A Trichotomy for Regular Trail Queries

Wim Martens
University of Bayreuth, Germany
wim.martens@uni-bayreuth.de

Matthias Niewerth
University of Bayreuth, Germany
matthias.niewerth@uni-bayreuth.de

Tina Trautner
University of Bayreuth, Germany
tina.trautner@uni-bayreuth.de

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 paper 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 et al. [5]. 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.

2012 ACM Subject Classification Information systems → Query languages for non-relational engines; Information systems → Information retrieval query processing; Theory of computation → Problems, reductions and completeness; Theory of computation → Regular languages

Keywords and phrases Regular languages, query languages, path queries, graph databases, databases, complexity, trails, simple paths

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.7

Related Version A full version of the paper is available at [21], `https://arxiv.org/abs/1903.00226`.

Funding Deutsche Forschungsgemeinschaft (DFG), grant MA 4938/4-1.

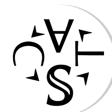
Acknowledgements We thank the participants of Shonan meeting No. 138 (and Hassan Chafi in particular), who provided significant inspiration for the first paragraph in the Introduction, Jean-Éric Pin for pointing us to positive C_{ne}-varieties of languages, and Jean-Éric Pin and Luc Segoufin for their help with the proof of Proposition 3.13(b). Furthermore, we want to thank the anonymous reviewers of STACS 2020 for useful comments.

1 Introduction

Graph databases are a popular tool to model, store, and analyze data [25, 33, 27, 35, 12]. 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.

© Wim Martens, Matthias Niewerth, and Tina Trautner;
licensed under Creative Commons License CC-BY

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Christophe Paul and Markus Bläser; Article No. 7; pp. 7:1–7:16



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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 paper, 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 [23, 6, 7, 20, 3] and SPARQL [34] focus on the first and last node of matching paths, Cypher [26] returns the entire paths. G-Core, a recent proposal by partners from industry and academia, sees paths as "first-class citizens" in graph databases [2].

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 [26, 14], 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 paper, 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 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 $\mathbf{FO}^2[<]$, $\mathbf{FO}^2[<, +]$, $\mathbf{FO}[<]$ (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 [29] and were recently discovered to appear in public query logs [10, 9]. Furthermore, every *single-occurrence regular expression* [8] 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 [11] 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

---

[1] Bagan et al. [5] called the class $\mathsf{C}_{\mathrm{tract}}$, 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.

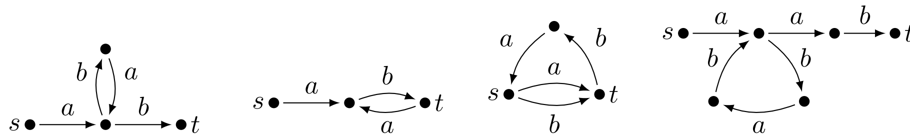


Figure 1 Directed, edge-labeled graphs that have a trail from s to t.

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 [36] 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 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 [23] 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. [13] and LaPaugh and Papadimitriou [19].

Our work is most closely related to the work of Bagan et al. [5] 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 paper 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 [22] refined this trichotomy of Bagan et al. [5] for *simple transitive expressions*, by analyzing the complexity where the input consists of both the expression and the graph.

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 [28, 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 paper 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 [19, 23], but – assuming that a and b-edges are different – 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 two rightmost graphs, there is exactly one path labeled $(ab)^*$, which is also a trail.)

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. A *word* is a finite sequence $w = a_1 \cdots a_n$ of symbols.

We consider edge-labeled directed graphs $G = (V, E)$, where V is a finite set of nodes and $E \subseteq V \times \Sigma \times V$ is a set of (labeled) edges. A *path* p from node s to t is a sequence $(v_1, a_1, v_2)(v_2, a_2, v_3) \cdots (v_m, a_m, v_{m+1})$ with $v_1 = s$ and $v_{m+1} = t$ and such that $(v_i, a_i, v_{i+1}) \in E$ for each $i \in [m]$. By $|p|$ we denote the number of edges of a path. A path is a *trail* if all the edges (v_i, a_i, v_{i+1}) are different and a *simple path* if all the nodes v_i are different. (Notice that each simple path is a trail but not vice versa.) We denote $a_1 \cdots a_m$ by $\mathrm{lab}(p)$. Given a language $L \subseteq \Sigma^*$, path p *matches* L if $\mathrm{lab}(p) \in L$. 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.

By $\delta(q, w)$ we denote the states reachable from state q by reading w. 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 $\mathrm{Loop}(q)$ 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 $w^{-1}L = \{w' \mid ww' \in L\}$.

A DFA is an NFA such that I is a singleton and for all $q \in Q, \sigma \in \Sigma$ $|\delta(q, \sigma)| \leq 1$. 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. 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 of an aperiodic language L does not have simple cycles labeled w^k for $k > 1$ and $w \neq \varepsilon$. Thus, for "large enough n" we have: $uw^n v \in L$ 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 [31].)

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

$\mathsf{RTQ}(L)$	
Given:	A graph $G = (V, E)$ 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. [5], is the *RSPQ problem*. The $\mathsf{RSPQ}(L)$ 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 $\mathsf{RTQ}(L)$ 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* [16]. Furthermore, they can be defined by a clean class of regular expressions (which was shown by Jullien [18] and later rediscovered by Abdulla et al. [1]), 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 [23], 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 $L_{q_2} \subseteq L_{q_1}$.[2] 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** ([1, 16, 18, 23])**.** *The following are equivalent:*
(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 $q_1, q_2 \in Q$:*

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

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 note that every downward closed language L satisfies the left-synchronized containment property.

▶ **Definition 3.4.** *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 $w_\ell w_1^n w_2^n w_r \in L$.*

[2] 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.

We note that Definition 3.4 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 3.4.

Next, we show that all conditions defined in Definitions 3.3 and 3.4 are equivalent for DFAs.

▶ **Theorem 3.5.** *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.*

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 3.6.** *A regular language L belongs to $\mathsf{T}_{\mathsf{tract}}$ if L satisfies one of the equivalent conditions in Theorem 3.5.*

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 3.7.** *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)* [8]. SOREs were studied in the context of learning schema languages for XML [8], since they occur very often in practical schema languages.

3.3 A Syntactic Characterization

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 3.8.** *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.*

The following theorem 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) emposes an additional restriction on condition (4), and we later use it to prove that $\mathsf{T}_{\mathsf{tract}} \subseteq \mathbf{FO}^2[<,+]$.

▶ **Theorem 3.9.** *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.*

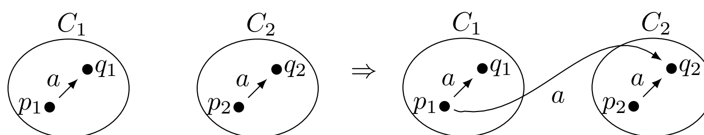


Figure 2 Consistent jump condition (simplified, i.e.: without preconditions, counter and update) used in Theorem 3.12. C_1 and C_2 are components (not necessarily different) such that C_2 is reachable from C_1.

We use *finite automata with counters or CNFAs* from Gelade et al. [15], that we slightly adapt to make the construction easier.[3] For convenience, we provide a full definition in Appendix A. 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 $k \in \mathbb{N}$;[4]

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.[5] 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 3.9. The implications (3) ⇒ (2) and (5) ⇒ (4) are trivial. We sketch the proofs of (1) ⇒ (5) ⇒ (3) and (4) ⇒ (2) ⇒ (1) below, establishing the theorem.

(1) ⇒ (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 4.3 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) ⇒ (3) and (4) ⇒ (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) ⇒ (1): One can show that the left-synchronized containment property is invariant under the powerset construction. ◀

[3] The adaptation is that we let counters decrease instead of increase. Furthermore, it only needs zero-tests.
[4] If q_2 is in a trivial component, then k should be 0 for the transition to be useful.
[5] The values of P and U depend on whether C_1 is the same as C_2 or not.

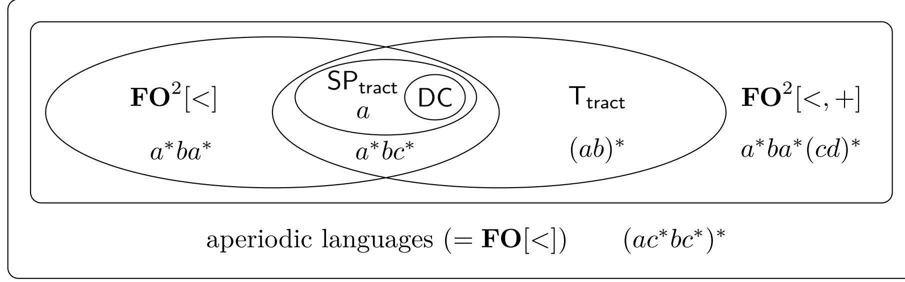


Figure 3 Expressiveness of subclasses of the aperiodic languages.

3.4 Comparison to Other Classes

We compare $\mathsf{T}_{\mathsf{tract}}$ to some closely related and yardstick languages to get an idea of its expressiveness. For example, every downward closed (DC) language is in $\mathsf{T}_{\mathsf{tract}}$, since $\mathsf{T}_{\mathsf{tract}}$ relaxed the containment property.

Bagan et al. [5] 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 3.10** (Theorem 2 in Bagan et al. [5])**.** *Let L be a regular language.*
(1) *If L is finite, then RSPQ$(L) \in AC^0$.*
(2) *If $L \in \mathsf{SP}_{\mathsf{tract}}$ and L is infinite, then RSPQ(L) is NL-complete.*
(3) *If $L \notin \mathsf{SP}_{\mathsf{tract}}$, then RSPQ$(L)$ is NP-complete.*
One characterization of $\mathsf{SP}_{\mathsf{tract}}$ is the following (Theorem 4 in [5]):

▶ **Definition 3.11.** $\mathsf{SP}_{\mathsf{tract}}$ *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 $w_\ell w_1^i w_2^i w_r \in L$.*

From this definition it is easy to see that every language in $\mathsf{SP}_{\mathsf{tract}}$ is also in $\mathsf{T}_{\mathsf{tract}}$, since our definition imposes an extra "synchronizing" condition on w_1 and w_2, namely that they share the same first (or last) symbol (Definition 3.4). We now fully classify the expressiveness of $\mathsf{T}_{\mathsf{tract}}$ and $\mathsf{SP}_{\mathsf{tract}}$ compared to yardsticks as DC, $\mathbf{FO}^2[<]$, and $\mathbf{FO}^2[<,+]$ (see also Figure 3).

Here, $\mathbf{FO}^2[<]$ and $\mathbf{FO}^2[<,+]$ are the two-variable restrictions of $\mathbf{FO}[<]$ and $\mathbf{FO}[<,+]$ over words, respectively. By $\mathbf{FO}[<,+]$ 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). $\mathbf{FO}[<]$ is $\mathbf{FO}[<,+]$ without the binary predicate $+1$.

▶ **Theorem 3.12.**
(a) $\mathsf{DC} \subsetneq \mathsf{SP}_{\mathsf{tract}} \subsetneq (\mathbf{FO}^2[<] \cap \mathsf{T}_{\mathsf{tract}})$
(b) $\mathsf{T}_{\mathsf{tract}} \subsetneq \mathbf{FO}^2[<,+]$
(c) $\mathsf{T}_{\mathsf{tract}}$ *and* $\mathbf{FO}^2[<]$ *are incomparable*
Since $\mathbf{FO}^2[<,+] \subsetneq \mathbf{FO}[<]$, we also have $\mathsf{T}_{\mathsf{tract}} \subsetneq \mathbf{FO}[<]$. Thus, every language in $\mathsf{T}_{\mathsf{tract}}$ is aperiodic.

Next, we show where $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ are in the concatenation hierarchy (also known as Straubing-Thérien hierarchy) and the dot-depth hierarchy (also known as Brzozowski hierarchy).

▶ **Proposition 3.13.**
(a) $\mathsf{SP}_{\mathsf{tract}}$ *is in $\mathcal{V}_{3/2}$, the 3/2th level of the concatenation hierarchy.*
(b) *Every language L in $\mathsf{T}_{\mathsf{tract}} \cap \Sigma^+$ is in $\mathcal{B}_1$, the 1st level of the dot-depth hierarchy.*

Thus Proposition 3.13 implies that every language in $\mathsf{SP}_{\mathsf{tract}}$ can be described by a formula in $\Sigma_2[<]$ and every language in tractable language $\mathsf{T}_{\mathsf{tract}}$ by a boolean combination of formulas in $\Sigma_1[<, +, \min, \max]$, see Pin [30, Theorem 4.1].

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.*

We will prove Theorem 4.1 only for simple graphs, but it extends to graphs with multi-edges, which are graphs with multiple edges with the same label between the same nodes. Equivalently, this can be seen as a variation of the problem where edges are accompanied with numbers that say how often they can be traversed. For example, we could say that e_1 may be used at most twice, while e_2 may be used at most 30 times. Here, the number of occurrences of edges can even be encoded in binary. We discuss this in Section 4.4.

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 an FO-formula. Since we can also test in FO that no edge is used more than once, the graphs for which $\mathsf{RTQ}(L)$ holds are FO-definable. By Immerman [17], 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 [5, Theorem 2]. 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 first describe the NL algorithm. Then we observe that, if the algorithm answers "yes", we can also output a shortest trail. 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 *path summary*. We will define path 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. Let $A = (Q, \Sigma, I, F, \delta)$ be an NFA. We extend δ to paths, in the sense that we denote by $\delta(q, p)$ the set of states that A can reach from q after reading $\mathrm{lab}(p)$. For $q_0 \in Q$, we say that a *run from* q_0 of A over a path $p = (v_1, a_1, v_2)(v_2, a_2, v_3) \cdots (v_m, a_m, v_{m+1})$ is a sequence $q_0 \cdots q_m$ of states such that $q_i \in \delta(q_{i-1}, a_i)$, for every $i \in [m]$. When A is a DFA and q_0 its initial state, we also simply call it *the run of A over p*.

▶ **Definition 4.2.** *Let $p = e_1 \cdots e_m$ be a path and $r = q_0 \cdots q_m$ the run of A_L over 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. To this end, we use the following synchronization property for A_L.

▶ **Lemma 4.3.** *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 $\delta_L(q_1, w) = \delta_L(q_2, w)$.*

The lemma motivates the use of *summaries*, which we define next.

▶ **Definition 4.4.** *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 4.5.** *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 and all edges appearing in S are distinct. 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 4.3, 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 4.6.** *Let L be a language in $\mathsf{T}_{\mathsf{tract}}$. 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 lemma, Corollary 4.6 can be used to obtain an NL algorithm 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 4.7.** *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 an NL 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.*

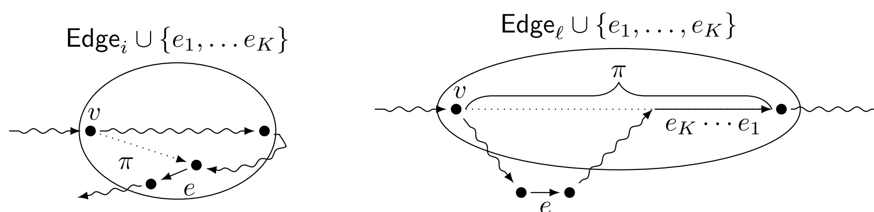


Figure 4 Sketch of case (1) and (2) in the proof of Lemma 4.10.

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 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. [5, Definition 5]. 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 simple 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 $\delta_L(q, \pi) \in C$.

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 $p_1, \ldots, p_k$ 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 $\mathsf{RTQ}(L)$, 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 4. We construct the new trail by substituting the respective subtrail with π. ◀

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 graph $G = (V, E)$, 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 $L \in \mathsf{T}_{\mathsf{tract}}$.*

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

▶ **Corollary 4.13.** *Let $L \in \mathsf{T}_{\mathsf{tract}}$, G be a graph, 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).*

4.3 Languages not in $\mathsf{T}_{\mathsf{tract}}$

The proof of Theorem 4.1(3) is by reduction from the following NP-complete problem:

	TwoEdgeDisjointPaths
Given:	A language L, a graph $G = (V, E)$, 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. [5, Lemma 2] (which is a reduction from the two vertex-disjoint paths problem).

4.4 Extension to Multigraphs

We believe that Theorem 4.1 can be extended to graphs with multi-edges. For each edge e, denote by max_e the number of occurrences of e in the multigraph. First, consider the case where L is a finite language. Let m be the length of longest word in L. Notice that m is a constant, since it only depends on L. Every edge can be used at most m times in paths that match L, so we can check in FO whether an edge e is used at most $n_e = min\{m, max_e\}$ times. The rest of the argument is analogous to Section 4.1.

We now turn to languages in $\mathsf{T}_{\mathsf{tract}}$. The length of the candidate summaries S (Definition 4.5) only depends on L and is therefore constant. Instead of testing whether all edges appearing in S are distinct, we have to check if they occur at most the maximal number of times. (Therefore, listing all candidate summaries is still in $O((\log |G|)^N)$, and thus in polynomial time.) For the local edge domains (Definition 4.8), we define E_1 as an ordinary graph, i.e., non-multigraph, containing all edges that have not exhausted their maximal number of occurrences in S already. With this graph we can continue just as for ordinary graphs.

5 Recognition and Closure Properties

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

▶ **Theorem 5.1.** *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.*

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 [32, 24] has no subword that is a cube (i.e., no subword of the form w^3) [32, 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 $\mathsf{T}_{\mathsf{tract}}$.

▶ **Lemma 5.2.** *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.*

This lemma implies that $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ each are a positive C_{ne}-variety of languages, i.e., a positive variety of languages that is closed under inverse non-erasing homomorphisms.

▶ **Lemma 5.3.** *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 5.2 (vi) that there do not exist best lower or upper approximations for regular languages outside $\mathsf{SP}_{\mathsf{tract}}$ or $\mathsf{T}_{\mathsf{tract}}$.

▶ **Corollary 5.4.** *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 [8] 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 [36] transfers to the setting of enumerating trails matching L.

▶ **Theorem 6.1.** *Let L be a regular language, G be a graph 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 [36] that enumerates the k shortest simple paths for some given number k, similar to what was done by Martens and Trautner [22, Theorem 18]. It uses the algorithm from Corollary 4.13 as a subprocedure. ◀

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 tradeoffs 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 [29] and appear in query logs [10, 9].
- Both $\mathsf{SP}_{\mathsf{tract}}$ and $\mathsf{T}_{\mathsf{tract}}$ can be syntactically characterized but, currently, the characterization for $\mathsf{SP}_{\mathsf{tract}}$ (Section 3.5 in [5]) 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 [11] 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 directed graphs: it only has bidirectional navigation of the form $(a + \hat{}a)$. It has been open problem since 1991 whether evaluating $(aaa)^*$ on undirected graphs is in P or NP-complete [4].

References

1 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.

2 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.

3 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.

4 Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *J. ACM*, 38(2):255–274, 1991.
5 Guillaume Bagan, Angela Bonifati, and Benoît Groz. A trichotomy for regular simple path queries on graphs. In *Symposium on Principles of Database Systems (PODS)*, pages 261–272, 2013.
6 Pablo Barceló. Querying graph databases. In *Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013.
7 Pablo Barceló, Leonid Libkin, and Juan L. Reutter. Querying graph patterns. In *PODS*, pages 199–210. ACM, 2011.
8 Geert Jan Bex, Frank Neven, Thomas Schwentick, and Stijn Vansummeren. Inference of concise regular expressions and dtds. *ACM Trans. Database Syst.*, 35(2):11:1–11:47, 2010.
9 Angela Bonifati, Wim Martens, and Thomas Tim. Navigating the maze of wikidata query logs. In *The Web Conference (WWW)*. ACM, 2019. To appear.
10 Angela Bonifati, Wim Martens, and Thomas Timm. An analytical study of large SPARQL query logs. *PVLDB*, 11(2):149–161, 2017.
11 Angela Bonifati, Wim Martens, and Thomas Timm. DARQL: deep analysis of SPARQL queries. In *WWW (Companion Volume)*, pages 187–190. ACM, 2018.
12 Dbpedia. `https://wiki.dbpedia.org`.
13 Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science (TCS)*, 10(2):111–121, 1980.
14 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 *SIGMOD Conference*, pages 1433–1445. ACM, 2018.
15 Wouter Gelade, Marc Gyssens, and Wim Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM J. Comput.*, 41(1):160–190, 2012.
16 Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969.
17 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.
18 Pierre Jullien. *Contribution à l'étude des types d'ordres dispersés*. PhD thesis, Universite de Marseille, 1969.
19 Andrea S. LaPaugh and Christos H. Papadimitriou. The even-path problem for graphs and digraphs. *Networks*, 14(4):507–513, 1984.
20 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.
21 Wim Martens, Matthias Niewerth, and Tina Trautner. A trichotomy for regular trail queries. *CoRR*, abs/1903.00226, 2019. `arXiv:1903.00226`.
22 Wim Martens and Tina Trautner. Evaluation and enumeration problems for regular path queries. In *ICDT*, volume 98 of *LIPIcs*, pages 19:1–19:21. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.
23 Alberto O. Mendelzon and Peter T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, December 1995.
24 Harold Marston Morse. Recurrent geodesics on a surface of negative curvature. *Transactions of the American Mathematical Society*, 22(1):84–100, January 1921. `doi:10.2307/1988844`.
25 Neo4j. `https://neo4j.com/`.
26 Cypher query language reference, version 9, mar. 2018. `https://github.com/opencypher/openCypher/blob/master/docs/openCypher9.pdf`.
27 Oracle spatial and graph. `https://www.oracle.com/database/technologies/spatialandgraph.html`.
28 Yehoshua Perl and Yossi Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, 25(1):1–9, 1978.

29 Neo4J Petra Selmer. Personal communication.
30 Jean-Éric Pin. The dot-depth hierarchy, 45 years later. In *The Role of Theory in Computer Science*, pages 177–202. World Scientific, 2017.
31 Marcel Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.
32 Axel Thue. *Über unendliche Zeichenreihen*. Skrifter udg. af Videnskabs-Selskabet i Christiania : 1. Math.-Naturv. Klasse. Dybwad [in Komm.], 1906.
33 Tigergraph. `https://www.tigergraph.com/`.
34 SPARQL 1.1 query language. `https://www.w3.org/TR/sparql11-query/`, 2013. World Wide Web Consortium.
35 Wikidata. `https://www.wikidata.org/`.
36 Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

A Background on NFAs with Counters

We recall the definition of counter NFAs from Gelade et al. [15]. 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 A.1.** *A* non-deterministic 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 is of the form $c := k$ with $k \leq \tau$.*

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.)