Logical Methods in Computer Science
Volume 19, Issue 4, 2023, pp. 20:1–20:38
https://lmcs.episciences.org/

Submitted Oct. 26, 2021
Published Dec. 07, 2023

A TRICHOTOMY FOR REGULAR TRAIL QUERIES

WIM MARTENS, MATTHIAS NIEWERTH, AND TINA POPP

University of Bayreuth, Germany

Abstract. Regular path queries (RPQs) are an essential component of graph query languages. Such queries consider a regular expression r and a directed edge-labeled graph G and search for paths in G for which the sequence of labels is in the language of r. In order to avoid having to consider infinitely many paths, some database engines restrict such paths to be *trails*, that is, they only consider paths without repeated edges. In this article we consider the evaluation problem for RPQs under trail semantics, in the case where the expression is fixed. We show that, in this setting, there exists a trichotomy. More precisely, the complexity of RPQ evaluation divides the regular languages into the finite languages, the class $\mathsf{T}_{\mathsf{tract}}$ (for which the problem is tractable), and the rest. Interestingly, the tractable class in the trichotomy is larger than for the trichotomy for *simple paths*, discovered by Bagan, Bonifati, and Groz [JCSS 2020]. In addition to this trichotomy result, we also study characterizations of the tractable class, its expressivity, the recognition problem, closure properties, and show how the decision problem can be extended to the enumeration problem, which is relevant to practice.

1. Introduction

Graph databases are a popular tool to model, store, and analyze data [Neo, Tig, Ora, Wik, DBp]. They are engineered to make the *connectedness of data* easier to analyze. This is indeed a desirable feature, since some of today's largest companies have become so successful because they understood how to use the connectedness of the data in their specific domain (e.g., Web search and social media). One aspect of graph databases is to bring tools for analyzing connectedness to the masses.

Regular path queries (RPQs) are a crucial component of graph databases, because they allow reasoning about arbitrarily long paths in the graph and, in particular, paths that are longer than the size of the query. A regular path query essentially consists of a regular expression r and is evaluated on a graph database which, for the purpose of this article, we view as an edge-labeled directed graph G. When evaluated, the RPQ r searches for paths in G for which the sequence of labels is in the language of r. The return type of the query varies: whereas most academic research on RPQs [MW95, Bar13, BLR11, LM13, ACP12] and SPARQL [W3C13] focus on the first and last node of matching paths, Cypher [Ope]

Key words and phrases: Regular languages, query languages, path queries, graph databases, databases, complexity, trails, simple paths.

* A shorter version of this article has appeared in STACS 2020, see [MNT20].

returns the entire paths. G-Core, a recent proposal by partners from industry and academia, sees paths as "first-class citizens" in graph databases [AAB$^+$18].

In addition, there is a large variation on which types of paths are considered. Popular options are *all paths*, *simple paths*, *trails*, and *shortest paths.* Here, *simple paths* are paths without repeated nodes and *trails* are paths without repeated edges. Academic research has focused mostly on *all paths*, but Cypher 9 [Ope, FGG$^+$18], which is perhaps the most widespread graph database query language at the moment, uses *trails.* Since the trail semantics in graph databases has received virtually no attention from the research community yet, it is crucial that we improve our understanding.

In this article, we study the *data complexity* of RPQ evaluation under trail semantics. That is, we study variants of RPQ evaluation in which the RPQ r is considered to be fixed. As such, the input of the problem only consists of an edge-labeled (multi-)graph G and a pair (s, t) of nodes and we are asked if there exists a trail from s to t on which the sequence of labels matches r. One of our main results is a trichotomy on the RPQs for which this problem is in AC^0, NL-complete, or NP-complete, respectively. By $\mathsf{T}_{\mathsf{tract}}$, we refer to the class of tractable languages (assuming NP $\neq$ NL).

In order to increase our understanding of $\mathsf{T}_{\mathsf{tract}}$, we study several important aspects of this class of languages. A first set of results is on characterizations of $\mathsf{T}_{\mathsf{tract}}$ in terms of closure properties and syntactic and semantic conditions on their finite automata. In a second set of results, we therefore compare the expressiveness of $\mathsf{T}_{\mathsf{tract}}$ with yardstick languages such as $\mathbf{FO}^2[<]$, $\mathbf{FO}^2[<, +1]$, $\mathbf{FO}[<]$ (or *aperiodic languages*), and $\mathsf{SP}_{\mathsf{tract}}$. The latter class, $\mathsf{SP}_{\mathsf{tract}}$, is the closely related class of languages for which the data complexity of RPQ evaluation under *simple path* semantics is tractable.[1] Interestingly, $\mathsf{T}_{\mathsf{tract}}$ is strictly larger than $\mathsf{SP}_{\mathsf{tract}}$ and includes languages outside $\mathsf{SP}_{\mathsf{tract}}$ such as a^*bc^* and $(ab)^*$ that are relevant in application scenarios in network problems, genomic datasets, and tracking provenance information of food products [PS] and were recently discovered to appear in public query logs [BMT17, BMT19]. Furthermore, every *single-occurrence regular expression* [BNSV10] is in $\mathsf{T}_{\mathsf{tract}}$, which can be a convenient guideline for users of graph databases, since *single-occurrence* (every alphabet symbol occurs at most once) is a very simple syntactical property. It is also popular in practice: we analyzed the 50 million RPQs found in the logs of [BMT18] and discovered that over 99.8% of the RPQs are single-occurrence regular expressions.

We then study the recognition problem for $\mathsf{T}_{\mathsf{tract}}$, that is: given an automaton, does its language belong to $\mathsf{T}_{\mathsf{tract}}$? This problem is NL-complete (resp., PSPACE-complete) if the input automaton is a DFA (resp., NFA). We also treat closure under common operations such as union, intersection, reversal, quotients and morphisms.

We conclude by showing that also the *enumeration problem* is tractable for $\mathsf{T}_{\mathsf{tract}}$. By tractable, we mean that the paths that match the RPQ can be enumerated with only *polynomial delay* between answers. Technically, this means that we have to prove that we cannot only solve a decision variant of the RPQ evaluation problem, but we also need to find witnessing paths. We prove that the algorithms for the decision problems can be extended to return *shortest paths.* This insight can be combined with Yen's Algorithm [Yen71] to give a polynomial delay enumeration algorithm.

Related Work. RPQs on graph databases have been studied since the end of the 80's and are now finding their way into commercial products. The literature usually considers the

[1]Bagan et al. [BBG20] called the class $\mathsf{C}_{\mathsf{tract}}$, which stands for "tractable class". We distinguish between $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ here to avoid confusion between simple paths and trails.

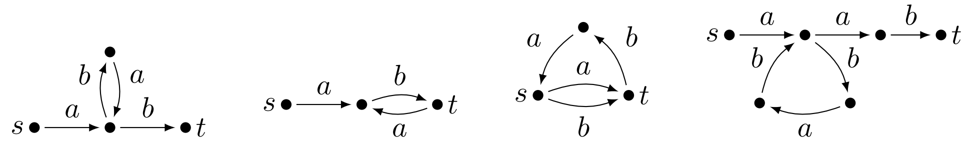

Figure 1: Directed, edge-labeled graphs that have a trail from s to t.

variant of RPQ evaluation where one is given a graph database G, nodes s, t, and an RPQ r, and then needs to decide if G has a path from s to t (possibly with loops) that matches r. For arbitrary and shortest paths, this problem is well-known to be tractable, since it boils down to testing intersection emptiness of two NFAs.

Mendelzon and Wood [MW95] studied the problem for simple paths, which are paths without node repetitions. They observed that the problem is already NP-complete for regular expressions a^*ba^* and $(aa)^*$. These two results rely heavily on the work of Fortune et al. [FHW80] and LaPaugh and Papadimitriou [LP84].

Our work is most closely related to the work of Bagan et al. [BBG20] who, like us, studied the complexity of RPQ evaluation where the RPQ is fixed. They proved a trichotomy for the case where the RPQ should only match simple paths. In this article we will refer to this class as $\mathsf{SP}_{\mathsf{tract}}$, since it contains the languages for which the *simple path* problem is tractable, whereas we are interested in a class for *trails*. Martens and Trautner [MT19] refined this trichotomy of Bagan et al. [BBG20] for *simple transitive expressions*, by analyzing the complexity where the input consists of both the expression and the graph.

Paperman has integrated the classes $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ in his tool called Semigroup Online [Pap22]. The tool can process a regular expression as input and can tell the user whether the language is in $\mathsf{SP}_{\mathsf{tract}}$, $\mathsf{T}_{\mathsf{tract}}$, and/or in many other important classes of languages.

Trails versus Simple Paths. We conclude with a note on the relationship between simple paths and trails. For many computational problems, the complexities of dealing with simple paths or trails are the same due to two simple reductions, namely: (1) constructing the line graph or (2) splitting each node into two, see for example Perl and Shiloach [PS78, Theorem 2.1 and 2.2]. As soon as we consider labeled graphs, the line graph technique still works, but not the nodes-splitting technique, because the labels on paths change. As a consequence, we know that finding trails is at most as hard as finding simple paths, but we do not know if it has the same complexity when we require that they match a certain RPQ r.

In this article we show that the relationship is strict, assuming NL $\neq$ NP. An easy example is the language $(ab)^*$, which is NP-hard for simple paths [LP84, MW95], but—assuming that a-labeled edges are different from b-labeled edges—in NL for trails. This is because every path from s to t that matches $(ab)^*$ can be reduced to a trail from s to t that matches $(ab)^*$ by removing loops (in the path, not in the graph) that match $(ab)^*$ or $(ba)^*$. In Figure 1 we depict four small graphs, all of which have trails from s to t. (In the three rightmost graphs, there is exactly one path labeled $(ab)^*$, which is also a trail.)

Outline. We note that this is a full version of the work presented in [MNT20]. In addition to adding the full proofs, we generalize our results to multigraphs throughout the article. In Section 2 we define our notation, Section 3 introduces the class $\mathsf{T}_{\mathsf{tract}}$, which contains exactly the regular languages for which finding a trail from s to t is in polynomial time (assuming $P \neq NP$). We prove this dichotomy in Section 4. (The article is named trichotomy because we can also differentiate between finite and infinite languages. For the first, finding such a path is in AC^0 while NL-hard for the latter.) After giving some interesting closure properties of $\mathsf{T}_{\mathsf{tract}}$ in Section 5 and extending the algorithm for languages in $\mathsf{T}_{\mathsf{tract}}$ to an enumeration algorithm, we conclude our work in Section 7. The most complex part of this article is in Section 3, where we give several equivalent definitions of $\mathsf{T}_{\mathsf{tract}}$, some of which are needed for the proof of its tractability, others, like the syntactic definition given in Theorem 3.31 might be useful for database engineers, while others are used to compare $\mathsf{T}_{\mathsf{tract}}$ to well-known classes such as $\mathbf{FO}$ or $\mathbf{FO}^2[<, +1]$.

2. Preliminaries

We use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. By Σ we always denote a finite alphabet, i.e., a finite set of *symbols*. We always denote symbols by a, b, c, d and their variants, like a', a_1, b_1, etc. The regular expressions we use in this article are defined as follows: $\emptyset, \varepsilon$, and every symbol in Σ is a regular expression. When r and s are regular expressions, then (rs), $(r + s)$, $(r?)$, (r^*), and (r^+) are also regular expressions. We use the usual precedence rules to omit parentheses. For $n \in \mathbb{N}$, we use r^n to abbreviate the n-fold concatenation $r \cdots r$ of r. The language $L(r)$ of a regular expression r is defined as usual. For readability, we often omit the $L(\cdot)$ and only write r for the language of r. A *word* is a finite sequence $w = a_1 \cdots a_n$ of symbols.

We consider edge-labeled directed multigraphs $G = (V, E, \mathcal{E})$, where V is a finite set of nodes, E is a finite set of edges, and $\mathcal{E} \colon E \to V \times \Sigma \times V$ is a function that maps each edge identifier to a tuple (v_1, a, v_2) describing the origin, the label, and the destination node of the edge. We denote v_1 by origin(e), a by lab(e) and v_2 by destination(e). We emphasize that $\mathcal{E}$ does not need to be injective, i.e., there might be several edges with identical origin, label, and destination. The size of G is defined as $|V| + |E|$. A (simple) graph is a multigraph where $\mathcal{E}$ is injective. A *path* p from node s to t is a sequence $e_1 \cdots e_m$ of edges such that origin$(e_1) = s$, destination$(e_m) = t$, and for $1 \leq i < m$ it holds that destination$(i) =$ origin$(i + 1)$. By $|p|$ we denote the number of edges of a path. A path is a *trail* if every edge e appears at most once[2] and a *simple path* if all the nodes in origin(e_1) and destination$(e_1), \ldots,$ destination(e_m) are different. We note that each simple path is a trail but not vice versa. We denote lab$(e_1) \cdots$ lab(e_m) by lab(p). Given a language $L \subseteq \Sigma^*$, path p *matches* L if lab$(p) \in L$. For a subset $E' \subseteq E$, path p is E'-restricted if every edge of p is in E'. Given a trail p and two edges e_1 and e_2 in p, we denote the subpath of p from e_1 to e_2 by $p[e_1, e_2]$.

We define an NFA A to be a tuple $(Q, \Sigma, I, F, \delta)$ where Q is the finite set of states; $I \subseteq Q$ is a set of initial states; $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation; and $F \subseteq Q$ is the set of accepting states. Strongly connected components of (the graph of) A are simply called *components*. Unless noted otherwise, components will be non-trivial, i.e., containing at least one edge. We write $C(q)$ to denote the strongly connected component of state q.

By $\delta(q, w)$ we denote the states reachable from state q by reading w. Given a path p, we also slightly abuse notation and write $\delta(q, p)$ instead of $\delta(q, \text{lab}(p))$. We denote by $q_1 \rightsquigarrow q_2$ that state q_2 is reachable from q_1. Finally, L_q denotes the set of all words accepted from q and $L(A) = \bigcup_{q \in I} L_q$ is the set of words accepted by A. For every state q, we denote by Loop(q) the set $\{w \in \Sigma^+ \mid \delta_L(q, w) = q\}$ of all non-empty words that allow to loop on q. For a word w and a language L, we define $wL = \{ww' \mid w' \in L\}$ and $w^{-1}L = \{w' \mid ww' \in L\}$.

[2]We note that it is allowed that for $i \neq j$ it holds that $\mathcal{E}(e_i) = \mathcal{E}(e_j)$.

A DFA is an NFA such that I is a singleton and for all $q \in Q$ and $\sigma \in \Sigma$: $|\delta(q,\sigma)| \leq 1$. Let L be a regular language. We denote by $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ the (complete) minimal DFA for L and by N the number $|Q_L|$ of states. For $q_0 \in Q$, we say that a *run from* q_0 of A on $w = a_1 \cdots a_n$ is a sequence $q_0 \to \cdots \to q_n$ of states such that $q_i \in \delta(q_{i-1}, a_i)$, for every $i \in \{1, \ldots, n\}$. When A is a DFA and q_0 its initial state, we also simply call it *the run of* A *on* w. The product of multigraph $G = (V, E, \mathcal{E})$ and NFA $A = (Q, \Sigma, I, F, \delta)$ is a graph $(V', E', \mathcal{E}')$ with $V' = V \times Q$, $E' = \{(e,(q_1,q_2)) \mid (q_1, \mathrm{lab}(e), q_2) \in \delta\}$ and $\mathcal{E}'((e,(q_1,q_2))) = ((\mathrm{origin}(e), q_1), \mathrm{lab}(e), (\mathrm{destination}(e), q_2))$.

A language L is *aperiodic* if and only if $\delta_L(q, w^{N+1}) = \delta_L(q, w^N)$ for every state q and word w. Equivalently, L is aperiodic if and only if its minimal DFA does not have simple cycles labeled w^k for $k > 1$ and $w \neq \varepsilon$. Thus, for "large enough n" we have: $uw^nv \in L$ iff $uw^{n+1}v \in L$. So, a language like $(aa)^*$ is not aperiodic (take $w = a$ and $k = 2$), but $(ab)^*$ is. (There are many characterizations of aperiodic languages [Sch65].)

We study the *regular trail query (RTQ) problem* for a regular language L.

RTQ(L)	
Given:	A (multi-)graph $G = (V, E, \mathcal{E})$ and $(s,t) \in V \times V$.
Question:	Is there a trail from s to t that matches L?

A similar problem, which was studied by Bagan et al. [BBG20], is the *RSPQ problem.* The RSPQ(L) problem asks if there exists a *simple path* from s to t that matches L.

3. The Tractable Class

In this section, we define and characterize a class of languages of which we will prove that it is exactly the class of regular languages L for which RTQ(L) is tractable (if NL $\neq$ NP).

3.1. Warm-Up: Downward Closed Languages.

It is instructive to first discuss the case of downward closed languages. A language L is *downward closed* (DC) if it is closed under taking subsequences. That is, for every word $w = a_1 \cdots a_n \in L$ and every sequence $0 < i_1 < \cdots < i_k < n + 1$ of integers, we have that $a_{i_1} \cdots a_{i_k} \in L$. Perhaps surprisingly, *downward closed languages are always regular* [Hai69]. Furthermore, they can be defined by a clean class of regular expressions (which was shown by Jullien [Jul69] and later rediscovered by Abdulla et al. [ACBJ04]), which is defined as follows.

Definition 3.1. An *atomic expression* over Σ is an expression of the form $(a + \varepsilon)$ or of the form $(a_1 + \cdots + a_n)^*$, where $a, a_1, \ldots, a_n \in \Sigma$. A *product* is a (possibly empty) concatenation $e_1 \cdots e_n$ of atomic expressions $e_1, \ldots, e_n$. A *simple regular expression* is of the form $p_1 + \cdots + p_n$, where $p_1, \ldots, p_n$ are products.

Another characterization is by Mendelzon and Wood [MW95], who show that a regular language L is downward closed if and only if its minimal DFA $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ exhibits the *suffix language containment property*, which says that if $\delta_L(q_1, a) = q_2$ for some symbol $a \in \Sigma$, then we have $L_{q_2} \subseteq L_{q_1}$.[3] Since this property is transitive, it is equivalent to require that $L_{q_2} \subseteq L_{q_1}$ for every state q_2 that is reachable from q_1.

Theorem 3.2 [ACBJ04, Hai69, Jul69, MW95]**.** *The following are equivalent:*

[3] They restrict q_1, q_2 to be on paths from i_L to some state in F_L, but the property trivially holds for q_2 being a sink-state.

(1) *L is a downward closed language.*
(2) *L is definable by a simple regular expression.*
(3) *The minimal DFA of L exhibits the suffix language containment property.*

Obviously, $\mathsf{RTQ}(L)$ is tractable for every downward closed language L, since it is equivalent to deciding if there exists a path from s to t that matches L. For the same reason, deciding if there is a *simple path* from s to t that matches L is also tractable for downward closed languages. However, there are languages that are not downward closed for which we show $\mathsf{RTQ}(L)$ to be tractable, such as a^*bc^* and $(ab)^*$. For these two languages, the simple path variant of the problem is intractable.

3.2. **Main Definitions and Equivalence.**

The following definitions are the basis of the class of languages for which $\mathsf{RTQ}(L)$ is tractable.

Definition 3.3. An NFA A satisfies the *left-synchronized containment property* if there exists an $n \in \mathbb{N}$ such that the following implication holds for all $q_1, q_2 \in Q$ and $a \in \Sigma$:

$$\text{If } q_1 \rightsquigarrow q_2 \text{ and if } w_1 \in \text{Loop}(q_1), w_2 \in \text{Loop}(q_2) \text{ with } w_1 = aw_1' \text{ and } w_2 = aw_2', \quad \text{then } w_2^n L_{q_2} \subseteq L_{q_1}.$$

Similarly, A satisfies the *right-synchronized containment property* if the same condition holds with $w_1 = w_1'a$ and $w_2 = w_2'a$.

We illustrate this definition in Figure 2. We note that the minimal DFA of any downward closed language satisfies the left-synchronized containment property.

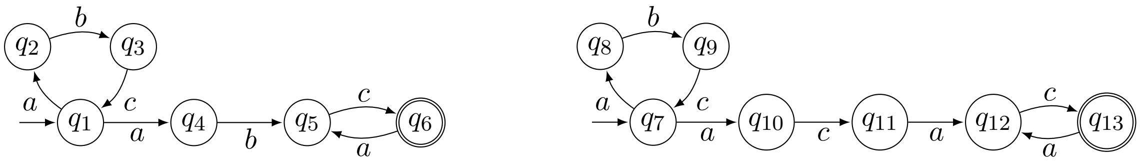

Figure 2: Example illustrating Definition 3.3. The left NFA does not satisfy the left-synchronized containment property as $(ac)^* L_{q_6} \cap L_{q_1} = \emptyset$. The right NFA satisfies the left-synchronized containment property with $n = 2$ as $(ac)^2 L_{q_{13}} \subseteq L_{q_7}$ and $(ca)^2 L_{q_{12}} \subseteq L_{q_9}$.

The *left-synchronizing length* of an NFA A is the smallest value n such that the implication in Definition 3.3 for the left-synchronized containment property holds. We define the *right-synchronizing length* analogously.

Observation 3.4. Let n_0 be the left-synchronizing length of an NFA A. Then the implication of Definition 3.3 is satisfied for every $n \geq n_0$. The reason is that $w_2 \in \text{Loop}(q_2)$.

Definition 3.5. A regular language L is *closed under left-synchronized power abbreviations* (resp., *closed under right-synchronized power abbreviations*) if there exists an $n \in \mathbb{N}$ such that for all words $w_\ell, w_m, w_r \in \Sigma^*$ and all words $w_1 = aw_1'$ and $w_2 = aw_2'$ (resp., $w_1 = w_1'a$ and $w_2 = w_2'a$) we have that $w_\ell w_1^n w_m w_2^n w_r \in L$ implies $w_\ell w_1^n w_2^n w_r \in L$.

We note that Definition 3.5 is equivalent to requiring that there exists an $n \in \mathbb{N}$ such that the implication holds for all $i \geq n$. The reason is that, given $i > n$ and a word of the form $w_\ell w_1^i w_m w_2^i w_r$, we can write it as $w'_\ell w_1^n w_m w_2^n w'_r$ with $w'_\ell = w_\ell w_1^{i-n}$ and $w'_r = w_2^{i-n} w_r$, for which the implication holds by Definition 3.5.

Lemma 3.6. *Consider a minimal DFA $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ with N states. Then the following is true:*

(1) *If A_L satisfies the left-synchronized containment property, then the left-synchronizing length is at most N.*
(2) *If A_L satisfies the right-synchronized containment property, then the right-synchronizing length is at most N.*

Proof. We only prove (1), (2) is symmetric. By Definition 3.3, there exists an $n \in \mathbb{N}$ such that: If $q_1, q_2 \in Q_A$ and $a \in \Sigma$ such that $q_1 \rightsquigarrow q_2$ and if $w_1 \in \text{Loop}(q_1), w_2 \in \text{Loop}(q_2)$ with $w_1 = aw'_1$ and $w_2 = aw'_2$, then $w_2^n L_{q_2} \subseteq L_{q_1}$.

If $n > N$, then there must be a loop in the w_2^n part that generates multiples of w_2. Applying the pigeonhole principle there is an $i < n$ for which $w_2^i L_{q_2} \subseteq L_{q_1}$ holds. By repetition, we obtain an i with $i < N$. □

From Definition 3.3, Observation 3.4, and Lemma 3.6, we get the following corollary.

Corollary 3.7. *Let A be a minimal DFA with N states, $q_1, q_2 \in Q_A$ with $q_1 \rightsquigarrow q_2$, $w_1 \in \text{Loop}(q_1)$, and $w_2 \in \text{Loop}(q_2)$. If A satisfies the*

- *left-synchronized containment property, $w_1 = aw'_1$, and $w_2 = aw'_2$, then $w_2^N L_{q_2} \subseteq L_{q_1}$.*
- *right-synchronized containment property, $w_1 = w'_1 a$, and $w_2 = w'_2 a$, then $w_2^N L_{q_2} \subseteq L_{q_1}$.*

We need two lemmas to prove Theorem 3.11. And their proofs require the following lemma:

Lemma 3.8 (Implicit in [BBG20], Lemma 3 proof)**.** *Every minimal DFA satisfying*

$$\text{for all } q_1, q_2 \in Q_L \text{ such that } q_1 \rightsquigarrow q_2 \text{ and } \text{Loop}(q_1) \cap \text{Loop}(q_2) \neq \emptyset : L_{q_2} \subseteq L_{q_1} \qquad \text{(P)}$$

accepts an aperiodic language.

Lemma 3.9. *If A_L has the left-synchronized containment property or right-synchronized containment property, then L is aperiodic.*

Proof. Let A_L satisfy the left- or right-synchronized containment property. We show that L satisfies Property (P), restated here for convenience.

$$L_{q_2} \subseteq L_{q_1} \text{ for all } q_1, q_2 \in Q_L \text{ such that } q_1 \rightsquigarrow q_2 \text{ and } \text{Loop}(q_1) \cap \text{Loop}(q_2) \neq \emptyset \qquad \text{(P)}$$

This proves the lemma since all languages satisfying Property (P) are aperiodic, see Lemma 3.8. Let $q_1, q_2 \in Q_L$ and w satisfy $q_1 \rightsquigarrow q_2$ and $w \in \text{Loop}(q_1) \cap \text{Loop}(q_2)$. By Corollary 3.7 we then have that $w^N L_{q_2} \subseteq L_{q_1}$. Since $w \in \text{Loop}(q_1)$, we have that $\delta(q_1, w^N) = q_1$, which in turn implies that $L_{q_2} \subseteq L_{q_1}$. □

Lemma 3.10. *If L is closed under left- or right-synchronized power abbreviations, then L is aperiodic.*

Proof. Let L be closed under left- or right-synchronized power abbreviations and $i \in \mathbb{N}$ be as in Definition 3.5. We show that A_L satisfies the Property (P). The aperiodicity then follows from Lemma 3.8.

Let $q_1, q_2 \in Q_L$ and w satisfy $q_1 \rightsquigarrow q_2$ and $w \in \text{Loop}(q_1) \cap \text{Loop}(q_2)$. Let $w_\ell, w_m \in \Sigma^*$ be such that $q_1 = \delta_L(i_L, w_\ell)$ and $q_2 = \delta_L(q_1, w_m)$. Let $w_r \in L_{q_2}$. Then, $w_\ell w^* w_m w^* w_r \subseteq L$

by construction. Especially, $w_\ell w^i w_m w^i w_r \in L$ and, by Definition 3.5, also $w_\ell w^i w^i w_r \in L$. Since $\delta_L(i_L, w_\ell w^i w^i) = q_1$, this means that $w_r \in L_{q_1}$. Therefore, $L_{q_2} \subseteq L_{q_1}$. □

Next, we show that all conditions defined in Definitions 3.3 and 3.5 are equivalent for DFAs.

Theorem 3.11. *For a regular language L with minimal DFA A_L, the following are equivalent:*
(1) *A_L satisfies the left-synchronized containment property.*
(2) *A_L satisfies the right-synchronized containment property.*
(3) *L is closed under left-synchronized power abbreviations.*
(4) *L is closed under right-synchronized power abbreviations.*

Proof. Let $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$. (1) $\Rightarrow$ (3): Let A_L satisfy the left-synchronized containment property. We will show that if there exists a word $w_\ell w_1^i w_m w_2^i w_r \in L$ with $i = N + N^2$ and w_1 and w_2 starting with the same letter, then $w_\ell w_1^i w_2^i w_r \in L$. To this end, let $w_\ell w_1^i w_m w_2^i w_r \in L$. Due to the pumping lemma, there are states q_1, q_2 and integers $h, j, k, \ell, m, n \leq N$ with $j, m \geq 1$ satisfying: $q_1 = \delta(i_L, w_\ell w_1^h)$, $q_1 = \delta(q_1, w_1^j)$, $q_2 = \delta(q_1, w_1^k w_m w_2^\ell)$, $q_2 = \delta(q_2, w_2^m)$, and $w_2^n w_r \in L_{q_2}$. This implies that

$$w_\ell w_1^h (w_1^j)^* w_1^k w_m w_2^\ell (w_2^m)^* w_2^n w_r \subseteq L \;.$$

Since A_L satisfies the left-synchronized containment property and by Corollary 3.7, we have $(w_2^m)^N L_{q_2} \subseteq L_{q_1}$ and therefore

$$w_\ell w_1^h (w_1^j)^* (w_2^m)^N w_2^n w_r \subseteq L \;.$$

Now we use that L is aperiodic, see Lemma 3.9:

$$w_\ell w_1^h (w_1^j)^N (w_1)^* (w_2^m)^N (w_2)^* w_2^n w_r \subseteq L$$

And finally, we use that $i = N + N^2$ and $h, j, m, n \leq N$ to obtain $w_\ell (w_1)^i (w_2)^i w_r \in L$.

(3) $\Rightarrow$ (4): Let L be closed under left-synchronized power abbreviations and let $j \in \mathbb{N}$ be the maximum of $|A_L|$ and $n+1$, where the n is from Definition 3.5. We will show that if $w_\ell (w_1 a)^j w_m (w_2 a)^j w_r \in L$, then $w_\ell (w_1 a)^j (w_2 a)^j w_r \in L$. If $w_\ell (w_1 a)^j w_m (w_2 a)^j w_r \in L$, then we also have $w_\ell (w_1 a)^j w_m (w_2 a)^{j+1} w_r \in L$ since L is aperiodic, see Lemma 3.10, and $j \geq |A_L|$. This can be rewritten as

$$w_\ell w_1 (a w_1)^{j-1} a w_m w_2 (a w_2)^{j-1} (a w_2 a w_r) \in L \;.$$

As L is closed under left-synchronized power abbreviations, and $n < j$, this implies

$$w_\ell w_1 (a w_1)^{j-1} (a w_2)^{j-1} (a w_2 a w_r) \in L \;.$$

This can be rewritten into $w_\ell (w_1 a)^j (w_2 a)^j w_r \in L$.

(4) $\Rightarrow$ (2): Let L be closed under right-synchronized power abbreviations. We will prove that A_L satisfies the right-synchronized containment property, that is, if there are two states q_1, q_2 in A_L with $q_1 \rightsquigarrow q_2$ and $w_1 \in \mathrm{Loop}(q_1)$, $w_2 \in \mathrm{Loop}(q_2)$, such that w_1 and w_2 end with the same letter, then $(w_2 a)^N L_{q_2} \subseteq L_{q_1}$. Let q_1, q_2 be such states. Then there exist w_ℓ, w_m with $q_1 = \delta_L(i_L, w_\ell)$ and $q_2 = \delta_L(q_1, w_m)$. If $L_{q_2} = \emptyset$, we are done. So let us assume there is a word $w_r \in L_{q_2}$. We define $w_r' = w_2^N w_r$. Due to construction, we have $w_\ell w_1^* w_m w_2^* w_r' \subseteq L$. Since L is closed under right-synchronized power abbreviations, there is an $i \in \mathbb{N}$ such that $w_\ell w_1^i w_2^i w_r' \in L$. Since we have a deterministic automaton and $q_1 = \delta_L(i_L, w_\ell w_1^i)$ this implies that $w_2^i w_r' = w_2^i w_2^N w_r \in L_{q_1}$. We now use that L is aperiodic due to Lemma 3.10 to infer that $w_2^N w_r \in L_{q_1}$.

(2) $\Rightarrow$ (1): Let A_L satisfy the right-synchronized containment property. We will show that if there exist states $q_1, q_2 \in Q_L$ and words $w_1, w_2 \in \Sigma^*$ with $aw_1 \in \text{Loop}(q_1)$ and $aw_2 \in \text{Loop}(q_2)$ and $q_1 \rightsquigarrow q_2$, then $(aw_2)^N L_{q_2} \subseteq L_{q_1}$. Let q_1, q_2 be such states and w_1, w_2 as above. We define $q_1' = \delta_L(q_1, w_1)$ and $q_2' = \delta_L(q_2, w_2)$. Since A_L is deterministic, the construction implies that $w_1 a \in \text{Loop}(q_1')$ and $w_2 a \in \text{Loop}(q_2')$. Furthermore, it holds that (i) $L_{q_1'} = a^{-1} L_{q_1}$ and (ii) $w_2 L_{q_2} \subseteq L_{q_2'}$. With this we will show that $(w_2 a)^N L_{q_2'} \subseteq L_{q_1'}$ implies $(aw_2)^N L_{q_2} \subseteq L_{q_1}$. Let $(w_2 a)^N L_{q_2'} \subseteq L_{q_1'}$. Adding an a left hand, yields $(aw_2)^N a L_{q_2'} \subseteq a L_{q_1'} \subseteq L_{q_1}$ because of (i). We use (ii) to replace $L_{q_2'}$ to get: $(aw_2)^{N+1} L_{q_2} \subseteq L_{q_1}$. Since L is aperiodic, see Lemma 3.9, this is equivalent to $(aw_2)^N L_{q_2} \subseteq L_{q_1}$. □

Corollary 3.12. *If a regular language L satisfies Definition 3.5 and $N = |A_L|$ then, for all $i > N^2 + N$ and for all words $w_\ell, w_m, w_r \in \Sigma^*$ and all words $w_1 = aw_1'$ and $w_2 = aw_2'$ (resp., $w_1 = w_1' a$ and $w_2 = w_2' a$) we have that $w_\ell w_1^i w_m w_2^i w_r \in L$ implies $w_\ell w_1^i w_2^i w_r \in L$.*

Proof. This immediately follows from the proof of (1) $\Rightarrow$ (3). □

In Theorem 4.1 we will show that, if NL $\neq$ NP, the languages L that satisfy the above properties are precisely those for which $\mathsf{RTQ}(L)$ is tractable. To simplify terminology, we will henceforth refer to this class as $\mathsf{T}_{\mathsf{tract}}$.

Definition 3.13. A regular language L belongs to $\mathsf{T}_{\mathsf{tract}}$ if L satisfies one of the equivalent conditions in Theorem 3.11.

For example, $(ab)^*$ and $(abc)^*$ are in $\mathsf{T}_{\mathsf{tract}}$, whereas a^*ba^*, $(aa)^*$ and $(aba)^*$ are not. The following property immediately follows from the definition of $\mathsf{T}_{\mathsf{tract}}$.

Observation 3.14. Every regular expression for which each alphabet symbol under a Kleene star occurs at most once in the expression defines a language in $\mathsf{T}_{\mathsf{tract}}$.

A special case of these expressions are those in which every alphabet symbol occurs at most once. These are known as *single-occurrence regular expressions (SORE)* [BNSV10]. SOREs were studied in the context of learning schema languages for XML [BNSV10], since they occur very often in practical schema languages.

3.3. **The inner Structure of minimal DFAs in $\mathsf{T}_{\mathsf{tract}}$.**

The components of minimal DFAs of languages in $\mathsf{T}_{\mathsf{tract}}$ have a very special form. The insights provided in this section are used in Section 4 to show trichotomy results for $\mathsf{T}_{\mathsf{tract}}$, and in Section 3.4 to give a syntactic characterization of languages in $\mathsf{T}_{\mathsf{tract}}$.

Lemma 3.15. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, $a \in \Sigma$, C be a component of A_L, and $q_1, q_2 \in C$. If there exist $w_1 a \in \text{Loop}(q_1)$ and $w_2 a \in \text{Loop}(q_2)$, then, for all $\sigma \in \Sigma$, we have that $\delta_L(q_1, \sigma) \in C$ if and only if $\delta_L(q_2, \sigma) \in C$.*

Proof. Let $q_1 \neq q_2$ be two states in C. Let σ satisfy $\delta_L(q_1, \sigma) \in C$ and let $w \in \text{Loop}(q_1) \cap \sigma \Sigma^* a$. Such a w exists since $\delta_L(q_1, \sigma) \in C$ and $\delta_L(q_1, w_1 a) = q_1$. Let $q_3 = \delta_L(q_2, w^N)$. We will prove that $q_1 = q_3$, which implies that $\delta_L(q_2, \sigma) \in C$. As L is aperiodic, $w \in \text{Loop}(q_3)$. Consequently, there is an $n \in \mathbb{N}$ such that $w^n L_{q_3} \subseteq L_{q_1}$ by Definition 3.3. Since $w \in \text{Loop}(q_1)$, this also implies $L_{q_3} \subseteq L_{q_1}$. Furthermore, q_2 has a loop ending with a and A_L satisfies the right-synchronized containment property, so $w^N L_{q_1} \subseteq L_{q_2}$ by Corollary 3.7. Hence, $L_{q_1} \subseteq (w^N)^{-1} L_{q_2}$ and, by definition of q_3, we have $(w^N)^{-1} L_{q_2} = L_{q_3}$. So we showed $L_{q_3} \subseteq L_{q_1}$ and $L_{q_1} \subseteq L_{q_3}$ which, by minimality of A_L, implies $q_1 = q_3$. □

The following is a direct consequence thereof.

Corollary 3.16. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, $a \in \Sigma$, C be a component of A_L, and $q_1, q_2 \in C$. If there exist $w_1 a \in \mathrm{Loop}(q_1)$ and $w_2 a \in \mathrm{Loop}(q_2)$, then $\delta_L(q_1, w) \in C$ if and only if $\delta_L(q_2, w) \in C$ for all words $w \in \Sigma^*$.*

Lemma 3.17. *Let A_L satisfy the left-synchronized containment property. If states q_1 and q_2 belong to the same component of A_L and $\mathrm{Loop}(q_1) \cap \mathrm{Loop}(q_2) \neq \emptyset$, then $q_1 = q_2$.*

Proof. Let q_1, q_2 be as stated and let w be a word in $\mathrm{Loop}(q_1) \cap \mathrm{Loop}(q_2)$. According to Definition 3.3, there exists an $n \in \mathbb{N}$ such that $w^n L_{q_2} \subseteq L_{q_1}$. Since $w \in \mathrm{Loop}(q_1)$, this implies that $L_{q_2} \subseteq L_{q_1}$. By symmetry, we have $L_{q_2} = L_{q_1}$, which implies $q_1 = q_2$, since A_L is the minimal DFA. □

To this end, we obtain the following synchronization property for A_L.

Lemma 3.18. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, let C be a component of A_L, let $q_1, q_2 \in C$, and let w be a word of length N^2. If $\delta_L(q_1, w) \in C$ and $\delta_L(q_2, w) \in C$, then $\delta_L(q_1, w) = \delta_L(q_2, w)$.*

Proof. Assume that $w = a_1 \cdots a_{N^2}$. For each i from 0 to N^2 and $\alpha \in \{1, 2\}$, let $q_{\alpha,i} = \delta_L(q_\alpha, a_1 \cdots a_i)$. Since there are at most N^2 distinct pairs $(q_{1,i}, q_{2,i})$, there exist i, j with $0 \leq i < j \leq N^2$ such that $q_{1,i} = q_{1,j}$ and $q_{2,i} = q_{2,j}$. Since $\delta_L(q_1, w) \in C$ and $\delta_L(q_2, w) \in C$, $q_{1,i}, q_{2,i} \in C$. Let $w' = a_{i+1} \cdots a_j$. We have $w' \in \mathrm{Loop}(q_{1,i}) \cap \mathrm{Loop}(q_{2,i})$, hence $q_{1,i} = q_{2,i}$ by Lemma 3.17. As a consequence, $\delta_L(q_1, w) = \delta_L(q_2, w)$. □

Furthermore, we show that every language in $\mathsf{T}_{\mathsf{tract}}$ satisfies an inclusion property which is stronger than indicated by Definition 3.3. That is, we show that it is not necessary to repeat some word w_2 multiple times. Instead, we show that any word w that stays in a component, given that w is long enough and starts with a suitable symbol, already implies an inclusion property.

Lemma 3.19. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, $a \in \Sigma$ and let q_1, q_2 be two states such that $q_1 \rightsquigarrow q_2$ and $\mathrm{Loop}(q_1) \cap a\Sigma^* \neq \emptyset$. Let C be the component of A_L that contains q_2. Then,*

$$L_{q_2} \cap L^a_{q_2} \Sigma^* \subseteq L_{q_1}$$

where $L^a_{q_2}$ is the set of words w of length N^2 that start with a and such that $\delta_L(q_2, w) \in C$.

Proof. If $\mathrm{Loop}(q_2) = \emptyset$, then $L_{q_2} \cap L^a_{q_2} \Sigma^* = \emptyset$ and the inclusion trivially holds. Therefore we assume from now on that $\mathrm{Loop}(q_2) \neq \emptyset$. Since the proof of this lemma requires a number of different states and words, we provide a sketch in Figure 3. Let $w \in L_{q_2} \cap L^a_{q_2} \Sigma^*$, u be the prefix of w of length N^2 and w' be the suffix of w such that $w = uw'$. Since q_2 and $\delta_L(q_2, u)$ are both in the same component C, there exists a word v with $uv \in \mathrm{Loop}(q_2)$. Corollary 3.7 implies that

$$(uv)^N L_{q_2} \subseteq L_{q_1} \; . \tag{3.1}$$

Let $q_3 = \delta_L(q_1, (uv)^N)$. Due to aperiodicity we have $uv \in \mathrm{Loop}(q_3)$. Since A_L is deterministic, this implies $L_{q_3} = ((uv)^N)^{-1} L_{q_1}$ and, together with Equation (3.1) that

$$L_{q_2} \subseteq L_{q_3} \; . \tag{3.2}$$

We now show that there is a prefix u_1 of u such that $\delta_L(q_1, u_1) = q$ and $\delta_L(q_3, u_1) = q'$ with $\mathrm{Loop}(q) \cap \mathrm{Loop}(q') \neq \emptyset$. Assume that $u = a_1 \cdots a_{N^2}$. Let $q_{\alpha,0} = q_\alpha$ and, for each i from 1 to N^2 and $\alpha \in \{1, 3\}$, let $q_{\alpha,i} = \delta_L(q_\alpha, a_1 \cdots a_i)$. Since there are at most N^2 distinct pairs $(q_{1,i}, q_{3,i})$, there exist i, j with $0 \leq i < j \leq N^2$ such that $q_{1,i} = q_{1,j}$ and $q_{3,i} = q_{3,j}$.

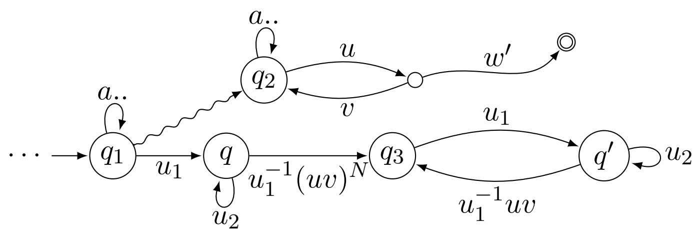

Figure 3: Sketch of the proof of Lemma 3.19

Let $u_1 = a_1 \cdots a_i$ and $u_2 = a_{i+1} \cdots a_j$. We have $u_2 \in \text{Loop}(q_{1,i}) \cap \text{Loop}(q_{3,i})$. We define $q = \delta_L(q_1, u_1)$ and $q' = \delta_L(q_3, u_1)$. Since $q \rightsquigarrow q'$ and $u_2 \in \text{Loop}(q) \cap \text{Loop}(q')$, Corollary 3.7 implies $u_2^N L_{q'} \subseteq L_q$. Since $u_2 \in \text{Loop}(q)$, we also have that

$$L_{q'} \subseteq L_q \ . \tag{3.3}$$

By definition of q and the determinism of A_L, we have that $L_q = u_1^{-1} L_{q_1}$. Thus, Equation (3.3) implies $L_{q'} \subseteq u_1^{-1} L_{q_1}$. The definition of q' implies that $L_{q'} = u_1^{-1} L_{q_3}$, so $u_1^{-1} L_{q_3} \subseteq u_1^{-1} L_{q_1}$. In other words, we have $L_{q_3} \cap u_1 \Sigma^* \subseteq L_{q_1} \cap u_1 \Sigma^*$. Since u_1 is a prefix of u, and by Equation (3.2), we also have $L_{q_2} \cap u\Sigma^* \subseteq L_{q_1}$. This implies that $w \in L_{q_1}$, which concludes the proof. □

3.4. A Syntactic Characterization.

The goal of this section is to give a better understanding of languages in $\mathsf{T}_{\mathsf{tract}}$. We provide a syntactic definition, which will allow to construct languages in $\mathsf{T}_{\mathsf{tract}}$. More precisely, we will show that every language of a “memoryless component” is in $\mathsf{T}_{\mathsf{tract}}$. And if memoryless components are connected with “consistent jumps”, then the language is again in $\mathsf{T}_{\mathsf{tract}}$. We show that all languages in $\mathsf{T}_{\mathsf{tract}}$ can be constructed in this way. Using this modular principle, systems with graphical search queries could enable users to “click” a language in $\mathsf{T}_{\mathsf{tract}}$ together. Note that this section is quite technical and detached from the rest of the article, thus it can be skipped.

As we have seen before, regular expressions in which every symbol occurs at most once define languages in $\mathsf{T}_{\mathsf{tract}}$. We will define a similar notion on automata.

Definition 3.20. A component C of some NFA A is called *memoryless*, if for each symbol $a \in \Sigma$, there is at most one state q in C, such that there is a transition (p, a, q) with p in C.

In this section, we will prove the following theorem which provides (in a non-trivial proof that requires several steps) a syntactic condition for languages in $\mathsf{T}_{\mathsf{tract}}$. The syntactic condition is item (4) of the theorem, which we define after its statement. Condition (5) imposes an additional restriction on condition (4).

Theorem 3.21. *For a regular language L, the following properties are equivalent:*
(1) $L \in \mathsf{T}_{\mathsf{tract}}$
(2) *There exists an NFA A for L that satisfies the left-synchronized containment property.*
(3) *There exists an NFA A for L that satisfies the left-synchronized containment property and only has memoryless components.*
(4) *There exists a detainment automaton for L with consistent jumps.*
(5) *There exists a detainment automaton for L with consistent jumps and only memoryless components.*

To define detainment automata, we use *finite automata with counters or CNFAs* from Gelade et al. [GGM12], which we slightly adapt to make the construction easier.[4]

We recall the definition of counter NFAs from Gelade et al. [GGM12]. We introduce a minor difference, namely that counters count down instead of up, since this makes our construction easier to describe. Furthermore, since our construction only requires a single counter, zero tests, and setting the counter to a certain value, we immediately simplify the definition to take this into account.

Let c be a *counter variable*, taking values in $\mathbb{N}$. A *guard* on c is a statement γ of the form true or $c = 0$. We denote by $c \models \gamma$ that c satisfies the guard γ. In the case where γ is true, this is trivially fulfilled and, in the case where γ is $c = 0$, this is fulfilled if c equals 0. By G we denote the set of guards on c. An *update* on c is a statement of the form $c := c - 1$, $c := c$, or $c := k$ for some constant $k \in \mathbb{N}$. By U we denote the set of updates on c.

Definition 3.22. A *nondeterministic counter automaton* (CNFA) with a single counter is a 6-tuple $A = (Q, I, c, \delta, F, \tau)$ where Q is the finite set of states; $I \subseteq Q$ is a set of initial states; c is a counter variable; $\delta \subseteq Q \times \Sigma \times G \times Q \times U$ is the transition relation; and $F \subseteq Q$ is the set of accepting states. Furthermore, $\tau \in \mathbb{N}$ is a constant such that every update of the form $c := k$ has $k \leq \tau$.

Intuitively, A can make a transition $(q, a, \gamma; q', \pi)$ whenever it is in state q, reads a, and $c \models \gamma$, i.e., guard γ is true under the current value of c. It then updates c according to the update π, in a way we explain next, and moves into state q'. To explain the update mechanism formally, we introduce the notion of configuration. A *configuration* is a pair (q, ℓ) where $q \in Q$ is the current state and $\ell \in \mathbb{N}$ is the value of c. Finally, an update π defines a function $\pi : \mathbb{N} \to \mathbb{N}$ as follows. If $\pi = (c := k)$ then $\pi(\ell) = k$ for every $\ell \in \mathbb{N}$. If $\pi = (c := c - 1)$ then $\pi(\ell) = \max(\ell - 1, 0)$. Otherwise, i.e., if $\pi = (c := c)$, then $\pi(\ell) = \ell$. So, counters never become negative.

An *initial configuration* is $(q_0, 0)$ with $q_0 \in I$. A configuration (q, ℓ) is *accepting* if $q \in F$ and $\ell = 0$. A configuration $\alpha' = (q', \ell')$ *immediately follows* a configuration $\alpha = (q, \ell)$ by reading $a \in \Sigma$, denoted $\alpha \to_a \alpha'$, if there exists $(q, a, \gamma; q', \pi) \in \delta$ with $c \models \gamma$ and $\ell' = \pi(\ell)$.

For a string $w = a_1 \cdots a_n$ and two configurations α and α', we denote by $\alpha \Rightarrow_w \alpha'$ that $\alpha \to_{a_1} \cdots \to_{a_n} \alpha'$. A configuration α is *reachable* if there exists a string w such that $\alpha_0 \Rightarrow_w \alpha$ for some initial configuration α_0. A string w is *accepted* by A if $\alpha_0 \Rightarrow_w \alpha_f$ where α_0 is an initial configuration and α_f is an accepting configuration. We denote by $L(A)$ the set of strings accepted by A.

It is easy to see that CNFA accept precisely the regular languages. (Due to the value τ, counters are always bounded by a constant.)

Let A be a CNFA with one counter c. Initially, the counter has value 0. The automaton has transitions of the form $(q_1, a, P; q_2, U)$ where P is a precondition on c and U an update operation on c. For instance, the transition $(q_1, a, c = 5; q_2, c := c - 1)$ means: if A is in state q_1, reads a, and the value of c is five, then it can move to q_2 and decrease c by one. If we decrease a counter with value zero, its value remains zero. We denote the precondition that is always fulfilled by true.

We say that A is a *detainment automaton* if, for every component C of A:

- every transition inside C is of the form $(q_1, a, \mathsf{true}; q_2, c := c - 1)$;
- every transition that leaves C is of the form $(q_1, a, c = 0; q_2, c := k)$ for some $k \in \mathbb{N}$;[5]

[4]The adaptation is that we let counters decrease instead of increase. Furthermore, it only needs zero-tests.

[5]If q_2 is in a trivial component, then k should be 0 for the transition to be useful.

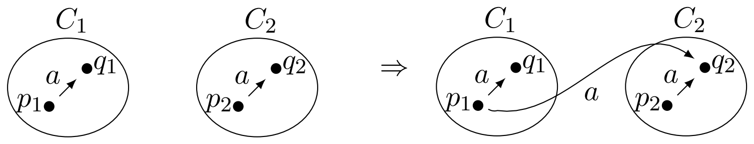

Figure 4: Consistent jump condition (simplified, i.e.: without preconditions, counter and update) used in Theorem 3.31. C_1 and C_2 are components (not necessarily different) such that C_2 is reachable from C_1.

Intuitively, if a detainment automaton enters a non-trivial component C, then it must stay there for at least some number of steps, depending on the value of the counter c. The counter c is decreased for every transition inside C and the automaton can only leave C once $c = 0$. We say that A has *consistent jumps* if, for every pair of components C_1 and C_2, if $C_1 \rightsquigarrow C_2$ and there are transitions $(p_i, a, \mathsf{true}; q_i, c := c - 1)$ inside C_i for all $i \in \{1, 2\}$, then there is also a transition $(p_1, a, P; q_2, U)$ for some $P \in \{\mathsf{true}, c = 0\}$ and some update U.[6] We illustrate this in Figure 4. We note that C_1 and C_2 may be the same component. The consistent jump property is the syntactical counterpart of the left-synchronized containment property. The *memoryless* condition carries over naturally to CNFAs, ignoring the counter.

Proof sketch of Theorem 3.21. The implications (3) $\Rightarrow$ (2) and (5) $\Rightarrow$ (4) are trivial. We sketch the proofs of (1) $\Rightarrow$ (5) $\Rightarrow$ (3) and (4) $\Rightarrow$ (2) $\Rightarrow$ (1) below, establishing the theorem.

(1) $\Rightarrow$ (5) uses a very technical construction that essentially exploits that—if the automaton stays in the same component for a long time—the reached state only depends on the last N^2 symbols read in the component. This is formalized in Lemma 3.18 and allows us to merge any pair of two states p, q which contradict that some component is memoryless. To preserve the language, words that stay in some component C for less than N^2 symbols have to be dealt with separately, essentially avoiding the component altogether. Finally, the left-synchronized containment property allows us to simply add transitions required to satisfy the consistent jumps property without changing the language.

(5) $\Rightarrow$ (3) and (4) $\Rightarrow$ (2): We convert a given CNFA to an NFA by simulating the counter (which is bounded) in the set of states. The consistent jump property implies the left-synchronized containment property on the resulting NFA. The property that all components are memoryless is preserved by the construction.

(2) $\Rightarrow$ (1): One can show that the left-synchronized containment property is invariant under the powerset construction.

The following lemma is the implication (1) $\Rightarrow$ (5) from Theorem 3.21

Lemma 3.23. *If $L \in \mathsf{T}_{\mathsf{tract}}$, then there exists a detainment automaton for L with consistent jumps and only memoryless components.*

Proof. Let $A_L = (Q_L, \Sigma, i_L, F_L, \delta_L)$ be the minimal DFA for L. The proof goes as follows: First, we define a CNFA A with two counters. Second, we show that we can convert A to an equivalent CNFA A' with only one counter that is a detainment automaton with consistent jumps and only memoryless components. This conversion is done by simulating one of the counters using a bigger set of states. Last, we show that $L(A) = L(A_L)$, which shows the lemma statement as $L(A) = L(A')$.

[6]The values of P and U depend on whether C_1 is the same as C_2 or not.

Before we start we need some additional notation. We write $p_1 \curvearrowright^a q_2$ to denote that $C(p_1) \rightsquigarrow C(q_2)$ and there are states $q_1 \in C(p_1)$ and $p_2 \in C(q_2)$ such that $(p_i, a, q_i) \in \delta_L$ for $i \in \{1, 2\}$. Let q be a state, then we write $\Sigma^{\circlearrowright}(q)$ to denote the set of symbols a, such that there is a word $w = aw' \in \text{Loop}(q)$.

Let $\sim \subseteq Q_L \times Q_L$ be the smallest equivalence relation over Q_L that satisfies $p \sim q$ if $C(p) = C(q)$ and $\Sigma^{\circlearrowright}(p) \cap \Sigma^{\circlearrowright}(q) \neq \emptyset$. For $q \in Q_L$, we denote by $[q]$ the equivalence class of q. By $[Q_L]$ we denote the set of all equivalence classes. We also write $[C]$ to denote the equivalence classes that only use states from some component C. We extend the notion $C(q)$ to $[Q_L]$, i.e., $C([q]) = C(q)$ for all $q \in Q_L$.

We will use the following observation that easily follows from Lemma 3.15 using the definition of $\sim$.

Observation 3.24. Let q_1, q_2 be states with $[q_1] = [q_2]$, then for all $a \in \Sigma$ it holds that $\delta_L(q_1, a) \in C(q_1)$ if and only if $\delta_L(q_2, a) \in C(q_1)$.

We define a CNFA $A = (Q, I, c, d, \delta, F, N^2)$ that has two counters c and d. The counter c is allowed to have any initial value from $[0, N^2]$, while the counter d has initial value 0. We note that we will eliminate counter c when converting to a one counter automaton, thus this is not a contradiction to the definition of CNFA with one counter that we use.

We use $Q' = Q_L \cup [Q_L]$, i.e., we can use the states from A_L and the equivalence classes of the equivalence relation $\sim$. The latter will be used to ensure that components are memoryless, while the former will only be used in trivial components. We use $I = \{i_L, [i_L]\}$ and $F = F_L$.

$$
\begin{aligned}
\delta^1_{\circlearrowright} &= \{\ (q_1, a, \{c > 0, d = 0\}; q_2, \{c := c - 1\})\ \mid\ (q_1, a, q_2) \in \delta_L, C(q_1) = C(q_2)\ \}\\
\delta^2_{\circlearrowright} &= \{\ ([q_1], a, \{c = N^2\}; [q_2], \{d := d - 1\})\ \mid\ (q_1, a, q_2) \in \delta_L, C(q_1) = C(q_2)\ \}\\
\delta^3_{\circlearrowright} &= \{\ ([q_1], a, \{c = N^2, d = 0\}; q_2, \{c := c - 1\})\ \mid\ (q_1, a, q_2) \in \delta_L, C(q_1) = C(q_2)\ \}\\
\delta^1_{\rightarrow} &= \{\ (q_1, a, \{c = 0, d = 0\}; q_2, \{c := i\})\ \mid\ (q_1, a, q_2) \in \delta_L, C(q_1) \neq C(q_2), i \in [0, N^2 - 1]\ \}\\
\delta^2_{\rightarrow} &= \{\ (q_1, a, \{c = 0, d = 0\}; [q_2], \{c := N^2\})\ \mid\ (q_1, a, q_2) \in \delta_L, C(q_1) \neq C(q_2)\ \}\\
\delta_{\curvearrowright} &= \{\ ([q_1], a, \{c = N^2, d = 0\}; [q_2], \{d := N^2\})\ \mid\ q_1 \curvearrowright^a q_2, C(q_1) \neq C(q_2)\ \}\\
\delta &= \delta^1_{\circlearrowright}\ \cup\ \delta^2_{\circlearrowright}\ \cup\ \delta^3_{\circlearrowright}\ \cup\ \delta^1_{\rightarrow}\ \cup\ \delta^2_{\rightarrow}\ \cup\ \delta_{\curvearrowright}
\end{aligned}
$$

We say that a component C of A_L is a *long run component* of a given word $w = a_1 \cdots a_n$, if $|\{i \mid \delta(i_L, a_1 \cdots a_i) \in C\}| > N^2$, i.e., if the run stays in C for more than N^2 symbols. All other components are *short run components.*

For short run components, we use states from Q_L. We use the counter c to enforce that these parts are indeed short. For long run components, we first use states in $[Q_L]$. Only the last N^2 symbols in the component are read using states from Q_L. The left-synchronized containment property guarantees that for long run components the precise state is not important, which allows us to make these components memoryless.

The transition relation is divided into transitions between states from the same component of A_L (indicated by $\delta_{\circlearrowright} = \delta^1_{\circlearrowright} \cup \delta^2_{\circlearrowright} \cup \delta^3_{\circlearrowright}$) and transitions between different components (indicated by $\delta_{\rightarrow} = \delta^1_{\rightarrow} \cup \delta^2_{\rightarrow}$). Transitions in $\delta_{\curvearrowright}$ are added to satisfy the consistent jumps property. They are the only transitions that increase the counter d. This is necessary, as the left-synchronized containment property only talks about the language of the state reached after staying in the component for some number of symbols. If we added the transitions in $\delta_{\curvearrowright}$

without using the counter, we would possibly add additional words to the language. This concludes the definition of A.

We now argue that the automaton $A' = (Q' \times [0, N^2], i_L, d, \delta', F \times \{0\}, N^2)$ derived from A by pushing the counter c into the states is a detainment automaton with consistent jumps that only has memoryless components. The states of A' have two components, first the state of A and second the value of the second counter that is bounded by N^2. We do not formally define δ'. It is derived from δ in the obvious way, i.e., by doing the precondition checks that depend on c on the second component of the state. Similarly, updates of c are done on the second component of the states.

It is straightforward to see that A' is a detainment automaton with consistent jumps that only has memoryless components using the following observations:

- Every transition in A that does not have $c = N^2$ before and after the transition requires $d = 0$.
- Let Cuts be the set of components of A, then the set of components of A' is $\{[C] \times \{N^2\} \mid C \in \mathsf{Cuts}\}$.

The consistent jumps are guaranteed by the transitions in $\delta_{\curvearrowright}$. As A' only has memoryless components, the consistent jump property is trivially satisfied for states inside the same component.

We now show that $L(A_L) \subseteq L(A)$. Let $w = a_1 \cdots a_n$ be some string in $L(A_L)$ and $q_0 \to \cdots \to q_n$ be the run of A_L on w. $\mathsf{countdown}\colon \mathbb{N} \to \mathbb{N}$ that gives us how long we stay inside some component as $\mathsf{countdown}\colon i \mapsto j - i$, where j is the largest number such that $C(q_j) = C(q_i)$.

It is easy to see by the definitions of the transitions in $\delta_{\to}$ and $\delta_{\circlearrowright}$, that the run

$$\big(p_0, \min(N^2, \mathsf{countdown}(0)), 0\big) \;\to\; \cdots \;\to\; \big(p_n, \min(N^2, \mathsf{countdown}(n)), 0\big)$$

is an accepting run of A, where p_i is q_i if $c_i < N^2$ and $[q_i]$ otherwise. We note that the counter d is always zero, as we do not use any transitions from $\delta_{\curvearrowright}$. The transitions in $\delta_{\curvearrowright}$ are only there to satisfy the consistent jumps property. This shows $L(A_L) \subseteq L(A)$.

Towards the lemma statement, it remains to show that $L(A) \subseteq L(A_L)$. Let therefore $w = a_1 \cdots a_n$ be some string in $L(A)$, $(p_0, c_0, d_0) \to \cdots \to (p_n, c_n, d_n)$ be an accepting run of A, and $q_0 \to \cdots \to q_n$ be the unique run of A_L on w.

We now show by induction on i that there are states $\hat{q}_1, \ldots, \hat{q}_n$ in Q_L such that the following claim is satisfied. The claim easily yields that $q_n \in F_L$, as both counters have to be zero for the word to be accepted.

$$L_{\hat{q}_i} \cap a_{i+1} \cdots a_{i+d_i} \Sigma^* \subseteq L_{q_i} \quad \text{and} \quad \hat{q}_i \in \begin{cases} \{p_i\} & \text{if } c_i = d_i = 0 \\ [p_i] & \text{if } c_i + d_i > 0 \text{ and } p_i \in Q_L \\ p_i & \text{if } c_i + d_i > 0 \text{ and } p_i \in [Q_L] \end{cases}$$

The base case $i = 0$ is trivial by the definition of I. We now assume that the induction hypothesis holds for i and are going to show that it holds for $i+1$. Let $\rho = (p_{i-1}, a_i, P; p_i, U)$ be the transition used to read a_i. We distinguish several cases depending on ρ.

Case $\rho \in \delta_{\to}$: In this case, $c_i = 0$ by the definition of $\delta_{\to}$. Therefore, the claim for $i+1$ follows with $\hat{q}_{i+1} = p_{i+1}$, as $\hat{q}_i = p_i$ by the induction hypothesis and $(p_i, a, p_{i+1}) \in \delta_L$ by the definition of $\delta_{\to}$.

Case $\rho \in \delta^2_{\circlearrowleft}$: We note that $p_i, p_{i+1} \in [Q_L]$. The claim for $i+1$ follows with $\hat{q}_{i+1} = \delta(\hat{q}_i, a_{i+1})$ using $C(q') = C(\delta(q', a_{i+1})$ for all $q' \in [p_i]$ (by Observation 3.24), $C(p_i) = C(p_{i+1})$ (by definition of $\delta_{\circlearrowleft}$), and $\hat{q}_i \in p_i$ (by the induction hypothesis).

Case $\rho \in \delta^3_{\circlearrowleft}$: We want to show that $L_{p_{i+N^2}} \subseteq L_{q_{N^2}}$ establishing the claim directly for the position $i + N^2$ using $\hat{q}_{i+N^2} = p_{i+N^2}$. Therefore, we first want to apply Lemma 3.18 to show that $\delta(\hat{q}_i, a_{i+1} \cdots a_{i+N^2}) = p_{i+N^2}$. The preconditions of the lemma require us to show that (i) $C(\hat{q}_i) = C(p_i)$, (ii) $C(p_i) = C(p_{i+N^2})$, and (iii) $C(\hat{q}_i) = C(\delta_L(\hat{q}_i, a_{i+1} \cdots a_{i+N^2}))$. Precondition (i) is given by the induction hypothesis, precondition (ii) is by the definition of $\delta_{\circlearrowleft}$, i.e., that all transitions in $\delta_{\circlearrowleft}$ are inside the same component of A_L, and precondition (iii) is by the fact that each transition in $\delta_{\circlearrowleft}$ has a corresponding transition in δ_L that stays in the same component. Therefore, we can actually apply Lemma 3.18 to conclude that $\delta(\hat{q}_i, a_{i+1} \cdots a_{i+N^2}) = p_{i+N^2}$. As we furthermore have that $L_{\hat{q}_i} \cap a_{i+1} \cdots a_{i+d_i}\Sigma^* \subseteq L_{q_i}$ by the induction hypotheses, we can conclude that $L_{p_{i+N^2}} \subseteq L_{q_{N^2}}$. This establishes the claim for position $i + N^2$ using $\hat{q}_{i+N^2} = p_{i+N^2}$. As we only need the claim for position n (and not for all smaller positions), we can continue the induction at position $i + N^2$. Especially there is no need to look at the case where $\rho \in \delta^1_{\circlearrowleft}$.

Case $\rho \in \delta_{\curvearrowright}$: By the definition of $\delta_{\curvearrowright}$, we have that $p_i, p_{i+1} \in [Q_L]$. Furthermore, there are transitions (p_i, a_{i+1}, p') and (p'', a_{i+1}, p_{i+1}) in δ_L such that $C(p') = C(p_i)$, $C(p'') = C(p_{i+1})$, and $p' \rightsquigarrow p''$. This (and the fact that $\hat{q}_i \in p_i$ by the induction hypothesis) allows us to apply Observation 3.24, which yields $\delta(\hat{q}_i, a_{i+1}) \in C(p_i)$. From $p' \rightsquigarrow p''$ and $\hat{q}_i \in C(p')$ we can conclude that $\hat{q}_i \rightsquigarrow p''$. We now can apply Lemma 3.19 that gives us $L_{p''} \cap L^{a_{i+1}}_{p''}\Sigma^* \subseteq L_{\hat{q}_i}$.

Now we argue that the subword $a_{i+2} \cdots a_{i+N^2+1}$ is in $L^{a_{i+1}}_{p''}$. By the definition of $\delta_{\curvearrowright}$, we have $d_{i+1} = N^2$, enforcing that the next N^2 transitions are all from $\delta^2_{\circlearrowleft}$, as these are the only transitions that allow $d > 0$ in the precondition. Applying Observation 3.24 N^2 times yields that $\delta(p'', a_{i+2} \cdots a_{i+N^2+1}) \in C(p'')$ and therefore $a_{i+2} \cdots a_{i+N^2+1} \in L^{a_{i+1}}_{p''}$. Using this and $L_{p''} \cap L^{a_{i+1}}_{p''}\Sigma^* \subseteq L_{\hat{q}_i}$, we get that $L_{\delta(p'', a_{i+1})} \cap a_{i+2} \cdots a_{i+N^2+1}\Sigma^* \subseteq L_{\delta(\hat{q}_i, a_{i+1})}$ yielding the claim for $i + 1$. This concludes the proof of the lemma. □

We now continue with the rest of the proof of Theorem 3.21.

Proof of Theorem 3.21. We show (1) ⇒ (5) ⇒ (3) ⇒ (2) ⇒ (1) and (5) ⇒ (4) ⇒ (2).

(1) ⇒ (5): Holds by Lemma 3.23.

(5) ⇒ (3) and (4) ⇒ (2): Let $A = (Q, I, c, \delta, F, \ell)$ be a detainment automaton with consistent jumps. We compute an equivalent NFA $A' = (Q \times \{0, \ldots, \ell\}, \delta', I \times \{0\}, F \times \{0\})$ in the obvious way, i.e., $((p, i), a, (q, j)) \in \delta'$ if and only if A can go from configuration (p, i) to configuration (q, j) reading symbol a. By the definition of detainment automata, we get that the components of A' are

$$\{ \ C \times \{0\} \ \mid \ C \text{ is a component of } A \ \}$$

This directly shows that A' only has memoryless components if A only has memoryless components.

To prove the left-synchronizing containment property, we choose $n = \ell$. Let now $(q_1, c_1), (q_2, c_2) \in Q \times \{0, \ldots, \ell\}$, $a \in \Sigma$, and $w_1', w_2' \in \Sigma^*$ be such that $(q_1, c_1) \rightsquigarrow (q_2, c_2)$, $w_1 = aw_1' \in \text{Loop}((q_1, c_1))$, and $w_2 = aw_2' \in \text{Loop}((q_2, c_2))$. We have to show that

$$w_2^n L_{(q_2,c_2)} \ \subseteq \ L_{(q_1,c_1)} \ . \tag{3.4}$$

We distinguish two cases. If q_1 and q_2 are in the same component, we know that there is a transition $(q_1, a, \mathsf{true}; q_3; c := c - 1) \in \delta$, as A has consistent jumps. Therefore, there is a transition $((q_1, 0), a, (q_2, 0)) \in \delta'$, which directly yields (3.4).

If q_1 and q_2 are in different components, then there is a transition $(q_1, a, c = 0; q_3; c := k) \in \delta$, as A has consistent jumps. Therefore, there is a transition $((q_1, 0), a, (q_2, k)) \in \delta'$ for some $k \in [0, \ell]$. We have $w_2 \in \mathrm{Loop}(q_2)$. The definition of detainment automata requires that every transition inside a component—thus every transition used to read w_2 using the loop—is of the form $(p, a, \mathsf{true}; q, c := c - 1)$, i.e., it does not have a precondition and it decreases the counter by one. Therefore in A', we have that $\delta'((q_2, k), w^\ell) \supseteq \delta((q_2, 0), w^\ell)$. This concludes the proof of (5) $\Rightarrow$ (3) and (4) $\Rightarrow$ (2)

(5) $\Rightarrow$ (4) and (3) $\Rightarrow$ (2): Trivial.

(2) $\Rightarrow$ (1): Let $A = (Q, \Sigma, \delta, I, F)$ be an NFA satisfying the left-synchronized containment property and A_L be the minimal DFA equivalent to A. We show that A_L satisfies the left-synchronized containment property establishing (1).

Let M be the left synchronizing-length of A and $q_1, q_2 \in Q_L$ be states of A_L such that

- $q_1 \rightsquigarrow q_2$; and
- there are words $w_1 \in \mathrm{Loop}(q_1)$ and $w_2 \in \mathrm{Loop}(q_2)$ that start with the same symbol a.

We need to show that there exists an $n \in \mathbb{N}$ with $w_2^n L_{q_2} \subseteq L_{q_1}$. Let w be a word such that $\delta(q_1, w) = q_2$. Let $P_1 \subseteq Q$ be a state of the powerset automaton of A with $L_{P_1} = L_{q_1}$ and let $P_2 = \delta(P_1, w w_2^*)$ be the state in the powerset automaton of A that consists of all states reachable from P_1 reading some word from $w w_2^*$.

It holds that $L_{P_2} = L_{q_2}$, as $\delta(q_1, w w_2^*) = q_2$ and $L_{q_1} = L_{P_1}$.

We define

$$\begin{aligned} P_2' &= \{\, p \in P_2 \mid w_2^i \in \mathrm{Loop}(p) \text{ for some } i > 0 \,\} \\ P_2'' &= \delta(P_1, w_2^{|A|}) \end{aligned}$$

We obviously have $P_2'' \subseteq P_2' \subseteq P_2$. Furthermore, we have

$$L_{P_2} = L_{q_2} = L_{\delta(q_2, w_2^{|A|})} = L_{P_2''}$$

The second equation is by $\delta(q_2, w_2^{|A|}) = q_2$. We can conclude that $L_{q_2} = L_{P_2'}$.

Let $\rho \colon Q \to Q$ be a function that selects for every state $p_2 \in P_2'$ a state $p_1 \in P_1$ such that $p_1 \rightsquigarrow p_2$. By definition of P_2', such states exist. Using the fact that A satisfies the left-synchronized containment property, we get that $w_2^M L_{p_2} \subseteq L_{\rho(p_2)}$ for each $p_2 \in P_2$. We can conclude

$$w_2^M L_{q_2} = w_2^M L_{P_2'} = \bigcup_{p_2 \in P_2'} w_2^M L_{p_2} \subseteq \bigcup_{p_2 \in P_2'} L_{\rho(p_2)} \subseteq L_{P_1} = L_{q_1}$$

and therefore $w_2^{|A|+M} L_{q_2} \subseteq L_{q_1}$. So A_L satisfies the left-synchronized containment property with $n = M$, where M is the left synchronizing-length of A. This concludes the proof for (2) $\Rightarrow$ (1) and thus the proof of the theorem. $\square$

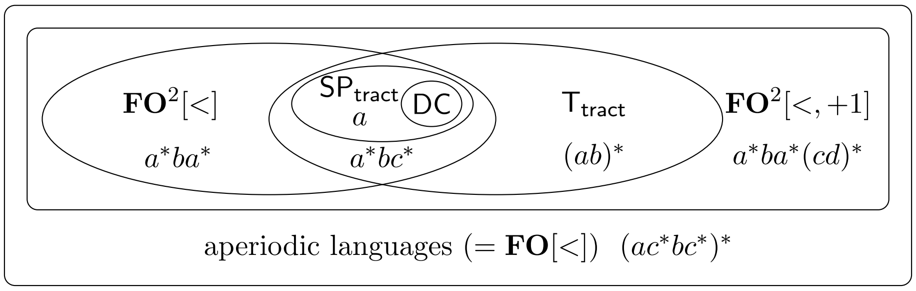
$\mathbf{FO}^2[<]$
a^*ba^*
$\mathsf{SP}_{\mathsf{tract}}$
a
DC
a^*bc^*
$\mathsf{T}_{\mathsf{tract}}$
$(ab)^*$
$\mathbf{FO}^2[<,+1]$
$a^*ba^*(cd)^*$
aperiodic languages ($= \mathbf{FO}[<]$) $(ac^*bc^*)^*$

Figure 5: Expressiveness of subclasses of the aperiodic languages

3.5. **Regular Simple Path Queries.** Bagan et al. [BBG20] analyzed the problem of computing all simple paths that satisfy a given regular path query. They introduced the class $\mathsf{SP}_{\mathsf{tract}}$, which characterizes the class of regular languages L for which the *regular simple path query (RSPQ)* problem is tractable.

Theorem 3.25 [BBG20, Theorem 3]**.** *Let L be a regular language.*
(1) *If L is finite, then $\mathsf{RSPQ}(L) \in AC^0$.*
(2) *If $L \in \mathsf{SP}_{\mathsf{tract}}$ and L is infinite, then $\mathsf{RSPQ}(L)$ is NL-complete.*
(3) *If $L \notin \mathsf{SP}_{\mathsf{tract}}$, then $\mathsf{RSPQ}(L)$ is NP-complete.*

One characterization of $\mathsf{SP}_{\mathsf{tract}}$ is the following (Theorem 6 in [BBG20]):

Theorem 3.26. $\mathsf{SP}_{\mathsf{tract}}$ *is the set of regular languages L such that there exists an $i \in \mathbb{N}$ for which the following holds: for all $w_\ell, w, w_r \in \Sigma^*$ and $w_1, w_2 \in \Sigma^+$ we have that, if $w_\ell w_1^i w w_2^i w_r \in L$, then $w_\ell w_1^i w_2^i w_r \in L$.*

Comparing the characterization with Definition 3.5, we see that Definition 3.5 imposes an extra "synchronizing" condition on w_1 and w_2, namely that they share the same first (or last) symbol. We therefore have the following observation:

Observation 3.27. The class $\mathsf{SP}_{\mathsf{tract}}$ is contained in $\mathsf{T}_{\mathsf{tract}}$.

3.6. **An algebraic Characterization of $\mathsf{T}_{\mathsf{tract}}$ and $\mathsf{SP}_{\mathsf{tract}}$.** We now provide an algerbaic characterization of $\mathsf{T}_{\mathsf{tract}}$ and $\mathsf{SP}_{\mathsf{tract}}$. We use this characterization for two things: First, we use the characterizations to fully classify the expressiveness of both classes with respect to some well known fragments of first order logic. The results are depicted in Figure 5. Later, in Section 5, we will conclude a bunch of closure properties for both classes. These properties follow from Observation 3.29, which is the only result from this subsection that is used outside of it.

We refer the reader to the book [Pin97] for a general overview of syntactic semigroups and the different hierarchies. We use the following notation. The syntactic preorder of a language L of Σ^* is the relation $\leq_L$ defined on Σ^* by $x \leq_L y$ if and only if for all $u, v \in \Sigma^*$ we have $uxv \in L \Rightarrow uyv \in L$. The syntactic congruence of L is the associated equivalence relation $\sim_L$ defined by $x \sim_L y$ if and only if $x \leq_L y$ and $y \leq_L x$. The quotient $\Sigma^+/\sim_L$ ($\Sigma^*/\sim_L$) is called the syntactic semigroup (monoid) of L. A word $e \in \Sigma^*$ is *idempotent* if $e^2 = e$. Given a finite semigroup S, it is folklore that there is an integer $\omega(S)$ (denoted by ω when S is understood) such that for all $s \in S$, s^ω is idempotent. More precisely, s^ω is the limit of the Cauchy sequence $(s^{n!})_{n \geq 0}$.

Let **SP** denote the variety of semigroups defined by the profinite inequalities $x^\omega u y^\omega \leq x^\omega y^\omega$. Let **T** denote the variety of semigroups defined by the set of profinite inequalities $(xy)^\omega u (xz)^\omega \leq (xy)^\omega (xz)^\omega$. Note that, by choosing $x = y$ and/or $x = z$, this last inequality implies that the profinite inequalities $(x)^\omega u (xz)^\omega \leq (x)^\omega (xz)^\omega$, $(xy)^\omega u (x)^\omega \leq (xy)^\omega (x)^\omega$ and $(x)^\omega u (x)^\omega \leq (x)^\omega$ must also be satisfied.

Theorem 3.28.
(1) *A regular language $L \in \Sigma^+$ is in* $\mathsf{SP}_{\mathsf{tract}}$ *if and only if its syntactic semigroup belongs to* ***SP****.*
(2) *A regular language $L \in \Sigma^+$ is in* $\mathsf{T}_{\mathsf{tract}}$ *if and only if its syntactic semigroup belongs to* ***T****.*

Proof. Item (1) follows from Theorem 3.26 and the observation that if there exists an i for which Theorem 3.26 holds, then it also holds for each $i' \geq i$. This can easily be seen by choosing $w'_\ell = w_\ell w_1^{i'-i}$ and $w'_r = w_2^{i'-i} w_r$. Item (2) follows from Theorem 3.11, Definition 3.5 and the paragraph after the definition. □

Observation 3.29. The theorem immediately implies that $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ are varieties of semigroups and ne-varieties [Pin97, PS05].

We now fully classify the expressiveness of $\mathsf{T}_{\mathsf{tract}}$ and $\mathsf{SP}_{\mathsf{tract}}$ compared to yardstick classes such as DC, $\mathbf{FO}^2[<]$, and $\mathbf{FO}^2[<,+1]$ (see also Figure 5). Here, $\mathbf{FO}^2[<]$ and $\mathbf{FO}^2[<,+1]$ are the two-variable restrictions of $\mathbf{FO}[<]$ and $\mathbf{FO}[<,+1]$ over words, respectively. By $\mathbf{FO}[<,+1]$ we mean the first-order logic with unary predicates P_a for all $a \in \Sigma$ (denoting positions carrying the letter a) and the binary predicates $+1$ and $<$ (denoting the successor relation and the order relation among positions). The logic $\mathbf{FO}[<]$ is $\mathbf{FO}[<,+1]$ without the successor predicate.

We use the characterizations from Theorem 3.28 to classify $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ wrt. the Straubing-Thérien hierarchy [Str81, Thé81]) and the dot-depth hierarchy (also known as Brzozowski hierarchy [CB71]). Both hierarchies are particular instances of concatenation hierarchies, which means that they can be built through a uniform construction scheme. Pin [Pin17] summarized numerous results and conjectures around these hierarchies.

Thomas [Tho82] showed that the dot-depth hierarchy corresponds, level by level, to the quantifier alternation hierarchy of first-order formulas, defined as follows. A formula is a Σ_n-formula if it is equivalent to a formula $Q(x_1, \ldots, x_k)\varphi$, where φ is quantifier free and $Q(x_1, \ldots, x_k)$ is a sequence of n blocks of quenatifiers such that the first block contains only existental quantifiers. The class Σ_n is the class of languages which can be defined by Σ_n-formulas. The class Π_n is defined by starting with a block of universal instead of existential quantifiers. A language L is in $\Pi_1[<,+1]$ if and only if its complement L^c is in $\Sigma_1[<,+1]$. The class of downward closed languages DC is exactly the class $\Pi_1[<]$.

Theorem 3.30. $\mathsf{DC} \subsetneq \mathsf{SP}_{\mathsf{tract}} \subsetneq \mathsf{T}_{\mathsf{tract}} \subsetneq \Pi_1[<,+1]$

Proof. We first show $\mathsf{DC} \subsetneq \mathsf{SP}_{\mathsf{tract}}$. As DC is definable by simple regular expressions, we have for each downward closed language L that $w_\ell w_1^i w w_2^i w_r \in L$ implies $w_\ell w_1^i w_2^i w_r \in L$ for every integer $i \in \mathbb{N}$ and all words $w_\ell, w_1, w, w_2, w_r \in \Sigma^*$. Therefore, $L \in \mathsf{SP}_{\mathsf{tract}}$ by Theorem 3.26. The language $\{a\}$ is not downward closed, but in $\mathsf{SP}_{\mathsf{tract}}$ using Theorem 3.26 with $i = 1$.

The subset relation $\mathsf{SP}_{\mathsf{tract}} \subseteq \mathsf{T}_{\mathsf{tract}}$ was already obserbed earlier (Observation 3.27) and a^*bc^* is a language in $\mathsf{T}_{\mathsf{tract}}$ which is not in $\mathsf{SP}_{\mathsf{tract}}$, showing that the containment is strict.

We now prove that $\mathsf{T}_{\mathsf{tract}}$ is strictly contained in $\Pi_1[<,+1]$. The class $\Sigma_1[<,+1]$ is defined by the equation $x^\omega u x^\omega \geq x^\omega$, see Pin and Weil [PW02]. As $\Pi_1[<,+1]$ is the dual variety of $\Sigma_1[<,+1]$, it is defined by $x^\omega u x^\omega \leq x^\omega$. By Theorem 3.28 it immediately follows that $\mathsf{T}_{\mathsf{tract}}$ is in the class satisfied by $x^\omega u x^\omega \leq x^\omega$ and therefore $\mathsf{T}_{\mathsf{tract}} \subseteq \Pi_1[<,+1]$. On the other hand, $(a+b)^*a(a+b)^*$ is an example of a language definable in $\Pi_1[<,+1]$ which is not in $\mathsf{T}_{\mathsf{tract}}$. □

So $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ are between $\Pi_1[<]$ and $\Pi_1[<,+1]$.

While $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ behave similar when the number of alternations of a first-order formula is restricted, restricting the number of variables ($\mathbf{FO}^2$) leads to a different behavior:

Theorem 3.31.

(a) $\mathsf{SP}_{\mathsf{tract}} \subsetneq \mathbf{FO}^2[<]$
(b) $\mathsf{T}_{\mathsf{tract}}$ *and* $\mathbf{FO}^2[<]$ *are incomparable*
(c) $\mathsf{T}_{\mathsf{tract}} \subsetneq \mathbf{FO}^2[<,+1]$

Proof. We first show (a). Thérien and Wilke [TW98] proved that $\mathbf{DA}{=}\mathbf{FO}^2[<]$, where $\mathbf{DA}$ is defined by the identity $(xyz)^\omega y(xyz)^\omega = (xyz)^\omega$. Thus we only have to prove that each syntactic semigroup of a language in $\mathsf{SP}_{\mathsf{tract}}$ satisfies this identity. Let $L \in \mathsf{SP}_{\mathsf{tract}}$. By Theorem 3.28, it immediately follows that the syntactic semigroup of L satisfies $(xyz)^\omega y(xyz)^\omega \leq (xyz)^\omega$. Thus it remains to show that there exists an n' such that for each $n \geq n'$ and all $u, v, x, y, z \in \Sigma^*$ it holds that:

$$u(xyz)^n y(xyz)^n v \in L \quad \text{if} \quad u(xyz)^n v \in L\ .$$

For this direction, we use that Bagan et al. [BBG20, Theorem 6] give a definition of $\mathsf{SP}_{\mathsf{tract}}$ in terms of regular expressions, showing that each component can be represented as $(A^{\geq k} + \varepsilon)$ for some set $A \subseteq \Sigma$ and $k \in \mathbb{N}$. So if there is $xyz \in \Sigma^*$ with $u(xyz)^M v \in L$ for some $u, v \in \Sigma^*$, then we also have $u(xyz)^M(\mathrm{Alph}(xyz))^*(xyz)^M v \subseteq L$, where $\mathrm{Alph}(x)$ denotes the set of symbols x uses. Thus we especially have $u(xyz)^M y(xyz)^M v \in L$, which proves the other direction. The same holds for each $M' \geq M$. This concludes the proof of (a).

Statement (b) simply follows from the facts that the language a^*ba^* is in $\mathbf{FO}^2[<]$ but not in $\mathsf{T}_{\mathsf{tract}}$ whereas the language $(ab)^*$ is in $\mathsf{T}_{\mathsf{tract}}$ but not in $\mathbf{FO}^2[<]$.

It remains to prove (c), which follows from Theorem 3.30 as $\Pi_1[<,+1]$ is a subset of the 1st level of the dot-depth hierarchy, which in turn is a subset of $\mathbf{FO}^2[<,+1]$. The language a^*ba^* is an example of a language in $\mathbf{FO}^2[<,+1]$ and not in $\mathsf{T}_{\mathsf{tract}}$. □

Proposition 3.32. $\mathsf{SP}_{\mathsf{tract}}$ *is in* $\mathcal{V}_{3/2}$*, the 3/2th level of the Straubing-Thérien hierarchy.*

Proof. Let $L \in \mathsf{SP}_{\mathsf{tract}}$. The 3/2th level of the Straubing-Thérien hierarchy is defined by the profinite inequality $x^\omega \leq x^\omega y x^\omega$ where $\mathrm{Alph}(x) = \mathrm{Alph}(y)$ [Pin97, Theorem 8.9]. This means that we have to show that there exists an n' such that for all $n \geq n'$ and words w_ℓ, w_r it holds: if $w_\ell x^n w_r$ in L, then also $w_\ell x^n y x^n w_r$ in L. We can easily see that every language in $\mathsf{SP}_{\mathsf{tract}}$ satisfies this: The components have the form $(A^{\geq k} + \varepsilon)$ for some set of symbols A by the definition of $\mathsf{SP}_{\mathsf{tract}}$ in terms of regular expressions, see [BBG20, Theorem 6]. Therefore, the implication immediately holds for all y with $\mathrm{Alph}(y) = \mathrm{Alph}(x)$. □

4. The Trichotomy

This section is devoted to the proof of the following theorem.

Theorem 4.1. *Let L be a regular language.*
(1) *If L is finite then $\mathsf{RTQ}(L) \in AC^0$.*
(2) *If $L \in \mathsf{T}_{\mathsf{tract}}$ and L is infinite, then $\mathsf{RTQ}(L)$ is NL-complete.*
(3) *If $L \notin \mathsf{T}_{\mathsf{tract}}$, then $\mathsf{RTQ}(L)$ is NP-complete.*

4.1. Finite Languages.

We now turn to proving Theorem 4.1. We start with Theorem 4.1(1). Clearly, we can express every finite language L as a FO-formula. Since we can also test in FO that no edge e is used more than once, the multigraphs for which $\mathsf{RTQ}(L)$ holds are FO-definable. By Immerman [Imm88], this implies that $\mathsf{RTQ}(L)$ is in AC^0.

4.2. Languages in $\mathsf{T}_{\mathsf{tract}}$.

We now sketch the proof of Theorem 4.1(2). We note that we define several concepts (trail summary, local edge domains, admissible trails) that have a natural counterpart for simple paths in Bagan et al.'s proof of the trichotomy for simple paths [BBG20]. However, the underlying proofs of the technical lemmas are quite different. For instance, components of languages in $\mathsf{SP}_{\mathsf{tract}}$ behave similarly to A^* for some $A \subseteq \Sigma$, while components of languages in $\mathsf{T}_{\mathsf{tract}}$ are significantly more complex. Furthermore, the trichotomy for trails leads to a strictly larger class of tractable languages.

For the remainder of this section, we fix the constant $K = N^2$.

We will show that in the case where L belongs to $\mathsf{T}_{\mathsf{tract}}$, we can identify a number of edges that suffice to check if the path is (or can be transformed into) a trail that matches L. This number of edges only depends on L and is therefore constant for the $\mathsf{RTQ}(L)$ problem. These edges will be stored in a *summary*. We will define summaries formally and explain how to use them to check whether a trail between the input nodes that matches L exists. To this end, we need a few definitions.

Definition 4.2. Let $p = e_1 \cdots e_m$ be a path and $r = q_0 \to \cdots \to q_m$ the run of A_L over lab(p). For a set C of states of A_L, we denote by left_C the first edge e_i with $q_{i-1} \in C$ and by right_C the last edge e_j with $q_j \in C$. A component C of A_L is a *long run component of* p if left_C and right_C are defined and $|p[\mathsf{left}_C, \mathsf{right}_C]| > K$.

Next, we want to reduce the amount of information that we require for trails. The synchronization property, see Lemma 3.18, motivates the use of *summaries*, which we define next.

Definition 4.3. Let Cuts denote the set of components of A_L and $\mathsf{Abbrv} = \mathsf{Cuts} \times (V \times Q) \times E^K$. A *component abbreviation* $(C, (v, q), e_K \cdots e_1) \in \mathsf{Abbrv}$ consists of a component C, a node v of G and state $q \in C$ to start from, and K edges $e_K \cdots e_1$. A trail π *matches* a component abbreviation, denoted $\pi \models (C, (v, q), e_K \cdots e_1)$, if $\delta_L(q, \pi) \in C$, it starts at v, and its suffix is $e_K \cdots e_1$. Given an arbitrary set of edges E', we write $\pi \models_{E'} (C, (v, q), e_K \cdots e_1)$ if $\pi \models (C, (v, q), e_K \cdots e_1)$ and all edges of π are from $E' \cup \{e_1, \ldots, e_K\}$. For convenience, we write $e \models_\emptyset e$.

If p is a trail, then the *summary* S_p of p is the sequence obtained from p by replacing, for each long run component C, the subsequence $p[\mathsf{left}_C, \mathsf{right}_C]$ by the abbreviation $(C, (v, q), p_{\mathsf{suff}})$, where v is the source node of the edge left_C, q is the state in which A_L is immediately before reading left_C, and p_{suff} is the suffix of length K of $p[\mathsf{left}_C, \mathsf{right}_C]$.

We note that the length of a summary is always bounded by $O(N^3)$, i.e., a constant that depends on L. Indeed, A_L has at most N components and, for each of them, we store at most $K+3$ many things (namely, C, v, q, and K edges). Our goal is to find a summary S and replace all abbreviations with matching pairwise edge-disjoint trails which do not use any other edge in S, because this results in a trail that matches L. However, not every sequence of edges and abbreviations is a summary, because a summary needs to be obtained from a trail. So, we will work with candidate summaries instead.

Definition 4.4. A *candidate summary* S is a sequence of the form $S = \alpha_1 \cdots \alpha_m$ with $m \leq N$ where each α_i is either (1) an edge $e \in E$ or (2) an abbreviation $(C, (v,q), e_K \cdots e_1) \in \mathsf{Abbrv}$. Furthermore, all components in S are distinct and each edge e occurs at most once. A path p that is derived from S by replacing each $\alpha_i \in \mathsf{Abbrv}$ by a trail p_i such that $p_i \models \alpha_i$ is called a *completion* of the candidate summary S.

The following corollary is immediate from the definitions and Lemma 3.18, as the lemma ensures that the state after reading p inside a component does not depend on the whole path but only on the labels of the last K edges, which are fixed.

Corollary 4.5. *Let L be a language in $\mathsf{T_{tract}}$. Let S be the summary of a trail p that matches L and let p' be a completion of S. Then, p' is a path that matches L.*

Together with the following lemmas, Corollary 4.5 can be used to obtain an nondeterministic logarithmic space algorithm[7] that gives us a completion of a summary S. The lemma heavily relies on other results on the structure of components in A_L.

Lemma 4.6. *There exists a nondeterministic logarithmic space algorithm that, given a directed graph G and nodes s and t, outputs a shortest path from s to t in G.*

Proof. We show that Algorithm 1 can output a shortest path in nondeterministic logarithmic space. Recall that nondeterministic algorithms with output either give up, or produce a correct output and that at least one computation does not give up. We note that Algorithm 1 is a mixture of the Immermann-Szelepscényi Theorem [Imm88, Sze88] and reachability. To this end, $S(k)$ denotes the set of nodes reachable from s with k edges. Using the algorithm given by Immermann [Imm88] and Szelepscényi [Sze88] to show that non-reachability is in NL, we can find in lines 1–27 the smallest n such that a path from s to t of length n but none of length $n-1$ exists. Indeed, we only added a test in line 19 to find the smallest k for which $t \in S(k)$—this k is the length of a shortest path from s to t. After line 28 we then use the smallest k (which we name n) together with a standard reachability algorithm to nondeterministically output a path of this length. (If we are only interested in the length of a shortest path, we can return n instead.) We note that one can easily change the algorithm to avoid outputting edges of paths that will give up. This would require an extra test if there exists a path of length $n-p$ from w_p to t before outputting the edge from w_{p-1} to w_p. We omitted this extra test for readability (and because at this point we know that there is a solution and non-deterministic algorithms will always return the correct output).

That Algorithm 1 runs in nondeterministic logarithmic space follows from the Immermann-Szelepscényi Theorem and reachability being in NL. □

We explain how to use the algorithm described in Lemma 4.6 to output a shortest path that satisfies some additional constraints.

[7]That is, a nondeterministic Turing Machine with read-only input and write-only output that only uses $O(\log n)$ space on its working tapes.

ALGORITHM 1: Extension of the Immermann-Szelepscényi Theorem

```
   Input: A directed graph $G = (V, E, \mathcal{E})$, nodes $s, t$ in $G$, $s \neq t$
   Output: A shortest path from $s$ to $t$ in $G$ or "no" if no path from $s$ to $t$ exists
1  $n \leftarrow -1$        ▷ $n$ will be the length of a shortest path from $s$ to $t$
2  $|S(0)| \leftarrow 1$
3  for $k = 1, 2, \ldots, |V| - 1$ do        ▷ Compute $|S(k)|$ from $|S(k-1)|$
4      $\ell \leftarrow 0$
5      foreach $u \in V$ do        ▷ Test if $u \in S(k)$
6          $m \leftarrow 0$
7          $reply \leftarrow$ false
8          foreach $v \in V$ do        ▷ Test if $v \in S(k-1)$
9              $w_0 \leftarrow s$
10             for $p = 1, \ldots, k-1$ do
11                 guess a node $w_p$
12                 if $(w_{p-1}, w_p)$ is not an edge in $G$ then
13                     give up
14             if $w_{k-1} \neq v$ then
15                 give up
16             $m \leftarrow m + 1$
17             if $(v, u)$ is an edge in $G$ then
18                 $reply \leftarrow$ true
19                 if $u = t$ then
20                     $n \leftarrow k$
21                     continue in line 28
22         if $m < |S(k-1)|$ then
23             give up
24         if $reply =$ true then
25             $\ell \leftarrow \ell + 1$
26     $|S(k)| \leftarrow \ell$
27 return "no"        ▷ $t \notin S(k)$ for any $k$
28 $w_0 \leftarrow s$
29 for $p = 1, \ldots, n$ do
30     guess a node $w_p$
31     if $(w_{p-1}, w_p)$ is not an edge in $G$ then
32         give up
33     output an edge from $w_{p-1}$ to $w_p$ in $G$
34 if $w_n \neq t$ then
35     give up
```

Lemma 4.7. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, let $(C, (v, q), e_K \cdots e_1)$ be an abbreviation and $E' \subseteq E$. There exists a nondeterministic logarithmic space algorithm that outputs a shortest trail p such that $p \models_{E'} (C, (v, q), e_K \cdots e_1)$ if it exists and rejects otherwise.*

Proof. Let G be a directed (labeled) multigraph. In order to find a path in G that matches C, ends on $e_K \cdots e_1$, and uses edges $\{e_1, \ldots, e_K\}$ only once, we use Algorithm 1 on the product of G (restricted to the edges $E' \cup \{e_1, \ldots, e_K\}$) and C extended with numbers $\ell \in [K]$. Since we cannot store the product in $O(\log n)$ space, we will construct it on-the-fly. Intuitively, the value of ℓ will tell us if we are in the last K edges and if so, then which of the last K edges we expect next. The "product" of G, C, and $[K]$ is a directed multigraph $G^* = (V^*, E^*, \mathcal{E}^*)$ defined as follows: $V^* = V \times Q \times [K]$ and

$$\begin{aligned} E^* = \{(e_\ell, (q_1, q_2, \ell)) \mid \ell \in [K] \text{ and } (q_1, \text{lab}(e_\ell), q_2) \in C\} \\ \cup \{(e, (q_1, q_2, K)) \mid e \in E' - \{e_1, \ldots, e_K\} \text{ and } (q_1, \text{lab}(e), q_2) \in C\} \end{aligned}$$

$$\begin{aligned} \mathcal{E}^*((e, (q_1, q_2, K))) &= ((\text{origin}(e), q_1, K), \text{lab}(e), (\text{destination}(e), q_2, K)) \text{ if } e \neq e_K \\ \mathcal{E}^*((e, (q_1, q_2, K))) &= ((\text{origin}(e), q_1, K), \text{lab}(e), (\text{destination}(e), q_2, K-1)) \text{ if } e = e_K \\ \mathcal{E}^*((e_\ell, (q_1, q_2, \ell))) &= ((\text{origin}(e_\ell), q_1, \ell), \text{lab}(e_\ell), (\text{destination}(e_\ell), q_2, \ell - 1)) \text{ if } \ell < K \end{aligned}$$

Since K is a constant, the size of each state in V^* is logarithmic in the input, and for two states $x, y \in V^*$, we can test in logarithmic space if there is an edge $e \in E^*$ such that $\mathcal{E}^*(e) = (x, \text{lab}(e), y)$. This is necessary for lines 12, 17, and 31.

We then output a shortest path from (v, q, K) to $(t, q', 1)$ for t being the target node of e_1 and some $q' \in C$.[8] More precisely, since we want a path in G and not in the product, we project away the unnecessary state and number and only output the corresponding edge in G in line 33.

It remains to show that p is a trail (in G). Assume towards contradiction that $p = d_1 \cdots d_m e_K \cdots e_1$ is not a trail. Then there exists an edge $d_i = d_j$ that appears at least twice in p. Note that d_j is not in the suffix $e_K \cdots e_1$ by definition of p. We define

$$p' = d_1 \cdots d_i d_{j+1} \cdots d_m e_K \cdots e_1$$

and show that p' is a shorter than p but meets all requirements. Let $q_1 = \delta(q, d_1 \cdots d_i)$ and $q_2 = \delta(q, d_1 \cdots d_j)$. By definition, $q_1, q_2 \in C$ and both have an incoming edge with label $\text{lab}(d_i) = \text{lab}(d_j)$. This allows us to use Corollary 3.16 to ensure that

$$\delta(q_1, d_{j+1} \cdots d_m e_K \cdots e_1) \in C.$$

We can then apply Lemma 3.18 to prove that

$$\delta(q_1, d_{j+1} \cdots d_m e_K \cdots e_1) = \delta(q_2, d_{j+1} \cdots d_m e_K \cdots e_1) .$$

So p' is indeed a trail satisfying $p' \models_{E'} (C, (v, q), e_K \cdots e_1)$. Furthermore, p' is shorter than p, contradicting our assumption. □

Using the algorithm of Lemma 4.7 we can, in principle, output a completion of S that matches L using nondeterministic logarithmic space. However, such a completion does not necessarily correspond to a trail. The reason is that, even though each trail p_C we guess for some abbreviation involving a component C is a trail, the trails for different components

[8]Algorithm 1 can also output a shortest path from s to some node in a set T by testing $u \in T$ in line 19 and $w_{n-1} \notin T$ in line 34.

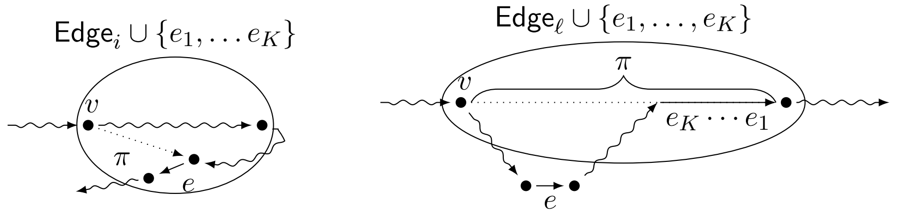
$\mathsf{Edge}_i \cup \{e_1, \dots e_K\}$
v
π
e
$\mathsf{Edge}_\ell \cup \{e_1, \dots, e_K\}$
π
v
$e_K \cdots e_1$
e

Figure 6: Sketch of case (1) and (2) in the proof of Lemma 4.10

may not be disjoint. Therefore, we will define pairwise disjoint subsets of edges that can be used for the completion of the components.

The following definition fulfills the same purpose as the local domains on nodes in Bagan et al. [BBG20, Definition 7]. Since our components can be more complex, we require extra conditions on the states (the $\delta_L(q, \pi) \in C$ condition).

Definition 4.8 (Local Edge Domains)**.** Let $S = \alpha_1 \cdots \alpha_k$ be a candidate summary and $E(S)$ be the set of edges appearing in S. We define the local edge domains $\mathsf{Edge}_i \subseteq E_i$ inductively for each i from 1 to k, where E_i are the remaining edges defined by $E_1 = E \setminus E(S)$ and $E_{i+1} = E_i \setminus \mathsf{Edge}_i$. If there is no trail p such that $p \models \alpha_i$ or if α_i is a single edge, we define $\mathsf{Edge}_i = \emptyset$.

Otherwise, let $\alpha_i = (C, (v, q), e_K \cdots e_1)$. We denote by m_i the minimal length of a trail p with $p \models_{E_i} \alpha_i$ and define Edge_i as the set of edges used by trails π that start at v, only use edges in E_i, are of length at most $m_i - K$, and satisfy $\delta_L(q, \pi) \in C$.

By definition of Edge_i, we can conclude that $\mathcal{E}(e_i) \neq \mathcal{E}(e_j)$ for all $e_i \in \mathsf{Edge}_i, e_j \in \mathsf{Edge}_j, i \neq j$, as $e_i \in \mathsf{Edge}_i$ and $\mathcal{E}(e_i) = \mathcal{E}(e_j)$ imply that $e_j \in \mathsf{Edge}_i$. We note that a shortest trail using e_i but not e_j can use e_j instead of e_i. We note that the sets $E(S)$ and $(\mathsf{Edge}_i)_{i \in [k]}$ are always disjoint.

Definition 4.9 (Admissible Trail)**.** We say that a trail p is *admissible* if there exist a candidate summary $S = \alpha_1 \cdots \alpha_k$ and trails $p_1, \dots, p_k$ such that $p = p_1 \cdots p_k$ is a completion of S and $p_i \models_{\mathsf{Edge}_i} \alpha_i$ for every $i \in [k]$.

We show that *shortest* trails that match L are always admissible. Thus, the existence of a trail is equivalent to the existence of an admissible trail.

Lemma 4.10. *Let G and (s, t) be an instance for RTQ(L), with $L \in \mathsf{T}_{\mathsf{tract}}$. Then every shortest trail from s to t in G that matches L is admissible.*

Proof sketch. We assume towards a contradiction that there is a shortest trail p from s to t in G that matches L and is not admissible. That means there is some $\ell \in \mathbb{N}$, and an edge e used in p_ℓ with $e \notin \mathsf{Edge}_\ell$. There are two possible cases: (1) $e \in \mathsf{Edge}_i$ for some $i < \ell$ and (2) $e \notin \mathsf{Edge}_i$ for any i. In both cases, we construct a shorter trail p that matches L, which then leads to a contradiction. We depict the two cases in Figure 6. We construct the new trail by substituting the respective subtrail with π. □

Proof. In this proof, we use the following notation for trails. By $p[e_1, e_2)$ we denote the prefix of $p[e_1, e_2]$ that excludes the last edge (so it can be empty). Notice that $p[e_1, e_2]$ and $p[e_1, e_2)$ are always well-defined for trails. Let $p = d_1 \cdots d_m$ be a shortest trail from s to t that matches L. Let $S = \alpha_1 \cdots \alpha_k$ be the summary of p and let $p_1, \dots, p_k$ be trails such that

$p = p_1 \cdots p_k$ and $p_i \models \alpha_i$ for all $i \in [k]$. We denote by left_i and right_i the first and last edge in p_i. By definition of p_i and the definition of summaries, left_i and right_i are identical with left_C and right_C if $\alpha_i \in \mathsf{Abbrv}$ is an abbreviation for the component C.

Assume that p is not admissible. That means there is some edge e used in p_ℓ such that $e \notin \mathsf{Edge}_\ell$. There are two possible cases:

(1) $e \in \mathsf{Edge}_i$ for some $i < \ell$; and
(2) $e \notin \mathsf{Edge}_i$ for any i.

In case (1), we choose i minimal such that some edge $e \in \mathsf{Edge}_i$ is used in p_j for some $j > i$. Among all such edges $e \in \mathsf{Edge}_i$, we choose the edge that occurs latest in p. This implicitly maximizes j for a fixed i. Especially no edge from Edge_i is used in $p_{j+1} \cdots p_k$.

Let $\alpha_i = (C_i, (v,q), e_K \cdots e_1)$. By definition of Edge_i, there is a trail π from v, ending with e, with $\delta_L(q, \mathrm{lab}(\pi)) \in C_i$, and that is shorter than the subpath $p[\mathsf{left}_i, \mathsf{right}_i]$ and therefore shorter than $p[\mathsf{left}_i, e]$.

We now show that $p' = p_1 \cdots p_{i-1} \pi p(e, d_m]$ is a trail. Since p is a trail, it suffices to prove that the edges in π are disjoint with other edges in p'. We note that all intermediate edges of π belong to Edge_i. By minimality of i, no edge in $p_1 \cdots p_{i-1}$ can use any edge of Edge_i and by our choice of e, no edge in p after e can use any edge of Edge_i. This shows that p' is a trail.

We now show that p' matches L. Since e appears in p_j, there is a path from left_j to right_j over e that stays in C_j. Let q_1 and q_2 be the states of A_L before and after reading e in p and, analogously, q_1' and q_2' the states of A_L before and after reading e in p'. That is

$$\begin{aligned} q_1 &= \delta_L(i_L, p[d_1, e)) & \qquad q_2 &= \delta_L(q_1, e) \\ q_1' &= \delta_L(i_L, p'[d_1, e)) & \qquad q_2' &= \delta_L(q_1', e) \end{aligned}$$

We note that in p', e is at the end of the subtrail π.

We can conclude that the states q_1 and q_1' both have loops starting with $a = \mathrm{lab}(e)$, as the transition $(q_1, \mathrm{lab}(e), q_2)$ is read in C_j and the transition $(q_1', \mathrm{lab}(e), q_2')$ is read in C_i. Furthermore, $q_1' \rightsquigarrow q_1$, since $q_1' \in C_i$ and $q_1 \in C_j$. Therefore, Lemma 3.19 implies that $L_{q_1} \cap L^a_{q_1}\Sigma^* \subseteq L_{q_1'}$ where $L^a_{q_1}$ denotes all words w of length K that start with a and such that $\delta_L(q_1, w) \in C_j$.

We have that $\mathrm{lab}(p[e, d_m]) \in L_{q_1}$ by the fact that p matches L. We have that $\mathrm{lab}(p[e, d_m]) \in L^a_{q_1}\Sigma^*$, as, by the definition of summaries, A_L stays in C_j for at least K more edges after reading e in p. We can conclude that $\mathrm{lab}(p[e, d_m]) \in L_{q_1'}$, which proves that p' matches L.

This concludes case (1). For case (2), we additionally assume w.l.o.g. that there is no edge $e \in \mathsf{Edge}_i$ that appears in some p_j with $j > i$, i.e., no edge satisfies case (1). By definition of Edge_ℓ, there is a trail π with $\pi \models_{\mathsf{Edge}_\ell} \alpha_\ell$ that is shorter than $p[\mathsf{left}_\ell, \mathsf{right}_\ell]$. We choose p' as the path obtained from p by replacing p_ℓ with π.

We now show that $p' = p_1 \cdots p_{\ell-1} \cdot \pi \cdot p_{\ell+1} \cdots p_k$ is a trail. Since p is a trail, it suffices to prove that the edges in π are disjoint with other edges in p'. We note that all intermediate edges of π belong to Edge_ℓ.

By definition of Edge_ℓ, no edge in $p_1 \cdots p_{\ell-1}$ is in Edge_ℓ. And by the assumption that there is no edge satisfying case (1), no edge in $p_{\ell+1} \cdots p_k$ is in Edge_ℓ. Therefore, p' is a trail.

It remains to prove that p' matches L. Let $(C, (v, \hat{q}), e_K \cdots e_1) = \alpha_\ell$ and let q and q' be the states in which A_L is before reading e_K in p and p', respectively. By definition of a summary, we have that $\delta_L(q, e_K \cdots e_1) \in C$ and, by definition of $\models$, we have that $\delta_L(q', e_K \cdots e_1) \in C$.

By Lemma 3.18 we can conclude that $\delta_L(q, e_K \cdots e_1) = \delta_L(q', e_K \cdots e_1)$. As p matches L, we can conclude that also p' matches L. □

So, if there is a solution to $\mathsf{RTQ}(L)$, we can find it by enumerating the candidate summaries and completing them using the local edge domains. We next prove that testing if an edge is in Edge_i can be done in logarithmic space. We will name this decision problem $P_{\mathsf{edge}}(L)$ and define it as follows:

	$P_{\mathsf{edge}}(L)$
Given:	A (multi-)graph $G = (V, E, \mathcal{E})$, nodes s, t, a candidate summary S, an edge $e \in E$ and an integer i.
Question:	Is $e \in \mathsf{Edge}_i$?

Lemma 4.11. *$P_{\mathsf{edge}}(L)$ is in NL for every $L \in \mathsf{T}_{\mathsf{tract}}$.*

Proof. The proof is similar to the proof of Lemma 17 by Bagan et al. [BBG20], which is based on the following result due to Immerman [Imm88]: $\mathrm{NL}^{\mathrm{NL}} = \mathrm{NL}$. In other words, if a decision problem P can be solved by an NL algorithm using an oracle in NL, then this problem P belongs to NL. Let, for each $k \geq 0$, $P^{\leq k}_{\mathsf{edge}}(L)$ be the decision problem $P_{\mathsf{edge}}(L)$ with the restriction $i \leq k$, i.e., (G, s, t, S, e, i) is a positive instance of $P^{\leq k}_{\mathsf{edge}}(L)$ if and only if (G, s, t, S, e, i) is a positive instance of $P_{\mathsf{edge}}(L)$ and $i \leq k$. Notice that i belongs to the input of $P^{\leq k}_{\mathsf{edge}}(L)$ while this is not the case for k. Obviously, $P_{\mathsf{edge}}(L) = P^{\leq |S|}_{\mathsf{edge}}(L)$. We prove that $P^{\leq k}_{\mathsf{edge}}(L) \in \mathrm{NL}$ for each $k \geq 0$. If $k = 0$, $P^{\leq 0}_{\mathsf{edge}}(L)$ always returns False because Edge_i is not defined for $i = 0$. So $P^{\leq 0}_{\mathsf{edge}}(L)$ is trivially in NL. Assume, by induction, that $P^{\leq k}_{\mathsf{edge}}(L) \in \mathrm{NL}$. It suffices to show that there is an NL algorithm for $P^{\leq k+1}_{\mathsf{edge}}(L)$ using $P^{\leq k}_{\mathsf{edge}}(L)$ as an oracle. Since $\mathrm{NL}^{\mathrm{NL}} = \mathrm{NL}$, this implies that $P^{\leq k+1}_{\mathsf{edge}}(L) \in \mathrm{NL}$.

Let (G, s, t, S, e, i) be an instance of $P^{\leq k+1}_{\mathsf{edge}}(L)$. If $i \leq k$, we return the same answer as the oracle $P^{\leq k}_{\mathsf{edge}}(L)$. If $i = k + 1$ and $\alpha_i \in E$, we return False, as $\mathsf{Edge}_i = \emptyset$. If $i = k + 1$ and $\alpha_i \in \mathsf{Abbrv}$, we first compute the length m of a minimal trail p such that $p \models_{E_i} \alpha_i$ using the NL algorithm of Lemma 4.7. We note that we can compute E_i using the NL algorithm for $P^{\leq k}_{\mathsf{edge}}$.

To test whether the edge e can be used by a trail from some (v, q) in at most $m - K$ steps, we use the on-the-fly product of G and A_L restricted to the edges in E_i and states in C. We search for a shortest path from (v, q) to some $(v', q') \in V \times C$ that ends with e. We recall that reachability is in NL.

We note that this trail in the product graph might correspond to a path p with a cycle in G. As we project away the states, some distinct edges in the product graph are actually the same edge in G. However, by Lemma 3.15, we can remove all cycles from p without losing the property that $\delta_L(q, p) \in C$. This concludes the proof. □

With this, we can finally give an NL algorithm that decides whether a candidate summary can be completed to an admissible trail that matches L.

Lemma 4.12. *Let L be a language in $\mathsf{T}_{\mathsf{tract}}$. There exists an NL algorithm that given an instance G, (s, t) of $\mathsf{RTQ}(L)$ and a candidate summary $S = \alpha_1 \cdots \alpha_k$ tests whether there is a trail p from s to t in G with summary S that matches L.*

Proof. We propose the following algorithm, which consists of three tests:

(1) Guess, on-the-fly, a path p from S by replacing each α_i by a trail p_i such that $p_i \models_{\mathsf{Edge}_i} \alpha_i$ for each $i \in [k]$. This test succeeds if and only if this is possible.
(2) In parallel, check that p matches L.
(3) In parallel, check that S is a summary of p.

We first prove that the algorithm is correct. First, we assume that there is a trail with summary S from s to t that matches L. Then, there is also a shortest such trail and, by Lemma 4.10, this trail is admissible. Therefore, the algorithm will succeed.

Conversely, assume that the algorithm succeeds. Since $E(S)$ and all the sets Edge_i are mutually disjoint, the path p is always a trail. By tests (2) and (3), it is a trail from s to t that matches L.

We still have to check the complexity. We note that the sets Edge_i are not stored in memory: we only need to check on-the-fly if a given edge belongs to those sets, which only requires logarithmic space according to Lemma 4.11. Therefore, we use an on-the-fly adaption of the NL algorithm from Lemma 4.7, which requires a set Edge_i as input, which we will provide on-the-fly.

Testing if p matches L can simply be done in parallel to test (1) on an edge-by-edge basis, by maintaining the current state of A_L in memory. If we do so, we can also check in parallel if $S = \alpha_1 \cdots \alpha_k$ is a summary of p. This is simply done by checking, for each α_i of the form $(C, (v, q), e_K \cdots e_1)$ and $\alpha_{i+1} = e$, whether $e \notin C$. This ensures that, after being in C for at least K edges, the path p leaves the component C, which is needed for summaries. Furthermore, we test if there is no substring $\alpha_i \cdots \alpha_j$ in S that purely consists of edges that are visited in the same component C, but which is too long to fulfill the definition of a summary. Since this maximal length is a constant, we can check it in NL. □

We eventually show the main Lemma of this section, proving that $\mathsf{RTQ}(L)$ is tractable for every language in $\mathsf{T}_{\mathsf{tract}}$.

Lemma 4.13. *Let $L \in \mathsf{T}_{\mathsf{tract}}$. Then, $RTQ(L) \in NL$.*

Proof. We simply enumerate all possible candidate summaries S w.r.t. (L, G, s, t), and apply on each summary the algorithm of Lemma 4.12. We return true if this algorithm succeeds and false otherwise. Since the algorithm succeeds if and only if there exists an admissible path from s to t that matches L, and consequently, if and only if there is a trail from s to t that matches L (Lemma 4.10), this is the right answer. Since L is fixed, there is a polynomial number of candidate summaries, each of logarithmic size. Consequently, they can be enumerated within logarithmic space. □

Lemma 4.14. *Let $L \in \mathsf{T}_{\mathsf{tract}}$ and L be infinite. Then, $RTQ(L)$ is NL-complete.*

Proof. The upper bound is due to Lemma 4.13, the lower due to reachability in directed graphs being NL-hard. □

Corollary 4.15. *Let $L \in \mathsf{T}_{\mathsf{tract}}$, G be a multigraph, and s, t be nodes in G. If there exists a trail from s to t that matches L, then we can output a shortest such trail in polynomial time (and in nondeterministic logarithmic space).*

Proof. For each candidate summary S, we first use Lemma 4.12 to decide whether there exists an admissible trail with summary S. With the algorithm in Lemma 4.7, we then compute the minimal length m_i of each p_i. The sum of these m_is then is the length of a shortest trail that is a completion of S. We will keep track of a summary of one of the shortest trails and finally recompute the overall shortest trail completing this summary and outputting it. Notice that this algorithm is still in NL since the summaries have constant size and overall counters never exceed $|E|$. □

4.3. **Languages not in $\mathsf{T}_{\mathsf{tract}}$.** In this section we prove that $\mathsf{RTQ}(L)$ is NP-hard for languages $L \notin \mathsf{T}_{\mathsf{tract}}$, even if the input is restricted to graphs. Therefore, NP-completeness also follows for multigraphs. The proof of Theorem 4.1(3) is by reduction from the following NP-complete problem:

	TwoEdgeDisjointPaths
Given:	A language L, a graph $G = (V, E, \mathcal{E})$, and two pairs of nodes (s_1, t_1), (s_2, t_2).
Question:	Are there two paths p_1 from s_1 to t_1 and p_2 from s_2 to t_2 such that p_1 and p_2 are edge-disjoint?

The proof is very close to the corresponding proof for simple paths by Bagan et al. [BBG20, Lemma 4] (which is a reduction from the two vertex-disjoint paths problem).

Lemma 4.16. *TwoEdgeDisjointPaths is NP-complete.*

Proof. Fortune et al. [FHW80] showed that the problem variant of TwoEdgeDisjointPaths that asks for node-disjoint paths is NP-complete. The reductions from LaPaugh and Rivest [LR80, Lemma 1 and 2] or Perl and Shiloach [PS78, Theorem 2.1 and 2.2] then imply that the NP completeness also holds for edge-disjoint paths. □

To prove the lower bound, we first show that every regular language that is not in $\mathsf{T}_{\mathsf{tract}}$ admits a witness for hardness, which is defined as follows.

Definition 4.17. A *witness for hardness* is a tuple (q, w_m, w_r, w_1, w_2) with $q \in Q_L$, $w_m, w_r, w_1, w_2 \in \Sigma^*$, $w_1 \in \mathrm{Loop}(q)$ and there exists a symbol $a \in \Sigma$ with $w_1 = aw_1'$ and $w_2 = aw_2'$ and satisfying

- $w_m(w_2)^* w_r \subseteq L_q$, and
- $(w_1 + w_2)^* w_r \cap L_q = \emptyset$.

Before we prove that each regular language that is not in $\mathsf{T}_{\mathsf{tract}}$ has such a witness, recall Property P:

$$L_{q_2} \subseteq L_{q_1} \text{ for all } q_1, q_2 \in Q_L \text{ such that } q_1 \rightsquigarrow q_2 \text{ and } \mathrm{Loop}(q_1) \cap \mathrm{Loop}(q_2) \neq \emptyset$$

Lemma 4.18. *Let L be a regular language that does not belong to $\mathsf{T}_{\mathsf{tract}}$. Then, L admits a witness for hardness.*

Proof. Let L be a regular language that does not belong to $\mathsf{T}_{\mathsf{tract}}$. Then there exist $q_1, q_2 \in Q_L$ and words w_1, w_2 with $w_1 = aw_1'$ and $w_2 = aw_2'$ such that $w_1 \in \mathrm{Loop}(q_1), w_2 \in \mathrm{Loop}(q_2)$, and $q_1 \rightsquigarrow q_2$ such that $w_2^M w_r' \notin L_{q_1}$ for a $w_r' \in L_{q_2}$. Let w_m be a word with $q_2 = \delta_L(q_1, w_m)$. We set $w_r = w_2^M w_r'$.

We now show that the so-defined tuple $(q_1, w_m, w_r, w_1, w_2)$ is a witness for hardness. By definition, we have $w_m(w_2)^* w_r \subseteq L_{q_1}$. We distinguish two cases, depending on whether L satisfies Property P or not. If L does not satisfy P, we can assume w.l.o.g. that in our tuple we have $w_1 = w_2$ and since $w_2^M w_r' \notin L_{q_1}$, we also have $w_2^* w_r \cap L_{q_1} \neq \emptyset$, so it is indeed a witness for hardness.

Otherwise, L is aperiodic, see Lemma 3.8. We assume w.l.o.g. that $w_1 = (w_1')^M$ for some word w_1'. Then, we claim that $L_{q'} \subseteq L_{q_1}$ for every q' in $\delta_L(q_1, \Sigma^* w_1)$. Indeed, every $q' \in \delta_L(q_1, \Sigma^* w_1)$ loops over w_1 by the pumping lemma and aperiodicity of L, hence $w_1 \in \text{Loop}(q_1) \cap \text{Loop}(q')$ and therefore $L_{q'} \subseteq L_{q_1}$ due to Property P.

It remains to prove that $(w_1 + w_2)^* w_r \cap L_{q_1} = \emptyset$. Every word in $(w_1 + w_2)^* w_r$ can be decomposed into uv with $u \in \varepsilon + (w_1 + w_2)^* w_1$ and $v \in w_2^* w_r$. For $q' = \delta_L(q_1, u)$ we have proved that $L_{q'} \subseteq L_{q_1}$, so it suffices to show that $v \notin L_{q_1}$. This is immediate from $w_r = w_2^M w_r' \notin L_{q_1}$ and the aperiodicity of L. So we have $uv \notin L_{q_1}$ and the tuple (q, w_m, w_r, w_1, w_2) is indeed a witness for hardness. □

We can now show the following

Lemma 4.19. *Let L be a regular language that does not belong to $\mathsf{T}_{\mathsf{tract}}$. Then, $\mathsf{RTQ}(L)$ is NP-complete.*

Proof. The proof is almost identical to the reduction from two node-disjoint paths to the $\mathsf{RSPQ}(L)$ problem by Bagan et al. [BBG20]. Clearly, $\mathsf{RTQ}(L)$ is in NP for every regular language L, since we only need to guess a trail of length at most $|E|$ from s to t and verify that the word on the trail is in L. Let $L \notin \mathsf{T}_{\mathsf{tract}}$. We exhibit a reduction from $\mathsf{TwoEdgeDisjointPaths}$ to $\mathsf{RTQ}(L)$. According to Lemma 4.18, L admits a witness for hardness (q, w_m, w_r, w_1, w_2). Let w_ℓ be a word such that $\delta_L(i_L, w_\ell) = q$. By definition of a witness we get $w_\ell(w_1 + w_2)^* w_r \cap L = \emptyset$ and $w_\ell w_1^* w_m w_2^* w_r \subseteq L$. Let $a \in \Sigma$ and $w_1', w_2' \in \Sigma^*$ such that $w_1 = a w_1'$ and $w_2 = a w_2'$. If w_1' or w_2' is empty, we replace it with a.

In the following construction, whenever we say that we add a path from v_0 to v_n labeled by a word $w = a_1 \cdots a_n$, denoted by $v_0 \overset{w}{\rightsquigarrow} v_n$ we mean that we add $n-1$ new nodes $v_1, \ldots, v_{n-1}$ and n new edges $e_1, \ldots, e_n$ such that $\mathcal{E}'(e_i) = (v_{i-1}, a_i, v_i)$.

Let $G = (V, E)$ be an unlabeled and simple input graph for the $\mathsf{TwoEdgeDisjointPaths}$ problem and s_1, t_1, s_2, t_2 be nodes in V. We build from G a graph $G' = (V', E', \mathcal{E}')$ such that (G, s_1, t_1, s_2, t_2) is a yes-instance of $\mathsf{TwoEdgeDisjointPaths}$ if and only if there is a trail from s to t matching L in G'. We start with the nodes from G and add two new nodes s and t and three paths $s \overset{w_\ell}{\rightsquigarrow} s_1$, $t_1 \overset{w_m}{\rightsquigarrow} s_2$, and $t_2 \overset{w_r}{\rightsquigarrow} t$. Furthermore, for each edge (v_1, v_2) in G, we add a new node v_{12} and three paths $v_1 \overset{a}{\rightsquigarrow} v_{12}$, $v_{12} \overset{w_1'}{\rightsquigarrow} v_2$, and $v_{12} \overset{w_2'}{\rightsquigarrow} v_2$. An example for the language $da^*c(abc)^*ef$ and some graph G can be seen in Figure 7.

By construction, two edge disjoint paths p_1 and p_2 in G going from s_1 to t_1 and from s_2 to t_2 correspond to a trail p from s to t in G' that contains the path $t_1 \overset{w_m}{\rightsquigarrow} s_2$. Such a trail p matches a word in $w_\ell(w_1 + w_2)^* w_m (w_1 + w_2)^* w_r$. And, as paths labeled w_1 and w_2 can be used interchangeably, we find a trail matching $w_\ell w_1^* w_m w_2^* w_r \subseteq L$.

For the other direction, we have to show two things. First, we show that every trail p in G' from s to t that uses the path $t_1 \overset{w_m}{\rightsquigarrow} s_2$ proves the existence of two edge disjoint paths p_1 and p_2 in G from s_1 to t_1 and from s_2 to t_2. Indeed p_1 and p_2 can be computed from p by keeping only those nodes that are from G and splitting p between t_1 and s_2. The paths are disjoint, as otherwise some edge v_i to v_{ij} has to be used twice by p. Second, we show that there can be no trail p from s to t in G' that matches L and does not use the path $t_1 \overset{w_m}{\rightsquigarrow} s_2$. Indeed, every trail p from s to t in G' that does not contain the path $t_1 \overset{w_m}{\rightsquigarrow} s_2$ matches a word in $w_\ell(w_1 + w_2)^* w_r$. By definition of witness for hardness, no such word is in L. Thus, $\mathsf{RTQ}(L)$ returns true for (G', s, t) if and only if there is a trail from s to t in G' that contains the edge (t_1, w_m, s_2) that is, if and only if $\mathsf{TwoEdgeDisjointPaths}$ returns true for (G, s_1, t_1, s_2, t_2). □

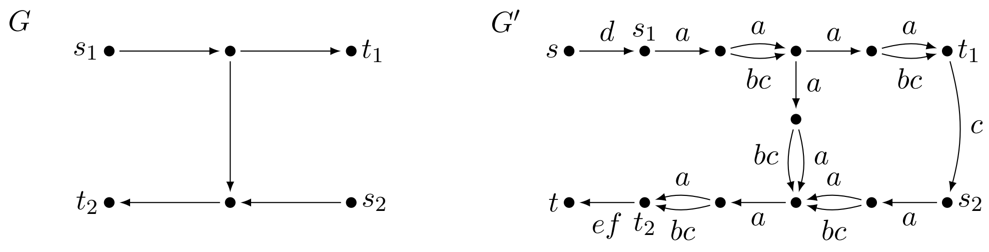

Figure 7: Example of the reduction in Lemma 4.19 for the language $da^*c(abc)^*ef$. We use $w_\ell = d$, $w_m = c$, $w_r = ef$, $w_1 = aa$, and $w_2 = abc$ for the construction. For the ease of readability, we omit the intermediate nodes on the bc and ef paths.

5. Recognition and Closure Properties

The following theorem establishes the complexity of deciding if a regular language is in $\mathsf{T}_{\mathsf{tract}}$.

Before we establish the complexity of deciding for a regular language L whether $L \in \mathsf{T}_{\mathsf{tract}}$, we need some lemmas. The first has been adapted from the simple path case (Lemma 6 in [BBG20]).

Lemma 5.1. *Let L be a regular language. Then, L belongs to $\mathsf{T}_{\mathsf{tract}}$ if and only if for all pairs of states $q_1, q_2 \in Q_L$ and symbols $a \in \Sigma$ such that $q_1 \rightsquigarrow q_2$ and $\text{Loop}(q_1) \cap a\Sigma^* \neq \emptyset$, the following statement holds: $(\text{Loop}(q_2) \cap a\Sigma^*)^N L_{q_2} \subseteq L_{q_1}$.*

Proof. The (if) implication is immediate by Corollary 3.7. Let us now prove the (only if) implication. Since the proof of this lemma requires a number of different states and words, we provide a sketch in Figure 8. Assume $L \in \mathsf{T}_{\mathsf{tract}}$. Let q_1, q_2 be two states such that $\text{Loop}(q_1) \cap a\Sigma^* \neq \emptyset$ and $q_1 \rightsquigarrow q_2$. If $\text{Loop}(q_2) \cap a\Sigma^* = \emptyset$, the statement follows immediately. So let us assume w.l.o.g. that $\text{Loop}(q_2) \cap a\Sigma^* \neq \emptyset$. Let $v_1, \ldots, v_N \in (\text{Loop}(q_2) \cap a\Sigma^*)$ be arbitrary words and $q_3 = \delta_L(q_1, v_1 \cdots v_N)$. We want to prove $L_{q_2} \subseteq L_{q_3}$. For some i, j with $0 \leq i < j \leq N$, we get $\delta_L(q_1, v_1 \cdots v_i) = \delta_L(q_1, v_1 \cdots v_j)$ due to the pumping Lemma. (We have $\delta_L(q_1, v_1 \cdots v_i) = q_1$ for $i = 0$.) Let $u_1 = v_1 \cdots v_i, u_2 = v_{i+1} \cdots v_j$ and $u_3 = v_{j+1} \cdots v_k$. Let $q_4 = \delta_L(q_1, u_1)$.

We claim that $L_{q_2} \subseteq L_{q_4}$. The result then follows from $L_{q_2} = u_3^{-1} L_{q_2} \subseteq u_3^{-1} L_{q_4} = L_{q_3}$. To prove the claim, let $w = u_1 u_2^N$ and $q_5 = \delta_L(q_1, w^N)$. As $w \in \text{Loop}(q_2)$, we can use Corollary 3.7 to obtain $w^N L_{q_2} \subseteq L_{q_1}$. Together with $L_{q_5} = (w^N)^{-1} L_{q_1}$ this implies $L_{q_2} \subseteq L_{q_5}$. Furthermore, u_2 belongs to $\text{Loop}(q_5)$ because L is aperiodic. To conclude the proof, we observe that $L_{q_5} \subseteq L_{q_4}$, by Corollary 3.7 with q_5, q_4 and u_2, and because $\delta_L(q_4, u_2^N) = q_4$ and $u_2 \in \text{Loop}(q_5)$. □

Theorem 5.2. *Testing whether a regular language L belongs to $\mathsf{T}_{\mathsf{tract}}$ is*
(1) *NL-complete if L is given by a DFA and*
(2) *PSPACE-complete if L is given by an NFA or by a regular expression.*

Proof. The proof is inspired by Bagan et al. [BBG20]. The upper bound for (1) needs several adaptations, the lower bound for (1) and the proof for (2) works exactly the same as in [BBG12], a preliminary version of [BBG20] (just replacing $\mathsf{SP}_{\mathsf{tract}}$ by $\mathsf{T}_{\mathsf{tract}}$).

We first prove (1). W.l.o.g., we can assume that L is given by the minimal DFA A_L, as testing Nerode-equivalence of two states is in NL.

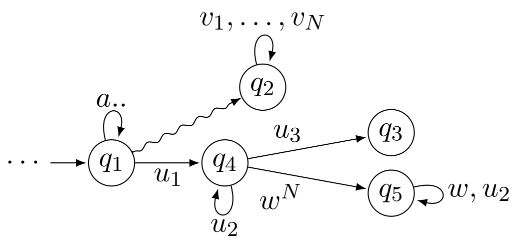

Figure 8: Sketch of the proof of Lemma 5.1

By Lemma 5.1, we need to check for each pair of states q_1, q_2 and symbol $a \in \Sigma$ whether

(i) $q_1 \rightsquigarrow q_2$;
(ii) $\text{Loop}(q_1) \cap a\Sigma^* \neq \emptyset$; and
(iii) $(\text{Loop}(q_2) \cap a\Sigma^*)^N L_{q_2} \setminus L_{q_1} = \emptyset$.

Statements (i) and (ii) are easily verified using an NL algorithm for transitive closure. For (iii), we test emptiness of $(\text{Loop}(q_2) \cap a\Sigma^*)^N L_{q_2} \setminus L_{q_1}$ using an NL algorithm for reachability in the product automaton of A_L with itself, starting in the state (q_2, q_1). More precisely, the algorithm checks whether there does not exist a string that is in L_{q_2}, is not in L_{q_1}, starts with an a, and leaves the state q_2 (in the left copy of A_L) at least N times with an a-transition.

The remainder of the proof is from [BBG12] and only included for self containedness.

For the lower bound of (1), we give a reduction from the Emptiness problem. Let $L \subseteq \Sigma^*$ be an instance of Emptiness given by a DFA A_L. W.l.o.g. we assume that $\varepsilon \notin L$, since this can be checked in constant time. Furthermore, we assume that the symbol 1 does not belong to Σ. Let $L' = 1^+ L 1^+$. A DFA $A_{L'}$ that recognizes L' can be obtained from A_L as follows. We add a state q_I that will be the initial state of $A_{L'}$. and a state q_F that will be the unique final state of $A_{L'}$. The transition function $\delta_{L'}$ is the extension of δ_L defined as follows:

- $\delta_{L'}(q_I, 1) = q_I$ and $\delta_{L'}(q_I, a) = i_L$ for every symbol $a \in \Sigma$.
- For every final state $q \in F_L$, $\delta_{L'}(q, 1) = q_F$.
- $\delta_{L'}(q_F, 1) = q_F$.

We will show that $L' \in \mathsf{T_{tract}}$ if and only if L is empty. If L is empty, then $L' = \emptyset$ belongs to $\mathsf{T_{tract}}$. For the other direction, assume that L is not empty. Let $w \in L$. Then, for every $n \in \mathbb{N}$, $1^n w 1^n \in L'$ and $1^n 1^n \notin L'$. Thus $L' \notin \mathsf{T_{tract}}$.

For the upper bound of (2), we first observe the following fact: Let A, B be two problems such that $A \in$ NL and let t be a reduction from B to A that works in polynomial space and produces an exponential output. Then B belongs to PSPACE. Thus, we can apply the classical powerset construction for determinization on the NFA and use the upper bound from (1).

For the lower bound of (2), we give a reduction from Universality. Let $L \subseteq \{0,1\}^*$ be an instance of Universality given by an NFA or a regular expression. Consider $L' = (0+1)^* a^* b a^* + L a^*$ over the alphabet $\{0, 1, a, b\}$. We show that $L = \{0,1\}^*$ if and only if $L' \in \mathsf{T_{tract}}$. Our reduction associates L' to L and keeps the same representation (NFA or regular expression). If $L' = \{0,1\}^*$, then $L' = (0+1)^* a^* (b + \varepsilon) a^*$ and thus $L' \in \mathsf{T_{tract}}$. Conversely, assume $L \neq \{0,1\}^*$. Let $w \in \{0,1\}^* \setminus L$. Then, for every $n \in \mathbb{N}$, $w a^n b a^n \in L'$ and $w a^n a^n \notin L'$. Thus $L' \notin \mathsf{T_{tract}}$. □

We wondered if, similarly to Theorem 3.2, it could be the case that languages closed under left-synchronized power abbreviations are always regular, but this is not the case. For example, the (infinite) Thue-Morse word [Thu06, Mor21] has no subword that is a cube (i.e., no subword of the form w^3) [Thu06, Satz 6]. The language containing all prefixes of the Thue-Morse word thus trivially is closed under left-synchronized power abbreviations (with $i = 3$), yet it is not regular.

We now give some closure properties of $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$. We note that Bagan et al. [BBG20] already observed that $\mathsf{SP}_{\mathsf{tract}}$ is closed under finitie unions, intersections, and reversal.

Lemma 5.3. *Both classes* $\mathsf{SP}_{\mathsf{tract}}$ *and* $\mathsf{T}_{\mathsf{tract}}$ *are closed under* (i) *finite unions,* (ii) *finite intersections,* (iii) *reversal,* (iv) *left and right quotients,* (v) *inverses of non-erasing morphisms,* (vi) *removal and addition of individual strings.*

Proof. The closure properties (i) to (vi) follow immediately from Observation 3.29, i.e., that $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ are ne-varieties, see [Pin97, PS05]. □

Lemma 5.4. *The classes* $\mathsf{SP}_{\mathsf{tract}}$ *and* $\mathsf{T}_{\mathsf{tract}}$ *are not closed under complement.*

Proof. Let $\Sigma = \{a, b\}$. The language of the expression b^* clearly is in $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$. Its complement is the language L containing all words with at least one a. It can be described by the regular expression $\Sigma^* a \Sigma^*$. Since $b^i a b^i \in L$ for all i, but $b^i b^i \notin L$ for any i, the language L is neither in $\mathsf{SP}_{\mathsf{tract}}$ nor in $\mathsf{T}_{\mathsf{tract}}$. □

It is an easy consequence of Lemma 5.3 (vi) that for regular languages outside of $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$ there do not exist best lower or upper approximations.

Corollary 5.5. *Let $\mathcal{C} \in \{\mathsf{SP}_{\mathsf{tract}}, \mathsf{T}_{\mathsf{tract}}\}$. For every regular language L such that $L \notin \mathcal{C}$ and*

- *for every upper approximation L'' of L (i.e., $L \subsetneq L''$) with $L'' \in \mathcal{C}$ it holds that there exists a language $L' \in \mathcal{C}$ with $L \subsetneq L' \subsetneq L''$;*
- *for every lower approximation L'' of L (i.e., $L'' \subsetneq L$) it holds that there exists a language $L' \in \mathcal{C}$ with $L'' \subsetneq L' \subsetneq L$.*

The corollary implies that Angluin-style learning of languages in $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$ is not possible. However, learning algorithms for single-occurrence regular expressions (SOREs) exist [BNSV10] and can therefore be useful for an important subclass of $\mathsf{T}_{\mathsf{tract}}$.

6. Enumeration

In this section we state that—using the algorithm from Theorem 4.1—the enumeration result from [Yen71] transfers to the setting of enumerating trails matching L.

Theorem 6.1. *Let L be a regular language, G be a multigraph and (s, t) a pair of nodes in G. If NL $\neq$ NP, then one can enumerate trails from s to t that match L in polynomial delay in data complexity if and only if $L \in \mathsf{T}_{\mathsf{tract}}$.*

Proof sketch. The algorithm is an adaptation of Yen's algorithm [Yen71] that enumerates the k shortest simple paths for some given number k, similar to what was done by Martens and Trautner [MT19]. It uses the algorithm from Corollary 4.15 as a subprocedure. □

ALGORITHM 2: Yen's algorithm changed to work with trails on multigraphs

```
   Input: Multigraph $G = (V, E, \mathcal{E})$, nodes $s, t \in V$, a language $L \in \mathsf{T}_{\mathsf{tract}}$
   Output: All trails from $s$ to $t$ in $G$ that match $L$ under bag semantics
1  $A \leftarrow \emptyset$                    ▷ $A$ is the set of trails already written to output
2  $B \leftarrow \emptyset$                    ▷ $B$ is a set of trails from $s$ to $t$ matching $L$
3  $p \leftarrow$ a shortest trail from $s$ to $t$ matching $L$     ▷ $p \leftarrow \bot$ if no such trail exists
4  while $p \neq \bot$ do
5      output $p$
6      Add $p$ to $A$
7      for $i = 0$ to $|p|$ do
8          $G' \leftarrow (V, E', \mathcal{E}|_{E'})$ with $E' = E \setminus E(p[1,i])$     ▷ Remove edges used in $p[1,i]$
9          $S \leftarrow \{e \in E \mid p[1,i] \cdot e$ is a prefix of a trail in $A\}$
10         $p_1 \leftarrow$ a shortest trail from destination$(p[1,i])$ to $t$ in $G'$ that matches
            $((\mathrm{lab}(p[1,i]))^{-1}L) \setminus \{\varepsilon\}$ and does not start with an edge from $S$
11         Add $p[1,i] \cdot p_1$ to $B$
12     $p \leftarrow$ a shortest trail in $B$                ▷ $p \leftarrow \bot$ if $B = \emptyset$
13     Remove $p$ from $B$
```

Notice that we cannot simply use the line graph construction and solve this problem for simple paths since the class of regular languages that is tractable for simple paths is a strict subset of $\mathsf{T}_{\mathsf{tract}}$. So this method would not, for example, solve the problem for $L = (ab)^*$.

Instead, we can change Yen's algorithm [Yen71] to work with trails instead of simple paths. The changes are straightforward: instead of deleting nodes, we only delete edges. In Algorithm 2 we changed Yen's algorithm to enumerate L-labeled trails. Note that we only need $L \in \mathsf{T}_{\mathsf{tract}}$ to ensure that the subroutines in lines 3 and 10 are in polynomial time.

Explanation of Algorithm 2. In the algorithm, $p[1,i]$ denotes the prefix of p containing exactly i edges and destination(p) denotes the last node of p. In line 3, we can use the algorithm explained in the proof of Theorem 4.1 (more concretely, Corollary 4.15) to find a L-labeled trail from s to t in G in NL if one exists. In the for-loop in line 7 we use quotients of the last trail written to the output to find new candidates. Intuitively, for all $i \in \mathbb{N}$, we regard all paths that share the prefix of length exactly i with the last path and do not share a prefix of length $i+1$ with any path outputted so far. In line 10, we search for a suffix to the prefix $p[1,i]$ by again using the algorithm explained in the proof of Theorem 4.1. We recall that $\mathsf{T}_{\mathsf{tract}}$ is closed under left derivatives and removal of individual strings, see Lemma 5.3, i.e., $(\mathrm{lab}(p[1,i]))^{-1}L \setminus \{\varepsilon\}$ is in $\mathsf{T}_{\mathsf{tract}}$. However, to prevent finding a trail that was already in the output, we do not allow the suffix to start with some edge from S. We note that the algorithm from Corollary 4.15 can be easily modified to check for this additional condition. We repeat this procedure with all trails in B, until we do not find any new trails.

Set vs. Bag Semantics in Multigraphs. Let us consider a small multigraph $G = (V, E, \mathcal{E})$ with two nodes $\{v_1, v_2\}$ and two edges e_1, e_2 with $\mathcal{E}(e_1) = \mathcal{E}(e_2) = (v_1, a, v_2)$. If we want to enumerate all paths from v_1 to v_2 that match a, how many paths should we get? Under set semantics, we will only obtain a single answer, whereas, under bag semantics, we will consider e_1 and e_2 as different edges and therefore return two answers. Algorithm 2 enumerates

ALGORITHM 3: Yen's algorithm for trails with set semantics

```
Input: Multigraph $G = (V, E, \mathcal{E})$, nodes $s, t \in V$, a language $L \in \mathsf{T}_{\mathsf{tract}}$
Output: All trails from $s$ to $t$ in $G$ that match $L$ under bag semantics
1  $A \leftarrow \emptyset$                         ▷ $A$ is the set of trails already written to output
2  $B \leftarrow \emptyset$                         ▷ $B$ is a set of trails from $s$ to $t$ matching $L$
3  $p \leftarrow$ a shortest trail from $s$ to $t$ matching $L$     ▷ $p \leftarrow \bot$ if no such trail exists
4  $j \leftarrow 0$                                 ▷ tells where the deriviation should start
5  while $p \neq \bot$ do
6      output $p$
7      Add $p$ to $A$
8      for $i = j$ to $|p|$ do
9          $G' \leftarrow (V, E', \mathcal{E}|_{E'})$ with $E' = E \setminus E(p[1,i])$   ▷ Remove edges used in $p[1,i]$
10         $S \leftarrow \{e' \in E \mid \exists e$ with $\mathcal{E}(e) = \mathcal{E}(e')$ and $p[1,i] \cdot e$ is a prefix of a trail in $A\}$
11         $p_1 \leftarrow$ a shortest trail from destination($p[1,i]$) to $t$ in $G'$ that matches
            $((\mathrm{lab}(p[1,i]))^{-1}L) \setminus \{\varepsilon\}$ and does not start with an edge from $S$
12         Add $(p[1,i] \cdot p_1, i)$ to $B$
13     $(p, j) \leftarrow$ a pair from $B$ with $p$ being a shortest trail      ▷ $p \leftarrow \bot$ if $B = \emptyset$
14     Remove $(p, j)$ from $B$
```

trails according to bag semantics. This is because all edges are considered to be different. Algorithm 3 enumerates trails according to set semantics.

The changes between Algorithm 2 and Algorithm 3 are the following:

(1) The set S is computed differently. In Algorithm 2 the set S contains exactly the edges that were already used to continue the path $p[1, i]$, whereas in Algorithm 3 the set S contains all edges whose origin, destination, and label are identical to some edge already used to continue the path $p[1, i]$.

(2) In Algorithm 2 the set B can contain paths that are identical under set semantics, while this is forbidden Algorithm 3. One possibility to realize this is in Algorithm 3 is to use Lawler's [Law72] extension of Yen's algorithm. Lawler observed that if a path $p' = p[1, i] \cdot p_1$ was added to A, in the next iteration it is sufficent to start the derivation from i, as the derivatives of $p[1, i]$ have already been added to B. This change impacts the for-loop in line 8 of Algorithm 3.

We note that Lawler's extension of Yen's algorithm can also be used under bag semantics to improve the running time, as observed by Lawler [Law72]. The following example shows that (1) alone is not sufficient to enumerate multigraphs under set semantics because it would be possible that different paths are added to B which are identical under set semantics, but different under bag semantics.

Example 6.2. Consider the graph illustrated in Figure 9 with nodes $\{s, v, t\}$ and edges $\{e_1, \ldots, e_5\}$. Assume S in line 9 in Algorithm 2 is calculated as in (1). If the algorithm starts with the ab-path e_1e_2, it can add one db-path e_4e_2 and the ab-path e_1e_3 to B. If it picks the ab-path e_1e_3 from B in the next step, the algorithm might add the other db-path e_5e_2 to B. Thus $B = \{e_4e_2, e_5e_2\}$ contains two paths that are identical under set semantics.

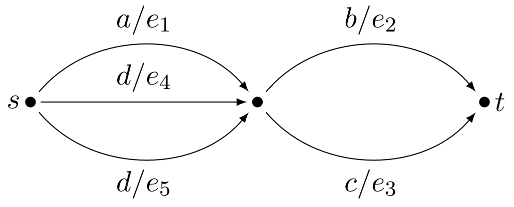

Figure 9: Example graph for enumeration under set semantics. Edges are annotated with label and edge identifier.

7. Conclusions and Lessons Learned

We have defined the class $\mathsf{T}_{\mathsf{tract}}$ of regular languages L for which finding trails in directed graphs that are labeled with L is tractable iff NL $\neq$ NP. We have investigated $\mathsf{T}_{\mathsf{tract}}$ in depth in terms of closure properties, characterizations, and the recognition problem, also touching upon the closely related class $\mathsf{SP}_{\mathsf{tract}}$ (for which finding *simple paths* is tractable) when relevant.

In our view, graph database manufacturers can have the following trade-offs in mind concerning simple path ($\mathsf{SP}_{\mathsf{tract}}$) and trail semantics ($\mathsf{T}_{\mathsf{tract}}$) in database systems:

- $\mathsf{SP}_{\mathsf{tract}} \subsetneq \mathsf{T}_{\mathsf{tract}}$, that is, there are strictly more languages for which finding regular paths under trail semantics is tractable than under simple path semantics. Some of the languages in $\mathsf{T}_{\mathsf{tract}}$ but outside $\mathsf{SP}_{\mathsf{tract}}$ are of the form $(ab)^*$ or a^*bc^*, which were found to be relevant in several application scenarios involving network problems, genomic datasets, and tracking provenance information of food products [PS] and appear in query logs [BMT17, BMT19].
- Both $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ can be syntactically characterized but, currently, the characterization for $\mathsf{SP}_{\mathsf{tract}}$ (Section 7 in [BBG20]) is simpler than the one for $\mathsf{T}_{\mathsf{tract}}$. This is due to the fact that connected components for automata for languages in $\mathsf{T}_{\mathsf{tract}}$ can be much more complex than for automata for languages in $\mathsf{SP}_{\mathsf{tract}}$.
- On the other hand, the *single-occurrence* condition, i.e., each alphabet symbol occurs at most once, is a sufficient condition for regular expressions to be in $\mathsf{T}_{\mathsf{tract}}$. This condition is trivial to check and also captures languages outside $\mathsf{SP}_{\mathsf{tract}}$ such as $(ab)^*$ and a^*bc^*. Moreover, the condition seems to be useful: we analyzed the 50 million RPQs found in the logs of [BMT18] and discovered that over 99.8% of the RPQs are single-occurrence.
- In terms of closure properties, learnability, or complexity of testing if a given regular language belongs to $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$, the classes seem to behave the same.
- The tractability for the decision version of RPQ evaluation can be lifted to the enumeration problem, in which case the task is to output matching paths with only a polynomial delay between answers.

As an open question remains the trichotomy for 2RPQs, that is, when we allow RPQs to follow a directed edge also in its reverse direction. We briefly discuss why this is challenging. Let us denote by $\hat{\ }a$ the backward navigation of an edge labeled a. Then, the case of ordinary RPQs can be seen as a special case of 2RPQs on undirected graphs: it only has bidirectional navigation of the form $(a + \hat{\ }a)$. It has been open since 1991 whether evaluating $(aaa)^*$ on undirected graphs is in P or NP-complete [APY91].

Acknowledgments. This work was supported by DFG grant MA 4938/4-1. We thank the participants of Shonan meeting No. 138 (and Hassan Chafi in particular), who provided significant inspiration for the first paragraph in the Introduction, Jean-Éric Pin for pointing us to positive C_{ne}-varieties of languages, and Charles Paperman for useful comments and pointing out that $\mathsf{T}_{\mathsf{tract}} \subsetneq \Pi[<, +1]$ (see Theorem 3.30). We also thank Jean-Éric Pin and Luc Segoufin for their help with the proof with a weaker statement. Furthermore, we want to thank the anonymous reviewers for valuable comments.

References

[AAB$^+$18] Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter A. Boncz, George H. L. Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan F. Sequeda, Oskar van Rest, and Hannes Voigt. G-CORE: A core for future graph query languages. In *International Conference on Management of Data (SIGMOD)*, pages 1421–1432, 2018.

[ACBJ04] Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

[ACP12] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. In *International Conference on World Wide Web (WWW)*, pages 629–638, 2012.

[APY91] Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *Journal of the ACM*, 38(2):255–274, 1991.

[Bar13] Pablo Barceló. Querying graph databases. In *Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013.

[BBG12] Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. *CoRR*, abs/1212.6857, 2012.

[BBG20] Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. *Journal of Computer and System Sciences*, 108:29–48, 2020.

[BLR11] Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In *Symposium on Principles of Database Systems (PODS)*, pages 199–210. ACM, 2011.

[BMT17] Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *Proceedings of the VLDB Endowment*, 11(2):149–161, 2017.

[BMT18] Angela Bonifati, Wim Martens, and Thomas Timm. DARQL: deep analysis of SPARQL queries. In *The Web Conference (WWW) (Companion Volume)*, pages 187–190. ACM, 2018.

[BMT19] Angela Bonifati, Wim Martens, and Thomas Timm. Navigating the maze of wikidata query logs. In *The Web Conference (WWW)*, pages 127–138. ACM, 2019.

[BNSV10] Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtds. *ACM Transactions on Database Systems*, 35(2):11:1–11:47, 2010.

[CB71] Rina S. Cohen and Janusz A. Brzozowski. Dot-depth of star-free events. *Journal of Computer and System Sciences*, 5(1):1–16, 1971.

[DBp] Dbpedia. `wiki.dbpedia.org`.

[FGG$^+$18] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. Cypher: An evolving query language for property graphs. In *International Conference on Management of Data (SIGMOD)*, pages 1433–1445. ACM, 2018.

[FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science (TCS)*, 10(2):111–121, 1980.

[GGM12] Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM Journal on Computing*, 41(1):160–190, 2012.

[Hai69] Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

[Jul69] Pierre Jullien. *Contribution à l'étude des types d'ordres dispersés*. PhD thesis, Universite de Marseille, 1969.

[Law72] Eugene L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18(7):401—-405, mar 1972.

[LM13] Katja Losemann and Wim Martens. The complexity of regular expressions and property paths in SPARQL. *ACM Transactions on Database Systems*, 38(4):24:1–24:39, 2013.

[LP84] Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.

[LR80] Andrea S. LaPaugh and Ronald L. Rivest. The subgraph homeomorphism problem. *Journal of Computer and System Sciences*, 20(2):133–149, 1980.

[MNT20] Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 154 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[Mor21] Harold Marston Morse. Recurrent geodesics on a surface of negative curvature. *Transactions of the American Mathematical Society*, 22(1):84–100, Jan 1921. doi:10.2307/1988844.

[MT19] Wim Martens and Tina Trautner. Dichotomies for evaluating simple regular path queries. *ACM Transactions on Database Systems*, 44(4):16:1–16:46, 2019.

[MW95] Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 12 1995.

[Neo] Neo4j. neo4j.com.

[Ope] Cypher query language reference, version 9, mar. 2018. https://github.com/opencypher/openCypher/blob/master/docs/openCypher9.pdf.

[Ora] Oracle spatial and graph. www.oracle.com/technetwork/database/options/spatialandgraph/.

[Pap22] Charles Paperman. Semigroup online. https://paperman.name/semigroup/, 2022.

[Pin97] Jean-Eric Pin. Syntactic semigroups. In *Handbook of Formal Languages (1)*, pages 679–746. Springer, 1997.

[Pin17] Jean-Éric Pin. The dot-depth hierarchy, 45 years later. In *The Role of Theory in Computer Science*, pages 177–202. World Scientific, 2017.

[PS] Neo4J Petra Selmer. Personal communication.

[PS78] Yehoshua Perl and Yossi Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *Journal of the ACM*, 25(1):1–9, 1978.

[PS05] Jean-Eric Pin and Howard Straubing. Some results on $\mathcal{C}$-varieties. *RAIRO Theoretical Informatics and Applications*, 39(1):239–262, 2005.

[PW02] Jean-Eric Pin and Pascal Weil. The wreath product principle for ordered semigroups. *Communications in Algebra*, 30:5677–5713, 2002.

[Sch65] Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

[Str81] Howard Straubing. A generalization of the schützenberger product of finite monoids. *Theoretical Computer Science (TCS)*, 13:137–150, 1981.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Thé81] Denis Thérien. Classification of finite monoids: The language approach. *Theoretical Computer Science (TCS)*, 14:195–208, 1981.

[Tho82] Wolfgang Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

[Thu06] Axel Thue. *Über unendliche Zeichenreihen*. Skrifter udg. af Videnskabs-Selskabet i Christiania : 1. Math.-Naturv. Klasse. Dybwad [in Komm.], 1906.

[Tig] Tigergraph. www.tigergraph.com.

[TW98] Denis Thérien and Thomas Wilke. Over words, two variables are as powerful as one quantifier alternation. In *ACM Symposium on Theory of Computing (STOC)*, pages 234–240. ACM, 1998.

[W3C13] SPARQL 1.1 query language. https://www.w3.org/TR/sparql11-query/, 2013. World Wide Web Consortium.

[Wik] Wikidata. wikidata.org.

[Yen71] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.