^*; \{R\}_{R \in \text{Rel}} \rangle$. Some examples of automatic relations are the prefix relation $\cdot \preceq \cdot$, the equal length relation *eq-len*$(\cdot,\cdot)$ and the unary relation $lst_a(\cdot)$ stating that the last letter of the word is an $a \in \Gamma$. In fact, in the relational calculus, these relations coincide with the automatic relations, in the sense that the following are equivalent: (1) R is an automatic relation from Rel of arity k; (2) there is $\varphi(x_1, \ldots, x_k) \in \text{FO}(\Gamma^*, \preceq, \textit{eq-len}, \{lst_a\}_{a \in \Gamma})$ such that $R = [\![\varphi]\!] = \{\bar{u} \in (\Gamma^*)^k : \Gamma^* \models \varphi[\bar{u}]\}$ [23]. This is why FO($\Gamma^*, \preceq, \textit{eq-len}, \{lst_a\}_{a \in \Gamma}$) is sometimes called the "universal automatic structure" [12]. Another example of an automatic relation is the *edit-distance-k* binary relation (see [7]), for any fixed k. Observe that regular languages are exactly the class of automatic relations of arity 1.

We assume that words are represented as a list of letters, and relations as lists of lists of words. The *size* of a word is its length. We represent automatic relations inside formulas via an NFA recognizing the language L in (1).

Example 4.1. Consider a phylogenetic database, with each node defining a DNA sequence (and perhaps some other data). Using FO($\mathbf{T}_{\text{Aut}}, \sigma$) we can query, whether the DNA sequence of the current element is at edit distance at most 5 from parameter p:

$$\varphi(\textit{curr}, p) = \exists y\ \textit{dna-seq}(\textit{curr}, y) \wedge \textit{edit-distance-5}(y, p),$$

where *dna-seq*(*curr*, *y*) specifies that y is the DNA sequence of the current node *curr* and *edit-distance-5*$(\cdot,\cdot)$ is an automatic relation (of being at edit distance at most 5).

Unfortunately, model checking of FO over $\mathbf{T}_{\text{Aut}}$-embedded structures is hard for the polynomial hierarchy (PH) [11, Prop. 4.12], implying untractability for RDPQ querying:

Proposition 4.2 ([11]). RDPQ(FO, $\mathbf{T}_{\text{Aut}}, \sigma$) *is hard for every level of PH in data complexity, even without parameters.*

Hence, we focus on the EFO fragment of FO-formulas in prenex form using no universal quantifiers. The set of EFO formulas using no negation is denoted by EFO^+.

Lemma 4.3 (proof in Appendix). EFO($\mathbf{T}_{\text{Aut}}, \sigma$) *has effective RQC.*

Thus, model checking for EFO is tractable, since model checking for $\text{FO}_{\text{act}}(\mathbf{T}_{\text{Aut}}, \sigma)$ is in NC^1 (proof in Appendix).

Lemma 4.4. *[11, proof of Corollary 4.15] The model checking problem for* $\text{FO}_{act}(\mathbf{T}_{\text{Aut}}, \sigma)$ *is in* NC^1 *in data complexity.*

By Lemma 4.3, every EFO($\mathbf{T}_{\text{Aut}}, \sigma$) formula is effectively equivalent to a $\text{FO}_{\text{act}}(\mathbf{T}_{\text{Aut}}, \sigma)$ formula which in turn, by Lemma 4.4, is in NC^1, hence in NL. Thus, by Lemma 3.5 the bound transfers to RDPQ evaluation.

Corollary 4.5 (of Lemma 4.3, Lemma 4.4, and Lemma 3.5). *Evaluation of* RDPQ(EFO, $\mathbf{T}_{\text{Aut}}, \sigma$) *and* RDPQ($\text{FO}_{act}, \mathbf{T}_{\text{Aut}}, \sigma$) *without parameters is NL-complete in data complexity.*

However, as soon as unrestricted parameters are allowed, we lose tractability, as we show next (proof in Appendix E).

Lemma 4.6. *Evaluation of* RDPQ($\text{EFO}^+, \mathbf{T}_{\text{Aut}}, \sigma$) *is NP-hard in data complexity, even with no registers and one parameter.*

To regain tractability we assume LogH on the $(\mathbf{T}_{\text{Aut}}, \sigma)$-structure $\mathbf{S}$, that is, that for some constant c, every word $w \in adom$ satisfies $|w| \leq c \cdot \log(|\mathbf{S}|)$. In fact, LogH implies that parameters can be taken to have a logarithmic size, and in light of Lemmas 4.3 and 3.5, we deduce that RDPQ(EFO, $\mathbf{T}_{\text{Aut}}, \sigma$) evaluation (under LogH) is in NL. To prove this, we use a corollary of the proof of Lemma 4.3 (in Appendix G):

Corollary 4.7 (of Lemma 4.3). *Every* EFO($\mathbf{T}_{\text{Aut}}, \sigma$) *formula* $\psi(\bar{z})$ *is effectively equivalent to a disjunction of* $\text{EFO}_{act}(\mathbf{T}_{\text{Aut}}, \sigma)$ *formulas* ψ' *of the form* $A'(\bar{z}) \wedge \exists \bar{r} \in adom\ \tau(\bar{z}\bar{r})$, *where* A' *is a* $\mathbf{T}_{\text{Aut}}$*-atom,* τ *is a* $\text{EFO}_{act}(\sigma)$ *formula.*

Let $N = \max_{w \in adom} |w|$ *and let* $v : \bar{z} \to \Gamma^*$ *be a satisfying assignment for* $\psi'(\bar{z})$, *and* $z \in \bar{z}$ *such that* $|v(z)| > N$. *Suppose there is a word* $u \in \Gamma^*$ *such that* $|u| > N$ *and* $\Gamma^*, v' \models A'(\bar{z})$ *for* $v' = v[z \mapsto u]$. *Then,* v' *is also a satisfying assignment for* ψ'.

Theorem 4.8. *Evaluation of* RDPQ(EFO, $\mathbf{T}_{\text{Aut}}, \sigma$) *with parameters under LogH is NL-complete in data complexity.*

Proof. Let $\mathbf{G}$ be a $\mathbf{T}_{\text{Aut}}$-embedded graph over σ, $F = \exists \mathcal{P} : x \to_{\mathcal{A},\mathcal{P}} y$ be an RDPQ(EFO, $\mathbf{T}_{\text{Aut}}, \sigma$) query, and $(s,t) \in V^2$ be two vertices of $\mathbf{G}$. Let N be the maximum size of a word in the active domain (of logarithmic size by LogH).

Let $s = v_0 \to_{a_1} \cdots \to_{a_n} v_n = t$ be an accepting $\mathbf{G}$-path with transitions $t_0, \ldots, t_n$, where $t_i = (q_i, (a_i, \psi_i), q_{i+1})$ and $a_0 = \#$. Then for some register valuations $\mu_0, \ldots, \mu_{n+1} \in V^{\mathcal{R}}$ the formula $\varphi(\mathcal{P}) = \bigwedge_i \psi_i(v_i, \mathcal{P}, \mu_i, \mu_{i+1})$ is satisfiable. We show that if φ is satisfiable, then it is satisfied with a valuation for $\mathcal{P}$ with log-size words.

Suppose there is a parameter valuation $\nu \in U^{\mathcal{P}}$ which satisfies φ. By Corollary 4.7, for each i, ψ_i can be written as a disjunction of formulas of the form $R_i(\bar{z}) \wedge \exists \bar{r} \in adom\ \tau(\bar{z}\bar{r})$ where R_i is a $\mathbf{T}_{\text{Aut}}$-atom and τ is a $\text{FO}_{\text{act}}(\sigma)$ formula. Consider the DFA A_i over $\Gamma_\perp^{n_i}$ corresponding to the relation denoted by R_i, let it have n_i free variables. Let $\bar{v}_i$ be a satisfying assignment for $\bar{y}_i$. Hence, there is an accepting run of A_i over the convolution w of the words $v_i \nu \mu_i \mu_{i+1} \bar{v}_i$. These words are of logarithmic size, since they are from *adom*, with a possible exception of those from ν, which are parameters. Let w' be the N-prefix of w, and let q_i be the state of A_i after reading w' from the initial state. We define A_i' to be the same automaton as A_i but (1) setting q_i as initial state, and (2) removing any transition reading a non-$\perp$ symbol on any component which is not of the parameters. It then follows that, if w'' is any word in the language of A_i', then $w' \cdot w''$ is in the language of A_i. Let $\pi(A_i')$ be the automaton corresponding to the projection onto the components of the parameters of $L(A_i')$. Consider a word $u \in \bigcap_i L(\pi(A_i')) \subseteq (\Gamma_\perp^{|\mathcal{P}|})^*$; observe that its description is bounded by a function of the (fixed) RDPQ F, and hence u is of constant size. It then follows, by Corollary 4.7, that $u' \cdot u$ witnesses also a satisfying assignment for φ, where u' is the N-prefix of ν. This shows that we can assume that the parameter valuation for witnessing a word in the language is also of logarithmic size. Since parameters can be taken to have a logarithmic size, by Lemmas 4.3, 4.4, and 3.5 we get an NL bound. □

5 WORD EQUATIONS

This section deals with the theory of word equations with concatenation and regular languages, i.e., let $\mathbf{T}_{\text{WE}}$ be the theory of $\langle \Gamma^*; \cdot, \{R\}_{R \in \text{Reg}} \rangle$, where Γ is a finite alphabet, $\cdot$ is the concatenation, and Reg is the set of all regular languages (over Γ). The data complexity of FO($\mathbf{T}_{\text{WE}}, \sigma$) is undecidable [11], so we consider two fragments of FO($\mathbf{T}_{\text{WE}}, \sigma$): (1) $\mathbb{L}^+_{\text{WE}} = \text{EFO}^+_{\text{act}}(\mathbf{T}_{\text{WE}}, \sigma)$ of existential positive formula with active-domain quantifiers, and (2) $\mathbb{L}_{\text{PM}}$ of existential positive formulas without regular relations and whose allowed $\mathbf{T}_{\text{WE}}$-formulas have atoms of the form $\exists \bar{y}(\bigwedge_{i \in I} x_i = \beta_i)$ with the left-hand side x_i being a single element from $\mathit{curr} \cup \mathcal{R} \cup \mathcal{R}' \cup \mathcal{P}$ and the right-hand side β_i being a concatenation of elements in $\bar{y} \cup \mathcal{R} \cup \mathcal{R}' \cup \{\mathit{curr}\}$ and constants, and that each existentially quantified variable occurs at most once in an equation whose left-hand side is from $\mathcal{P}$. (Note that it can occur at multiple equations in which the left-hand side is from $\mathit{curr} \cup \mathcal{R} \cup \mathcal{R}'$.)

Example 5.1. Consider a query in a phylogenetic database, defined in Example 4.1: *is there a path, such that all DNA sequences on it are at edit distance at most k from some (unknown) DNA sequence*?

Answering such query is related to the closest string problem [15, 28]. It is known [7, Eq. (3)], that the edit distance between p, w is at most k if and on only if

$$\bigvee_{\substack{a_1,\ldots,a_k \in \Gamma \cup \{\varepsilon\} \\ b_1,\ldots,b_k \in \Gamma \cup \{\varepsilon\}}} \exists x_0, \ldots, x_k \;\; p = x_0 a_1 x_1 \cdots x_{k-1} a_k x_k \wedge w = x_0 b_1 x_1 \cdots x_{k-1} b_k x_k \;\; .$$

When p is a parameter and w in *curr*, this formula is in $\mathbb{L}_{\text{PM}}$ and so this query is expressible in RDPQ($\mathbb{L}_{\text{PM}}, \mathbf{T}_{\text{WE}}, \sigma$).

Theorem 5.2. RDPQ($\mathbb{L}^+_{\mathrm{WE}}, \mathbf{T}_{\mathrm{WE}}, \sigma$) *has* NL*-complete data complexity, similarly* RDPQ($\mathbb{L}_{\mathrm{PM}}, \mathbf{T}_{\mathrm{WE}}, \sigma$).

Proof idea. As in both cases the guards on a G-path are positive formulas, the guard on any G-path π are a positive Boolean combination of word equations over some variables and traversing such a path can be seen as constructing a system of word equations. In particular, given a valuation of the parameters and existentially quantified variables such that $\mathbf{G}, \pi \models \mathcal{A}$ we can determine, which atomic equations were true and this valuation can be seen as a solution to the system of word equations (i.e. the equations that are made true in the guards). Hence evaluating a query can be seen as solving a system of word equations, though this system is not given explicitly. The best algorithms solving word equations run in PSPACE, so in order to answer RDPQ queries in NL, we need some insight into the structure of the systems of equations that we obtain in this case. Those turn out to be different for $\mathbb{L}^+_{\mathrm{WE}}$ and $\mathbb{L}_{\mathrm{PM}}$.

For $\mathbb{L}_{\mathrm{PM}}$: each atomic equation is of the form $p = \beta$ or $\alpha = \beta$, where p is a parameter, $\alpha \in \mathit{curr} \cup \mathcal{R} \cup \mathcal{R}'$ and β is a sequence of (existentially quantified) variables and elements from $\Gamma \cup \mathit{curr} \cup \mathcal{R} \cup \mathcal{R}'$. In the second case $\alpha \in \mathit{adom}(\mathbf{S})$, so every (existentially quantified) variable in β is a substring of an element in $\mathit{adom}(\mathbf{S})$, so it can be represented in NL. For equations $p = \beta$, fix a parameter p and consider all such equations in the system created while traversing the G-path. Every variable at the right-hand side is used once in the system and not used in any other such subsystem. Such system is satisfiable if and only if each of the equations satisfies a condition on a prefix and suffix of its sides; in particular, we do not need to consider all the equations simultaneously, they can be checked one by one, see Lemma H.1 in the appendix. The NL algorithm guesses for each parameter p an appropriate prefix and suffix, it can be shown that those can be represented as concatenations of at most $|\mathcal{A}|$ many constants and substrings of elements from $\mathit{adom}(\mathbf{S})$. Then in G it guesses consecutive edge views and evaluates the guard φ using the condition on prefixes and suffixes mentioned above, i.e. the one from Lemma H.1.

Concerning $\mathbb{L}^+_{\mathrm{WE}}$, for simplicity of presentation, we ignore regular relations; the appendix contains a brief discussion how to generalize to this case. The (existential) active domain quantifiers are instantiated by elements of $\mathit{adom}(\mathbf{S})$ using nondeterministic guesses. Therefore the equations occurring in guards are only in variables that are parameters. We first observe that if words substituted for parameters are polynomial-length and there is an NL access to individual letters in them, then RDPQ($\mathbb{L}^+_{\mathrm{WE}}, \mathbf{T}_{\mathrm{WE}}, \sigma$) has NL data complexity, as desired.

Lemma 5.3. *Assume a logarithmic-size representation of valuations of parameters $\mathcal{P}$, such that each parameter is of polynomial (in $|\mathbf{G}|$) length and individual letter of each parameter and length of each parameter can be accessed in* NL*. Then we can evaluate a given* RDPQ($\mathbb{L}^+_{\mathrm{WE}}, \mathbf{T}_{\mathrm{WE}}, \sigma$) *query for those parameters on* G *in* NL.

Proof. We guess consecutive vertices on a G-path. At a given node we guess the next node in G, the letter and transition of $\mathcal{A}$, which yields a guard φ that should be satisfied. To verify φ, we guess the existentially quantified variables, which are an element from *adom*, and then evaluate the atomic equations in the guards one by one. For a given equation $u(\bar{x}) = v(\bar{x})$ we already have the substitutions for each variable in $\bar{x}$, which is either an active-domain quantified (so was guessed) or a parameter (so we have a valuation for it). Hence we verify the equation symbol by symbol, which can be done in NL, as each substituted value is of polynomial length and individual letters can be accessed in NL. As each atom of φ is evaluated in NL, we can also evaluate φ in NL.

If we have ended in t then the traversed path π yields $\mathbf{G}, \pi \models \mathcal{A}$. On the other hand, if there is π such that $\mathbf{G}, \pi \models \mathcal{A}$ then we guess π's consecutive nodes, the transitions of $\mathcal{A}$ and substitutions for variables, for which we end up at t, as desired. □

We show that if there is an accepting G-path π, and parameter valuation $\nu : \mathcal{P} \to \Gamma^*$ such that $\mathbf{G}, \pi \models \mathcal{A}$ then this also holds for a valuation ν' such that each assigned value is polynomial-size and its letters can be accessed in NL.

So assume such a path π and corresponding sequence of transitions in $\mathcal{A}$. For any guard φ take the equations made true under ν, they form a system of word equations in k variables $\mathcal{P}$; the sum of lengths of equations is polynomial in $|\mathbf{G}|$: the length $|\pi|$ is polynomial (discussion after Proposition 2.3), and each atom in it is from $\Gamma^* \cup \mathit{curr} \cup \mathcal{R} \cup \mathcal{R}'$, hence of linear size. Lemma H.5 in the Appendix shows that such a system has a polynomial-size solution. Moreover, it is known [44] that for (some) such solutions the *length* of the substitutions together with the system of equations uniquely determines the solution and yield access to letters in the substitutions, in this case the access is in NL; see the appendix. However, the *size* of this representation is polynomial, as there are (potentially) polynomially many equations.

We exploit the form of equations in the system: fix an atomic equation $u(\bar{p}, \bar{x}) = v(\bar{p}, \bar{x})$, where $\bar{p}$ are from $\mathcal{P}$ and $\bar{x}$ are from $\mathit{curr}, \mathcal{R}, \mathcal{R}'$ and active domain quantified variables. For different substitutions for $\bar{x}$, the obtained equations have the same sequence of variables (from $\mathcal{P}$), separated with different words; call such equations *similar*. We show that given two similar equations we can deduce a partial valuation of variables, which makes those two equations equivalent (in appropriate sense). By considering consecutive equations we get a partial substitution that makes increasing set of equations equivalent. Moreover, this process "simplifies" the resulting equation and we can perform it $O(m)$ times, where m is the number of occurrences of variables in the equation. As a result, among a system of similar equations we can choose $O(m)$ which unify this system. This is done for each equation $u(\bar{p}, \bar{x}) = v(\bar{p}, \bar{x})$ in $\mathcal{A}$, yielding a system of $O(m|\mathcal{A}|)$ equations, which can be used to access the substitution for parameters. See appendix for details. □

6 GENERALIZATIONS AND CONCLUSIONS

We conclude by mentioning related issues and future work.

Conjunctive Queries. NL data complexity of RDPQ easily extends to CRDPQ [40], which enriches RDPQ by conjunctions. Here we enrich RDPQ($\mathbb{L}, \mathbf{T}, \sigma$) not only by conjunctions, but also by an additional constraint in $\mathbb{L}$. Namely, queries in CRDPQ($\mathbb{L}, \mathbf{T}, \sigma$) are conjunctions of the form

$$Q(\bar{z}) \leftarrow \bigwedge_{i=1}^{n} x_i \to_{\mathcal{A}_i} y_i \wedge \varphi(\mathcal{P}, \bar{x}, \bar{y})$$

where $\bar{z} \subseteq \bar{x} \cup \bar{y}$, $x_i \rightarrow_{\mathcal{A}_i} y_i$ is an RDPQ($\mathbb{L}, \mathbf{T}, \sigma$)-query, and $\varphi(\mathcal{P}, \bar{x}, \bar{y}) \in \mathbb{L}$. Here, $\mathcal{P}$ contains all parameters used across all register automata in Q, which may among themselves share parameters. Given a $(\mathbf{T}, \sigma)$-structure $\mathbf{S}$ with active domain U, let $[\![Q]\!]_{\mathbf{S}}$ be the set of all $\beta : \bar{z} \to U$ such that for some $\nu : \bar{x} \cup \bar{y} \to U$, with $\beta(z) = \nu(z)$ for each $z \in \bar{z}$, and $\mu : \mathcal{P} \to \mathcal{D}$, it is the case that $\mathbf{S} \models \varphi(\mu(\mathcal{P}), \nu(\bar{x}), \nu(\bar{y}))$ and $\nu(x_i) \rightarrow_{\mathcal{A}_i} \nu(y_i)$. In other words, each path constraint $x_i \rightarrow_{\mathcal{A}_i} y_i$ is satisfied where a parameter valuation μ that conforms to φ. One example of a CRDPQ query is the query that there are two actors with finite Bacon graphs whose ages differ from Bacon by an even number k, i.e., this is of the form

$$Q(y, z) \leftarrow x \rightarrow_{\mathcal{A}} y \wedge x \rightarrow_{\mathcal{A}} z \wedge k \equiv 0 \pmod 2 \wedge x = \text{Bacon},$$

where $\mathcal{A}$ uses one active-domain register r to save the birthyear y_0 of the first person in the path, and a parameter k that is nondeterministically guessed and is checked against the difference between y_0 and the birthyear of the final person in the path. We sketch in the appendix a log-space reduction from CRDPQ($\mathbb{L}, \mathbf{T}, \sigma$) to RDPQ($\mathbb{L}, \mathbf{T}, \sigma$) query evaluation, where the output query depends only on the input query. This extends NL data complexity to CRDPQ($\mathbb{L}, \mathbf{T}, \sigma$).

Property Graphs. Property graphs are the data models used by most modern graph database systems [17]. A property graph associates unique IDs to both nodes and edges in a graph, i.e., they are both first-class citizens. The main difference to the standard data graph model in database theory (e.g., see [40]) is three-fold. First, some edges could be unordered and there could be multiple edges from a node to another node. second, nodes (as well as edges) may also carry labels from Σ. Finally, edges (as well as nodes) may be associated with a property value. Since we allow first-order views, we may easily view a property graph $\mathbf{G}$ as a data graph $\mathbf{G}'$ in the sense of [40] by interpreting nodes and edges in $\mathbf{G}$ as nodes in $\mathbf{G}'$. Indeed, view definitions allow a huge amount of flexibility, e.g., one can reverse the edge relation, make an edge relation symmetric, etc. This allows us to also obtain NL data complexity of RDPQ($\mathbb{L}, \mathbf{T}, \sigma$) on the property graph data model.

Combined Complexity. The primary concern of our paper is developing data path query languages with NL data complexity. The consideration of data complexity is standard and sensible for database query evaluation since typically the size of the database is very large and the size of the query small. The combined complexity of our query languages is higher than that of the basic RDPQ from [40] (i.e. PSPACE), primarily because of the high complexity of the theories that we consider, e.g., the best known algorithm for FO($\mathbb{R}_{\times,+}$) is double-exponential time (see [8]) and FO($\mathbb{Z}_{LA}$) would require at least double exponential-time [37]. We leave it for future work to identify fragments of FO($\mathbf{T}, \sigma$) that could match the PSPACE combined complexity of the basic RDPQ.

Integrating views into the automaton. Thus far, our presentation has kept the view definitions $G = (V, \{E_a\}_{a \in \Sigma})$ of the relational database $\mathbf{S}$ rather "separate" from the automaton $\mathcal{A}$ in the query $x \rightarrow_{\mathcal{A}} y$. In effect, the automaton $\mathcal{A}$ is forced to explore the relational database $\mathbf{S}$ through the graph views as defined by G. Although this is more in line with the data model of SQL/PGQ (e.g. see [17] and https://pgql-lang.org/), a further generalization that enables $\mathcal{A}$ to directly traverse the relational structure $\mathbf{S}$ is possible, which we will discuss next.

For a start, we dispense completely with the view definitions over the database $\mathbf{S}$. How does $\mathcal{A}$ then traverse $\mathbf{S}$? The answer is simple. Each transition of $\mathcal{A}$ can be associated with a *next* variable indicating which element in $\mathbf{S}$ it can go to, i.e., it is of the form

$$(q, \varphi(\mathit{curr}, \mathit{next}, \mathcal{R}, \mathcal{R}', \mathcal{P}, \mathcal{P}'), q'),$$

where φ is a formula over the vocabulary σ of $\mathbf{S}$. Notice that Σ does not exist, and therefore an automata transition does not have any Σ entry. The semantics of a transition is the same as defined in § 2, except that by taking this transition the automaton $\mathcal{A}$ goes from *curr* to *next*. This setting is strictly more general than the setting defined in § 2 in that the graph views allowed by this extended automaton might depend on the register and parameter values. All the results in this paper generalize easily to this extended data model.

Future Work. We now mention several open questions. Firstly, in what way can our query languages be extended with aggregation (e.g., summing values along a path, taking average, etc.) without sacrificing NL data complexity? To this end, we conjecture that decidable subclasses of counter automata (e.g. [35, 46]) could prove useful. Secondly, identify the fragment of the graph pattern matching language (GPML) over property graphs [17] that can be captured in CRDPQ($\mathbb{L}, \mathbf{T}, \sigma$). Concerning word equations, can we extend the methods used for $\mathbb{L}^{+}_{\text{WE}}$ to obtain NL data complexity for RDPQ over fragment with negation and/or with existential quantification?

ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for the helpful feedback. We also thank Michael Benedikt, Leonid Libkin, Domagoj Vrgoč, and Zhilin Wu for the discussion and insightful comments. Diego Figueira is partially supported by ANR QUID, grant ANR-18-CE40-0031. Artur Jeż work is supported under National Science Centre, Poland project number 2017/26/E/ST6/00191. Anthony Lin was supported by the ERC Starting Grant 759969 (AV-SMP) and a Max-Planck Society Fellowship.

REFERENCES

[1] Étienne André. 2019. What's decidable about parametric timed automata? *Int. J. Softw. Tools Technol. Transf.* 21, 2 (2019), 203–219. https://doi.org/10.1007/s10009-017-0467-0

[2] Renzo Angles and Claudio Gutiérrez. 2008. Survey of graph database models. *ACM Comput. Surv.* 40, 1 (2008), 1:1–1:39. https://doi.org/10.1145/1322432.1322433

[3] Pablo Barceló Baeza. 2013. Querying graph databases. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, Richard Hull and Wenfei Fan (Eds.). ACM, 175–188. https://doi.org/10.1145/2463664.2465216

[4] Jorge Baier, Dietrich Daroch, Juan L. Reutter, and Domagoj Vrgoc. 2017. Evaluating Navigational RDF Queries over the Web. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media, HT 2017, Prague, Czech Republic, July 4-7, 2017*, Peter Dolog, Peter Vojtás, Francesco Bonchi, and Denis Helic (Eds.). ACM, 165–174. https://doi.org/10.1145/3078714.3078731

[5] Abebe Alemu Balcha. 2020. Curve Fitting and Least Square Analysis to Extrapolate for the Case of COVID-19 Status in Ethiopia. *Advances in Infectious Diseases* 10 (2020). Issue 3.

[6] Pablo Barceló, Gaëlle Fontaine, and Anthony Widjaja Lin. 2015. Expressive Path Queries on Graph with Data. *Log. Methods Comput. Sci.* 11, 4 (2015). https://doi.org/10.2168/LMCS-11(4:1)2015

[7] Pablo Barceló, Leonid Libkin, Anthony Widjaja Lin, and Peter T. Wood. 2012. Expressive Languages for Path Queries over Graph-Structured Data. *ACM Trans. Database Syst.* 37, 4 (2012), 31:1–31:46. https://doi.org/10.1145/2389241.2389250

[8] Saugata Basu. 2014. Algorithms in Real Algebraic Geometry: A Survey. *CoRR* abs/1409.1534 (2014). arXiv:1409.1534 http://arxiv.org/abs/1409.1534

[9] Michael Benedikt and Leonid Libkin. 1996. On the Structure of Queries in Constraint Query Languages. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*. IEEE Computer Society, 25–34. https://doi.org/10.1109/LICS.1996.561300

[10] Michael Benedikt and Leonid Libkin. 2000. Relational queries over interpreted structures. *J. ACM* 47, 4 (2000), 644–680. https://doi.org/10.1145/347476.347477

[11] Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. 2003. Definable relations and first-order query languages over strings. *J. ACM* 50, 5 (2003), 694–751. https://doi.org/10.1145/876638.876642

[12] Achim Blumensath and Erich Grädel. 2000. Automatic Structures. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 51–62. https://doi.org/10.1109/LICS.2000.855755

[13] Achim Blumensath and Erich Grädel. 2004. Finite presentations of infinite structures: Automata and interpretations. *Theory of Computing Systems* 37, 6 (2004), 641–674.

[14] Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. 2011. Two-variable logic on data words. *TOCL* 12, 4 (2011), 27:1–27:26. https://doi.org/10.1145/1970398.1970403

[15] Christina Boucher and Bin Ma. 2011. Closest string with outliers. *BMC Bioinform.* 12, S-1 (2011), S55. https://doi.org/10.1186/1471-2105-12-S1-S55

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. 2000. Containment of Conjunctive Regular Path Queries with Inverse. In *KR 2000, Principles of Knowledge Representation and Reasoning Proceedings of the Seventh International Conference, Breckenridge, Colorado, USA, April 11-15, 2000*, Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman (Eds.). Morgan Kaufmann, 176–185.

[17] Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Hannes Voigt, Oskar van Rest, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. 2021. Graph Pattern Matching in GQL and SQL/PGQ. arXiv:2112.06217 [cs.DB]

[18] Alin Deutsch and Val Tannen. 2001. Optimization Properties for Classes of Conjunctive Regular Path Queries. In *Database Programming Languages, 8th International Workshop, DBPL 2001, Frascati, Italy, September 8-10, 2001, Revised Papers (Lecture Notes in Computer Science, Vol. 2397)*, Giorgio Ghelli and Gösta Grahne (Eds.). Springer, 21–39. https://doi.org/10.1007/3-540-46093-4_2

[19] Volker Diekert. 2002. Makanin's Algorithm. In *Algebraic Combinatorics on Words*, M. Lothaire (Ed.). Encyclopedia of Mathematics and its Applications, Vol. 90. Cambridge University Press, Chapter 12, 387–442.

[20] Volker Diekert, Claudio Gutiérrez, and Christian Hagenah. 2005. The existential theory of equations with rational constraints in free groups is PSPACE-complete. *Inf. Comput.* 202, 2 (2005), 105–140. http://dx.doi.org/10.1016/j.ic.2005.04.002

[21] Volker Diekert, Artur Jeż, and Wojciech Plandowski. 2016. Finding all solutions of equations in free groups and monoids with involution. *Inf. Comput.* 251 (2016), 263–286. https://doi.org/10.1016/j.ic.2016.09.009 Conference version in Proc. CSR 2014, LNCS 8476 (2014).

[22] Volker Diekert, Artur Jeż, and Wojciech Plandowski. 2016. Finding all solutions of equations in free groups and monoids with involution. *Inf. Comput.* 251 (2016), 263–286. https://doi.org/10.1016/j.ic.2016.09.009

[23] S Eilenberg, C.C Elgot, and J.C Shepherdson. 1969. Sets recognized by n-tape automata. *Journal of Algebra* 13, 4 (1969), 447–464. https://doi.org/10.1016/0021-8693(69)90107-0

[24] Herbert B. Enderton. 2001. *Introduction to Mathematical Logic* (2 ed.). Academic Press.

[25] Daniela Florescu, Alon Y. Levy, and Dan Suciu. 1998. Query Containment for Conjunctive Queries with Regular Expressions. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington, USA*, Alberto O. Mendelzon and Jan Paredaens (Eds.). ACM Press, 139–148. https://doi.org/10.1145/275487.275503

[26] Jörg Flum and Martin Ziegler. 1999. Pseudo-Finite Homogeneity and Saturation. *J. Symb. Log.* 64, 4 (1999), 1689–1699. https://doi.org/10.2307/2586806

[27] E. Grädel, P. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. 2007. *Finite Model Theory and Its Applications*. Springer.

[28] Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. 2003. Fixed-Parameter Algorithms for CLOSEST STRING and Related Problems. *Algorithmica* 37, 1 (2003), 25–42. https://doi.org/10.1007/s00453-003-1028-3

[29] Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. 2009. Reachability in Succinct and Parametric One-Counter Automata. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings (Lecture Notes in Computer Science, Vol. 5710)*, Mario Bravetti and Gianluigi Zavattaro (Eds.). Springer, 369–383. https://doi.org/10.1007/978-3-642-04081-8_25

[30] Jelle Hellings, Bart Kuijpers, Jan Van den Bussche, and Xiaowang Zhang. 2013. Walk logic as a framework for path query languages on graph databases. In *Joint 2013 EDBT/ICDT Conferences, ICDT '13 Proceedings, Genoa, Italy, March 18-22, 2013*, Wang-Chiew Tan, Giovanna Guerrini, Barbara Catania, and Anastasios Gounaris (Eds.). ACM, 117–128. https://doi.org/10.1145/2448496.2448512

[31] Wilfrid Hodges. 1997. *A Shorter Model Theory*. Cambridge University Press.

[32] Artur Jez. 2012. Recompression: a simple and powerful technique for word equations. *CoRR* abs/1203.3705 (2012). arXiv:1203.3705 http://arxiv.org/abs/1203.3705v1 version v1.

[33] Artur Jeż. 2016. Recompression: a simple and powerful technique for word equations. *J. ACM* 63, 1 (Mar 2016), 4:1–4:51. https://doi.org/10.1145/2743014

[34] Michael Kaminski and Nissim Francez. 1994. Finite-Memory Automata. *Theor. Comput. Sci.* 134, 2 (1994), 329–363. https://doi.org/10.1016/0304-3975(94)90242-9

[35] Eryk Kopczynski and Anthony Widjaja To. 2010. Parikh Images of Grammars: Complexity and Applications. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*. 80–89. https://doi.org/10.1109/LICS.2010.21

[36] Antoni Kościelski and Leszek Pacholski. 1996. Complexity of Makanin's Algorithm. *J. ACM* 43, 4 (1996), 670–684. https://doi.org/10.1145/234533.234543

[37] Dexter C. Kozen. 2006. *Theory of Computation*. Springer.

[38] G. M. Kuper, L. Libkin, and J. Paredaens (Eds.). 2000. *Constraint Databases*. Springer.

[39] Leonid Libkin. 2004. *Elements of Finite Model Theory*. Springer.

[40] Leonid Libkin, Wim Martens, and Domagoj Vrgoc. 2016. Querying Graphs with Data. *J. ACM* 63, 2 (2016), 14:1–14:53. https://doi.org/10.1145/2850413

[41] Leonid Libkin and Domagoj Vrgoc. 2012. Regular path queries on graphs with data. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, Alin Deutsch (Ed.). ACM, 74–85. https://doi.org/10.1145/2274576.2274585

[42] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.* 24, 6 (1995), 1235–1258. https://doi.org/10.1137/S009753979122370X

[43] Wojciech Plandowski. 2019. On *PSPACE* generation of a solution set of a word equation and its applications. *Theor. Comput. Sci.* 792 (2019), 20–61. https://doi.org/10.1016/j.tcs.2018.10.023

[44] Wojciech Plandowski and Wojciech Rytter. 1998. Application of Lempel-Ziv Encodings to the Solution of Word Equations. In *ICALP (LNCS, Vol. 1443)*, Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel (Eds.). Springer, 731–742. https://doi.org/10.1007/BFb0055097

[45] Luc Segoufin and Szymon Torunczyk. 2011. Automata based verification over linearly ordered data domains. In *28th International Symposium on Theoretical Aspects of Computer Science, STACS 2011, March 10-12, 2011, Dortmund, Germany (LIPIcs, Vol. 9)*, Thomas Schwentick and Christoph Dürr (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 81–92. https://doi.org/10.4230/LIPIcs.STACS.2011.81

[46] Anthony Widjaja To. 2009. Model Checking FO(R) over One-Counter Processes and beyond. In *Computer Science Logic, 23rd international Workshop, CSL 2009, 18th Annual Conference of the EACSL, Coimbra, Portugal, September 7-11, 2009. Proceedings*. 485–499. https://doi.org/10.1007/978-3-642-04027-6_35

[47] Heribert Vollmer. 1999. *Introduction to Circuit Complexity*. Springer.

[48] Domagoj Vrgoc, Carlos Rojas, Renzo Angles, Marcelo Arenas, Diego Arroyuelo, Carlos Buil Aranda, Aidan Hogan, Gonzalo Navarro, Cristian Riveros, and Juan Romero. 2021. MillenniumDB: A Persistent, Open-Source, Graph Database. *CoRR* abs/2111.01540 (2021). arXiv:2111.01540 https://arxiv.org/abs/2111.01540

[49] Peter T. Wood. 2012. Query languages for graph databases. *SIGMOD Rec.* 41, 1 (2012), 50–60. https://doi.org/10.1145/2206869.2206879

A PROOF OF PROPOSITION 2.3

The input consists of a $(\mathbf{T}, \mathbf{S})$-graph $\mathbf{G}$, a RDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$-query $x \to_{\mathcal{A}} y$, and two nodes $s, t \in V$. A *configuration* C is a tuple $(v, q, \bar{r}, \bar{p}, \bar{m}) \in V \times Q \times U^l \times \mathcal{D}^h \times \{0, \ldots, b\}^h$. The values in the component $\bar{m}$ are to record the number of rewrites to the unrestricted registers performed thus far. In the following, we refer to $\bar{m}$ as the *mode* of C. An initial configuration is of the form $(s, q_0, \bar{r}, \bar{p}, \bar{0})$ with $q_0 \in Q_0$, whereas a final coniguration is of the form $(t, q_F, \bar{r}, \bar{p}, \bar{m})$. Let $\pi := C_0, \cdots, C_n$ be sequence of configurations witnessing a path from s to t in $\mathbf{G}$ satisying $s \to_{\mathcal{A}} t$. We may assume that π has no repetition of configurations (otherwise, one may simply cut out the segment in between two repeating configurations). Now, notice that the modes in π either stay the same or increase (in a component-wise manner). This means that π can be divided into a sequence of subsequences $\pi_1, \ldots, \pi_N$ such that each π_i has exactly the same mode (and no π_i and π_j with $i \neq j$ share the same mode). Thus $N \leq h(b+1)$. Notice also that the values of the unrestricted registers stay the same throughout each π_i. So, each π_i now has length at most $|V| \times |Q| \times |U|^l$. Thus, we deduce that the length of π is at most $M := |V| \times |Q| \times |U|^l \times (b+1)^h$.

To conclude, we simply need to guess π of length at most M, but keep the values of the unrestricted registers symbolically. Let $\mathbf{m} := \bar{m}_1, \ldots, \bar{m}_N$ be the sequence of modes of each path segment $\pi_1, \ldots, \pi_N$ (as above) and a sequence $\bar{l} := l_1, \ldots, l_n$ of lengths of these paths. We also guess the sequence of transitions in $\mathcal{A}$ that takes us through π with guards $\psi_1, \ldots, \psi_n$. [We separately need to ensure that the edge labels in π through the graph and $\mathcal{A}$ also agree.] We can then write a formula φ_π in $\mathbb{L}$ with free variables $X := \{p_i^j : i \in [1, h], j \in [0, b]\}$ that asserts that π with can be realized by X. Here, we use p_i^j to denote p_i but after j rewrites. Thus, from $\mathbf{m}$ and $\bar{l}$, we may compute an appropriate sequence $\bar{p}_0, \ldots, \bar{p}_n$ of variables from X consisting of exactly one p_i^j for each $i \in [1, h]$. If the node (resp. the active-domain register values) in C_i is denoted by v_i (resp. $\bar{r}_i$), then

$$\varphi := \bigwedge_{i=1}^{n} \psi_i(v_{i-1}, \bar{r}_{i-1}, \bar{r}_i, \bar{p}_{i-1}, \bar{p}_i)$$

Thus, query evaluation reduces to query evaluation of

$$\exists X \varphi$$

which is a formula in $\mathbb{L}$ and can be solved by the query evaluation algorithm for $\mathbb{L}$.

B PROOF OF PROPOSITION 2.4

Let $x \to_{\mathcal{A}} y$ be a RDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$-query. Let $\mathcal{P} = \{p_1, \ldots, p_h\}$ be the unrestricted registers used in $\mathcal{A}$. The new $\mathcal{A}'$ will use $\mathcal{P}' := \{p_i^j : i \in [1, h], j \in [0, b]\}$, where p_i^j records the value of p_i after j rewrites. To ensure that $\mathcal{A}'$ uses the right p_i^j, we could record the number of rewrites for p_i in the state, i.e., the set Q' of states of $\mathcal{A}'$ will be $Q \times [0, b]^h$, where Q is the set of states of $\mathcal{A}$. Although this shows equi-expressivity, the resulting $\mathcal{A}'$ is exponentially bigger in $\mathcal{A}$. This is immaterial for data complexity.

C RRC FOR LINEAR ARITHMETICS

We follow the notation from the proof for $\mathbb{R}_{\times,+}$.

We may assume (see proof of [27, Proposition 5.6.10]) that each $\varphi(curr, \mathcal{P}, \mathcal{R}, \mathcal{R}')$ in $\mathcal{A}$ is of the form

$$Q_1 y_1 \in adom \ldots Q_m y_m \in adom \alpha(\bar{y}, p_k, Z_k)$$

where α is a boolean combination of formulas of the form

(1) atomic σ-formula $R(\bar{u})$ with $\bar{u} \subseteq \bar{y}$.
(2) linear constraints of the form $0 \sim t(\bar{y}, Z_k, p_k)$
(3) congruence constraint of the form $x \equiv c \pmod{M}$ for $x \in \bar{y} \cup \cup Z_k \cup \{p_k\}$ and c, M constant integers.

We may also assume that each linear constraint of type (2) of the form $cp_k \sim t(\bar{y}, Z_k)$ satisfies that either $c = 0$ or $c = 1$. This can be done by a standard normalization step in quantifier elimination of Presburger Arithmetic [24]. More precisely, we collect all coefficients $c_1, \ldots, c_s$ such that $c_i p_k$ appears in some linear constraint of the form (2) in $\mathcal{A}$. Let ℓ be the least common multiple of $c_1, \ldots, c_s$. By multiplying with suitable constants, each linear constraint of type (2) and type (3) containing p_k will have the same term ℓp_k. Note that $p_k \equiv c \pmod{M}$ will become $\ell p_k \equiv \ell c \pmod{\ell M}$. Use a new parameter p_k' and replace each ℓp_k by p_k'. Finally replace each resulting formula φ in a transition by $\varphi \wedge p_k' \equiv 0 \pmod{\ell}$.

In addition, by taking the least common multiple of all the divisors M across all congruence constraints (i.e. of type (3)) in transitions in $\mathcal{A}$, we may also assume that there is a single $M_{\mathcal{A}}$ such that each congruence constraint is of the form $x \equiv c \pmod{M_{\mathcal{A}}}$. In summary, α is now a boolean combination of the form:

(1) atomic σ-formula $R(\bar{u})$ with $\bar{u} \subseteq \bar{y}$.
(2a) linear constraints of the form $p_k \sim t(\bar{y}, Z_k)$
(2b) linear constraints of the form $0 \sim t(\bar{y}, Z_k)$
(3) congruence constraint of the form $x \equiv c \pmod{M_{\mathcal{A}}}$ for $x \in \bar{y} \cup \cup Z_k \cup \{p_k\}$ and c constant integers.

For simplicity, we assume that each formula φ in $\mathcal{A}$ uses precisely m active-domain variables $y_1, \ldots, y_m$. Letting $Y_k = \bar{y} \cup Z_k$, we may enumerate all linear term t of type (2a) occuring in some formula in $\mathcal{A}$ as

$$t_1(Y_k), \ldots, t_m(Y_k)$$

We introduce an active-domain register $r(z)$ for each $z \in Y_k$. Extend this to sets of variables, i.e., $r(Y_k)$ is defined. The idea is to replace p_k by either one of the terms in the set

$$S := \{t_i(r(Y_k)) + c : c \in [-M, M] \wedge 1 \leq i \leq m\}$$

Define $Q' := Q \times S$, $Q_0' = Q_0 \times S$, and $F_0' := F_0 \times S$. For each transition $(q, \varphi(Z_k, p_k), q')$ and each $t \in S$, we add to Δ' the transition

$$((q, t), \varphi'(Z_k, r(Y_k), r(Y_k')), (q', t))$$

where φ' is defined as

$$r(Y_k) = r(Y_k') \wedge \varphi(Z_k, t(r(Y_k)))$$

By RQC of FO$(\mathbf{T}, \sigma)$, we may assume each φ' has only active-domain quantifiers.

We claim now that $[\![\mathcal{A}]\!]_{\mathbf{G}} = [\![\mathcal{A}']\!]_{\mathbf{G}}$ for all $(\mathbf{T}, \sigma)$-graph $\mathbf{G}$. The direction $[\![\mathcal{A}]\!]_{\mathbf{G}} \supseteq [\![\mathcal{A}']\!]_{\mathbf{G}}$ is immediate since $\mathcal{A}'$ restricts the range of values that are permitted to p_k in $\mathcal{A}$ (i.e. to values in S, where each $r(Y_k)$ can be instantiated by any active-domain

values). To prove the other direction $[[\mathcal{A}]]_{\mathrm{G}} \subseteq [[\mathcal{A}']]_{\mathrm{G}}$, assume an accepting computation of $\mathcal{A}$

$$\pi := (q_0, v_0) \to_{(a_1,\varphi_1)} \cdots \to_{(a_n,\varphi_n)} (q_n, v_n)$$

for a valuation $\mu : \mathcal{P} \to \mathbb{Z}$, i.e., $(\mathbf{T}, \mathbf{G}), \mu \models \exists p_k \Phi(Z_k, p_k)$, where $\Phi(Z_k, p_k)$ is:

$$\exists \bar{s}, \bar{R} \in adom \bigwedge_{i=0}^{n-1} (\varphi_i(s_i, R_i, R_{i+1}, \mathcal{P}) \wedge E_{a_i}(s_i, s_{i+1})).$$

By the proof of [39, Proposition 5.6.10], there exist $c \in [-M, M]$, $i \in [1, m]$, and a tuple $\bar{a}$ over $adom(\mathbf{G})$ of length $|Y_k|$ such that $(\mathbf{T}, \mathbf{G}), \mu \models \Phi(Z_k, t_i(\bar{a})+c)$. Hence, $\mathcal{A}'$ may simply instantiate $r(Y_k)$ with $\bar{a}$ and keep them constant throughout the computation, as is done in the definition of $\mathcal{A}'$. Extend each v_i with this instantiation yielding v_i'. Let $\mu' := \mu_{|_{Z_k}}$ be the restriction of μ to Z_k. The following path is then an accepting path of $\mathcal{A}'$ with valuation μ':

$$\pi' := (q_0, v_0') \to_{(a_1,\delta_0')} \cdots \to_{(a_n,\delta_n')} (q_n, v_n).$$

D PROOF OF LEMMA 3.5

Fix a $(\mathbf{T}, \sigma)$-register automaton with parameters

$$\mathcal{A} = (\mathcal{R}, \mathcal{P}, Q, Q_0, F_0, \Delta),$$

and a logarithmic function f. Say $|\mathcal{R}| = k$. Given two vertices s, t of a given graph $\mathbf{G}$ with set V of nodes and universe $\mathcal{D}$ of data values, we want to know if there is some parameter assignment $\mu : \mathcal{P} \to \{d \in \mathcal{D} : |d| \leq f(|\mathrm{G}|)\}$ for which the query returns (s, t). The NL machine T first guesses μ (using logarithmic space) and then simply traverses $\mathbf{G}$, while simultaneously simulating $\mathcal{A}$ (as in standard product automata construction). More precisely, it will initially guess a state $q_0 \in Q_0$ in the simulation, and set the register values in $\mathcal{R}$ to an arbitrary active-domain values $\mathbf{r}_0 \in \mathcal{D}^k$, and start the procedure from configuration $(s, q_0, \mathbf{r})$. From any configuration $(v, q, \mathbf{r})$, the machine T guesses a node v' and a label $a \in \Sigma$ such that $(v, v') \in E_a$, the next register (active-domain) values $\mathbf{r}'$, and a transition

$$(q, (a, \varphi(curr, \mathcal{P}, \mathcal{R}, \mathcal{R}')), q')$$

such that $\mathbf{G} \models \varphi(v, \mu, \mathbf{r}, \mathbf{r}')$. The simulation proceeds to the next configuration $(v', q', \mathbf{r}')$. It will stop and accept as soon as a configuration of the form $(t, q_F, \mathbf{r})$, for some $q_F \in F$ and register values $\mathbf{r}$, is visited.

Recall that an NL machine have an input tape, and a log-space working memory tape. To ensure an NL simulation, we use the standard trick of using pointers that refer to the parts of the input tape that contain the node/datum of interest in $\mathbf{G}$. Each such a pointer reference uses space that is logarithmic in the size of the input (i.e. $\mathbf{G}$).

E PROOF OF LEMMA 4.6

We reduce from 3-SAT. Consider a $(\sigma, \mathbf{T}_{\mathrm{Aut}})$-register automaton with no registers and having just three states q_0, q, q_F, where q_0 is initial and q_F is final, one parameter p, and three transitions:

- $(q_0, (\#, \top), q)$,
- $(q, (a, \top), q_F)$, and
- $(q, (a, \psi(curr, p)), q)$,

with

$$\psi(curr, p) = \exists p', x', x \; value(curr, x) \wedge x' \prec x \wedge p' \prec p \wedge eq\text{-}len(p', x') \wedge lst_1(x') \wedge ((lst_1(p') \wedge lst_1(x)) \vee (lst_0(p') \wedge lst_0(x))).$$

Note that ψ says that there is a position i so that the current value x is such that: (1) the i-th letter of the parameter and the last letter of x coincide, and (2) x has a 1 at position i. The idea will be that p encodes a satisfying assignment (where the i-th variable is true iff the i-th position of p is 1), and x is of the form $0^{i-1}10 \cdots 0b$ if it encodes a literal of the i-th variable appearing either positive if $b = 1$ or negative if $b = 0$.

Given a 3-SAT formula

$$\varphi := \bigwedge_{i=1}^{m} (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$$

Consider the $\mathbf{T}_{\mathrm{Aut}}$-embedded graph over a singleton relation signature $\{a\}$, an attribute-assigning relation $value(\cdot, \cdot)$, a vertex set $V = \{v_{i,j} : i \in [1, m], j \in \{1, 2, 3\}\} \dot\cup \{v_0, v_F\}$ and edges

$$E_a = \{(v_0, v_{1,j}), (v_{m,j}, v_F) : j \in \{1, 2, 3\}\} \cup \{(v_{i,j}, v_{i+1,j'}) : i \in [1, n-1], j, j' \in \{1, 2, 3\}\}.$$

The interpretation of $value$ contains (v_0, ε), (v_F, ε); and for any other $v_{i,j}$ such that $\ell_{i,j} = x_k$ (resp. $\ell_{i,j} = \neg x_k$) it contains $(v_{i,j}, u)$, where u is the string over $\{0, 1\}$ of length $m + 1$ such that:

- its k-th position has a 1,
- its last position has a 1 (resp. 0), and
- every other position has a 0.

If φ is satisfiable by setting variables $x_{i_1}, \ldots, x_{i_r}$ to true and the remaining ones to false, then there is a path from v_0 to v_F in the language with the parameter assignment $\{0, 1\}^{m+1}$ which has a 1-symbol on the i_j-th position, for every $1 \leq j \leq r$, and a 0-symbol otherwise. Conversely, for any path from v_0 to v_F in the language, the m-prefix of the parameter assignment witnessing membership yields a satisfying assignment for φ. □

F PROOF OF LEMMA 4.4

Assume an input formula $\varphi \in \mathrm{FO}_{\mathrm{act}}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$ in prenex normal form. For each atomic formula of the form $H(\bar{x})$, where $H \in \sigma$ and assignment $\mu : \bar{x} \to \Gamma^*$, testing $\mu \models H$ is log-uniform AC^0 (therefore, NC^1) because it reduces to checking if a tuple of strings is in a list of tuples of strings. For each atomic formula of the form $R(\bar{x})$, where $R \in \mathbf{T}_{\mathrm{Aut}}$ is an automatic relation, and for each assignment $\mu : \bar{x} \to \Gamma^*$, testing $\mu \models R$ is log-uniform NC^1 because it reduces to checking if the convolution of μ is in some regular language. We can then connect the NC^1 circuits via $\wedge$, $\vee$, and $\neg$ in the quantifier-free subformula of φ by induction. We may furthermore replace $\exists y_i \in adom$ by $\bigvee_{n \in adom}$ and $\forall y_i \in adom$ by $\bigwedge_{n \in adom}$, and replace these with $\vee$ and $\wedge$ of unbounded fan-in, which can easily be simulated with bounded fan-in log-depth circuits. Since φ is a fixed formula, the depth of the circuit stays logarithmic and the size stays polynomial. □

G PROOF OF LEMMA 4.3

We actually show that, in terms of expressive power, we have:

(1) $\mathrm{EFO}^+(\mathbf{T}_{\mathrm{Aut}}, \sigma) \subseteq \mathrm{EFO}^+_{\mathrm{act}}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$ and
(2) $\mathrm{EFO}(\mathbf{T}_{\mathrm{Aut}}, \sigma) \subseteq \mathrm{FO}_{\mathrm{act}}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$.

To prove (1), first note that an existential-positive input formula $\varphi(\bar{x})$ is equivalent to a disjunction of formulas of the form $\varphi'(\bar{x}) = \exists \bar{y}\ \psi(\bar{z}) \wedge \bigwedge_{i \in I} H_i(\bar{z}_i)$, where variables from $\bar{z}$ and $\bar{z}_i$ range over $\bar{x}\bar{y}$, each H_i is a relation from the schema σ, and ψ is a formula of $\mathrm{EFO}^+(\mathbf{T}_{\mathrm{Aut}})$. We can further push any existential quantification of variable from $\bar{y}$ which is not in $\bigcup_{i \in I} \bar{z}_i$ inside ψ. In other words, we can assume that every variable of $\bar{y}$ is either free (i.e., in $\bar{x}$) or in some $\bar{z}_i$. Now we can consider the automatic relation $R_\psi = \{\bar{u} : \Gamma^* \models \psi[\bar{u}]\}$ and redefine φ' as the equivalent formula $\exists \bar{y} \in adom\ R_\psi(\bar{z}) \wedge \bigwedge_{i \in I} H_i(\bar{z}_i)$.

The proof for (2) is similar to the existential positive case with a twist. By a similar reasoning as before, it suffices to assume that we deal with the formula $\varphi(\bar{x})$ of the form $\exists \bar{y}\ \big(\bigwedge_{i \in I}(\neg) R_i(\bar{z}_i) \wedge A(\bar{z})\big)$, where $y = y_1, \ldots, y_n$, $R_i \in \sigma$ and A is an automatic relation. Furthermore, we may assume that $\bar{y} \cap \bar{z}_i \neq \emptyset$ and $\bar{y} \cap \bar{z} \neq \emptyset$; otherwise, we pull the corresponding atom out of the scope of $\bar{y}$. Let $<_{lex}$ be the 'shortlex' lexicographic order on Σ-strings (here, we implicitly assume a total ordering on Σ). That is, $v <_{lex} w$ iff (i) $|v| < |w|$, or (ii) $|v| = |w| = n$, $v = a_1 \cdots a_n$, $w = b_1 \cdots b_n$, and there is $i \in [1, n]$ such that $a_i < b_i$ and $a_1 \cdots a_{i-1} = b_1 \cdots b_{i-1}$. Observe that $<_{lex}$ is a total linear order and is also automatic. We assume below the variables $x, x_1, x_2, \ldots$ do not appear in $\bar{z}$. Define $\theta(x_1, x_2)$ and $\max_{adom}(x)$ as follows:

$$\theta(x_1, x_2) := x_1 <_{lex} x_2 \wedge \neg \exists x \in adom(x_1 <_{lex} x <_{lex} x_2),$$
$$\max_{adom}(x) := \forall x_1 \in adom(x_1 <_{lex} x),$$
$$\min_{adom}(x) := \forall x_1 \in adom(x <_{lex} x_1).$$

Then, φ is equivalent to a big disjunction of $\psi_{\bar{C}}$ formulas, where $\bar{C} = (C_1, \ldots, C_n)$ and $C_i \in \{\min_{adom}(r_i), \theta(r_i, r_i'), \max_{adom}(r_i)\}$. The idea is that by a satisfying valuation of C_i we guess where the valuation of y_i appears w.r.t. the $<_{lex}$ order: either before the first *adom* word (case $C_i = \min_{adom}(r_i)$), or between two consecutive *adom* words (case $C_i = \theta(r_i, r_i')$), or after the last *adom* word, should there be one (case $C_i = \max_{adom}(r_i)$). Let $Y_+ \subseteq \bar{y}$ be the set of all variables z appearing in a positive literal $R_i(\bar{z}_i)$ with $z \in \bar{z}_i$. Here $\psi_{\bar{C}}$ is defined as

$$\exists r_1, r_1', \ldots, r_n, r_n' \in adom \left(\bigwedge_{i=1}^{n} C_i \wedge \bigvee_{Y \subseteq (\bar{y} \setminus Y_+)} \chi_Y \right)$$

where Y is intuitively the variables whose valuations are guessed to be not in *adom* (which of course cannot intersect Y_+). The formula χ_Y is defined as follows. Let η_Y be the variable substitution replacing each $y_i \notin Y$ by r_i, and for any formula ξ, let us write $\xi[\eta_Y]$ to denote the result of substituting the variables according to η_Y in ξ. Thus, χ_Y is a conjunction consisting of each $R(\bar{z}_i)[\eta_Y]$ appearing positive, each negative $\neg R(\bar{z}_i)[\eta_Y]$, where $\bar{z}_i \cap Y = \emptyset$, and the formula

$$\chi_Y' := \exists Y (A(\bar{z})[\eta_Y] \wedge \mu(Y))$$

where $\mu(Y) = \bigwedge_{y_i \in Y} v_i(y_i)$ with

$$v_i(y_i) := \begin{cases} y_i <_{lex} r_i & \text{if, } C_i = \min_{adom}(r_i) \\ r_i <_{lex} y_i <_{lex} r_i' & \text{if, } C_i = \theta(r_i, r_i') \text{ and,} \\ r_i <_{lex} y_i & \text{if, } C_i = \max_{adom}(r_i) \end{cases}$$

Observe that χ_Y' is automatic, and hence can be written as an atom $A'(\bar{z})$ for some $A' \in \mathrm{Rel}$. The reason why $\neg R(\bar{z}_i)[\eta_Y]$, with $\bar{z}_i \cap Y \neq \emptyset$, can be removed is that the above formula already ensures that at least one of the arguments in $\bar{z}_i[\eta_Y]$ will not be in the active domain, which implies $\neg R(\bar{z}_i)[\eta_Y]$. Observe that χ_Y' is in $\mathrm{FO}(\mathbf{T}_{\mathrm{Aut}})$ and so is an automatic relation. Thus, we obtain a formula in $\mathrm{FO}_{act}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$. □

Corollary G.1 (implying Corollary 4.7). *Every $\mathrm{EFO}(\mathbf{T}_{\mathrm{Aut}}, \sigma)$ formula $\psi(\bar{z})$ is effectively equivalent to a $\mathrm{EFO}_{act}(\mathbf{T}_{\mathrm{Aut}}, \sigma')$ formula ψ', where σ' is the extension of σ with*

$$\{\theta(\cdot,\cdot), \max_{adom}(\cdot), \min_{adom}(\cdot)\},$$

interpreted as in the proof of Lemma 4.3. Further, $\psi'(\bar{z})$ is a disjunction of formulas of the form $A'(\bar{z}) \wedge \exists \bar{r} \in adom\ \tau(\bar{z}\bar{r})$, where A' is a $\mathbf{T}_{\mathrm{Aut}}$-atom, τ is a conjunction of σ'-atoms and negated σ-atoms.

Let $N = \max_{w \in adom} |w|$ and let $v : \bar{z} \to \Gamma^*$ be a satisfying assignment for $\psi'(\bar{z})$, and $z \in \bar{z}$ such that $|v(z)| > N$. Suppose there is a word $u \in \Gamma^*$ such that $|u| > N$ and $\Gamma^*, v' \models A'(\bar{z})$ for $v' = v[z \mapsto u]$. Then, v' is also a satisfying assignment for ψ'.

H WORD EQUATIONS: PROOF OF THEOREM 5.2

We consider $\mathrm{RDPQ}(\mathbb{L}_{\mathrm{PM}}, \mathbf{T}_{\mathrm{WE}}, \sigma)$ and first give the proof of Theorem 5.2 in this case.

We begin a simple technical characterization of systems of word equations that need to be solved in this case.

Lemma H.1. *Consider a system of word equations of the form*

$$x = v_i(\bar{y}) \text{ for } i = 1, \ldots, h\ ,$$

where x is a fixed variable other than variables in $\bar{y}$ and each variable from $\bar{y}$ appears at most once in $v_1, \ldots, v_h$ and each v_i contains at least one variable (from $\bar{y}$).

Let w_i, s_i be the longest prefix and suffix of each v_i consisting of letters, i.e. no variables are allowed. Then the system has a solution if and only if there are w, s such that $w_i \preceq w$ and $s \succeq s_i$ for each i.

Proof. Suppose that the condition does not hold, by symmetry, suppose that there is no such w. This means that for some i, j the $w_i[\ell] \neq w_j[\ell]$ for some $\ell \leq \min(|w_i|, |w_j|)$. But then the two equations $x = v_i(\bar{y})$ and $x = v_j(\bar{y})$ cannot be simultaneously satisfied, as they differ already at the constants. Symmetric argument applies to the suffixes.

So suppose that the condition is satisfied. We will explicitly define a solution in this case. For each v_i let v_i' be the v_i with the prefix w_i, suffix s_i and all variables removed. We assign $x := w v_1' v_2' \cdots v_h' s$, in the following we will define the substitution for remaining variables. Consider an equation $x = v_i(\bar{y})$, by the assumption, a variable that appears in $v_i(\bar{y})$ does not appear in any other equation. Let $w = w_i w_i'$, $s = s_i' s_i$, those are well defined as by case assumption $w_i \preceq w$ and $s \succeq s_i$. Let y_i be the first variable in v and y_i' the last; we set all other variables in v_i to ε. There are two subcases, depending on whether $y_i = y_i'$ or not. In the first case we have that v_i is $w_i y_i s_i$, then we assign $y_i := w_i' v_1' v_2' \cdots v_h' s_i'$. Then $v_i(\bar{y})$ evaluates to

$$w_i w_i' v_1' v_2' \cdots v_h' s_i' s_i = w v_1' v_2' \cdots v_h' s$$

and so $x = v(\bar{y})$ is satisfied.

Otherwise, we assign $y_i := w'_i v'_1 \cdots v'_{i-1}$ and $y'_i := v'_{i+1} \cdots v'_h s'_i$. Then $v_i(\bar{y})$ evaluates to (recall that any variable other than y_i and y'_i is set to ε)

$$\begin{aligned} w_i y_i v'_i y'_i s_i &= w_i \underbrace{w'_i v'_1 \cdots v'_{i-1}}_{y_i} v'_i \underbrace{v'_{i+1} \cdots v'_h s'_i}_{y'_i} s_i \\ &= w v'_1 \cdots v'_h s \end{aligned}$$

and so $x = v_i(\bar{y})$ is satisfied. □

We now give an NL algorithm for RDPQ($\mathbb{L}_{\mathrm{PM}}, \mathbf{T}_{\mathrm{WE}}, \sigma$). For each parameter p_i for $i = 1, \ldots, k$ we first guess, whether its value can be represented as a concatenation of at most m substrings of elements in the active domain and constants, where m is the longest right-hand side of the equations in formulas labelling the transitions in $\mathcal{A}$. If so, then we guess such a representation. If not, then we guess such a representation for a prefix w_i and suffix s_i of value of p_i (those will be the w and s from Lemma H.1 for p_i). Note that this all fits in NL, as m and number of parameters depend on the query. We then guess the path π in $\mathbf{G}$ node by node, verify, whether they are connected by an edge (i.e. evaluate the node-view formula for each node and the edge-view formula for the two nodes) guess the transition of $\mathcal{A}$ and evaluate the guard φ of the transition. The view formulas can be computed in time independent of $|\mathbf{G}|$; for a given guard φ, we choose a subset of equations that make this formula true (this can be checked after the choice) and treat it as a system of equations. We then verify this system of equations; note that in general we need to verify all such equations along the whole path π, but Lemma H.1 will allow us to verify this system in isolation of other equations.

We first deal with the equations of the form $x = v(\bar{y})$, where x is an element from the active domain (be it a register or some current value) or parameter whose whole value was initially guessed. As the left-hand side is known, we can guess the lengths of all (existentially quantified) variables on the right-hand side and this yields the values of those variables; note that each such valuation of a variable is a substring of a value from active domain. Then we verify the consistency of those guesses (i.e. whether the value of existentially quantified variable y in each equation is the same) and the equations, which in both cases is done letter by letter, so in NL.

We then verify the equation of the form

$$p_i = v_i(\bar{y})$$

for p_i that have guessed prefix and suffix. First we instantiate each variable y on the right-hand side, whose value was already set (when considering earlier equations); if this is the case then its value is a substring of an element in the active domain. Afterwards, if on the right-hand side there is no variable (whose value has not been yet fixed), then we reject, as we should have guessed the value of p_i: v consists of at most m symbols (by choice of m) and each of them was substituted with a substring of an element from the active domain.

Otherwise, we verify, whether the prefix of v_i before the first variable is a prefix of w_i and a suffix of v_i after the last variable is a suffix of s_i. If not, then we reject. Clearly those computations can be done in NL.

If at the end of the path we have not rejected, then we accept. We should show that if we accepted then indeed the graph $\mathbf{G}$ satisfied the query. Consider exactly the path that we simulated on the graph, let all variables be instantiated in the same way as the algorithm did. Note that the only variables that are not set in this way are the parameters p such that we have guessed only the prefix and suffix for them and some variables that occur at the right-hand side of equations for those parameters. Fix one such parameter x and consider all equations that were verified for it, let them be $x = v_i(\bar{y})$ for appropriate set $i \in I$, we instantiate on the right-hand side all variables that were instantiated during the computation. By the algorithm, each such an equation has at least one uninstantiated variable. Then this means that the algorithm explicitly verified the condition of Lemma H.1 for this system, and so $x = v_i(\bar{y})$ for $i \in I$ has a solution. By the definition of the fragment $\mathbb{L}_{\mathrm{PM}}$, no variable appearing in the right-hand side occurs in any other such subsystem and also by the algorithm it does not appear in any other equation (as then it would have been instantiated). Hence, we can consider the subsystem for each parameter p separately.

It remains to show that if $\mathbf{G}$ satisfies the query then we return a positive answer (for appropriate non-deterministic guess). Assume an accepting path

$$\pi := (q_0, v_0) \to_{(a_1,\varphi_1)} \cdots \to_{(a_n,\varphi_n)} (q_n, v_n)$$

For each formula guard φ_i take all equations that are satisfied in it, rename the variables so that existentially quantified variables from different instances of the formula are different. In the obtained system, call a variable or parameter fixed, when

(1) this variable is on the right-hand side of an equation whose left-hand side is from $curr, \mathcal{R}, \mathcal{R}'$ (in this case the variable is represented by a substring of an element from the active domain)
(2) this variable is a parameter and is a left-hand side of an equation whose right-hand side use only constants, $curr, \mathcal{R}, \mathcal{R}'$ and fixed variables, which need to be fixed in point 1. Observe that in this case the fixed variable can be represented as concatenation of at most m substrings of elements from the active domain (and constants).

For parameters that are fixed we can guess in the algorithm the appropriate values, for the remaining parameters, the system of equations that they satisfy is of the form as in the Lemma H.1. In particular, in the algorithm we can guess for the parameter p the prefix and suffix according to Lemma H.1. It can be represented as a concatenation of at most m substrings of elements from the active domain and constants, as the longest prefix and suffix consisting only of letters of right-hand sides of considered equations, as in Lemma H.1, can be represented in this way.

General properties of word equations. Our proof of Theorem 5.2 in case of $\mathbb{L}^{+}_{\mathrm{WE}}$ is based on general properties of word equations, which are independent from our setting. Thus we will introduce and prove using more standard terminology and only at the end show how those properties can be exploited in our case. The only departure from the standard usage is that we will use $u(\bar{x}) = v(\bar{x})$ or even $e(\bar{x})$ to denote an equation with variables $\bar{x}$ and $u(\bar{s}) = v(\bar{s})$, or simply $e(\bar{s})$, to denote the evaluation on the substitution $\bar{s}$. Due to our setting, we are focusing on equations with a fixed (or bounded)

number of variables and appearances of variables, as opposed to the length of those equations, which are not bounded. We first recall some general properties of word equations. In most cases their statement do not distinguish between the equations length, number of variables and number of occurrences of variables. We will state the refined versions of those bounds and sketch the needed improvements of the proofs.

Given a word w by $|w|$ we denote its length and also denote it as $w[1]w[2]\cdots w[|w|]$, with each $w[i]$ being a single letter, and $w[i\dots j] = w[i]\cdots w[j]$. We also use this notation to the equation of two words i.e. for an equation e equal to $u = v$, the $e(\bar{s})[i\dots j]$ is $u(\bar{s})[i\dots j]$, when $\bar{s}$ is a solution, note that this does not depend on the choice of the side. Also, we extend length to the equations: when e is $u = v$ then $|e(\bar{s})| = |u(\bar{s})| + |v(\bar{s})|$. Given an equation with substituted variables $e(\bar{s})$, we say that a letter comes from an equation (or is a position of a letter), when the corresponding letter was present in e, and that it comes from a variable (is a position of a variable), when this letter was substituted for a variable.

A solution $\bar{s}$ is length-minimal solution of a system of equations E, when for each other solution $\bar{s}'$ it holds that

$$\sum_{e\in E} |e(\bar{s})| \leq \sum_{e\in E} |e(\bar{s}')| \ .$$

Regular constraints. While we formally introduced the regular constraints by allowing unary symbols for regular relations, from the algorithmic perspective it is easier to fix the regular languages appearing in an instance and consider the transition functions (into Boolean matrices), i.e. a homomorphism $\rho : \Gamma \to \mathbb{M}_q$, where $\mathbb{M}_q$ are the $q \times q$ Boolean matrices with the standard (i.e. Boolean operations). The idea is that the $\mathbb{M}_q$ encodes the transition matrices of the automata specifying the regular languages involved in the instance. Regular constraints are then encoded by specifying ρ_x for various variables and requiring that for a variable x we consider only substitutions s such that $\rho(s) = \rho_x$. There is a folklore nondeterministic reduction that encodes the relational regular constraints into the ones specified by ρ.

When considering word equations with regular constraints it is useful [21] to close the original alphabet under the transition function: Given a finite alphabet Γ and a homomorphism $\rho : \Gamma^* \to \mathbb{M}_q$ we say that an alphabet Γ is *ρ-closed* if $\rho(\Gamma) = \rho(\Gamma^+)$, i.e. for each word $w \in \Gamma^+$ there exists a letter $a \in \Gamma$ such that $\rho(w) = \rho(a)$. Usually, an alphabet is not ρ-closed, however, we can naturally extend it with "missing" letters: for an alphabet Γ define a *ρ-closure* $\mathrm{cl}_\rho(\Gamma)$ of Γ:

$$\mathrm{cl}_\rho(\Gamma) = \Gamma \cup \{a_P \ : \ \text{there is } w \in \Gamma^+ \text{ such that } \rho(w) = P\} \ ,$$

where each a_P is a fresh letter not in Γ and $a_P \neq a_{P'}$ when $P \neq P'$. Then $\mathrm{cl}_\rho(\Gamma)$ is ρ-closed. Whenever clear from the context (or unimportant), we will drop ρ in the notation and talk about closure and cl.

The $\mathrm{cl}_\rho(\Gamma)$ might be in general large and it is better not to store it explicitly, instead, when we are given a letter a_P we can verify, whether $a_P \in \mathrm{cl}_\rho(\Gamma)$, which boils down to checking, whether P is a transition function for some word $w \in \Gamma^+$. This can be easily verified in NSPACE(q^2): we guess the word w letter by letter and store the transition matrix of the so-far guessed prefix. When the next letter is guessed, we multiply the matrix by the transition matrix of a guessed letter.

Viewing a system of word equations over Γ (with regular constraints) as a system of word equations (with regular constraints) over $\mathrm{cl}(\Gamma)$ does not change the satisfiability [21, Lemma 10].

Lemma H.2 ([21, Lemma 10]). *A system E of word equations with regular constraints (defined using a homomorphism $\rho : \Gamma \to \mathbb{M}_q$) over an alphabet Γ has a solution over Γ^* if and only if it has a solution over $\mathrm{cl}(\Gamma)^*$ (treating the input system as a system over $\mathrm{cl}(\Gamma)$).*

Similar equations. When we consider systems of equations that appear on a path in a graph data base, we often find equations of the form $\bigcup_s e(\bar{x};\bar{s})$. Those equations have the same variables in the same order, but the words of letters between them are different. Many bounds are easy to obtain for such equations and in fact those bounds hold for equations in which the variables appear in the same order. Formally, equations $u(\bar{x}) = v(\bar{x})$ and $u'(\bar{x}) = v'(\bar{x})$ are *similar*, when the sequence of variables (obtained by removing all letters) in $u(\bar{x})$ and $u'(\bar{x})$ is the same and the sequence of variables in $v(\bar{x})$ and $v'(\bar{x})$ is the same. Clearly, being similar is an equivalence relation.

We say that e and e' are *strongly similar*, when they are similar and moreover, the lengths of words between the corresponding variables in e and e' are the same. Again, being strongly similar is an equivalence relation.

If two equations are strongly similar then either they are the same equation or they can be split into the same system of (smaller) equations with the same solution.

Lemma H.3. *Suppose that equations $u(\bar{x}) = v(\bar{x})$ and $u'(\bar{x}) = v'(\bar{x})$ are strongly similar and that they have a solution $\bar{s}$. Then either they are the same equation or*

$$\begin{aligned} u &= w_0u_1w_1\cdots w_ru_rw_{r+1}\\ v &= w_0v_1w_1\cdots w_rv_rw_{r+1}\\ u' &= w'_0u_1w'_1\cdots w'_ru_rw'_{r+1}\\ v' &= w'_0v_1w'_1\cdots w'_rv_rw'_{r+1} \end{aligned}$$

where

- $r \geq 1$;
- $w_0, w'_0, \dots, w_{r+1}, w'_{r+1}$ *are non-empty words of letters and* $|w_i| = |w'_i|$ *for* $0 \leq i \leq r+1$;
- $u_1, v_1, \dots, u_r, v_r$ *are words of letters and variables;*
- *for* $1 \leq i \leq r$ *the equation* $u_i = v_i$ *contains at least one occurrence of variables and $\bar{s}$ is its solution.*

In particular, the system of equations $\{u_i = v_i\}_{i=1\dots,r}$ has a solution $\bar{s}$ and every solution $\bar{s}'$ of this system is also a solution of the two input equations.

Proof. Suppose that those are not the same equation. Consider $u(\bar{s})$ and $u'(\bar{s})$ and the first position (from the left) on which they differ, say the first has a and in the other: b. By the assumption that the equations are strongly similar, those letters come from the letters in the equation and so we can represent u as u_1au_2 and u' as $u_1bu'_2$, observe that u_1 may or may not have variables.

Consider the corresponding positions in $v(\bar{s})$ and $v'(\bar{s})$. Those cannot be positions coming from the variables, as this would imply that a substitution for a variable has a and b at the same position. So

those are positions coming from the letters. Hence, we can represent v as $v_1 a v_2$ and v' as $v_1 b v_2'$, where $u_1(\bar{s}) = v_1(\bar{s})$, note again that v_1 may or may not have a variable. We proceed with the equation $u_2 = v_2$ and $u_2' = v_2'$, which are strongly similar. If u_1 or v_1 contain a variable then we make them into u_1 and v_1 from the statement, if not, then $u_1 a$ and $v_1 a$ are the same string and we attach them to the initial word returned by the recursive call to get the words w_0 and w_0' from the statement.

The claims about the solution $\bar{s}$ and $\bar{s}'$ are clear from the construction. □

Exponent of periodicity, length-minimal solution bounds. For a word w the maximal k such that u^k, where $u \neq \varepsilon$, is a substring of w is the *exponent of periodicity* of w, denote it by $\mathrm{per}(w)$ and extend to a set of words by taking a maximum, i.e. $\mathrm{per}(W) = \max_{w \in W} \mathrm{per}(w)$. For a system of equations (with regular constraints), the $\mathrm{per}(E)$ is the maximum over length-minimal solutions $\bar{s}$ of $\mathrm{per}(E(\bar{s}))$. It is enough that we consider only simpler variant: per_1, whose definition is similar (for words, sets of words and systems of equations with regular constraints), but we additionally require that $|u| = 1$, i.e. we consider only the length of the maximal repetitions of a letter.

Lemma H.4. *Consider a system of equations E with regular constraints. Let $n = \max_{e \in E} |e|$ and $n_v = \max_{e \in E} |e|_{\bar{x}}$ i.e. the maximal number of occurrences of variables in e, k be the number of variables and let the constraints by expressed using Boolean matrices $\mathbb{M}_q$. Then*

$$\mathrm{per}_1(E) \leq O\left(nk\,(2q!n_v)^{2k}\right) \ .$$

Proof. Proving the bound on $\mathrm{per}_1(E)$ stated in the Lemma requires adaptations of existing proofs. The proof presented below it uses the existing proofs the most. In particular, the standard proof [20] in the presence of regular constraints extends [36], which processes the word equation letter by letter, introducing an integer variable for each letter of the equation. This is most likely an artifact of the proof and a sequence of constants could be processed in one step without much effect on the whole proof. Unfortunately, the existing proof is not modular enough and so verifying this claim essentially requires rewriting most of the existing definitions and proofs.

We first show the bound on per_1 in case of system of equations only, with no constraints. This bound is essentially given in the desired form in [33] (and it is a simplification of the proof in [36]). Then we consider the addition of the regular constraints, which requires a small improvement of this proof, which is described for instance in [21].

Consider a length-minimal solution $\bar{s}$ of E. We say that a^k is a *maximal power*, when it occurs in some $e(\bar{s})$ and it cannot be extended in $e(\bar{s})$. Consider the set of all such maximal powers in $E(\bar{s})$, let there be p of them. In [33, Section 8] it is shown (formally: for one equation, but many equations can be always encoded into one with constant-size increase; also exactly the same approach applies when we have a system of equations) that there are $p \leq 2n_E$ such different lengths and moreover, one can construct a system of p linear Diophantine expressions $n_1(\bar{z}), \ldots, n_p(\bar{z})$, where $\bar{z}$ is a sequence of natural-valued variables, corresponding to those lengths, in the sense that $\bar{z} = z_1, \ldots, z_{2k}$, i.e. there are at most 2 such variables for one word-variable, and $n_1(\bar{m}), \ldots, n_p(\bar{m})$ include all length of such maximal a-powers, where $\bar{m}$ is a sequence of lengths of a-maximal prefixes and suffixes of $s_1, \ldots, s_k$. We then create a system of equations, roughly speaking, if $n_i(\bar{m}) = n_j(\bar{m})$ then we add an equation $n_i(\bar{z}) = n_j(\bar{z})$. Then any other solution $\bar{m}'$ of this system leads to another solution $\bar{s}'$ of E, in which, in essence, we are replacing maximal a-powers of length $n_i(\bar{m})$ with $n_i(\bar{m}')$ (this is a simpler variant of an approach from [36]). As a simple example, an equation $a x_1 x_1 = x_2 x_2 x_2$ has maximal powers of a of length $1 + 2n_x$ and $3n_y$, where the substitution for x_1 is a^{m_1} and for x_2 is a^{m_2}. This leads to an (integer) equation $1 + 2z_1 = 3z_2$ and each solution m_1', m_2' of the latter system leads to a solution of the word equation $x_1 = a^{m_1'}, y = a^{m_2'}$.

It is shown that if $\bar{s}$ is a length-minimal solution then the corresponding solution of the system of Diophantine equations is minimal (i.e., minimal for pointwise comparison) and some known linear algebra and known estimations are used, see [33, Lemma 8.2], to bound the size of components of minimal solutions of a linear Diophantine systems. In essence, we need to bound the maximal value of determinant over square submatrices of the matrix of this system of Diophantine equations. The proof of [33, Lemma 8.2, Lemma 8.3] gives a bound

$$O(nk(2n_v)^{\ell}) \ . \tag{2}$$

Note that there the proof uses only one equation, but the case with many equations can be easily improved to the given bound. Then it maximizes it over n_v, ℓ as functions of $\sum_{e \in E} |e|$ and only the result of this maximization is stated explicitly. Here $2n_v$ means that the maximum sum of absolute values of coefficients at a single equation is at most $2n_v$ and ℓ means that we look at square submatrix of size $\ell \times \ell$. However, there are only $2k$ integer-valued variables, where k is the number of variables of the system of word equations, so (2) is at most

$$O\left(n\,(2n_v))^{2k}\right) \ . \tag{3}$$

When the regular constraints are taken into the account, the construction using the system of linear Diophantine equations still works, but for each (integer) variable x we add an equation of the form $x = x'q' + c$, see [22, Section 4.5.2], where $q' \leq q!$ is the minimal idempotent power of transition matrix for appropriate letter, $0 \leq c < q'$ and moreover x' is not used elsewhere in the equations. This formalizes the idea that the transition function of the a-power should be the same in all solutions. As x' is used only once and x has one more occurrence, this means that when we compute the bounds in (2) we need to multiply by $q'^{2k} \leq q!^{2k}$: In the submatrix we can include the columns and rows that use the $2k$ variables and they have only one element (q') per column, so we can eliminate them by using the Laplace expansion for columns (note that this expansion also takes care of the new occurrence of the old variables: the corresponding coefficients are eliminated). Hence the bound (2) turns to

$$O\left(nk\,(2q!n_v)^{2k}\right) \ , \tag{4}$$

as promised. □

Let us move to the bound on the solutions size of the length-minimal solution.

Lemma H.5. *Consider a system of equations E with regular constraints. Let $n_v = \max_{e\in E} |e|_{\bar{x}}$ and k be the number of variables. Let the constraints by expressed using Boolean matrices $\mathbb{M}_q$.*

Let $\bar{s}$ be a length-minimal solution (over some alphabet). Then

$$\sum_{e\in E} |e(\bar{s})| \leq p(\max_{e\in E} |e|) \cdot g(q, n_v, k) \ ,$$

where p is a polynomial (with degree depending on n_v, k) and g is triply exponential function not depending on $\max_{e\in E} |e|$.

Proof. The bound on the solution size can be obtained by using a graph representation of all solutions, which is available in many variants [22, 32, 33, 43]. Many details of newer constructions [22, 32, 33] are in fact interchangeable, one needs to update the bounds appropriately though. We will refer to [22], as it has the simplest definition and explicitly refers to regular constraints; all those constructions are based on applying compression operations to word equations and we will use the particular strategy described in [32], as it suits most our purposes. This means that the bounds on the space complexity are as in [32]. The [43] uses a much different approach, but the resulting structure is still similar.

The general idea [22, 32, 33] is that we define a (directed) graph, whose nodes are labelled with equations, and edges with families of transformations of substitutions (for the variables). One of the nodes corresponds to the input equation and the "final" nodes have only trivial solutions, i.e. $\bar{\varepsilon}$. The graph represents solutions in the following sense (see [22, Lemma 2] and the discussion after Lemma 2 there):

(1) If there is an edge from E to E' labelled with Φ and $\bar{s}'$ is a solution of E' then for each $\varphi \in \Phi$ the $\varphi(\bar{s}')$ is a solution of E.
(2) On the other hand, for every solution $\bar{s}$ of the initial system of equations E there is a (directed) path from E to a trivial equation with edges labelled with families $\Phi_1, \ldots, \Phi_m$ such that for some $\varphi_i \in \Phi_i$ it holds that $\bar{s} = \varphi_1(\varphi_2(\cdots \varphi_m(\bar{\varepsilon})))$; so in a sense each solution can be obtained in the way above.

As mentioned, an edge between E and E' means that E' can be obtained from E by applying some simple compression operations on letters in equations from E (as well as substitutions for variables of the form $x \to axb$). We will extend the above construction by allowing also an operation of splitting the equations: if a node of the graph is labelled with a system of equations $E \cup \{u = v\}$ then we may introduce another node, labelled with $E \cup \{u' = v', u'' = v''\}$, where $u = u'u''$, $v = v'v''$, where here the equality is understood as equality of sequences of letters and variables. Clearly, this operation has the properties 1–2.

It can be showed that if $\bar{s}$ is a length-minimal solution then the path in 2 does not contain a cycle, as removing it would yield a shorter solution, contradicting the length-minimality, see [33, Theorem 8.7] for a formal argument. Although this proof does not consider regular constraints, it generalizes easily, using exactly the same argument, when regular constraints are allowed.

We can also compare the lengths of the solutions words $e'(\bar{s}')$ and $e(\bar{s})$ for $e \in E$ and the corresponding equation $e' \in E'$. The $e(\bar{s})$ is obtained from $e'(\bar{s}')$ by substituting letters with longer words, their lengths are either constant (1 or 2) or are coordinates of a system of linear Diophantine equations, see construction in [22, Algorithm 2]; this system is the same as the one considered for per_1 in Lemma H.4 and it can be shown that if $\bar{s}$ is a length-minimal solution, then bound from Lemma H.4 applies to the length of the words substituted for a constant by φ; those bounds depend on the system E (maximal equation length n, number of variables k, maximal number of occurrences of variables in an equation n_v); again see [33, Theorem 8.7] for a formal argument; although it does not consider regular constraints, the proof is the same when regular constraints are allowed, as they only apply simple arguments on length-minimality. Observe that we need to treat the splitting of equations separately, however, for them the sum $\sum_e |e(\bar{s})|$ does not change.

The edge from E to E' is present, when we can obtain E' by one of "compression operations" and the main part of the proof is that by applying the operations in an appropriate way we can guarantee that we do not need to use equations longer than some bound. The exact bound depends on particular strategy that is used.

We want to obtain an upper bound on a length of a (loopless) path in such a graph. This bound is derived from a bound on maximum number of different systems of equations on such a path. The algorithm given in [32] works in phases. If we focus on a word w between two variables in the equation (or between a variable and one of the ends of the equation's side), then [32, Lemma 10] when we consider the corresponding word during a phase (the proof gives the bounds in terms of n, but in fact they hold for n_E):

- at most $4n_E + 2$ letters are introduced to this word during the phase
- at least $1/4|w|$ letters are removed from the word (due to compression) during a phase.

So at the end of the phase the word has length at most $3/4|w| + 4n_E + 2$.

As observed in [33, Lemma 5.2], we can modify the bound (on the number of "crossing pairs") to $2k$ instead of $2n_E$ at the expense of increasing the additive constant of introduced letters by 2. So that when considering a word between variables in one phase

- at most $4k + 4$ letters are introduced during a phase
- at least $1/4|w|$ letters are removed from the word (due to compression).

Hence, at the end of the phase the corresponding word has length at most $3/4|w| + 4k + 4$. Note that it could be that some variable is removed during a phase, so w' could be actually only a part of a word between the variables.

This bound implies that after $O\left(k \log\left(\max_{e\in E} |e|\right)\right) = O(k \log n)$ initial steps the system is shortened so that the word between any two variables is of length at most $32k$ (if any two words are joined because of variable removal, then they will be shortened to the length of the longest of them after $O(1)$ phases and there are at most k such removals, as the variables are not introduced).

Before we proceed, we observe that this system cannot have too many equations: there are at most k^{n_v+1} different equivalence classes of the (equations) similarity relation (as there are k variables and at most n_v occurrences of variables in an equation). As the length of each word between two variables is at most $32k$, we can also bound the number of equivalence classes of the (equations)

strong similarity relation:

$$k^{n_v+1} \cdot (33k)^{n_v+2} \leq 33^{n_v+2} k^{2n_v+3} \quad .$$

There are at most $n_v + 2$ words between the variables (and ends of words) and each has length from 0 to $32k$. Now, by Lemma H.3 given a system of equations and its solution $\bar{s}$ we can replace all strongly similar equations by a system of equations whose total length is the same as one such equation, each solution of this system is a solution of the previous one and $\bar{s}$ is a solution of the new system. Hence, we perform such replacement. Hence, we end up with a system of equations of total length at most

$$33^{n_v+2} k^{2n_v+3} \cdot (33k)(n_v + 2) \leq 33^{n_v+4} k^{2n_v+4} \quad .$$

The left part corresponds to the number of possible equations and the right to the maximal length: there are at most $n_v + 2$ words between letters and variables, each contains at most $32k$ letters and one variable.

In the following we consider the systems on this path that have total length at most $33^{n_v+4} k^{2n_v+4}$. No two systems can repeat on this path and we identify letters up to permutation, so the path length (after the initial steps) is $p^{33^{n_v+4} k^{2n_v+4}}$, where p is an upper bound on the size of the alphabet. As we take letters up to renaming, there are at most $2^{q^2} + 33^{n_v+4} k^{2n_v+4}$ different letters (we can distinguish letters in the equations and can also distinguish them by transition function). Hence in total the length of the path is

$$O(k \log n) + (2^{q^2} + 33^{n_v+4} k^{2n_v+4})^{33^{n_v+4} k^{2n_v+4}} \quad . \tag{5}$$

We now estimate, how much we can increase the length of the solution when going (backwards) on a path defining a solution. When there is an edge from E to E' and $\bar{s} = \varphi(\bar{s}')$ then $E'(\bar{s}')$ is obtained from $E(\bar{s})$ by replacing some substrings with single letters. Hence we can estimate $\sum_{e \in E} |e(\varphi(\bar{s}))|$ by multiplying $\sum_{e \in E'} |e(\varphi(\bar{s}'))|$ by a number that bounds the length of the longest substring that is replaced by a single letter; this bound corresponds to (3) applied to system E:

$$O\left(k \max_{e \in E} |e| \cdot (2q! n_v)^{2k}\right) \quad .$$

This bound is sufficient when $\max_{e \in E} |e| = O(33^{n_v+4} k^{2n_v+4})$, i.e. after the initial $O(k \log n)$ steps, but it is too large for the initial segment of the path, when we can only bound it in terms of n. To bound the size increase in this case, we need to be more subtle.

Fix an equation e from the initial equation and the consecutive equations $e = e_1, e_2, \ldots$ on the path corresponding to e. They are obtained by replacing some substrings by single letters and replacing all occurrences of a variable x with wxw' for some words w, w', the words w, w' are "popped" from the variable. We will say that an occurrence of a letter *represents original occurrences*, when this occurrence corresponds to an occurrence of a letter in the original equation e or it replaced occurrences that represented original occurrences. In other words, no popped letters were used in the compression leading to this letter. The remaining letters are said to *represent popped letters*, i.e. the compressions leading to this occurrence of a letter included at least one popped letter. Observe that when we go back in a path from e_i to e then each letter a such that a has an occurrence representing original occurrences, is replaced with a word of length at most n, as there are words of length at most n in the original equation e.

On the other hand, we can show that (in each e_i) the number of letters representing popped letters is at most $16kn_v$, the estimation is similar as for the length of the word between the variables: such letters are organized into sequences, at most 2 of them per occurrence of the variable (one to the left and one to the right). Similarly as in the case of the words between the variables, such sequence increases by at most $2k + 2$ letters during the popping (for the word between the variables this happens from both sides, so it is $4k + 4$). On the other hand, such sequence of letters is shortened by at least $1/4$ of its length from the beginning of the phase. Hence its length after one phase is at most:

$$|w'| \leq 3/4|w| + 2k + 2$$

This shows that the sequence is of length at most $16k$. There are at most 2 such sequences to a side of occurrence of a variable, and n_v occurrences of variables, so there are at most $32n_v k$ occurrences representing popped letters. (Such sequences can be joined, but then the estimate remains the same).

Now, the estimation of "n" in (4) actually refers to sum of number of variables and number of compressed letters. If we are replacing powers of a letter which does not have an occurrence representing original letter, this estimation is $O(32n_v k) = O(n_v k)$. If there are some letters representing original letters, this bound is $O(nk)$ (so much higher). However, afterwards each such letter is multiplied in total by at most n. Hence for the length increase, we either multiply by $O\left(n_v k\right) \cdot (2n_v q!)^{2k}$ and all other multiplication in total yield increase of

$$O\left(n^2 k\right) \cdot (2n_v q!)^{2k} \leq O\left(n^2 k \cdot (2n_v q!)^{2k}\right).$$

Hence in total the solution size is at most:

$$O\left(n^2 k \cdot (2n_v q!)^{2k}\right) \cdot O(n_v k)^{O(k \log n)} \cdot \left(33^{n_v+4} k^{2n_v+4}\right)^{(2^{q^2} + 33^{n_v+4} k^{2n_v+4})^{33^{n_v+4} k^{2n_v+4}}}$$

The third term depends only on k, q, n_v and is triply exponential, as promised; the first is just n^2 times an exponential function depending on k, q, n_v. Let us analyze the middle term:

$$O\left(n_v k\right)^{O(k \log n)} = \exp\left(O\left(k \log n \cdot \log(n_v k)\right)\right) = n^{(k \cdot \log(n_v k))}$$

which indeed is a polynomial of degree depending only on k, n_v. □

Solutions and length functions. In the following, we consider substitutions (and solutions) such that the lengths of substitutions for the variables are known. This approach is known [44] and formalized using *length functions*: a length function $f : \{x_1, \ldots, x_k\} \to \mathbb{N}$ simply assigns to each variable a natural number (i.e. the length of the substitution for this variable). Given a length function f, a substitution $\bar{s}$ is an f-substitution when $|s_i| = f(x_i)$ for each variable $i = 1, \ldots, k$. It is an f-solution, when it is f-substitution and a solution.

Given a length function we can relate positions that have the same letters for each f-solution [44, Lemma 4] (note that the original definition did not allow regular constraints): Given a system of word equations E over variables $\bar{x} = (x_1 \dots, x_k)$ and a length function $f : \{x_1, \dots, x_k\} \to \mathbb{N}$ for an equation $u = v$ from E define a word $f(u)$ as follows: going from left to right in u leave each letter intact and replace each variable x_i with symbols $(x_i^{(1)}, \dots, x_i^{(f(x_i))})$; define $f(v)$ analogously. Formally $x_i^{(j)}$ is a letter-variable, but we are not going to use it nor refer to it, it is just a marker to indicate letter coming from a substitution for a variable. We also use $f(u) = f(v)$ for equation $u = v$ and $f(E)$ for a set of such equations.

We say that a position with a letter $a \in \Sigma$ comes from a letter (is a position of a) and a position with $x_i^{(j)}$ comes from a variable x_i and that it is the j-th position of a variable x_i (we also use the notions first and last position of a variable, with an obvious meaning). By (i, L, e), where e is $u = v$ or $e \neq v$, we will denote the i-th position in $f(u)$ (so left-part) from e, we similarly use (i, R, e) for $f(v)$. We drop "e" and use (i, L), (i, R) when it is clear from the context or unimportant, similarly, we simply use i, when also the side is clear or unimportant. We will use positions to address positions in the system of equations, i.e. $f(E)[(i, L, e)]$ is a well-defined position. However, in most cases we will refer to an abstract position, without specifying the equation or its side, so we will use $f(E)[i]$ in this case.

Define a relation $\mathcal{R}_f$ that intuitively relates positions (in $f(E)$) that contain the same letters for each such f-substitution:

- $R_f((i, L, e), (i, R, e))$, i.e. the corresponding positions on the two sides of the equations are in the relation;
- $R_f(i, i')$ when both i, i' are j-th letters of some variable x.

Let $\mathcal{R}_f^*$ be reflexive, symmetric and transitive closure of $\mathcal{R}_f$. Again, we use $\mathcal{R}$ and $\mathcal{R}^*$, when f is clear from the context or unimportant.

The following Lemma is a slight generalization of [44, Lemma 4] to the case of regular constraints, the proof idea remains the same, though.

Lemma H.6 (cf. [44, Lemma 4]). *Given a system of equations E over words with regular constraints, and a length function f an f-substitution $\bar{s}$ is an f-solution if and only if the following conditions hold:*

- *for any two positions in a relation $\mathcal{R}_f^*$ the letters at those positions in $E(\bar{s})$ are the same*
- *the regular constraints are satisfied.*

Proof. If the two conditions are satisfied then the substitution is a solution: given any equation $u = v$ the corresponding positions at both sides of $u(\bar{s}), v(\bar{s})$ are in the relation $\mathcal{R}$, so they contain the same letter and so $u(\bar{s}) = v(\bar{s})$. Moreover, we explicitly require that the regular constraints are satisfied.

In the other direction, observe that when $\bar{s}$ is a solution, then straight from the definition of $\mathcal{R}_f$ and f-solution we have that letters at position in relation $\mathcal{R}$ are the same, and so also the letters at positions of the relation $\mathcal{R}^*$ are the same. Also, the regular constraints are satisfied. □

Given a length function f and a set of positions I by $\mathcal{R}_f^*(I)$ we denote the set of positions that are in relation with any position in I. We say that I fixes this set of positions and that it fixes a solution, when $\mathcal{R}_f^*(I) = f(E)$, i.e. it fixes all positions in the system of equations. The intuition is that I is a set of positions, for which we already know the letters (because, those are positions of the letters, or we already inferred, what those letters are). Then $\mathcal{R}^*(I)$ gives us all positions, on which we can deduce the letters using only the knowledge about the letters on I. Using a concrete set of positions, instead of, say, all position of letters, allows us to formally argue that we can choose a subset of equations in order to determine the whole solution.

Clearly

Lemma H.7. *The operation $\mathcal{R}^*(\cdot)$ is monotone (on sets) and, further, $\mathcal{R}^*(\mathcal{R}^*(I)) = \mathcal{R}^*(I)$. In particular, if $I \subseteq I' \subseteq \mathcal{R}^*(I)$ then $\mathcal{R}^*(I') = \mathcal{R}^*(I)$.*

By Lemma H.6, if I fixes a solution (for a given length function f) then it is enough to specify the letter at the position from I to uniquely determine the f-solution; note that it is possible that no such solution exists. Moreover, we can access an arbitrary position in the $E(\bar{s})$ in nondeterministic logarithmic space, assuming that we can access a letter on position $i \in I$ in logarithmic space.

Lemma H.8. *Suppose that we are given:*

- *a system E of word equations with k variables $\bar{x} = (x_1, \dots, x_k)$ such that each equation is of length at most ℓ*
- *a length-function f*
- *set of positions I that fixes an f-solution.*

Then for a set $\{a_i\}_{i \in I}$, there is at most one f-solution $\bar{s}$ such that $E(s)[i] = a_i$ for each $i \in I$.

If such a solution $\bar{s}$ exists, then we can compute $e(\bar{s})[i]$ for any $e \in E$ and any position $i \in f(e)$ in $\mathrm{NSPACE}(\log(\ell) + \max_{j=1,\dots,k} \log |s_j|)$, assuming that a letter $a_{i'}$ for $i' \in I$ can be retrieved in such space. Similarly, we can compute $s_j[i]$ for any $j \in \{1, \dots, k\}$ and $1 \leq i \leq |s_j|$ in the same space bounds.

Given a set $\{a_i\}_{i \in I}$, establishing, whether there is such a solution $\bar{s}$ can be decided in $\mathrm{NSPACE}(\log(\ell) + \max_{i=1,\dots,k} \log |s_i|)$,

Proof. From Lemma H.6 it follows that if $\mathcal{R}^*(i, i')$ then $E(s)[i] = E(s)[i']$. Since I fixes the solution, for every f-solution $\bar{s}$ each position i is in relation $\mathcal{R}^*$ with some position $i' \in I$. Hence the letter at position i is $a_{i'}$ and therefore there is at most one such solution $\bar{s}$.

Observe, that given a position $i \in f(e)$ we can determine in the given space, whether $e(\bar{s})[i]$ is a position of a letter (and which letter) or of a variable and which position within the variable: we scan the appropriate side of e from left to right, counting the number of used positions, adding 1 for a letter and $f(p)$ for a variable p. The length of this side is at most $\ell \cdot \max_{j=1,\dots,k} |s_j|$, so the available space $\mathrm{NSPACE}(\log(\ell) + \max_{j=1,\dots,k} \log |s_j|)$ is sufficient.

Consider a graph, whose nodes are positions in $f(E)$ and there is an edge between i, i' if and only if $\mathcal{R}(i, i')$. Description of a node takes $O(\log(\ell) + \max_{j=1,\dots,k} \log |s_j|)$ space and using the routine above we can establish in $\mathrm{NSPACE}(\log(\ell) + \max_{j=1,\dots,k} \log |s_j|)$ whether there is an edge between i, i'. Question, whether $\mathcal{R}^*(i, i')$ is exactly the reachability problem in such a graph, and so it can be answered in the same (nondeterministic) space as above, by guessing consecutive positions and verifying that there is an edge between them. As I fixes the solution, to establish the letter at

position i we guess i' such that $i' \in I$ (there is one by assumption), verify that $\mathcal{R}^*(i, i')$ and return $a_{i'}$.

If we want to establish a letter $s_j[i]$ then we choose $e \in E$ using s_j and proceed in a similar way.

Concerning the last claim of the Lemma, we are now given a set $\{a_i\} i \in I$ and want to validate, whether there is an f-solution $\bar{s}$ such that $E(s)[i] = a_i$ for each $i \in I$. First, we check, whether there are $i, i' \in I$ such that $\mathcal{R}^*(i, i')$ and $a_i \neq a_{i'}$. If this is so then clearly there is no such solution, by Lemma H.6. In other words, for every pair $i, i' \in I$ such that $a_i \neq a_{i'}$ we should check that $\neg\mathcal{R}^*(i, i')$. Observe that this is a co-reachability problem in the graph defined above and so it can be verified in coNSPACE($\log(\ell) + \max_{j=1,\ldots,k} \log |s_j|$). By the Immerman–Szelepcsényi theorem coNSPACE(g) = NSPACE(g) for any (reasonable and fast-growing enough) function g (and the translation is effective) and so we can verify this condition in NSPACE($\log(\ell) + \max_{j=1,\ldots,k} \log |s_j|$), as desired. Note that it is possible that there are $i, i' \in I$ such that $\mathcal{R}^*(i, i')$, but in such case $a_i = a_{i'}$.

We verify each equation one by one and letter by letter, by computing the corresponding positions of two sides of the equation, i.e. for position i we find $i' \in I$ such that $\mathcal{R}^*(i, i')$ and take the letter $a_{i'}$ and verifying their equality or inequality (in the case of first differences); note that the position i' may be not unique, but the letter $a_{i'}$ is defined uniquely. In the same space we can also compute the transition function of the sides of the equation. Lastly, we can verify the regular constraints for variables by computing the transition function for each variable in the same way, i.e. letter by letter.

We conclude, that by Lemma H.6 there is an f-solution $\bar{s}$. □

In view of Lemma H.8 we should show that a system of equations has a well-defined small set positions that fix a solution. For a system of word equations E with regular constraints and length-function f define $\text{Letters}_f(E)$ as a set of positions of letters in E and $\text{Last}_f(E)$ the set of last positions of all occurrences of variables in $f(E)$. As usual, we use $\text{Letters}(E), \text{Last}(E)$ when the length function is clear from the context or unimportant.

Lemma H.9. *Let $\bar{s}$ be a length-minimal solution over the alphabet* $\text{cl}(\Sigma)$ *and f be its length function. Then* $\text{Letters}_f(E) \cup \text{Last}_f(E)$ *fixes $\bar{s}$.*

Note that for a fixed variable x is enough to take one position of the last letter of x, as all such position are in relation $\mathcal{R}^*_f$, but we take all for simplicity of the notion. Observe also that the assumption that we consider a solution over a ρ-closed alphabet is crucial: in an equation $x = x$ a regular constraint may force x to be a long word and clearly the whole solution is not fixed by the last letter of x; on the other hand, when we consider the ρ-closed alphabet, there is a solution that use exactly one letter, and so it is indeed fixed by Last.

Proof. Let $I = \text{Last}_f(E) \cup \text{Letters}_f(E)$. Consider a position $i \notin \mathcal{R}^*(I)$. We claim that:

- $\mathcal{R}^*(i, i')$ if and only if $\mathcal{R}^*(i+1, i'+1)$;
- if $\mathcal{R}^*(i, i')$ then $i'+1$ is not a position of a letter.

Suppose first that the first condition does not hold, consider the shortest sequence of position $i = i_0, i_1, \ldots, i_\ell$ such that $\mathcal{R}(i_j, i_{j+1})$ and it does not hold that $\mathcal{R}^*(i+1, i_\ell+1)$. Then by the minimality $\mathcal{R}(i_j+1, i_{j+1}+1)$ for $j < \ell-1$ and $\neg\mathcal{R}(i_{\ell-1}+1, i_\ell+1)$. Let us consider, why $\mathcal{R}(i_{\ell-1}, i_\ell)$ holds. If this is because they are corresponding positions on the sides of the equation then also the same holds for the positions one to the right, i.e. $\mathcal{R}(i_{\ell-1}+1, i_\ell+1)$ holds, which cannot be. If this is because they are corresponding positions of a variable then since $\mathcal{R}(i_{\ell-1}+1, i_\ell+1)$ does not hold, we get that $i_{\ell-1}+1, i_\ell+1$ are not corresponding positions coming from the same variable as $i_{\ell-1}, i_\ell$. So $i_{\ell-1}, i_\ell$ are the last positions in the variable, contradiction with the assumption. The proof when $\mathcal{R}^*(i+1, i_\ell+1)$ and $\neg\mathcal{R}^*(i, i_\ell)$, is similar; note that here in the last case we conclude that $i_{\ell-1}+1$ is a first letter of a variable, but then $i_{\ell-1}$ is either a position of a letter or a last in a variable.

For the second claim observe that it would contradict the assumption that $i \notin \mathcal{R}^*(I)$: if $i'+1$ were a position of a letter then i' would be a position of a letter or a last position in a variable, so $i' \in \mathcal{R}^*(I)$ and so also $i \in \mathcal{R}^*(I)$.

Now consider the letters a, b at positions $i, i+1$ and take the letter $c = a_{\rho(ab)}$, this letter is available, as we work over the ρ-closed alphabet. We construct a new solution: we replace a on positions in the whole equivalence class of i with c and each b on position in the whole equivalence class of b with ε (i.e., we delete those letters). We claim that this is still a solution. It satisfies the regular constraints, as $\rho(c) = \rho(ab)$ and by the first showed claim, we replace a at position i' if and only if we replace b at position $i'+1$. For the equation e observe that the corresponding letters at both sides of $e(\bar{s})$ are in the relation $\mathcal{R}^*$ and so both sides are modified in the same way and so the equality is preserved. The new substitution is well defined, as all corresponding position coming from substitution for x are in the relation. Yet, the constructed solution has smaller length than $\bar{s}$, contradiction with the length-minimality of $\bar{s}$. □

The following two lemmata show that "fixing" of positions has natural properties that we associate with solving equation by substituting positions in the variables, on which we know the letters: if the first/last letter of x is known (fixed by I), we can substitute x with ax/xb (the same positions are fixed by a union of I and new positions of letters): this is formalized in Lemma H.10. In Lemma H.11 we show that if corresponding positions at two sides of the equation are positions of letters (so they have no impact on the solution), they can be removed and this does not impact the solution (the same positions are fixed).

Note that we need to show one more property: a given equation can be used "in transit" to fix some other positions. Hence we should show that if two positions i, j are not fixed, but are in the relation before some operation, they are also in the relation afterwards.

Lemma H.10. *Given a system $E(\bar{x})$ of equations with regular constraints and its solution $\bar{s}$, let $s_i = s'_i b'_i$ where $b_i \in \Gamma$ for some i; define $\bar{s}'$ as $\bar{s}$ with s_i replaced with s'_i. Consider a system $E'(\bar{x}')$ obtained by replacing x_i with $x_i b_i$. Then $\bar{s}'$ is a solution of E'.*

Let $f, \mathcal{R}$ be the old the length function and relation (so for E) and $f', \mathcal{R}'$ be the new ones (so for E'). Let I be any set of positions and I': the set of position of letters introduced to E'. If $I' \subseteq \mathcal{R}^*(I)$ then $\mathcal{R}^*(I) = \mathcal{R}'^*(I \cup I')$, i.e. I and $I \cup I'$ fix the same set of positions.

Moreover, if $i, j \notin \mathcal{R}^*(I)$ then $\mathcal{R}^*(i, j) \iff \mathcal{R}'^*(i, j)$.

Similar claims hold also for substitution $x_i = a_i x''_i$.

Proof. The proof of the first part is a straightforward manipulation of definitions. Consider any $e \in E$ and its counterpart $e' \in E'$. Observe that $e'(\bar{s}')$ is exactly $e(\bar{s})$: we take a letter b_i from s_i and add it to the variable; hence $e'(\bar{s}')$ holds.

Note, that the lengths of corresponding equations in $f(E)$ and $f'(E')$ are the same. Suppose that $j \in \mathcal{R}^*(I)$, let $j = j_0, j-1, \ldots, j_r$ be a sequence of positions such that any two consecutive are in relation $\mathcal{R}$ and $j_r \in I$. Consider the first index j_ℓ such that $\neg\mathcal{R}'(j_\ell, j_{\ell+1})$; if there is no such index then $\mathcal{R}'^*(j, j_r)$ and so $j \in \mathcal{R}'^*(I \cup I')$. The only reason why $\mathcal{R}'(j_\ell, j_{\ell+1})$ does not hold is that j_ℓ is a last position in x_i (in E) and so j_ℓ is a position of a letter in E'. But then $j_\ell \in I'$ and so $j \in \mathcal{R}'(I \cup I')$.

In the other direction, suppose that $j = j_0, j_1, \ldots, j_r$ are a sequence of positions such that any two consecutive are in the ration $\mathcal{R}'$ and $j_r \in I \cup I'$. As $\mathcal{R}' \subseteq \mathcal{R}$, we have $\mathcal{R}(j_r, j_{r+1})$, and so $\mathcal{R}^*(j, j_r)$. If $j_r \in I$ then we are done. Otherwise $j_r \in I' \subseteq \mathcal{R}^*(I)$ which also implies that $j \in \mathcal{R}^*(I)$.

Concerning the last claim, suppose that $i, j \notin \mathcal{R}^*(I)$ and $\mathcal{R}^*(i, j)$. When we consider the sequence of positions that show that $\mathcal{R}^*(i, j)$, none of them is from I', so exactly the same sequence shows $\mathcal{R}'^*(i, j)$. In the other direction it is enough to observe that $\mathcal{R}' \subseteq \mathcal{R}$.

The proofs for the prefix are done in a symmetric way. □

Lemma H.11. *Suppose that we are given a system of equations $E \cup \{e\}$ and a length function f. Suppose that $(i, L, e), (e, R, e)$ are both positions of letters in $f(e)$, i.e. the corresponding letters at both sides of e are letters, let I be a set of positions containing those two positions. Let e' be an equation obtained by removing those positions. Let $\mathcal{R}, \mathcal{R}'$ be the corresponding relations.*

Then

$$\mathcal{R}^*(I) = \mathcal{R}'^*(I \setminus \{(i, L, e), (i, R, e)\}) \cup \{(i, L, e), (i, R, e)\} \ ,$$

i.e. the same sets of positions are fixed in the corresponding systems of both equations for corresponding sets.

Moreover, if $j, j' \notin \mathcal{R}^*(I)$ then $\mathcal{R}^*(j, j') \iff \mathcal{R}'^*(j, j')$.

Proof. The proof is trivial: since $(i, L, e), (e, R, e)$ are the corresponding positions on both sides of the equation, they are not in the relation $\mathcal{R}$ with any other positions. Hence, $\mathcal{R}'^*(I \setminus \{(i, L, e), (j, R, e)\})$ contains each position of $\mathcal{R}^*(I)$ except for $\{(i, L, e), (j, R, e)\}$.

The proof of the second claim is similar: assume $j, j' \notin \mathcal{R}(I)$ and $\mathcal{R}^*(j, j')$. Then the sequence of positions that show that $\mathcal{R}^*(j, j')$ cannot use $(i, L, e), (j, R, e)$, as they are only in relation with themselves. □

From now, the analysis crucially exploits the equations that we are dealing with are similar. The next lemma shows that if we have several similar equations and a fixed length function, then we can choose a subset of them (the size depends on the number of occurrences of variables), such that the new subsystem fixes the same set of positions. Note that this is not obvious: in general removing equations makes the relation $\mathcal{R}$ smaller and so fewer positions are fixed.

Lemma H.12. *Let E be a system of similar equations and f be a length function f such that there is an f-solution, let $\mathcal{R}$ be the corresponding relation.*

Then there is an $E' \subseteq E$ of size $O(m)$, where m is the number of occurrences of variables in any equation in E, such that for the corresponding relation $\mathcal{R}'$ we have

$$\mathcal{R}'^*(\mathrm{Last}(E') \cup \mathrm{Letters}(E')) = \mathcal{R}^*(\mathrm{Last}(E) \cup \mathrm{Letters}(E)) \cap f(E') \ .$$

Moreover, if $i, j \in f(E')$ and $i, j \notin \mathcal{R}^*(\mathrm{Last}(E) \cup \mathrm{Letters}(E))$ then $\mathcal{R}^*(i, j) \iff \mathcal{R}'^*(i, j)$.

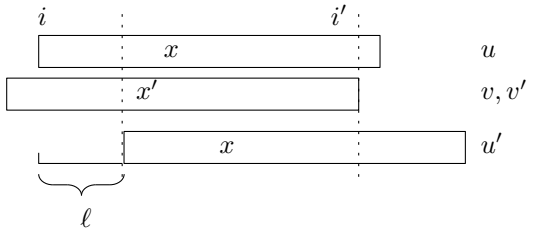

Figure 3

Proof. For the purpose of the proof we say that an equation (system of equations) equipped with a length function f is *simple*, when we cannot apply the operation from Lemma H.11. The simplification is a process of transforming an equation (system of equation) to the corresponding simple equation. It is easy to see that this does not depend on the order of application of the Lemma to different pairs of corresponding positions.

Suppose $u(\bar{x}) = v(\bar{x})$ (called also $e(\bar{x})$) and $u'(\bar{x}) = v'(\bar{x})$ (the $e'(\bar{x})$) are similar, i.e. the order of the variables is the same in u, u' and the same in v, v', and simple, i.e. the corresponding positions at sides of $u = v$ ($u' = v$) are not both positions coming from letters. If they are strongly similar, i.e. the corresponding sequence of letters between variables are of the same length, then by Lemma H.3 and the assumption that they are simple we conclude that they are the same equation, and so we are done.

So consider a left-most occurrence of a variable such that the corresponding occurrences in $f(u), f(u')$ (or $f(v)$ and $f(v')$) are different. By symmetry, let it begin at position i in $f(u)$ and $i + \ell$ in $f(u')$; take the occurrence which maximizes ℓ, if there are two possible choices. Let this variable be x. Consider the positions $[i \ldots i + \ell - 1]$ in $f(u')$. If any of them was a position of a variable, then this variable needs to occur in $f(u)$ as well, and it is on position before i, which contradicts our choice of i. So those are all position of letters, say of a word w, in particular, they are all fixed by $\mathrm{Letters}(\{e, e'\})$. Consider also, what is at position i in $f(v')$. By simplicity, this cannot be a letter, as there is a letter at this position in u'. So this is a position of a variable, say x'.

Consider the first position of this occurrence of x' in $f(v')$. Suppose first that it is a position earlier than i, see Fig. 3. Then by the choice of position i, also in $f(v)$ it begins at the same position. Let i' be the earliest of positions: end of positions coming from the variable x beginning at position i in $f(u)$ and the end of positions coming from the variable x' in $f(v)$ that include the position i. Then we conclude that all positions in $[i \ldots i']$ are fixed by the positions $[i \ldots i + \ell - 1]$, Fig. 3 shows that this word is periodic with period ℓ, in terms of fixing we can show that each position in $[i + \ell \ldots i']$ is in relation $\mathcal{R}^*$ with a position ℓ positions earlier:

- $\mathcal{R}((j, L, u = v), (j, R, u = v))$ (equation $u = v$),
- $\mathcal{R}((j, R, u = v), (j, R, u' = v'))$ (corresponding positions of a variable)

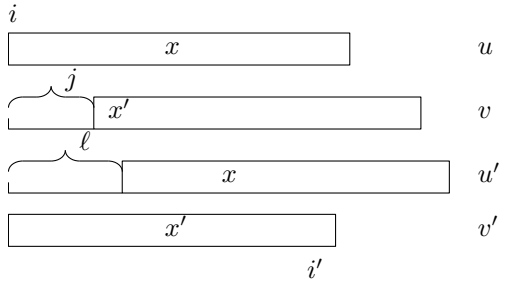

Figure 4

- $\mathcal{R}((j, R, u' = v'), (j, L, u' = v'))$ (equation $u' = v'$)
- $\mathcal{R}((j, L, u' = v'), (j - \ell, L, u = v))$ (corresponding positions in x).

Note that this takes into the account the special case when $i' \leq i+\ell$: in this case simply those positions are fixed.

We make a partial substitution to variable x at positions $[i \dots i']$. We change the length function appropriately. Observe that if i' is the last position of x then we have substituted x completely and if it is the last position of x' then in $f(u) = f(v)$ the x begins at position $i' + 1$ and x' and at position i', We then simplify both equations. By Lemma H.10, this does not change the set of fixed positions of the equation. We then repeat the whole step (beginning with removal of corresponding positions of letters).

The analysis, when the first position of x' in $f(v')$ is i is similar. Consider the first position of x' in $f(v)$. It cannot be earlier than i, as this would contradict the choice of i. If the first position of x' in v is i then the analysis is the same as in the previous case. So suppose that it is at position $i+j$; let i' be the earliest of the ends of positions of x (in $f(u')$) and x' (in $f(v')$). Consider first the case that $j \leq \ell$ (recall that x occurs at position $i + \ell$ in u'), see Fig. 4. We show that positions $[i \dots i']$ in $f(v')$ are fixed by positions of letters. First: the position $[i \dots i + j - 1]$ are positions of letters in $f(v)$ so the corresponding positions in $f(u)$ are fixed (corresponding sides of the equations), so also positions $[i + \ell \dots i + \ell + j - 1]$ are fixed in $f(u')$ (corresponding positions of the variable). The positions $[i \dots i+\ell-1]$ are positions of letters, so all positions $[i \dots i+\ell+j-1]$ are fixed in $f(u')$, so they are also fixed in $f(v')$ (corresponding sides of the equation); i.e. the first $\ell + j$ position of p' are fixed in $f(v')$. Consider now w position $j' + \ell + j \in [i + \ell + j \dots i']$ if $f(v')$. We show that it is in relation with positions j', which will end the proof.

- $\mathcal{R}((j'+\ell+j, R, u' = v'), (j'+\ell+j, L, u' = v'))$ (corresponding sides of the equation)
- $\mathcal{R}((j' + \ell + j, L, u' = v'), (j' + j, L, u = v))$ (corresponding variables)
- $\mathcal{R}((j' + j, L, u = v), (j' + j, R, u = v))$ (corresponding sides of the equation)
- $\mathcal{R}((j' + j, R, u = v), (j', R, u' = v'))$ (corresponding variables)

We proceed as in the previous case, making a substitution at positions $[i + \ell \dots i']$ to x in u' and to $[i \dots i']$ to x' in v'. Observe, that we have substituted either x (in u') or x' (in v') completely.

The last remaining case is that $\ell < j$. But this contradicts the choice of ℓ: we could choose variable x' instead of x, with x' occurring at positions i (in $f(v')$) and $i + j$ (in $f(v)$), and $j > \ell$, while ℓ was chosen as the maximal one.

Observe that in the third case the partial substitution fully substitutes a variable, so this can happen at most k times. In the first and second case it is also possible that a variable is removed, in which case the same analysis applies (with a common bound of k). The only remaining case is that we only substitute a prefix of x. In this case we analyze the *overlaps* between variables: two occurrences of a variable overlap in $f(u) = f(v)$, when there is a position i which comes from a variable in $f(u)$ and in $f(v)$. Observe that in the analysis in the first case above there was an overlap between occurrence of variables x, x' in $f(u') = f(v')$ and the partial substitution removes all such overlaps. Moreover, a partial substitution cannot increase the number of overlaps (it can decrease it though). It remains to estimate the maximal initial number of overlaps. Each overlap is associated with two ends of a substitution for variables, and initially there are at most $2m$ such ends. Each such step reduces this number by at least 1 and partial substitution cannot increase this number. So there are at most $2m$ such steps in total; note that a removal of variable also removes at least one such end, so we have a joint bound of $2m$.

Concerning the claim on relations: first, we can simplify the input system of equations. By Lemma H.11 the relation $\mathcal{R}$ remain the same after the simplification and each pair of removed positions were ir $\mathcal{R}$ only with themselves.

By E'_0 we denote the subsystem of equations for which the algorithm performed some simplification, we denote this subsystem after consecutive steps by $E'_1, E'_2, \dots$. We consider also the set of positions $I'_0 = \mathrm{Last}(E'_0) \cup \mathrm{Letters}(E'_0), I'_1, \dots$ and the relations $\mathcal{R}'_0, \mathcal{R}'_1, \dots$. For the purpose of the proof, let us also denote by $J'_0, J'_1, \dots$ the set of positions so far-removed from the subsystem. We inductively show that:

(1) $\mathcal{R}'^*_0(I'_0) \cap f(E'_{h+1}) = \mathcal{R}'^*_{h+1}(I'_{h+1})$
(2) $\mathcal{R}'^*_0(i, j) \iff \mathcal{R}'^*_h(i, j)$ for $i, j \notin \mathcal{R}'^*_0(I'_0)$;
(3) $\mathrm{Letters}(E'_h) \subseteq I'_h$;
(4) $J'_h \subseteq \mathcal{R}'^*_0(I'_0)$.

Suppose that we make a partial substitution, transforming E'_h with I'_h to E'_{h+1}. Define the new set of positions as the old ones union with the positions of introduced letters. As $\mathrm{Letters}(E'_h) \subseteq I'_h$, this means that $I'_{h+1} = I'_h \cup \mathrm{Letters}(E'_{h+1})$ and so 3 holds. By Lemma H.10 we get that $\mathcal{R}'^*_h(I'_h) = \mathcal{R}'^*_{h+1}(I'_{h+1})$, as $f(E'_{h+1}) = f(E'_h)$, this shows 1. Moreover, $\mathcal{R}'^*_h(i, j) \iff \mathcal{R}'^*_{h+1}(i, j)$ holds when $i, j \notin \mathcal{R}'^*_h(I_h)$ by Lemma H.10 and so 2 holds; and 4 holds as here nothing changed.

When the system of equations is transformed by removing corresponding positions of letters at two different sides of the equation, then those positions are in $\mathcal{R}'^*_h(I'_h)$, so they were also in $\mathcal{R}^*_0(I'_0)$ and so 4 holds. Define I'_{h+1} as I'_h minus the removed positions. Then 3 clearly holds and 1, 2 hold by Lemma H.11.

Consider similar sets for the whole system of equations, i.e. consider the system $E = E_0, E_1, \dots$, that are obtained using similar operations, i.e. by partial substitutions and simplifications, by J_h denote the position that were removed due to simplifications. Define $I_0 = \mathrm{Last}(E_0) \cup \mathrm{Letters}(E_0)$. Define also $\mathcal{R}_0, \mathcal{R}_1, \dots$. We show that:

(5) $\mathcal{R}_0^*(I_0) \cap f(E_{h+1}) = \mathcal{R}_{h+1}^*(I_{h+1})$
(6) $\mathcal{R}_0^*(i,j) \iff \mathcal{R}_h^*(i,j)$ for $i,j \notin \mathcal{R}_0^*(I_0)$;
(7) $\text{Letters}(E_h) \subseteq \text{Letters}(E_0) \cup \mathcal{R}_0^*(I_0')$
(8) $J_h \subseteq \text{Letters}(E_0) \cup \mathcal{R}_0^*(I_0')$.

Condition 5, 6 is shown as 1, 2. For 7 observe that those are either original letters or ones introduced due to partial substitutions and those are letters in $\mathcal{R}_h^*(I_h')$, which is equal to $\mathcal{R}_0^*(I_0')$. For 8 observe that we simplify positions such that both have letters on them, so this follows from 7.

At the end we have that all equations in E_h and E_h' are the same equation. Hence

$$\mathcal{R}_h^*(I_h) \cap f(E_h') = \mathcal{R}_h'^*(I_h')$$

By 1 and 5, we get

$$\mathcal{R}_0^*(I_0) \cap f(E_h') = \mathcal{R}_0'^*(I_0')$$

We are still missing positions that we deleted from E_0' to get E_h', those are J_h' and by 4 we have $J_h' \subseteq \mathcal{R}_0'^*(I_0')$ and so we get

$$\mathcal{R}_0^*(I_0) \cap f(E_0') = \mathcal{R}_0'^*(I_0')$$

For the last condition, when $i,j \in f(E_0')$ and $i,j \notin R_0^*(I_0)$ then we already know that $i,j \notin R_0'^*(I_0')$ and so

$$R_0'^*(i,j) \iff R_h'^*(i,j) \iff R_h^*(i,j) \iff R_0^*(i,j)$$

with the first equivalence following from 2, second by observation that for the final equations $\mathcal{R}'^*$ and $\mathcal{R}^*$ are the same on $f(E_h')$ and the last from 6. □

Lemma H.13. *Let $E = \bigcup_{i=1}^{h} E_i$ be a system of equations (with regular constraints), such that equations in each E_i are all similar, let each equation in E has at most m occurrences of variables. Let $\bar{s}$ be a length-minimal solution over an alphabet* $\text{cl}(A)$*, f be the length function of $\bar{s}$.*

The there is a subsystem $E_0 \subseteq E$ of size $O(mh)$ such that $\text{Last}(E_0) \cup \text{last}(E_0)$ *fixes $\bar{s}$ (for E_0).*

Proof. For each E_i we apply Lemma H.12, it returns a subsystem E_i' of size at most $2m$, define $E' = \bigcup_i E_i'$, it has the appropriate size. Let $\mathcal{R}$ be the relation in E and $\mathcal{R}'$ in E' and similarly $\mathcal{R}_i$ be the relation in E_i and $\mathcal{R}_i'$ in E_i'.

By Lemma H.9 the $\text{Last}(E) \cup \text{Letters}(E)$ fix the solution $\bar{s}$. Consider any position $i \in f(E')$, then $i \in \mathcal{R}^*(\text{Last}(E) \cup \text{Letters}(E))$, so consider a sequence of positions $i_0, i_1, \ldots i_h = i$ such that each two consecutive are in the relation $\mathcal{R}$ and $i_0 \in \text{Last}(E) \cup \text{Letters}(E)$, without loss of generality, no position except of i_0 is in $\text{Letters}(E) \cup \text{Last}(E)$ (if not then we can remove the prefix before this position). Let $e \in E$ be the equation such that $i_0 \in f(e)$, then also $i_1 \in f(e)$ and i_1 is a position within variable. So there is a position $i_1' \in f(E')$ that is a corresponding position of a variable, as for e we have chosen some similar equations to E'. By Lemma H.12, $i_1' \in \mathcal{R}'^*(\text{Last}(E') \cup \text{Letters}(E'))$.

Consider the sequence $i_1, \ldots, i_r$ and subdivide it into subsequences, such that one subsequence uses positions from similar equations. Take any such subsequence $i_j, i_{j+1}, \ldots, i_{j+q}$, say it corresponds to similar equations E_h. Then any two consecutive position are in relation $\mathcal{R}_h$. Since i_{j-1} is not in class of equations as i_j (by the division into subsequences), then they are both corresponding positions of the variable. In particular, there is a position $i_j' \in f(E_h')$ that is also the corresponding position of (the same) variable. Similarly, i_{j+q} and i_{j+q+1} are corresponding positions of the variables and there is a corresponding position $i_{j+q}' \in f(E_h')$. Then $\mathcal{R}_h^*(i_{j-1}', i_{j+q}')$. If either one of them is in $\mathcal{R}_h^*(\text{Last}(E_h) \cup \text{Letters}(E_h))$ then also the other one is and so by Lemma H.12 they are also both in $\mathcal{R}_h'^*(\text{Last}(E_h') \cup \text{Letters}(E_h')) \subseteq \mathcal{R}'^*(\text{Last}(E') \cup \text{Letters}(E'))$. If they are both not in $\mathcal{R}_h^*(\text{Last}(E_h) \cup \text{Letters}(E_h))$ then by the same Lemma $\mathcal{R}_h^*(i_{j-1}', i_{j+q}')$ implies $\mathcal{R}_h'^*(i_{j-1}', i_{j+q}')$ and so also $\mathcal{R}'^*(i_{j-1}', i_{j+q}')$.

Consider the whole such created sequence, take the last of its elements that is in $\mathcal{R}'^*(\text{Last}(E') \cup \text{Letters}(E'))$, we have shown that i_1' is there. Then each two consecutive are either corresponding positions in the same variable (and both are in $f(E')$), so are in $\mathcal{R}'^*$, or were shown to be in relation $\mathcal{R}'^*$. Hence the last element, i.e. i, is also in $\mathcal{R}'^*(\text{Last}(E') \cup \text{Letters}(E'))$.

It remains to observe that $\text{Last}(E) \cup \text{Letters}(E)$ fix all position in $f(E')$, so also $\text{Last}(E') \cup \text{Letters}(E')$ fix all position in $f(E')$, so they fix the solution $\bar{s}$. □

Lemma H.14. *Assume accepting computation of $\mathcal{A}$*

$$\pi := (q_0, v_0) \to_{(a_1,\varphi_1)} \cdots \to_{(a_n,\varphi_n)} (q_n, v_n)$$

for some valuation of $\mathcal{P}$ in Γ^*.

Then there is another valuation $\mu : \mathcal{P} \to \text{cl}_\rho(\Gamma)^*$, such that π is accepting for it, $|\mu(p_i)|$ is polynomial (in graph size) and there is an NL *algorithm that given i,j returns letter of $\mu(p_i)[j]$.*

Proof. For each φ on the path π choose the set of equations and regular constraints that make φ satisfiable. Let n be the size of the graph and maximal size of data in it and m the maximal sum of length of equations in any φ_i in π. Let all the constraints be over $\mathbb{M}_q$. Suppose there are k parameters.

Without loss of generality π has length at most (see discussion after Proposition2.3) $n^r \times |Q| \times k$, here r is the number of registers. So the total length of equations in the constructed system E is $h(n)$, for some polynomial h with degree depending on the query, and the system is over variables $\mathcal{P}$, so there are k of them.

This system of equations is satisfiable, so by Lemma H.5, the length-minimal solution has length $q(n)$ times a function not depending on n, where q is a polynomial with degree depending on m, q, k (note that n_v from Lemma H.5, is at most m in our setting). We set those values as the values assigned by μ.

The NL algorithm initially guesses the length function f of $\mu(p_i)$ for each i, let $\mathcal{R}$ be the corresponding relation on the By Lemma H.9 the set of positions $\text{Last}(E) \cup \text{Letters}(E)$ fixes the solution $\mu(\bar{p})$. Then by Lemma H.13 there is a subsystem $E_0 \subseteq E$ of this system of equations of size $O(km)$ such that E_0 has at least one occurrence of each p_i and $\text{Last}(E_0) \cup \text{Letters}(E_0)$ fix the solution for E_0 (in particular, fix the $\mu(\bar{p})$). We guess the letters (over the ρ-closed alphabet) that are on the last positions of each parameter.

Finally, by Lemma H.8 in NL space we can answer queries concerning letters of $\mu(\bar{p})$. □

We can now prove Theorem 5.2. First, by Lemma H.2, we can consider valuation of $\mathcal{P}$ from $\text{cl}_\rho(\Gamma)$, as an existence of solutions of word equations (with regular constraints) over Γ and $\text{cl}_\rho(\Gamma)$ is equivalent. Suppose that there is an accepting path for the input query. Then by Lemma H.14 there are values of the parameters that are of polynomial length (in the graph size), we guess those

lengths, and there is an NL algorithm that returns a given position of the value of p_i; we guess this algorithm as well (inspection of the proof of Lemma H.14 shows that it applies Lemma H.8 to a guessed small subset of equations, for which the last letters of parameters are guessed). Now we follow the accepting path, nondeterministically guessing consecutive vertices in $\mathbf{G}$ and verifying that they are connected by a view, and transitions of the $\mathcal{A}$ and changes in the registers. Given a guard φ we can evaluate each of its atomic formula in NL: first, we treat the active domain quantifiers as conjunction/disjunction over whole $adom(\mathbf{S})$, those can be enumerated and accessed in NL. The length function is known and polynomial in n, and the substitution for the variables are accessible in NL, so this follows from Lemma H.8 (note that there are no variables except $\mathcal{P}$). Hence also φ can be evaluated in NL.

I REDUCTION FROM CRDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$ TO RDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$

Assume a CRDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$-query:

$$Q(\bar{z}) \leftarrow \bigwedge_{i=1}^{n} x_i \rightarrow_{\mathcal{A}_i} y_i \wedge \varphi(\mathcal{P}, \bar{x}, \bar{y}).$$

Assume w.l.o.g. that each $\mathcal{A}_i$ has precisely l active-domain registers $\mathcal{R} := \{r_1, \ldots, r_l\}$. Let $\mathbf{G}$ be the graph consisting of view definitions $(V, \{E_a\}_{a\in\Sigma})$ and the underlying $\mathbf{T}$-structure $\mathbf{S}$. Finally, let μ be the given instantiation of $\bar{z}$. We construct a new $\mathcal{A}$ with $l + 2n$ active-domain registers $\mathcal{R}_1 := \mathcal{R} \cup \bar{x} \cup \bar{y}$. Let Q^i be the set of states of $\mathcal{A}_i$. The set Q of states of $\mathcal{A}$ is defined to be $\{q_0\}$ union a disjoint union of all Q^i, say, consisting of states of the form (q, i) for each $q \in Q^i$. For each $z \in \bar{z}$, we add the new unary relation $S_z = \{\mu(z)\}$ to $\mathbf{S}$. Let $\Sigma' := \Sigma \bigcup [1, n]\}$ (assume $[1, n] \cap \Sigma = \emptyset$). Define $E_i(z, z') := \theta_i(z) \wedge \theta_i(z')$. Here, $\theta(z) := V(z)$ if $i = 1$ or $i > 1$ and $y_{i-1} \notin \bar{z}$; otherwise, $\theta(z) := \exists y \in adom(z = y \wedge S_{y_{i-1}}(y))$. Similarly, $\theta'(z) := V(z)$ if $x_i \notin \bar{z}$; otherwise, $\theta'(z) := \exists x \in adom(z = x \wedge S_{x_i}(x))$. The output RDPQ$(\mathbb{L}, \mathbf{T}, \sigma)$ query evaluation problem is $Q' := x \rightarrow_{\mathcal{A}} y$ with two arbitrary start and final nodes s, t. The automaton $\mathcal{A}$ starts with q_0, from which there is a transition $(q_0, (1, \varphi(\mathcal{P}, \bar{x}, \bar{y}), q_0^1)$, for each start state q_0^1 of $\mathcal{A}_1$. $\mathcal{A}$ will then simulate $\mathcal{A}_1, \ldots, \mathcal{A}_n$ in this order, while transitioning from $\mathcal{A}_i$ to $\mathcal{A}_{i+1}$ using E_{i+1}. To keep the instantiations of $\bar{x}$ and $\bar{y}$ consistent, we do a check $curr = x_i$ in any outgoing transition from (q_0^i, i), for each start state q_0^i of $\mathcal{A}_i$, and $curr = y_i$ in any incoming transition to (q_F, i), for each final state q_F of $\mathcal{A}_i$. [WLOG, we assume that each start state has no incoming transitions, and each final state has no outgoing transitions.]

It is easy to see that the above reduction can be performed in logspace, and the output query Q' depends only on Q.